Empirical Software Engineering (2019) 24:1369–1417
https://doi.org/10.1007/s10664-018-9659-9

Discovering community patterns in open-source: a systematic approach and its evaluation

Damian A. Tamburri[1] · **Fabio Palomba**[2] · **Alexander Serebrenik**[3] · **Andy Zaidman**[4]

Published online: 13 November 2018

Abstract

"There can be no vulnerability without risk; there can be no community without vulnerability; there can be no peace, and ultimately no life, without community." - [M. Scott Peck]

The open-source phenomenon has reached the point in which it is virtually impossible to find large applications that do not rely on it. Such grand adoption may turn into a risk if the community regulatory aspects behind open-source work (e.g., contribution guidelines or release schemas) are left implicit and their effect untracked. We advocate the explicit study and automated support of such aspects and propose YOSHI (**Y**ielding **O**pen-**S**ource **H**ealth **I**nformation), a tool able to map open-source communities onto *community patterns*, sets of known organisational and social structure types and characteristics with measurable core attributes. This mapping is beneficial since it allows, for example, (a) further investigation of community health measuring established characteristics from organisations research, (b) reuse of pattern-specific best-practices from the same literature, and (c) diagnosis of organisational anti-patterns specific to open-source, if any. We evaluate the tool in a quantitative empirical study involving 25 open-source communities from GitHub, finding that the tool offers a valuable basis to monitor key community traits behind open-source development and may form an effective combination with web-portals such as OpenHub or Bitergia. We made the proposed tool open source and publicly available.

Keywords Community patterns · Community types · Open source systems and community analysis · Empirical software engineering

1 Introduction

Modern software engineering heavily relies on open-source software (Raju 2007; Crowston et al. 2012). Paraphrasing Crowston et al. (2012): *"Over the past ten years, [open-source*

Communicated by: Jeffrey C. Carver

✉ Damian A. Tamburri
d.a.tamburri@tue.nl

Extended author information available on the last page of the article.

software] has moved from an academic curiosity to a mainstream focus [...] there are now [hundreds of] thousands of active communities, spanning a wide range of applications". Despite their high popularity, open-source communities themselves do not commonly rely on governance insights from organisations research and/or tracking their organisational status using social networks analysis (SNA), e.g., to evaluate the current social and organisational characteristics describing their community structure.

On one side, open-source communities mostly emerge and organise organically (Sadowski et al. 2008), following often fairly implicit governance structures (Capra et al. 2008; Tullio and Staples 2014), and with little or no socio-technical tracking and monitoring. On the other side, for those communities which are big enough to care for their own emerging organisational and socio-technical processes and structure, there is very limited support. For example, for these big communities, there is limited support to find out the degree to which the community is capable of engaging more actively with newcomers or sponsoring organisations, e.g., so that external parties may engage in *shepherding* (Tamburri et al. 2016) the community with explicit and informed organisational decision-making.

Currently, online applications such as OpenHub[1] or Bitergia[2] do allow to grasp several organisational and social aspects (Gamalielsson and Lundell 2013; Schweik 2013) behind open-source organisational structures (e.g., amount of member activity), however their approach would benefit from considering theories, models, types, characteristics, and best practices from organisations and social-networks research (Tamburri et al. 2013a), since these theories and insights may prove vital to avoid abandonware or failure of entire open-source forges (e.g., there are several conjectured effects known for the failure of SourceForge[3] but not their root-cause). Moreover, recent studies in open-source organisations show the need to explore *sustainable open-source communities* (Hata et al. 2015), that is, software communities with clear, explicit, and measurable governance structures and characteristics. Similarly, the literature concerning open-source community failure (Tsirakidis et al. 2009; Capiluppi et al. 2003), suggests a latent but increasing need for (semi-)automated support of social, organisational, and socio-technical characteristics of these communities.

With the aim of providing community shepherds and practitioners with such a community-oriented dashboard, in this paper we built upon previous research made in an industrial environment (Tamburri et al. 2013b) by proposing a novel automated tool, called YOSHI (**Y**ielding **O**pen-**S**ource **H**ealth **I**nformation). YOSHI is designed to support two scenarios. First, it is able to measure the organisational status of an open-source community using six key open-source community characteristