UNIVERSITY OF SALERNO

DEPARTMENT OF COMPUTER SCIENCE

MASTER DEGREE IN COMPUTER SCIENCE

THESIS IN SOFTWARE ENGINEERING AND IT MANAGEMENT

Towards Understanding and Measuring the Health of Open-Source Software Development Communities

SUPERVISORS
Prof. Fabio Palomba
Dott. Stefano Lambiase
University of Salerno

CANDIDATE
Gianmario Voria
Mat. 0522501337

ACADEMIC YEAR 2022-2023

This thesis was realised in the

lab
sesa

SOFTWARE ENGINEERING
SALERNO

Abstract

Software development, particularly in the open-source context, is a social effort since such communities are naturally geo-dispersed and with no strict management protocols. Despite the critical role of communication and collaboration in projects' success, not enough attention is paid to these social aspects. Yet, communities often show some common social or organizational patterns, which have been defined as community patterns, and it has been shown that they are good indicators of community health. From a socio-technical point of view, communication and relations between developers of such communities are not trivial to measure, given their particular structure. Our objective is to identify underlying relations between two community-related aspects of open-source software development projects, i.e., community patterns and community smells. To observe community patterns, we implemented and validated TOAD (Towards Open-source communities health Analysis and Diagnosis), a tool capable of identifying common patterns in open-source GitHub communities. This tool is based on YOSHI (Yielding Open-Source Health Information), which has become unusable due to outdated APIs. To observe community smells, we used CSDETECTOR, an automated tool able to detect community smells in GitHub communities. We concluded our study by performing association-rule mining between community patterns and community smells, discovering relations between the two.

Contents

Contents **i**

List of Figures **iv**

List of Tables **v**

1 Introduction **1**

1.1 Context 1

1.2 Motivation and challenges 2

1.3 Objectives 2

1.4 Results 3

1.5 Thesis structure 3

2 State of the Art **4**

2.1 Software development community health 4

2.2 Community patterns 6

2.3 Community smells 8

2.4 Relations between community patterns and smells 11

3 TOAD: Towards Open-source communities health Analysis and Diagnosis 12

3.1 TOAD . . . 12

3.2 Technical details and Metrics computation . . . 16

3.2.1 Algorithmic Solution . . . 16

3.2.2 Structure . . . 17

3.2.3 Formality . . . 19

3.2.4 Longevity . . . 20

3.2.5 Engagement . . . 21

3.2.6 Dispersion . . . 22

3.3 Architecture . . . 26

3.4 Differences with YOSHI . . . 27

3.5 Limitations . . . 29

4 Research Methodology 34

4.1 Hypotheses and Research Questions . . . 35

4.2 Reseach Strategy . . . 36

4.3 RQ1: Validating TOAD . . . 37

4.4 RQ2: Relations between Community Patterns and Community Smells 41

5 Results 44

5.1 Validation of TOAD . . . 44

5.1.1 Comparison of Community Patterns . . . 45

5.1.2 Comparison of Metrics . . . 45

5.2 Analysis of relations . . . 48

5.3 Observations . . . 54

6 Threats to validity 55

6.1 Internal Validity . . . 56

6.2 Construct Validity 56
6.3 Reliability 56
6.4 External Validity 57

7 Conclusions and Future Work **58**
7.1 Conclusions 58
7.2 Future work 59

Bibliography **61**

List of Figures

3.1 The decision tree for organizational structures [1] from which the algorithm of TOAD has been extracted. 17
3.2 The architecture of TOAD 25

4.1 Overview of the research methodology. 37
4.2 Methodology to validate TOAD's results. 38
4.3 Methodology to mine relations between community patterns and smells. 42

List of Tables

2.1 Overview of community patterns. 7

2.2 Overview of community smells. 9

3.1 List of thresholds used to compute community patterns in TOAD . . 33

4.1 List of communities with characteristics analyzed by YOSHI 40

4.2 List of metrics computed by TOAD 41

5.1 Community patterns inferred by YOSHI and TOAD for the considered communities. Every analyzed community is considered an SN, and therefore SN has not been included in the table. 46

5.2 Community patterns inferred by YOSHI and how they contradict the reported empirical thresholds. 47

5.3 Metrics computed by YOSHI 49

5.4 (1/2) Community smells for the repositories analyzed (1 if present, 0 otherwise). 49

5.5 (2/2)Community smells for the repositories analyzed (1 if present, 0 otherwise). 50

5.6 (1/2) Community patterns for the repositories analyzed (1 if present, 0 otherwise). 51
5.7 (2/2) Community patterns for the repositories analyzed (1 if present, 0 otherwise). 52
5.8 Results of the association rule mining showing relations between community patterns and community smells. 53

CHAPTER 1

Introduction

1.1 Context

Software engineering is a social activity, and in their activities, software engineers give birth to communities. Among these communities, the most problematic ones are usually open-source software development communities. Such networks of developers usually arise naturally and involve people contributing to the same software project without managerial activities. Studies have shown the necessity of maintaining such communities and providing automated or semi-automated solutions to monitor their health and other socio-technical characteristics [2, 3]. Tamburri et al. proposed YOSHI (Yielding Open-Source Health Information), a tool able to map open-source software development communities into *community patterns*, i.e., sets of organizational and social structure types and characteristics [4].

Concerning the health of communities, Tamburri et al. focused their analysis on what causes *social debt*, i.e., unforeseen project costs connected to sub-optimal behaviors in the development community [5, 6]. In doing so, they introduced *community* *smells*, i.e., sets of socio-technical characteristics that may lead to the emergence of social debt.

1.2 Motivation and challenges

Many tools have been proposed in the literature to analyze the health of communities, from the perspective of community patterns [4] and community smells [7, 8]. Analyzing these aspects separately, though, looks like a wasted opportunity. Both analyses aim at understanding the underlying behaviors of communities: patterns try to understand how communities work from a high-level perspective, while smells try to understand communication and interaction-wise how the teams behave. Nonetheless, no one studied if the type of community from an organizational point of view could lead to sub-optimal social interactions that could lead to social debt.

1.3 Objectives

We aim to understand and monitor health information about open-source software development communities from a wider point of view, trying to understand whether there are underlying relations between community patterns and community smells. This will be done by analyzing several open-source communities with semi-automated tools to gather such information, and then mining relations with an association-rule process. In doing so, we will partially replicate YOSHI creating a new tool, named **TOAD: Towards Open-source communities health Analysis and Diagnosis**, that we will use to gather community patterns.

◎ **Our Goal.** Understand the underlying relations between community patterns and community smells.

1.4 Results

Finally, we were able to implement TOAD and validate its pattern identification. Also, we found many relations between patterns and smells, and we also analyzed their possible meaning. The main result of this thesis work is that we demonstrated how the organizational and social structure of open-source communities—health—can lead to social debt due to the relations between such patterns and community smells.

1.5 Thesis structure

The topics mentioned above will be detailed in the following chapters, which will be organized as follows:

- **Chapter 2 - State of the Art**, which contains the currently available works and discoveries regarding the field of interest of this thesis work;
- **Chapter 3 - Tool**, that will describe how the tool TOAD has been implemented and validated;
- **Chapter 4 - Research Methodology**, that will be the core of this work, illustrating the research questions and strategies applied to achieve the main objective;
- **Chapter 5 - Results**, in which the achieved results of the proposed research will be explained, analyzing such results and discussing observations;
- **Chapter 6 - Threats to validity**, which illustrates the threats to the validity of the study and the way they have been mitigated;
- **Chapter 7 - Conclusions**, which concludes the journey, summarizing the proposed research and showing a glimpse of future contributions that can follow the current work.

CHAPTER 2

State of the Art

This chapter summarizes the work that has already been developed on the definition of solutions to understand and measure health information of software development communities, outlining the state of the Art. Only some of the works on these areas are presented below, representing only a tiny part of the entire state of research.

2.1 Software development community health

Software development communities have been defined as specific varieties of social networks in which particular conditions hold, such as informal communication through open-source projects' communication channels [9]. In such definition, *community members* are people who interact in any way or shape within the context of pursuing a common objective, i.e., the development of a software product. Many works in current literature highlight the need to maintain the population of software communities to grant its sustainability [3] [10]. Hata et al. introduced the software population pyramid, discovering patterns of communication and contributions in

GitHub open-source projects, and highlighting how a well-prepared project structure, e.g., with the usage of clear documentation, and the hiring of new developers, helps the sustainability of such projects [3]. Crowston and Howison stated that healthy open-source software development communities are onion-shaped, i.e., the community shows significant differences in roles, having core developers and leaders at its center surrounded by layers of other developers, active users, and passive users [11]. They also found out that, to identify healthy communities, you can look at how members deal with heavy and difficult tasks: ideally, a healthy community should recognize and address such issues by itself.

Many papers have tried to understand which metrics could be used to measure open-source community health. In their work, Goggins et al., to understand how healthy a project can be based on analytical metrics, examined the results of the work made by the Linux Foundation named CHAOSS (Community Health Analytics Open Source Software)[1] in the first four years of its life-cycle, and they concluded that its metrics can be operationalized to understand open-source community health [12]. In a recent study, Paxton et al. performed a cognitive science study to understand patterns in communication and social aspects of open-source communities, analyzing text's sentiment and positive expressions among members and discovering how such attributes contribute to community health [13].

Researchers have shown that community health is a highly relevant factor for the success of a software development project. Catolino et al. stated that different community types, for example in terms of formality, show different outputs in terms of quality of the whole software production process and its maintenance [14]. Coelho and Valente performed a study to understand why open-source software projects fail. They provided many different reasons, and they concluded that maintenance practice among community members has a high influence on project success [2]. Kwan et al. found out that socio-technical congruence, i.e., the extent of alignment

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

between the social and the technical structure of the community [16], has an effect on projects' success. Palomba et al. studied how community-related aspects influence the presence of code smells, thus leading to technical debt, i.e., the unforeseen cost given by improper management of communication and coordination between developers.

2.2 Community patterns

The strong relationship between community health and software quality highlights the necessity of understanding how communities work from the inside and how such communities can be classified, with the aim of common *patterns*. Nakakoji et al., in their revolutionary work, studied evolution aspects in open-source software development not only from a technical point of view but also considering the associated communities, proposing a classification of such communities in three types: exploration-oriented, utility-oriented and service-oriented [18]. [19], based on its prior study on software population pyramid [10], proposed a classification that considers the different types of contributors in a development community [19]. Yamashita et al. classified communities into four different types based on project stickiness and magnetism, i.e., the tendency of an open-source software project to retain and attract contributors, finding a relationship between community health and such metrics [20].

Table 2.1: Overview of community patterns.

Community Pattern	Definition
Learning Community (LC)	A space for pure learning and explicit sharing of actionable knowledge. Leadership steers community practices, membership is subject to approval. LCs pursue personal and organizational goals equally.
Strategic Community (SC)	Meticulously selected group of people who try to proactively solve problems within strategic business areas of the organizational sponsor.
Informal Community (IC)	Usually sets of people part of an organization, with a common interest, often closely dependent on their practice. Informal interactions, usually across unbound distances.
Knowledge Community (KC)	Groups of people with a shared passion to create, use, and share new knowledge for tangible business purposes.
Problem Solving Community (PSC)	A specific instance of a strategic community focused on a particular problem. Consists generally of geographically and organizationally dispersed employees of the same discipline.
Community of Practice (CoP)	Groups of people sharing a concern, a set of problems, or a passion about a topic, who deepen their knowledge and expertise in this area by interacting frequently in the same geolocation.
Formal Network (FN)	Members are rigorously selected and prescribed by management (often in the form of FG), directed according to corporate strategy and mission.
Social Network (SN)	SNs can be seen as a supertype for all OSSs. To identify an SN, it is sufficient to split the structure of organizational patterns into macrostructure and microstructure.
Informal Network (IN)	Looser networks of ties between individuals that happen to come in contact in the same context. Their driving force is the strength of the ties between members.
Network of Practice (NoP)	A networked system of communication and collaboration connecting CoPs. Anyone can join. They span geographical and time distances alike.
Work Group (WG)	Groups of individuals who regularly work together to attain goals for the benefit of corporate sponsor(s).
Formal Group (FG)	People grouped by corporations to act on (or by means of) them. Each group has an organizational goal, called mission. Compared to FN, no reliance on networking technologies, local in nature.
Project Team (PT)	People with complementary skills who work together to achieve a common purpose for which they are accountable. Enforced by their organization and follow specific strategies or organizational guidelines.

Tamburri et al. defined a set of common patterns observable across different open-source software development communities, namely *community patterns*, that they described as sets of organizational and social structure types and characteristics, showing that each pattern has its characteristics and measurable attribute [4]. In such work, authors proposed a tool, named YOSHI (Yielding Open-Source Health Information), that is able to map open-source software development communities (as GitHub repositories) onto community patterns. The tool uses a decision tree [1] to classify a community using different metrics, using empirically-defined thresholds defined by di Nitto et al. in an ethnographic study performed to understand socio-technical metrics on open-source communities [21].

Table 2.1 reports the list of community patterns defined in the literature and that YOSHI is able to detect.

2.3 Community smells

From the social and human perspective in software engineering, most of the existing literature focused on *social debt*, i.e., unforeseen project costs connected to a suboptimal development community [5][6]. For instance, Tamburri et al. illustrated social debt by comparison with technical debt and discussed common real-life scenarios that exhibit "sub-optimal" development communities [6]. As an evolution of the research on social debt, Tamburri et al. defined *community smells*, i.e., a set of socio-technical characteristics (e.g., high formality) and patterns (e.g., recurrent condescending behavior, or rage-quitting), which may lead to the emergence of social debt [22]. Table 2.2 reports community smells identified by literature [22, 5, 23].

In the last years, community smells have begun to receive particular attention. One of the motivations resides in the development of tools able to detect such smells using various strategies. Tamburri et al. proposed the tool CODEFACE4SMELLS, an augmented version of CODEFACE by Joblin et al. [7], capable of detecting community smells [24]. Moreover, the authors assessed the detection capabilities of CODEFACE4SMELLS using a survey and performing an empirical evaluation on a set of 60 projects. Such activity confirmed the capacity of the tool in the detection

Table 2.2: Overview of community smells.

Community Smell	Definition
Organizational Silo	Siloed areas of the development community that do not communicate, except through one or two of their respective members.
Black Cloud	Excessive information overload due to a lack of structured communication or cooperation governance.
Lone Wolf	Defiant contributor who applies changes in the source code without considering the opinions of her peers.
Radio Silence	One member interposes herself into every interaction across sub-communities.
Prima Donna	Repeated condescending behavior, superiority, constant disagreement, uncooperativeness by one or few members.
Priggish Members	Demanding of others pointlessly precise conformity or exaggerated propriety, especially in a self-righteous or irritating manner.
Code Red	This smell identifies an area of code that is complex, dense, and dependent on 1-2 maintainers who are the only ones that can refactor it.
Unlearning	A new technological or organizational advancement or best practice that becomes infeasible when shared with older members.
Disengagement	Thinking the product is mature enough and sending it to operations even though it might not be ready.
Cognitive Distance	The distance developers perceive on the physical, technical, social, and cultural levels regarding peers with considerable background differences.

of community smells. In addition to that, Palomba et al. used CODEFACE4SMELLS to investigate the impact of such smells over *code smells*, i.e., poor implementation choices applied by developers during software evolution that often lead to critical flaws or failure. [25]. Such a study has reported that community smells represent key factors preventing developers from performing refactoring activities.

A more recent and promising trend is represented by the definition of modeling mechanisms able to describe the future structure of a community and alert project managers of the presence of social debt [26, 27, 28]. As a contribution to this, Almarimi et al. build a multi-label learning model based on genetic algorithms to detect eight common types of community smells. Its evaluation involved 103 open-source projects and 407 community smell instances and resulted in better performance indexes compared to other solutions (F-measure of 89%) [27]. This research led to the development of one of the best tools for the detection of community smells proposed in the literature, CSDETECTOR [8], which uses the above-mentioned model to detect community smells in a given GitHub repository.

Researchers have also investigated the significant factors that influence community smells and the way managers refactor them. Catolino et al. conducted an empirical study investigating the correlation between gender diversity and the emergence of four types of community smells. Such a paper showed how the emergence of community smells might be potentially reduced by the presence of women in development teams [29]. One of the most exciting results consists of women's influence on the emergence of smells correlated to the quality of communication (e.g., *Radio Silence* effect). Moreover, the authors proposed another work concerning refactoring strategies, studying how developers and managers can avoid or fix community smells. [30]. Through a survey conducted with 76 experts, the authors elicited a set of refactoring operations generally applied by practitioners to remove four types of smells. From the identified strategies emerge mentoring, monitoring, creation of a communication plan, conducting cohesion exercises, and restructuring the community. For which concern the impact that such community smells have on the whole software development field, various works have been conducted. Martini and Bosch [31] studied the impact of community smells on architecture debt, while Tamburri et al. [32] on organization structure types.

2.4 Relations between community patterns and smells

Prior to this work, De Stefano et al. performed preliminary work on the relation between community patterns and community smells. They analyzed the community patterns elicited from YOSHI and the community smells detected by CODEFACE4SMELLS, studying their relation through association-rule mining. They discovered several relations between patterns and smells in over 25 communities analyzed. Their results show that specific community smells may arise depending on the peculiarities of the community organization [33].

CHAPTER 3

TOAD: Towards Open-source communities health Analysis and Diagnosis

In this chapter, we discuss the implementation of TOAD: Towards Open-source communities health Analysis and Diagnosis[1]. As already mentioned, this tool is a partial replication of the tool YOSHI [4] by Tamburri et al. that has become unusable due to outdated APIs. **The tool uses empirically defined thresholds to classify open-source software development communities into community patterns on the base of several metrics**. We decided to keep metrics and the threshold as close as possible to the ones implemented in YOSHI while performing slight adjustments to better fit the concepts defined by di Nitto et al. and Tamburri et al..

3.1 TOAD

TOAD is a social network analysis tool able to detect open-source community types illustrated in Table 2.1. It is able to classify 8 of the 13 community types most

[1]`https://github.com/gianwario/TOAD`

relevant [9] by analyzing five key characteristics of open-source communities, namely (1) **structure**, (2) **geodispersion**, (3) **formality**, (4) **engagement** and (5) **longevity**. Note that the original tool YOSHI was also able to compute cohesion as the sixth metric that we did not implement. The reason is that it was only used to detect Work Groups, and Tamburri et al. stated that Formal Network, Informal Network, Network of Practice, and Informal Community are the community types most representative of open-source communities [9]. We speculated it would take too much time to re-implement cohesion, so we prioritized the other five metrics.

Functionalities TOAD's execution and computations are mostly based on public data retrieved from GitHub APIs[2]. To make such API calls, it is necessary to have a GitHub Personal Access Token[3], which is considered as an alternative to using passwords for authentication to GitHub. Instead of requiring this token in each execution, we decided to implement an OAuth Login[4], allowing users to authenticate with GitHub directly from our tool instead of manually generating the Personal Access Token. The authentication process makes it so that the token needed to use APIs will always be permanently stored in a local environment file, not requiring another authentication for future uses until the token expires.

The tool handles the input and output with *.csv* files. After authentication, the user is asked to insert (1) the name and path of an input file, (2) the directory of the output file, and (3) the name of the output file, that will be created and in which the computed community patterns will be reported. The input file must contain the following information separated by a comma:

- *repo_owner*: it represents the owner of the GitHub repository analyzed;

[2]`https://docs.github.com/en/rest?apiVersion=2022-11-28`

[3]`https://docs.github.com/en/authentication/keeping-your-account-and-data-secure/managing-your-personal-access-tokens`

[4]`https://docs.github.com/en/apps/oauth-apps/building-oauth-apps/authorizing-oauth-apps`

- *repo_name*: it represents the name of the GitHub repository analyzed;
- *end_date*: it represents the ending date of the 90-day time window of analysis.

More than one community can be specified in the input file, and each of them will be separately analyzed by TOAD. This is an example of the *input.csv* file:

```
netty,netty,2017-05-01
bundler,bundler,2017-05-01
composer,composer,2017-05-01
```

This is an example of the *output.csv* file containing the community patterns detected by TOAD for the given communities:

```
owner,name,start_date,end_date,SN,NoP,IN,FN,CoP,PT,FG,IC
netty,netty,2017-01-31,2017-05-01,
                True,True,False,True,False,False,False,True
bundler,bundler,2017-01-31,2017-05-01,
                True,False,False,False,True,True,False,True
composer,composer,2017-01-31,2017-05-01,
                True,True,False,True,False,False,False,False

```

TOAD works by computing several metrics and using them to classify the type of community. For each of the analyzed repositories, the tool produces two additional output files:

- *data*: we decided to save in a *JSON* file each of the major metrics computed by TOAD, in order to make more understandable the results of the execution;
- *graph*: since the tool is practically a social network analyzer, we implemented an output module that prints a weighted graph representing a community in two ways: (1) as an image and (2) as a *GEXF* file[5].

The following is an example of the metrics computed for one of the communities shown above. More information about these metrics will be in Section 3.2.

[5] https://gephi.org/

```
{
    "dispersion":
    {
        "geo_distance_variance": 17959025.225662425,
        "avg_geo_distance": 7122.194073794376,
        "cultural_distance_variance": 21.5851795725958
    },
    "engagement":
    {
        "m_comment_per_pr": 0.5,
        "mm_comment_dist": 2,
        "m_watchers": 0.0,
        "m_stargazers": 0.0,
        "m_active": 1.0,
        "mm_commit_dist": 0.0,
        "mm_filecollab_dist": 1.0
    },
    "formality":
    {
        "m_membership_type": 1.8372093023255813,
        "milestones": 271,
        "lifetime": 5466
    },
    "longevity": 21.205882352941178,
    "structure":
    {
        "repo_connections": true,
        "follow_connections": true,
        "pr_connections": true
    }
}

```

Use Case TOAD has been thought and developed with the same goals as YOSHI, but with improved usability and accessibility. Since the tool analyzes open-source repositories, the *Actor* can be any stakeholder of the project who wants to be informed about the healthiness of the community. To use the tool, one has to have a GitHub account. *On success*, the user finds himself with community patterns, metrics, and social network graphs of the analyzed community. As the *main scenario*, a flow of events could be that the owner of an open-source software project installs TOAD and executes it. He/She will authenticate with the GitHub Oauth process, and then the tool will prompt her/him with requests for the input and output file names. After the user inserts the needed inputs, the tool will start computing community patterns, and it will produce the outputs mentioned above.

3.2 Technical details and Metrics computation

As already stated, TOAD is a partial replication of YOSHI. We decided to implement this new tool in Python 3.11[6] for several reasons: (1) it has many available libraries that would help us during the implementation phase, (2) it allows us to interact with complex data structures easily, (3) it will make the tool easier to maintain and (4) libraries and APIs will be continuously updated and we will not incur in the risk of having the tool not working for such reason.

TOAD uses the GitHub REST API to retrieve the necessary GitHub data for a given repository, within a 3-month time window. This is estimated as the 90 days before the specified end date. Only data between this time window, e.g., commits, comments, and activities, will be analyzed. Observing organizational activity within a 3-month time window is a common practice in organization research [34].

YOSHI's code is poorly documented and its GitHub public repository is difficult to read. For this reason, we based our implementation of TOAD on the design described in Tamburri et al.'s work [4] and in di Nitto et al.'s work [21].

3.2.1 Algorithmic Solution

Algorithm 1 shows how TOAD detects the community patterns based on the values computed for the five characteristics. The thresholds used are listed in Table 3.1. Such values, also used in YOSHI [4], were extracted from the ethnographic study by di Nitto et al. [21]. It is important to notice that some of these thresholds were

[6]`https://docs.python.org/3.11/`

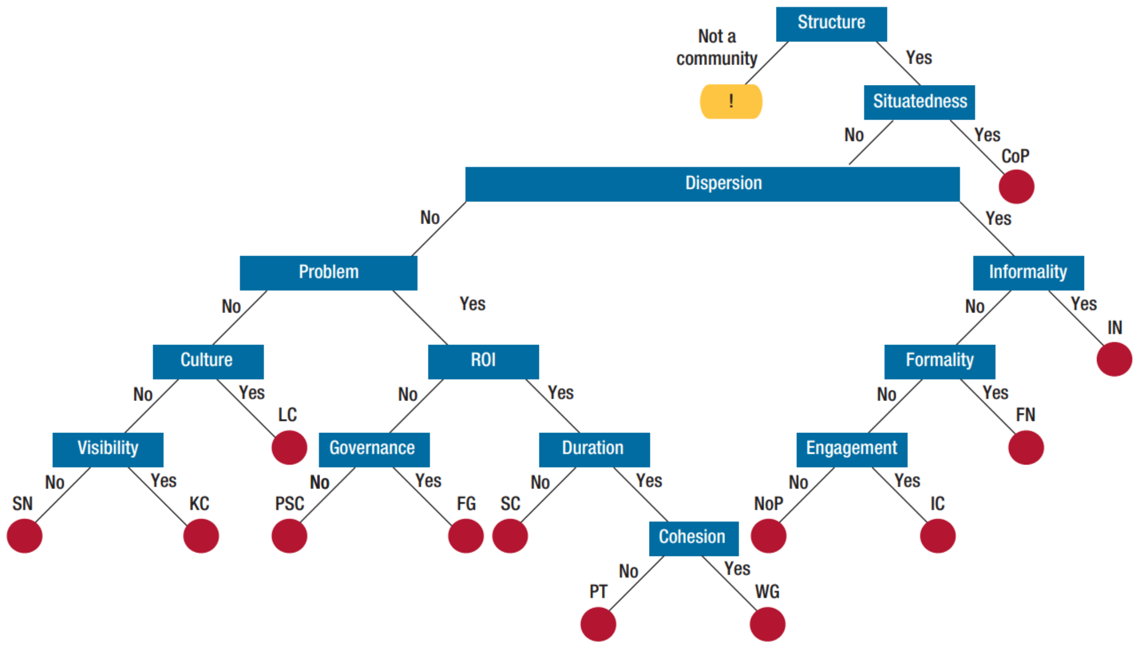

Figure 3.1: The decision tree for organizational structures [1] from which the algorithm of TOAD has been extracted.

modified by YOSHI's authors. For example, in the ethnographic study, there are separate thresholds for geographical distance and cultural distance when computing geodispersion, instead of a single value that is reported for YOSHI [4]. We decided to keep such values, other than metrics computation, as close as possible to di Nitto et al.'s original work. The algorithm comes from a prior work by Tamburri et al., in which they identified a decision tree, shown in Figure 3.1, for organizational structures [1]. Although this decision tree is not strictly followed in both implementations, it represents the base of the algorithmic solution of YOSHI and TOAD.

3.2.2 Structure

Structure is the key characteristic in the detection of community patterns. If a community is not structured, it cannot be considered a Social Network (SN), hence it is essential to evaluate the open-source software community. Community structure has been defined by Newman and Girvan as a property of a network in which connections within groups are dense but sparser between. TOAD computes structure in the same way that YOSHI does, but we added more details in the process: the community is represented by means of a graph, weighted in our case, in which nodes are community members and edges are social or organizational interactions between them. We decided to add weight to these edges, giving more importance to the connection the more interactions there are between developers. This information has not been operationalized in the computation of community patterns, since it is not considered in the original threshold evaluation, but it is left as future work to see how this weight can be used to evaluate the structure of a community.

Detection Method To evaluate the structure of a community, TOAD computes a network in which members are nodes and interactions between members are edges. A member of the community is considered if it has altered the source code of the software project in the 90-day time window analyzed. An edge between two members is added if one of the following relations exists:

- *Common Projects*: When the two members have at least one common repository in which they are contributing, and the weight equal to their number is added to the edge;
- *Follower/Following Relation*: When one of the two members (x) follows or is followed by the other one (y), the weight of the edge is incremented by 1 if only x follows y and by 2 if both x follows y and y follows x;
- *Pull Request Interaction*: When one member is the author of a pull request and another member comments, and therefore contributes to the pull request, and the weight of the edge is incremented by 1 for each pull request in common.

Structure

A structure exists if there is at least one edge in the computed social network, i.e., if one of the following relations exists between two community members: (1) Common Projects, (2) Follower/Following Relation, and (3) Pull Request Interaction.

3.2.3 Formality

Formality represents the level of control exercised or self-imposed on the community, i.e., access privileges, milestones, and regularity of contributions. This metric is essential for the identification of Formal Groups (FG) and Formal Networks (FN). As well as YOSHI, we did not operationalize informality, considering formality and informality as mutually exclusive and thus allowing this metric for the identification of Informal Networks (IN). TOAD implements such metric according to the original design. We consider a community to be formal based on the average number of privileged collaborators divided by the amount of structured work they have been able to carry out.

Detection Method To evaluate the formality of a community, as stated above, we need to establish access privileges, milestones, and lifetime of the project. For access privileges we mean the membership type of a community member, which can be *contributor* or *collaborator*. Collaborators have more privileges than contributors, such as read and write access to the repository, and have been invited by the project owner. A contributor does not have collaborator access but has contributed to the repository. Since we had no way of establishing membership types from GitHub APIs, we had to do an estimation based on their activities on the repository. The following three values were used to compute formality:

- *Mean Membership Type (MMT)*: MMT is computed by assigning each collaborator +2 and assigning each contributor +1, and then computing the mean;

- *Milestones (MS)*: the number of milestones created and used by the community;
- *Lifetime (LT)*: the number of days between the first and the last commit;

Formality

Formality is computed with the following equation:

$$FormalityLevel : \frac{MMT}{MS/LS}$$

That is, the formality level is computed as the Mean Membership Type, MMT, divided by the total number of milestones, MS, per project lifetime, LT.

3.2.4 Longevity

Longevity is represented by measuring how long committers—members—are a part of the community. It is the key characteristic to identify Project Teams (PT). We operationalized longevity following the definition in YOSHI.

Detection Method To compute longevity, TOAD extracts every community member that has been active (M), i.e., that has worked on the repository during the 3-month time window. After gathering active members, we compute the Mean Committer Longevity (CL), i.e., a measure of how long a committer has been part of the community. This value is computed as the number of days between the first and the last commit creation date for each active member.

Longevity

Longevity is computed with the following equation:

$$Longevity = \frac{1}{|M|} \sum_{m \in M} CL_m$$

That is, the mean committer longevity (CL) for the set of community members that were active within the 3-month period (M).

3.2.5 Engagement

Engagement reflects the participation level of the community, i.e., the amount of time a member is actively participating in community-related actions, and it is key to identify Informal Communities (IC). Kujala et al. have shown how engagement is important for community health, performing a case study and stating that developer engagement in software projects reflects user satisfaction.

Detection Method TOAD computes engagement the same way YOSHI does, i.e., through the following data:

- *Median Comments per PR*: the median number of comments for each of the Pull Requests that fall in the analyzed time window;
- *Median Active Members*: TOAD assigns each member 1 if they are active, i.e., they committed to the repository in the last 30 days, 0 otherwise, and computes the median;
- *Median Watchers*: TOAD assigns each member 1 if they are also watchers, i.e., registered to receive notifications on new discussions, as well as events, in the user's activity feed, 0 otherwise, and computes the median;
- *Median Stargazers*: TOAD assigns each member 1 if they are stargazers, i.e., "starred" the repository to show interest, 0 otherwise, and computes the median;

- *Median Monthly Distribution of Comments per Member*: TOAD extracts a list of commit comments and pull request comments. These lists are merged, and sorted by member. Then it computes each member's mean comments per month and takes the median from the resulting list.

- *Median Monthly Commit Distribution per Member*: TOAD first extracts a list of commits, then iterates over this list and assigns the committer date to the committer, and the author-date to the author. Then it computes the mean commits for each member per month, followed by taking the median from the resulting list.

- *Median Monthly Distribution of Collaborators per File*: TOAD extracts a list of commits and iterates through these commits to extract the list of files changed, splitting them per month. Then, it merges these files for each committer, and then the mean number of committers per file per month is computed, from which TOAD takes the median.

Engagement

Engagement is computed as the sum of the 7 metrics listed above

3.2.6 Dispersion

Dispersion is, after structure, the most significant characteristic identified by TOAD. It is key to identify Network of People (NoP) and Community of People (CoP), and it is used in thresholds of 6 out of 8 community patterns. This characteristic is made of two separate metrics, i.e., *geographical distance* and *cultural distance*. It reflects how much a development community is different in terms of cultural attitudes and geographical collocation of its members.

For geographical distance, we mean the spherical distance between the physical location of community members, like YOSHI [4].

For cultural distance, we decided to implement a different framework. Tamburri et al. used Hofsteade's 6-D framework, in which he defined six dimensions that assume values from 0 to 100 and which combination characterizes the culture of a specific country [37, 38]. Our implementation is based on another cultural framework, i.e., the GLOBE Framework, which defined 9 cultural dimensions, that assume values from 0 to 7, based on the interrelationships between societal culture, societal effectiveness, and organizational leadership [39]. The GLOBE's 9 dimensions are listed below [39].

- **Performance Orientation**: The degree to which a collective encourages and rewards (and should encourage and reward) group members for performance improvement and excellence.
- **Assertiveness**: The degree to which individuals are (and should be) assertive, confrontational, and aggressive in their relationship with others.
- **Future Orientation**: The extent to which individuals engage (and should engage) in future-oriented behaviors such as planning, investing in the future, and delaying gratification.
- **Humane Orientation**: The degree to which a collective encourages and rewards (and should encourage and reward) individuals for being fair, altruistic, generous, caring, and kind to others.
- **Institutional Collectivism**: The degree to which organizational and societal institutional practices encourage and reward (and should encourage and reward) collective distribution of resources and collective action.
- **In-Group Collectivism**: The degree to which individuals express (and should express) pride, loyalty, and cohesiveness in their organizations or families.
- **Gender Egalitarianism**: The degree to which a collective minimizes (and should minimize) gender inequality.

- **Power Distance**: The extent to which the community accepts and endorses authority, power differences, and status privileges.
- **Uncertainty Avoidance**: The extent to which a society, organization, or group relies (and should rely) on social norms, rules, and procedures to alleviate the unpredictability of future events. The greater the desire to avoid uncertainty, the more people seek orderliness, consistency, structure, formal procedures, and laws to cover situations in their daily lives.

Detection Method This characteristic, other than being one of the most influential for the identification of community patterns, happens to be also one of the most controversial in its definition. di Nitto et al. defined two different thresholds for the two metrics that make dispersion, i.e., one for geographical distance and one for cultural distance [21]. In their work, Tamburri et al. mention one single threshold, which is computed through a mathematical equation that reflects both metrics in one [4]. When analyzing the source code of YOSHI, though, we found out that this characteristic was not implemented as a single one, as stated in the paper, but was operationalized as two different values with two different thresholds as defined in the original ethnographic work [21]. Thus we decided to implement dispersion with two different metrics following di Nitto et al.'s definition. To compute such metrics, we first scan through community members and gather their location information, particularly *coordinates* and *country*. Then we compute the following two metrics:

- *Average Geographical Distance*: for each couple of members, we compute the spherical distance between their coordinates in kilometers using the Great Circle formula[7], and then we compute the average of these distances;
- *Cultural Distance Variance*: TOAD iterates through community members whose country's information is available, and stores GLOBE's value related to the

[7] `https://en.wikipedia.org/wiki/Great-circle_distance`

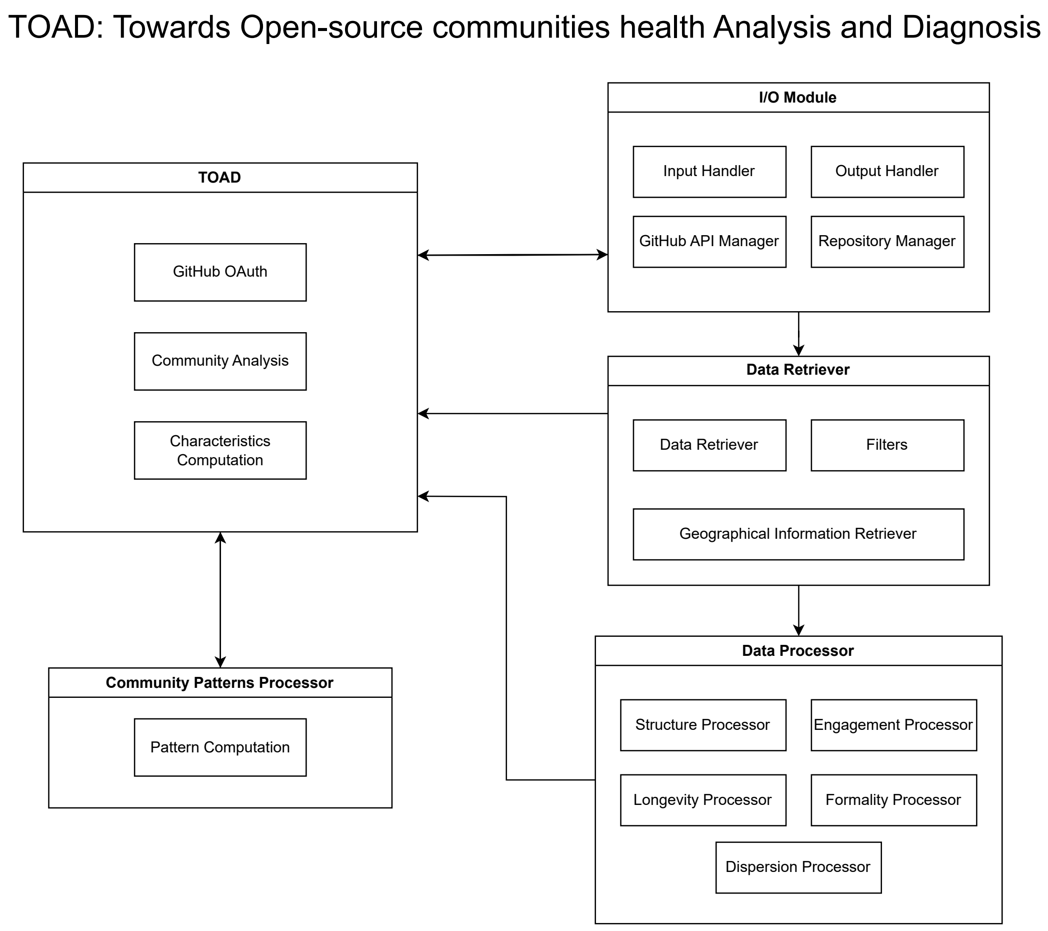

Figure 3.2: The architecture of TOAD

country in a separate list for each dimension. After gathering the values of each dimension for each member, we compute the average of the variance of each dimension, adjusting the value to a 0-100 scale to fit the threshold.

Dispersion

Dispersion is computed as the average geographical distance between each couple of members and the variance of the cultural distance between community members.

3.3 Architecture

Figure 3.2 shows the architecture of TOAD. Since YOSHI's components were unusable and couldn't be updated, we redesigned the architecture to fit our goals and intentions. TOAD is divided into multiple modules, divided by responsibilities. The main module, named **TOAD**, controls the whole flow of execution. It handles the authentication steps and then gives the start to the community analysis. During this step, many interactions **I/O Module** happen. First, it gets the input community from the Input Handler and downloads repositories from the Repository Manager. After getting the input, the **Data Retriever** module is interrogated, during which it retrieves every information needed by the GitHub API Manager from **I/O Module**. After retrieving every needed data to ensure that the community is valid, the control passes back to the main module **TOAD**, which starts the Characteristics Computation. This gathers each needed characteristic from the **Data Processor** module, which again uses the **Data Retriever** module to gather additional needed information. We decided to separate the data retrieval into two steps because it would be a waste of time to retrieve every possible information before verifying if the repository is valid. So, we decided to retrieve data for each characteristic step by step. After computing each characteristic, the flow goes back again to the main module. At this point, the tool creates a data object that is sent to the **Community Pattern Processor** module, which computes community patterns and sends them back to the **TOAD** module. After doing so, the flow ends by interrogating again **I/O Module** to handle the output of both community patterns and the other computed data.

We decided to separate the processing of characteristics and patterns from the main flow to have them as isolated components. We also implemented the modules to print characteristics and metrics as JSON files. This means that, as a future work, it will be possible to only use the Pattern Processor module and use it through a file containing self-computed metrics, without the necessity of using the ones computed by TOAD. For example, one could compute such metrics for a community of another platform, and then use our module to compute patterns.

3.4 Differences with YOSHI

TOAD has been implemented as a partial replication of YOSHI. Both tools are based on the algorithmic solution for the identification of community patterns shown in Algorithm 1 and the definition of characteristics and metrics defined by Tamburri et al.[4] and di Nitto et al.[21]. Nonetheless, there are some significant differences in their implementation, which will be listed in the current section. Note that these are only related to the source code of the tools since there is some difference between Tamburri et al.'s statements in the paper and the actual implementation of YOSHI. The conceptual differences, i.e., the decisions of considering values differently for some metrics, have already been reported in the detection method for each metric.

Authentication Process We decided to implement an actual authentication flow instead of having users provide a GitHub Personal Access Token (PAT) for each execution. This will strongly improve the usability of the tool. A regular PAT has to be created from the user's personal account on GitHub, and it has a limited time duration and a limited number of API calls available. With our OAuth authentication workflow, users only have to authenticate with their GitHub account once, and then a permanent PAT will be generated and stored locally to be used in the next executions.

Social Network Graph Computation As already pointed out in Section 3.2.2, we implemented a social network graph but with weighted edges. Although it has not been operationalized in TOAD, its implementation in the future would address one of the limitations of YOSHI: community structure has been designed as a binary characteristic, and it is set to be true if at least one connection (of any kind) is present between developers. As a future work, there could be a study on structure to find a threshold of graph's edges weights, in order to have a more accurate detection strategy for structure.

Alias Extraction YOSHI does not support alias resolution, but it would actually be difficult to measure community characteristics if members have multiple accounts. Since it is not uncommon for developers to contribute to repositories with different accounts, we decided to implement alias resolution in the workflow of TOAD. We used the same alias extraction method used by Almarimi et al. for CSDETECTOR. We used the Longest Common Subsequence Metric (LCS) to check the similarity of authors' IDs.

Membership Type Approximation YOSHI used the typical membership types from GitHub, i.e., Contributor and Collaborator, extracted from API publicly available at the time. As discussed in Section 3.2.3, this data is not available anymore, so in the new version we had to approximate which users are considered contributors and which collaborators.

Milestones in Formality In the original work [4], it is not clear which kind of milestone should be considered when computing formality. A milestone on GitHub can be opened or closed. YOSHI only uses closed ones, but we found discrepancies between this decision and pattern detection. We analyzed each of the repositories used by YOSHI for its validation, and we found that most of them have no closed milestones in the analyzed periods. Also, looking at the mathematical equations, having too few milestones—which is the case if we only considered closed ones—means to have a formality often ten times bigger than the upper limit of the defined threshold [21]. We found that considering all milestones instead of only closed ones produced more realistic results. Aside from a mathematical consideration, the number of milestones is used to assess the formality level of a community, and we consider a community to be formal even if it uses and updates milestones, instead of instantly closing them.

Geolocation API Since we decided to use a different programming language, i.e., Python, we decided to go for a different API choice for the geolocation of members' coordinates. We used the *geopy* library[8], which allows us to gather country names for the cultural dispersion computation starting from coordinates of developers' location. Geopy offers many third-party location services, and we choose to use *Nominatim*[9], which offers an infinite number of calls but limited to 1 per second.

Cultural Framework As already explained in Section 3.2.6, we decided to implement the GLOBE[39] framework instead of Hofstede's[37]. The difference between the two is that, where Hofstede focuses on single nations to gather dimensions, and these dimensions are only 6, GLOBE offers 9 dimensions that have been developed analyzing different cultural areas and leadership factors.

3.5 Limitations

In this section, we will describe some limitations of the tool.

General Limitations TOAD is only capable of detecting 8 out of the 13 community patterns discovered in the literature. This would affect the effectiveness of the tool in identifying the type of open-source communities since some characteristic is not computed (cohesion). Nevertheless, studies have shown that four of these patterns happen to be the most representative (FN, IN, NoP, and IC), and since TOAD is able to detect them, we consider the tool still partially effective.

Furthermore, TOAD only analyzes data from GitHub. It can happen that some open-source development community uses other platforms to perform social or organizational activity, making our tool's output limited in such a way. Also, the

[8]https://geopy.readthedocs.io/en/stable/

[9]https://nominatim.org/release-docs/develop/

tool analyzes GitHub history, but it could happen that the team migrated to/has migrated from another version control platform, making some data unavailable.

Structure Limitations By analyzing a snapshot of the community at the time of the retrieval, there is no way to intercept any interaction between members that has been deleted or gone unrecorded. For example, two members could have followed each other years before they started to work in the analyzed repository, and they could have no interaction in such a community. Yet, the tool would find a relationship and consider the community structured. Also, TOAD considers common projects, but something could go missing. For example, forks or deleted repositories have no way of being retrieved to the best of our knowledge, hence a structured community could be missed for such reasons. Moreover, one member could create a pull request outside of the 3-month time window and another member could interact with it inside this time window. This relation would be missed by TOAD.

Formality Limitations The main limitation in the computation of this characteristic is that we are obliged to assume the membership type of a community member due to limitations in GitHub APIs. Furthermore, there is a discrepancy between the mean membership type and the milestones per project lifetime ratio. The former is computed for active members in the 3-month time window analyzed, while the latter considers the whole lifetime of the repository. Also, the calculation of formality level needs the presence of at least one milestone, since a value of 0 would make formality infinite. However, the underuse of milestones risks increasing formality.

Engagement Limitations As already mentioned for structure, some pull request interactions could be missed due to the analysis in the 3-month time window. Moreover, we consider active members who interacted with the source code of the repository in recent times, but we did not consider members who participated in discussions even if they could be considered active in community activities. Finally, engagement considers each of the 7 metrics as equal in the final computation, but studies are needed to understand whether some metrics could have much more influence than others in the engagement level and should be weighted differently.

Dispersion Limitations The main limitation is that not every member of the community has public data regarding the location available. This means that dispersion could be computed only for a subset of the members, hence not fully representing the whole community. The decision to implement GLOBE instead of Hofstede addresses some limitations given by critiques on Hofstede's model, but even GLOBE does not lack limitations in its design.

Algorithm 1: TOAD's algorithm for community type detection.

```
patterns = {}
if $Structure$ = TRUE then
    patterns.add("SN")
    if $GeoDistance > 4000$ km and $CulturalDistance > 15$ then
        patterns.add("NoP")
        if $Formality\ levels < 0.1$ then
            patterns.add("IN")
        else if $Formality\ levels > 20$ then
            patterns.add("FN")
        end if
    else
        patterns.add("CoP")
        if $Longevity < 93$ days then
            patterns.add("PT")
        end if
        if $Formality\ levels \geq 0.1$ and $\leq 20$ then
            patterns.add("FG");
        end if
    end if
    if $Engagement\ Levels > 3.5$ then
        patterns.add("IC")
    end if
end if
return patterns
```

Table 3.1: List of thresholds used to compute community patterns in TOAD

Community Pattern	Thresholds
Communities of Practice (CoP)	GeoDistance <= 4000 km & CulturalDistance <= 15
Informal Networks (IN)	Formality Levels < 0.1 & GeoDistance > 4000 km & CulturalDistance > 15
Formal Networks (FN)	Formality Levels > 20 & GeoDistance > 4000 km & CulturalDistance > 15
Informal Communities	(IC) Engagement Levels > 3.5
Networks of Practice	(NoP) GeoDistance > 4000 km & CulturalDistance > 15
Project Teams (PT)	Longevity < 93 & GeoDistance <= 4000 km & CulturalDistance <= 15
Formal Groups (FG)	Formality Levels 0.1 and 20 & GeoDistance <= 4000 km & CulturalDistance <= 15
Social Networks (SN)	Structured Network = True

CHAPTER 4

Research Methodology

The following chapter will describe how the main objective will be achieved through research questions, strategies, and methodologies that will be used.

The main goal of the study is to understand better relationships among different open-source software development communities' health information, i.e., community patterns and community smells. We aim to provide a deeper understanding of how patterns in such development communities can lead to the emergence of socio-technical issues and, eventually, social debt.

The perspective is that of researchers and practitioners: the former seeks to identify any recurring pattern, or antipattern, that will more likely lead to the emergence of community smells and to some form of social debt; the latter are interested in avoiding organizational structures that are likely to incur social debt [22].

4.1 Hypotheses and Research Questions

Analyzing literature regarding health information in open-source software development communities, it is clear that measuring metrics to assess their healthiness is not an easy job. Also, the metrics used to detect both community smells and community patterns often have something in common, leading to the perception that there is some sort of connection between them [4] [8]. This leads to the development of the working hypotheses for this thesis work, stated as follows:

The social and organizational structure type—community pattern—of an open-source software development community influences socio-technical characteristics and patterns—community smell—of the project, leading to the emergence of social debt.

To achieve the objective of this study, which is reported below, following the hypotheses just defined, the research process will be separated into various steps, and mapped as research questions.

◎ **Our Goal.** Understand the underlying relations between community patterns and community smells.

First thing first, we had to gather automated tools to extract community patterns and community smells in open-source software development communities. As described in Chapter 2, there are various tools proposed in the literature to detect community smells. We decided to use csDetector since it is one of the most recent and the better performing one [8]. Concerning the detection of community patterns, YOSHI could be the most suitable tool to gather such community information [4]. Due to outdated APIs used in its original implementation, the tool YOSHI is not executable at the time of this study. This led us to the conclusion that there was a necessity to reimplement such a tool, also addressing some of its limitations. Having to reimplement the tool YOSHI leads to an inevitable question: are the community patterns extracted by our new tool consistent with the ones extracted by the original implementation? To understand if our results were comparable to the ones of YOSHI, we ran the tool on the original set of repositories on which it was validated, confronting both final results and metrics computation. We also performed a deep study on the thresholds used in the tool to define the type of a community, modifying eventually such values to fit our case study. Given our modification, our new tool **TOAD (Towards Open-source communities health Analysis and Diagnosis)** is a *partial replication* of YOSHI, but consistent with how metrics were calculated. The process of validation will answer our first research question:

🔍 **RQ$_1$.** *Is our implementation of the tool "YOSHI" consistent in the detection of community patterns?*

After validating the results of our replication, we found ourselves with two automated tools to detect community patterns and smells, i.e., the revised version of YOSHI and CSDETECTOR. The following step was to select a set of repositories, execute both tools and create a dataset. The selected repositories were as heterogeneous as possible. After gathering this set of repositories with both community patterns and community smells extracted, we studied the relations between the two through association-rule mining, to answer our main research question:

🔍 **RQ$_2$.** *Is there a relation between community patterns and community smells?*

The following sections will describe the research process we followed to answer our questions. A more detailed description of some steps of the research, results, and findings will be provided in the next chapters.

4.2 Reseach Strategy

Figure 4.1 provides an overview of the process we followed to answer the two research questions. We describe the process of implementing TOAD—Step 1 of the figure—in Chapter 3. During Step 1, while designing a new and improved way to compute the needed metrics, we performed an empirical evaluation of such metrics

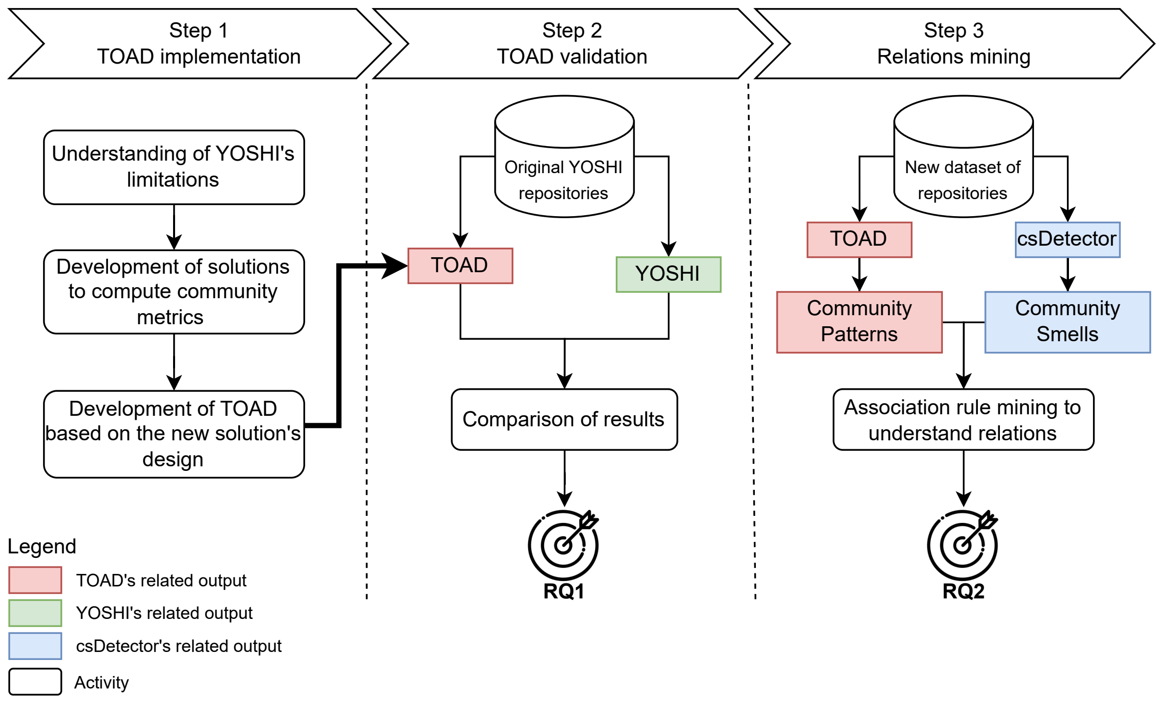
Step 1
TOAD implementation
Step 2
TOAD validation
Step 3
Relations mining
Understanding of YOSHI's limitations
Development of solutions to compute community metrics
Development of TOAD based on the new solution's design
Original YOSHI repositories
TOAD
YOSHI
Comparison of results
RQ1
New dataset of repositories
TOAD
csDetector
Community Patterns
Community Smells
Association rule mining to understand relations
RQ2
Legend
TOAD's related output
YOSHI's related output
csDetector's related output
Activity

Figure 4.1: Overview of the research methodology.

with respect to the current literature. Such a tool has been used as an input for Step 2, in which the results of TOAD's execution have been compared to the results of YOSHI's execution on the original set of repositories that Tamburri et al. used to validate the tool. This process will be detailed in Section 4.3. After having evaluated TOAD, a new set of repositories has been selected for Step 3. This dataset has been run onto both TOAD and csDetector, to gather community patterns and community smells and mine relations between the two, as described in Section 4.4.

4.3 RQ1: Validating TOAD

After TOAD's implementation, without going deeper into its characteristics, we had a new tool able to classify open-source software development communities into community patterns. Such classification, though, needed to be verified to understand whether or not it correctly reflected community types and characteristics. This section describes the process of *validating* TOAD *with respect to the original* YOSHI*'s results,*

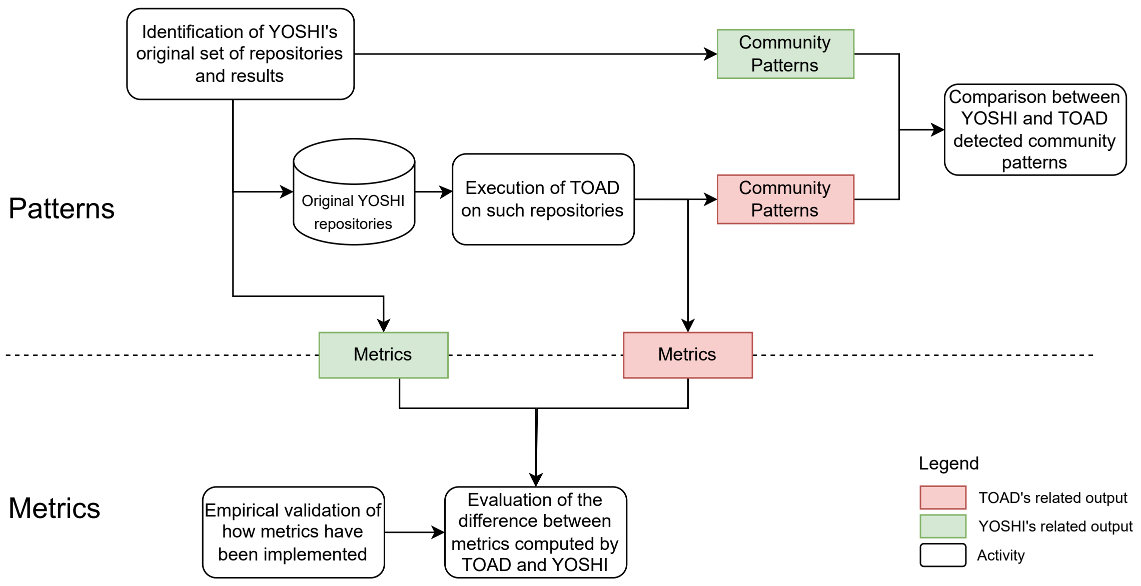
Identification of YOSHI's original set of repositories and results
Community Patterns
Comparison between YOSHI and TOAD detected community patterns
Patterns
Original YOSHI repositories
Execution of TOAD on such repositories
Community Patterns
Metrics
Metrics
Metrics
Empirical validation of how metrics have been implemented
Evaluation of the difference between metrics computed by TOAD and YOSHI
Legend
TOAD's related output
YOSHI's related output
Activity

Figure 4.2: Methodology to validate TOAD's results.

findings, and metrics definition [4], deepening Step 2 of Figure 4.1. As shown in Figure 4.2, the validation of the tool has been performed in two parallel ways: (1) examining the community pattern detected and (2) investigating how metrics are computed.

Detected Community Patterns To validate TOAD's detected community patterns, we had to gather YOSHI's execution results and the analyzed repository. Unfortunately, the research article that proposed the tool only mentions community names, and not directly GitHub repositories [4]. For this reason, we analyzed YOSHI's GitHub repository [1] and found a list of communities and their associated repositories. Among these, we found the exact list of repositories that were used to validate YOSHI in Tamburri et al.'s work. The list of communities, considered as GitHub repositories, is reported in table 4.1.

After retrieving the list of repositories that we needed, we had to select the most suitable time window to analyze. Unfortunately, no information about the time of execution was reported in YOSHI's documentation, except for the time in which they retrieved communities' characteristics, which was April 2017, which we could select

[1]`https://github.com/maelstromdat/YOSHI`

as the end date of the 90-day time window of the analysis since the tool analyzed the time window of 3 months (90 days). This led us to gather community patterns for most of the repositories used in YOSHI, and the results of both tools' execution were compared. More details on the results of the validation will be given in Chapter 5.

This approach, even if effective, is limited for the following reasons:

`Limitation 1` The selected time window for TOAD's analysis is based on the assumption that the repository was originally analyzed in a period between January 31, 2017, and April 30, 2017.

`Limitation 2` Projects' data on GitHub can be modified or even deleted. This might give different results compared to the ones obtained in 2017, even if analyzing the same exact time window.

To address Limitation 1, we performed our analysis with TOAD checking each 90-day time window from April 2017 going down until we find a suitable time window. The tool only works if several conditions are met, and these conditions ensure that in the given time period the community was sufficiently active. For this reason, checking the first 3-month time window before April 2017 in which the tool works means getting the closest and most significant results for our analysis. Addressing Limitation 2 was quite a hard job since we couldn't know whether a community had changed its history. We assumed that if community characteristics were too different compared to the original ones, the community had changed its history on GitHub.

Computed Metrics TOAD has been developed as a partial replication of YOSHI. Both tools work by retrieving public data from the GitHub APIs and computing several *metrics* that should represent community *characteristic*, e.g., structure, formality, and engagement. These metrics are listed in Table 4.2. We aimed to keep such characteristics as close as they were defined in the original works [21, 4], but we slightly changed their computation process. This led us to perform a first validation

Table 4.1: List of communities with characteristics analyzed by YOSHI

Community	# Rel.	# Commits	# Members	# Language	#KLOC	Domain
netty, netty	164	8,123	258	JavaScript	438	Software Tools
eoecn, android-app	3	132	14	Java	382	Library
arduino, Arduino	74	6,516	210	C	192	Rapid prototyping
angular-ui, bootstrap	55	2,067	389	JavaScript	378	Web libraries and fw.
boto, boto	86	7,111	495	Python	56	Web libraries and fw.
bundler, bundler	251	8,464	549	Java	112	Web libraries and fw.
c9, core	97	9,485	64	ShellScript	293	Application software
composer, composer	35	7,363	629	PHP	254	Software Tools
cucumber, cucumber	8	566	15	Java	382	Software Tools
emberjs, data	129	5,151	407	JavaScript	272	Web libraries and fw.
gollum, gollum	76	1,921	143	Gollum	182	App. fw.
EightMedia, hammer.js	25	1,193	84	C	199	Web libraries and fw.
h5bp, mobile-boilerplate	12	469	48	PHP	266	Web libraries and fw.
Modernizr, Modernizr	27	2,392	220	JavaScript	382	Web libraries and fw.
mongoid, mongoid	253	6,223	317	Ruby	187	App. fw.
xamarin, monodroid-samples	2	1,462	61	C	391	App. fw.
mozilla, pdf.js	43	9,663	228	JavaScript	398	Web libraries and fw.
scrapy, scrapy	78	6,315	242	Python	287	App. fw.
refinery, refinerycms	162	9,886	385	JavaScript	188	Software Tools
saltstack, salt	146	81,143	1,781	Python	278	Software Tools
sightmachine, SimpleCV	5	2,625	69	Python	389	App. and fw.
hawkthorne, hawkthorne-journey	116	5,537	62	Lua	211	Software Tools
square, SocketRocket	10	494	67	Obj-C	198	App. fw.

Table 4.2: List of metrics computed by TOAD

Characteristic	Metric
Structure	Common Projects
	Follower/Following relation
	Pull Request Interaction
Geodispersion	Average Geographical Distance
	Cultural Distance Variance
Formality	Mean Membership Type
	Number of Milestones
	Lifetime
Longevity	Mean Committer Longevity
Engagement	Median Number of Comments per PR
	Median Stargazer Member
	Median Watcher Member
	Median Active Member
	Median Monthly Distribution of Pull/Commit Comments per Member
	Median Monthly Distribution of Commits per Member
	Median Monthly Distribution of Collaborations on Files

of such metrics, in order to see whether or not they respected the definitions. After being sure that such metrics were computer in a way that didn't change their meaning, we compared the exact results of both tools for these metrics, with the aim of understanding differences and issues in our implementation.

4.4 RQ2: Relations between Community Patterns and Community Smells

Figure 4.3 shows how we mined relations between community patterns and community smells. We started from a dataset of repositories, each of them already analyzed by CSDETECTOR to detect community smells. Among these repositories, only 34 were suitable for their analysis with TOAD. By running this subset of reposi-

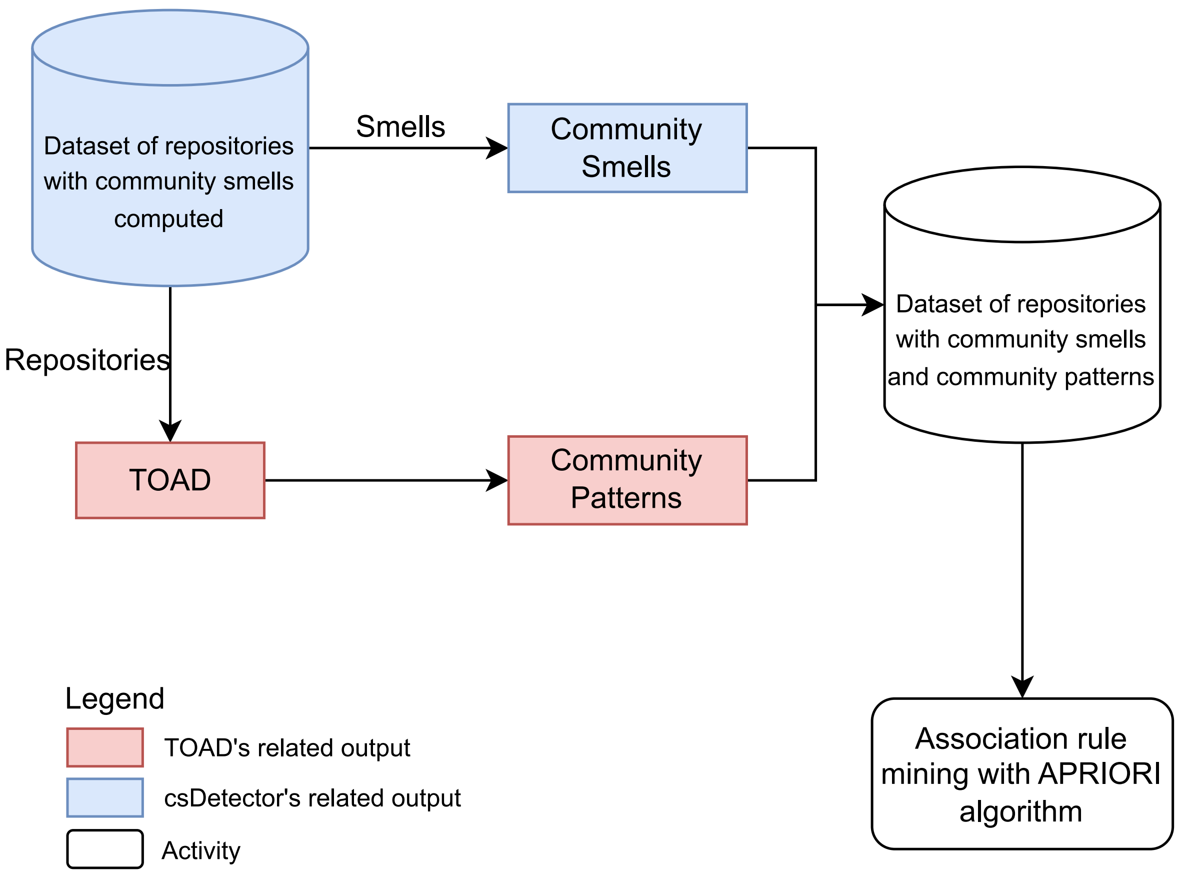
Dataset of repositories with community smells computed
Smells
Community Smells
Repositories
TOAD
Community Patterns
Dataset of repositories with community smells and community patterns
Association rule mining with APRIORI algorithm
Legend
TOAD's related output
csDetector's related output
Activity

Figure 4.3: Methodology to mine relations between community patterns and smells.

tories on our new tool, we created a new dataset of repositories with bot community smells and community patterns. At this point, we started mining association rules for detecting which community patterns and smells co-occur. An association rule $R_{left} \rightarrow R_{right}$ implies that if a project shows a certain community pattern (*left*), then the project is affected by a certain community smells (*right*). The strength of an association rule is determined by two metrics, i.e., support and confidence:

$$support = \frac{|R_{left} \cup R_{right}|}{T}$$

$$confidence = \frac{|R_{left} \cup R_{right}|}{R_{left}}$$

where T is the number of co-occurrences between community patterns and smells in our dataset. To implement association rule mining, we used the APRIORI algorithm [40], implemented in Python within the library MLXTEND [41]. We set thresholds to

0.2 for the support and 0.6 for the confidence to filter out weak relations. We also evaluated the lift metric, which measures the ability of a rule to correctly identify a relationship with respect to a random-choice model [40]. A lift value higher than 1 means that co-occurrences of the left-hand operator (community pattern) often imply the presence of the right-hand operator (community smell). Results and discussions of such process with the defined thresholds are reported in Chapter 5.

CHAPTER 5

Results

In this Chapter, we report and discuss the results of our research to answer the two research questions defined in Chapter 4.

5.1 Validation of TOAD

🔍 **RQ$_1$.** *Is our implementation of the tool "YOSHI" consistent in the detection of community patterns?*

The aim of this RQ is to perform an analysis of TOAD to understand if the detection of community patterns is consistent with the definitions of Tamburri et al. in YOSHI's design and implementation [4]. As already stated in Section 4.3, we performed this validation in two steps, namely (1) analyzing the pattern detected and (2) analyzing the metrics computed. To perform such validation, we analyzed the communities already listed in Table 4.1. Note that certain communities did not produce any result, so we will only report our findings on the communities that did produce an output. These repositories did not work since there were not enough commits in the analyzed date or there were no milestones available, hence the tool could not detect community patterns. We do not know how YOSHI was able to produce such results, but we think that they performed unreported modifications to the process.

5.1.1 Comparison of Community Patterns

Table 5.1 shows a comparison between the community patterns identified by YOSHI [4] and the ones identified by TOAD. As we slightly modified the implementation process, we already planned some differences between the results of the two tools. Results show that most of the repositories we analyzed have at least one common pattern with the one computed by YOSHI. Yet, when comparing results, we found out that there were discrepancies between the pattern inferred by YOSHI and the threshold defined. Due to the definition of the thresholds, reported in Table 3.1, some community patterns cannot appear together, or when a combination of patterns appears it would necessarily happen that another pattern should be identified, which is not the case for YOSHI. In Table 5.2 we report such contradictions. Analyzing such issues, we found no problems in our results, i.e., TOAD respects the empirically defined thresholds defined by Tamburri et al. [4] and di Nitto et al. [21]. Hence, even if the results are not exactly the same, we think that our implementation of the tool produces an acceptable subset of community patterns and respects their definition.

Finding 1. TOAD does not produce the same results as YOSHI, but due to contradictions in the original tool's result, we can affirm that our tool's results reflect community patterns in open-source software development communities.

5.1.2 Comparison of Metrics

Since the tool YOSHI is not executable due to outdated APIs, we had to use metrics computed for the first validation of the tool which we found in the public GitHub repository of YOSHI [1]. Among these data, we could not identify all of the

[1]https://github.com/maelstromdat/YOSHI

Table 5.1: Community patterns inferred by YOSHI and TOAD for the considered communities. Every analyzed community is considered an SN, and therefore SN has not been included in the table.

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

Table 5.2: Community patterns inferred by YOSHI and how they contradict the reported empirical thresholds.

Patterns	Communities	Contradictions
FN,FG	ember.js, mongoid, Scrapy, Salt	**FN**, **FG** cannot be reported together, formality levels and global distance contradict; if **FG** would be reported, **CoP** would be reported as well; if **FN** would be reported, **NoP** would be reported as well
IC,FN,FG	netty, android, Arduino, cloud9, composer, gollum	**FN**, **FG** cannot be reported together, formality levels and global distance contradict; if **FG** would be reported, **CoP** would be reported as well; if **FN** would be reported, **NoP** would be reported as well
IC,IN	boto	If **IN** would be reported, **NoP** would be reported as well
NoP,FG	bundler	**NoP**, **FG** cannot be reported together, global distance contradicts; if **FG** would be reported, **CoP** would be reported as well
NoP,IN,FG	Heroku, SocketRocket, scikit-learn	**NoP**, **FG** cannot be reported together, global distance contradicts; if **FG** would be reported, **CoP** would be reported as well; **IN**, **FG** cannot be reported together, formality levels and global distance contradict
IC,NoP,IN,FG	SimpleCV	**NoP**, **FG** cannot be reported together, global distance contradicts; if **FG** would be reported, **CoP** would be reported as well; **IN**, **FG** cannot be reported together, formality levels and global distance contradict
IC,IN,FG	Hawkthorne, Monodroid	**IN**, **FG** cannot be reported together, formality levels and global distance contradict
FG	refinerycms	If **FG** would be reported, **CoP** would be reported as well

computed metrics by TOAD and defined in the original paper, meaning that some of the metrics used by YOSHI were not actually the same defined in prior works. In Table 5.3 we report the metrics that we could find, while TOAD results and computed metrics are available in the online appendix[2]. We only reported common metrics based on the ones we were able to find for YOSHI. Analyzing both tools' results, we find that most metrics are different in YOSHI's implementation with respect to their definition. Among the analyzed metrics in common, we find that the ones that have a common representation, e.g., cultural and geographical distance, milestones per day, and active members, show close values. We can't say whether or not our metrics are similar to the one computed by YOSHI overall, yet we think our representation of the characteristics is well-suited.

Finding 2. The metrics computed by TOAD and YOSHI show many differences in their representation, yet it is hard to compare them. We still think that, based on the few comparable ones, our characteristics have been computed as defined in the original work.

Answer to RQ$_1$. TOAD's results do not precisely match YOSHI's, but we still can affirm that our partial replication of the tool is able to compute community pattern according to the original design.

5.2 Analysis of relations

RQ$_2$. *Is there a relation between community patterns and community smells?*

This research question aims to provide information about eventual relations between community patterns and community smells. As said in Chapter 4, we planned to do so by using our new tool TOAD to detect community patterns and CSDETECTOR [8] to detect community smells on a set of repositories. We reported in Tables 5.4, 5.5, 5.6, 5.7 the results of the execution of both tools on such repositories. After gathering

[2]Online Appendix: `https://github.com/gianwario/TOAD`

Table 5.3: Metrics computed by YOSHI

Community	milestones Per-Day	avgeFile Con-tributors	active Mem-bers	avgCommiter Longivity	avgDistance	avgCultural Distance
Netty	0.093	1.856	1	4.263	7742.387	20.811
Arduino	0.027	1.210	1	2.648	5224.618	19.206
Bootstrap	0.032	1.542	1	0.727	6824.103	16.679
Bundler	0.004	4.680	1	0.434	6500.530	16.430
Composer	0.011	3.579	1	0.208	5434.328	16.306
Ember-JS	0.008	5.884	3	0.308	5118.343	14.690
Modernizr	0.007	2.026	2	0.153	5978.121	19.086
PDF-JS	0.012	2.369	2	1.366	6693.698	20.488
RefineryCMS	0.012	2.780	2	0.984	7672.982	19.859
Salt	0.037	4.162	3	1.225	5708.846	18.394

Table 5.4: (1/2) Community smells for the repositories analyzed (1 if present, 0 otherwise).

repositoryAuthor	repositoryName	OSE	BCE	PDE	SV	OS	SD	RS	TF	UI	TC
microsoft	malmo	1	0	1	0	0	1	0	1	1	1
louismullie	treat	0	1	0	1	0	1	0	1	1	1
gunthercox	ChatterBot	1	1	1	1	0	1	1	0	1	1
leon-ai	leon	0	0	1	1	0	1	0	1	1	1
fossasia	susi_server	1	1	1	0	0	0	1	0	0	0
accord-net	framework	1	1	1	1	0	1	1	0	0	0
olivia-ai	olivia	1	0	1	0	0	1	0	0	1	0
asteroid-team	asteroid	1	0	1	1	0	1	0	0	0	0
microsoft	vscode-docker	1	0	0	0	0	0	0	0	0	1
microsoft	bobuilder-js	0	1	1	0	0	1	1	1	0	1
microsoft	BotFramework-Composer	0	1	0	1	0	1	1	1	0	1
google	oboe	0	0	0	1	0	0	0	1	0	1
google	WebFundamentals	1	1	0	1	0	1	0	0	0	1
PetrochukM	PyTorch-NLP	1	1	0	0	0	1	1	1	1	0
eclipse	leshan	1	1	0	0	1	0	0	0	0	1
evilsocket	pwnagotchi	0	0	0	0	0	1	0	1	1	0

Table 5.5: (2/2)Community smells for the repositories analyzed (1 if present, 0 otherwise).

repositoryAuthor	repositoryName	OSE	BCE	PDE	SV	OS	SD	RS	TF	UI	TC
tensorlayer	tensorlayer	0	1	0	0	0	1	0	0	0	1
OpenMined	PySyft	1	1	0	0	1	0	1	0	0	1
microsoft	PowerToys	1	1	0	0	0	0	1	0	0	1
microsoft	Microsoft365DSC	1	0	1	1	1	0	1	0	0	1
google	jax	0	1	0	1	0	0	1	0	0	1
google	blockly	1	1	0	0	1	1	1	1	0	1
uber	cadence	1	0	0	1	0	0	0	0	1	0
tensorflow	models	1	0	1	1	0	1	0	1	0	1
tensorflow	addons	0	0	0	0	0	1	1	0	1	0
NVIDIA	HugeCTR	0	0	1	0	0	0	0	0	1	0
microsoft	PowerStig	0	0	0	1	0	0	0	1	1	0
microsoft	vscode-cpptools	0	0	0	1	1	0	1	0	0	1
bardsoftware	ganttproject	1	0	1	0	1	1	1	0	1	0
eclipse	iceoryx	1	0	0	0	0	0	0	0	0	1
eclipse	californium	1	0	0	0	1	0	0	0	0	1
eclipse	ditto	0	0	0	1	1	0	0	0	0	1
eclipse	sw360	0	0	0	1	0	1	0	0	0	1
eclipse	nebula	1	1	1	1	1	0	1	0	0	1

Table 5.6: (1/2) Community patterns for the repositories analyzed (1 if present, 0 otherwise).

repositoryAuthor	repositoryName	SN	NoP	IN	FN	CoP	PT	FG	IC
microsoft	malmo	1	1	0	1	0	0	0	1
louismullie	treat	1	1	0	1	0	0	0	1
gunthercox	ChatterBot	1	0	0	0	1	1	0	1
leon-ai	leon	1	0	0	0	1	1	0	0
fossasia	susi_server	1	1	0	1	0	0	0	1
accord-net	framework	1	0	0	0	1	1	0	1
olivia-ai	olivia	1	0	0	0	1	1	0	1
asteroid-team	asteroid	1	0	0	0	1	1	0	1
microsoft	vscode-docker	1	1	0	1	0	0	0	1
microsoft	bobuilder-js	1	0	0	0	1	1	0	1
microsoft	BotFramework-Composer	1	0	0	0	1	1	0	1
google	oboe	1	1	0	1	0	0	0	1
google	WebFundamentals	1	1	0	1	0	0	0	1
PetrochukM	PyTorch-NLP	1	1	0	1	0	0	0	1
eclipse	leshan	1	0	0	0	1	1	0	1
evilsocket	pwnagotchi	1	1	0	1	0	0	0	1

Table 5.7: (2/2) Community patterns for the repositories analyzed (1 if present, 0 otherwise).

repositoryAuthor	repositoryName	SN	NoP	IN	FN	CoP	PT	FG	IC
tensorlayer	tensorlayer	1	0	0	0	1	1	0	1
OpenMined	PySyft	1	0	0	0	1	1	0	1
microsoft	PowerToys	1	1	0	1	0	0	0	0
microsoft	Microsoft365DSC	1	0	0	0	1	1	0	1
google	jax	1	0	0	0	1	1	0	1
google	blockly	1	0	0	0	1	1	0	1
uber	cadence	1	0	0	0	1	1	0	1
tensorflow	models	1	1	0	1	0	0	0	0
tensorflow	addons	1	1	0	1	0	0	0	1
NVIDIA	HugeCTR	1	0	0	0	1	1	0	1
microsoft	PowerStig	1	0	0	0	1	1	0	1
microsoft	vscode-cpptools	1	1	0	1	0	0	0	1
bardsoftware	ganttproject	1	1	0	1	0	0	0	1
eclipse	iceoryx	1	0	0	0	1	1	0	1
eclipse	californium	1	1	0	1	0	0	0	1
eclipse	ditto	1	0	0	0	1	1	0	1
eclipse	sw360	1	0	0	0	1	1	0	1
eclipse	nebula	1	0	0	0	1	1	0	1

Table 5.8: Results of the association rule mining showing relations between community patterns and community smells.

antecedents (Pattern)	consequents (Smell)	antecedent support	consequent support	support	confidence	lift
NoP	OSE	0,411764706	0,588235294	0,264705882	0,642857143	1,092857143
FN	OSE	0,411764706	0,588235294	0,264705882	0,642857143	1,092857143
CoP	SV	0,588235294	0,5	0,352941176	0,6	1,2
PT	SV	0,588235294	0,5	0,352941176	0,6	1,2
SN	TC	1	0,676470588	0,676470588	0,676470588	1
NoP	TC	0,411764706	0,676470588	0,264705882	0,642857143	0,950310559
FN	TC	0,411764706	0,676470588	0,264705882	0,642857143	0,950310559
CoP	TC	0,588235294	0,676470588	0,411764706	0,7	1,034782609
PT	TC	0,588235294	0,676470588	0,411764706	0,7	1,034782609
IC	TC	0,911764706	0,676470588	0,588235294	0,64516129	0,95371669

such a dataset of repositories with identified community patterns and smells, we used the APRIORI algorithm to mine relations. Table 5.8 shows the results of the association rule mining process. In these results, we can clearly see how every computed pattern by TOAD has co-occurrences with the smell Toxic Communication (TC), and in particular Community of Practice (CoP) and Project Team (PT) have a higher value in every computed metric. This could mean that when teams are heavily focused and have too much pressure from the organization, toxic communications could happen. The same two patterns also have a relation with Sharing Villainy (SV), which means that in the previously-mentioned situations community lacks high-level quality information exchange. Another interesting finding is that when the team is dispersed (geographically and culturally), showing a Network of Practice (NoP), and it is too formal in its interactions, showing a Formal Network (FN), it is probably to incur in Organizational Silo Effect (OSE), which means that there could be isolated subgroups that lack communication between them. More experimentation is needed to find actual responses to such hypotheses in real-world scenarios.

☞ **Answer to RQ$_2$.** There is a relation between community patterns and community smells.

5.3 Observations

We tried to validate TOAD with respect to YOSHI's pattern detection and characteristics computation, but we found out that the original tool has several contradictions and its results are inconsistent with the empirically defined thresholds that should have been used [21]. We still concluded that our tool correctly detects community patterns since we found no differences in the implementation of TOAD and the definitions of each metric. We concluded that our tool, even if not exactly comparable with YOSHI, is a working and accurate partial replication able to diagnose community patterns in open-source software development communities. Then we analyzed eventual relations between community patterns and smells, finding results that have both mathematical and logical reasoning. Even if such relations are not yet demonstrated, we believe that such behaviors in communities happen more frequently than we think, hence a more detailed study is needed.

CHAPTER 6

Threats to validity

To analyze validity threats in our work, we decided to adopt the model defined by Runeson and Höst in their work on guidelines to perform and report case studies in the software engineering field [42]. Such a model defines four different validity aspects to analyze:

- 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 Chapter, we will discuss such aspects separately.

6.1 Internal Validity

To answer RQ1, we decided to analyze the results of TOAD using the same repositories that were used by YOSHI for its validation. To reduce the chances of making mistakes, we decided to analyze both the community patterns identified and the metrics computed. Although we did our best to validate the tool, we could have made errors in the process invalidating the results obtained for RQ2. We could also have selected the wrong set of repositories in the context of RQ2, and the results could be biased by such a selection.

6.2 Construct Validity

The majority of construct threats are inherited by YOSHI. We already addressed several limitations, and yet there is more as we already discussed in Chapter 3. Concerning RQ2, it is trivial to identify a threat in the possible inaccuracy of the tools used to detect community patterns and community smells, hence invalidating the results obtained. We did our very best to validate the identification of community patterns of TOAD, supported by the usage of empirically defined thresholds and the original YOSHI design [4, 21], and for CSDETECTOR, it has already been validated and widely used in many other studies [8]. Another threat to construct validity is the difference in the representation of communities by the two tools. We tried to fill this gap by implementing alias resolution in TOAD, since CSDETECTOR does, and also considering a wider set of data, e.g., PRs instead of only issues, and all milestones instead of only closed ones.

6.3 Reliability

We analyzed YOSHI's repository to gather the communities used to validate the tool, but still, we could have missed some repositories and communities that were used by YOSHI's authors and that were not explicitly mentioned. We also tried to address some limitations, but in doing so we could have made biased assumptions of what could have been better, such as the use of GLOBE as the cultural framework, or the relaxing of some constraints like closed milestones. The main threat to reliability is the use of APRIORI algorithm to discover relations between community patterns and smells since there could be another more suitable method that was not considered by us. Concerning the algorithm we used, the threshold we set to consider the relation valid is also a threat to reliability.

6.4 External Validity

External threats happen to be the strongest ones. Since we found differences between the patterns computed by TOAD and YOSHI, we have no clue about how generalizable can their results be. The former has been implemented following as close as possible to the design of the original tool, adjusting some parts that we thought could improve the tool in its detection method without modifying it. The latter has shown many contradictions in its results. Hence, we think that whichever case study would be performed using both tools could produce different results. Also, the biggest threat is that the tools only analyze GitHub repositories. It could be that gathering community patterns and community smells on communities from other platforms, such as GitLab or BitBucket, could lead to different results in the relations between the two. Also, a wider set of repositories could have been used to investigate such relations.

CHAPTER 7

Conclusions and Future Work

This Chapter summarizes our research and the conclusion we gathered from this experience, also describing the future prospects of our work.

7.1 Conclusions

This thesis analyzes the underlying relations between health-related aspects in open-source software development communities, i.e., community patterns and community smells.

In doing so, we performed partial replication of a state-of-the-art tool to identify community patterns, YOSHI [4], creating our new tool **TOAD: Towards Open-source communities health Analysis and Diagnosis**. In order to use such a tool to analyze relations between patterns and smells, we needed to validate it in its detection and computations. Also, we tried to address some of the limitations of the original work, with the aim of providing a new tool able to consistently analyze the healthiness of communities.

We aimed at discovering such relations through the execution of our tool TOAD, to detect community patterns, and CSDETECTOR, to detect community smells, on a set of open-source repositories from GitHub, and the application of an association-rule mining process.

After applying our methodology, we can conclude that there are relations between community patterns and smells, and these relations have both mathematical and conceptual foundations. Also, we can affirm that the results of our new tool TOAD reflect the community types of open-source software repositories.

To sum up, our work provides the following contributions:

- An analysis of the tool YOSHI, providing contradictions and improvements in its development;
- A new tool, TOAD, which uses empirically defined thresholds and algorithmic representation [21, 4] to compute characteristics of open-source development communities and identify the community patterns successfully;
- A set of relations between community smells and community patterns that practitioners can use to identify eventual social debt in the software project.

7.2 Future work

Concerning our future research agenda, there are many things planned.

Of course, there is the need to perform a deeper and stronger analysis of TOAD, to understand better with more data if the detection of community patterns works as intended. Also, the tool has not fully addressed YOSHI's limitations and has inherited many of them, which will need to be fixed in the future. As for its scope, the tool only analyzes GitHub repositories for now, and it could be expanded to other platforms such as GitLab or BitBucket. We designed our architecture and developed our solution to make it scalable in that sense, allowing users to perform slight changes and be able to use parts of our tool independently.

For the relations, instead, we think that analyzing a wider set of repositories should have top priority as future work. Also, we could try and use other algorithms to perform association-rule mining to have a better view of the subject. Another step towards improving these relations is to use more tools to have a wider set of community smells and patterns analyzed.

Finally, we think that as a future work, we could integrate the relations we found in the workflow of TOAD, in order to have a tool able to both identify the community type and propose possible community smells that will arise, and eventually prevent the rise of social debt.

Bibliography

[1] D. A. Tamburri, P. Lago, and H. van Vliet, "Uncovering latent social communities in software development," *IEEE Software*, vol. 30, no. 1, pp. 29–36, 2013. (Citato alle pagine iv, 8 e 17)

[2] 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*, ser. ESEC/FSE 2017. New York, NY, USA: Association for Computing Machinery, 2017, p. 186–196. [Online]. Available: https://doi.org/10.1145/3106237.3106246 (Citato alle pagine 1 e 5)

[3] H. Hata, T. Todo, S. Onoue, and K. Matsumoto, "Characteristics of sustainable oss projects: A theoretical and empirical study," in *2015 IEEE/ACM 8th International Workshop on Cooperative and Human Aspects of Software Engineering*, 2015, pp. 15–21. (Citato alle pagine 1, 4 e 5)

[4] D. A. Tamburri, F. Palomba, A. Serebrenik, and A. Zaidman, "Discovering community patterns in open-source: A systematic approach and its evaluation," *Empirical Softw. Engg.*, vol. 24, no. 3, p. 1369–1417, jun 2019. [Online]. Available:

https://doi.org/10.1007/s10664-018-9659-9 (Citato alle pagine 1, 2, 7, 8, 12, 16, 17, 22, 23, 24, 27, 28, 35, 37, 38, 39, 44, 45, 56, 58 e 59)

[5] D. Tamburri, P. Kruchten, P. Lago, and H. van Vliet, "Social debt in software engineering: Insights from industry," *Journal of Internet Services and Applications*, 2015. (Citato alle pagine 1 e 8)

[6] D. A. Tamburri, P. Kruchten, P. Lago, and H. van Vliet, "What is social debt in software engineering?" in *2013 6th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)*. IEEE, 2013, pp. 93–96. (Citato alle pagine 1 e 8)

[7] M. Joblin, W. Mauerer, S. Apel, J. Siegmund, and D. Riehle, "From developer networks to verified communities: A fine-grained approach," in *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, vol. 1. IEEE, 2015, pp. 563–573. (Citato alle pagine 2 e 8)

[8] N. Almarimi, A. Ouni, M. Chouchen, and M. W. Mkaouer, "Csdetector: An open source tool for community smells detection," in *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ser. ESEC/FSE 2021. New York, NY, USA: Association for Computing Machinery, 2021, p. 1560–1564. [Online]. Available: https://doi.org/10.1145/3468264.3473121 (Citato alle pagine 2, 10, 28, 35, 48 e 56)

[9] D. A. Tamburri, P. Lago, and H. v. Vliet, "Organizational social structures for software engineering," *ACM Comput. Surv.*, vol. 46, no. 1, jul 2013. [Online]. Available: https://doi.org/10.1145/2522968.2522971 (Citato alle pagine 4 e 13)

[10] S. Onoue, H. Hata, and K. Matsumoto, "Software population pyramids: The current and the future of oss development communities," in *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and*

Measurement, ser. ESEM '14. New York, NY, USA: Association for Computing Machinery, 2014. [Online]. Available: https://doi.org/10.1145/2652524.2652565 (Citato alle pagine 4 e 6)

[11] K. Crowston and J. Howison, "The social structure of free and open source software development," *First Monday*, vol. 10, no. 2, Feb. 2005. [Online]. Available: https://firstmonday.org/ojs/index.php/fm/article/view/1207 (Citato a pagina 5)

[12] S. Goggins, K. Lumbard, and M. Germonprez, "Open source community health: Analytical metrics and their corresponding narratives," in *2021 IEEE/ACM 4th International Workshop on Software Health in Projects, Ecosystems and Communities (SoHeal)*, 2021, pp. 25–33. (Citato a pagina 5)

[13] A. Paxton, N. Varoquaux, C. Holdgraf, and R. S. Geiger, "Community, time, and (con)text: A dynamical systems analysis of online communication and community health among open-source software communities," *Cognitive Science*, vol. 46, no. 5, p. e13134, 2022. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1111/cogs.13134 (Citato a pagina 5)

[14] G. Catolino, F. Palomba, and D. A. Tamburri, "The secret life of software communities: What we know and what we don't know," in *BElgian-NEtherlands software eVOLution symposium*, 2019. [Online]. Available: https://api.semanticscholar.org/CorpusID:218757936 (Citato a pagina 5)

[15] I. Kwan, A. Schroter, and D. Damian, "Does socio-technical congruence have an effect on software build success? a study of coordination in a software project," *IEEE Transactions on Software Engineering*, vol. 37, no. 3, pp. 307–324, 2011. (Citato a pagina 5)

[16] 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 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, ser. ESEM '08. New York, NY, USA: Association for Computing Machinery, 2008, p. 2–11. [Online]. Available: https://doi.org/10.1145/1414004.1414008 (Citato a pagina 6)

[17] F. Palomba, D. Tamburri, F. Arcelli Fontana, R. Oliveto, A. Zaidman, and A. Serebrenik, "Beyond technical aspects: How do community smells influence the intensity of code smells?" *IEEE Transactions on Software Engineering*, vol. PP, pp. 1–1, 11 2018. (Citato a pagina 6)

[18] 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*, ser. IWPSE '02. New York, NY, USA: Association for Computing Machinery, 2002, p. 76–85. [Online]. Available: https://doi.org/10.1145/512035.512055 (Citato a pagina 6)

[19] 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 Transactions on Information and Systems*, vol. E99.D, no. 5, pp. 1304–1315, 2016. (Citato a pagina 6)

[20] K. Yamashita, S. McIntosh, Y. Kamei, and N. Ubayashi, "Magnet or sticky? an oss project-by-project typology," in *Proceedings of the 11th Working Conference on Mining Software Repositories*, ser. MSR 2014. New York, NY, USA: Association for Computing Machinery, 2014, p. 344–347. [Online]. Available: https://doi.org/10.1145/2597073.2597116 (Citato a pagina 6)

[21] E. di Nitto, S. Gatti, S. Invernizzi, and D. A. Tamburri, "Supporting awareness in open-source forges," 2013. [Online]. Available: https://tinyurl.com/ya3nhsqs (Citato alle pagine 8, 12, 16, 17, 24, 27, 28, 39, 45, 54, 56 e 59)

[22] D. A. Tamburri, R. Kazman, and H. Fahimi, "The architect's role in community shepherding," *IEEE Software*, vol. 33, no. 6, pp. 70–79, 2016. (Citato alle pagine 8 e 34)

[23] D. A. Tamburri, "Software architecture social debt: Managing the incommunicability factor," *IEEE Transactions on Computational Social Systems*, vol. 6, no. 1, pp. 20–37, 2019. (Citato a pagina 8)

[24] D. A. A. Tamburri, F. Palomba, and R. Kazman, "Exploring community smells in open-source: An automated approach," *IEEE Transactions on software Engineering*, 2019. (Citato a pagina 8)

[25] F. Palomba, D. Andrew Tamburri, F. Arcelli Fontana, R. Oliveto, A. Zaidman, and A. Serebrenik, "Beyond technical aspects: How do community smells influence the intensity of code smells?" *IEEE Transactions on Software Engineering*, vol. 47, no. 1, pp. 108–129, 2021. (Citato a pagina 10)

[26] N. Almarimi, A. Ouni, and M. W. Mkaouer, "Learning to detect community smells in open source software projects," *Knowledge-Based Systems*, vol. 204, p. 106201, 2020. (Citato a pagina 10)

[27] 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 *Proceedings of the 15th International Conference on Global Software Engineering*, 2020, pp. 43–54. (Citato a pagina 10)

[28] F. Palomba and D. A. Tamburri, "Predicting the emergence of community smells using socio-technical metrics: a machine-learning approach," *Journal of Systems and Software*, vol. 171, p. 110847, 2021. (Citato a pagina 10)

[29] 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 *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Society (ICSE-SEIS)*. IEEE, 2019, pp. 11–20. (Citato a pagina 10)

[30] ——, "Refactoring community smells in the wild: the practitioner's field manual," in *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Software Engineering in Society*, 2020, pp. 25–34. (Citato a pagina 10)

[31] A. Martini and J. Bosch, "Revealing social debt with the caffea framework: An antidote to architectural debt," in *2017 IEEE International Conference on Software Architecture Workshops (ICSAW)*. IEEE, 2017, pp. 179–181. (Citato a pagina 11)

[32] D. A. Tamburri, F. Palomba, A. Serebrenik, and A. Zaidman, "Discovering community patterns in open-source: a systematic approach and its evaluation," *Empirical Software Engineering*, vol. 24, no. 3, pp. 1369–1417, 2019. (Citato a pagina 11)

[33] M. De Stefano, E. Iannone, F. Pecorelli, and D. A. Tamburri, "Impacts of software community patterns on process and product: An empirical study," *Science of Computer Programming*, vol. 214, p. 102731, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167642321001246 (Citato a pagina 11)

[34] V. A. Traag, G. Krings, and P. Van Dooren, "Significant scales in community structure," *CoRR*, vol. abs/1306.3398, 2013. [Online]. Available: http://arxiv.org/abs/1306.3398 (Citato a pagina 16)

[35] M. E. J. Newman and M. Girvan, "Finding and evaluating community structure in networks," *Phys. Rev. E*, vol. 69, p. 026113, Feb 2004. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevE.69.026113 (Citato a pagina 17)

[36] 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. (Citato a pagina 21)

[37] G. Hofstede, G. J. Hofstede, and M. Minkov, *Cultures and organizations: Software of the mind*. Mcgraw-hill New York, 2005, vol. 2. (Citato alle pagine 23 e 29)

[38] G. Hofstede, "Dimensionalizing cultures: The hofstede model in context," *Online readings in psychology and culture*, vol. 2, no. 1, pp. 2307–0919, 2011. (Citato a pagina 23)

[39] R. House, P. Hanges, M. Javidan, P. Dorfman, and V. Gupta, "Leadership and organizations: The globe study of 62 societies," 01 2004. (Citato alle pagine 23 e 29)

[40] R. Agrawal, T. Imieliński, and A. Swami, "Mining association rules between sets of items in large databases," *SIGMOD Rec.*, vol. 22, no. 2, p. 207–216, jun 1993. [Online]. Available: https://doi.org/10.1145/170036.170072 (Citato alle pagine 42 e 43)

[41] S. Raschka, "Mlxtend: Providing machine learning and data science utilities and extensions to python's scientific computing stack," *The Journal of Open Source Software*, vol. 3, no. 24, Apr. 2018. [Online]. Available: https://joss.theoj.org/papers/10.21105/joss.00638 (Citato a pagina 42)

[42] P. Runeson and M. Höst, "Guidelines for conducting and reporting case study research in software engineering," *Empirical Software Engineering*, vol. 14, no. 2, pp. 131–164, Apr 2009. [Online]. Available: https://doi.org/10.1007/s10664-008-9102-8 (Citato a pagina 55)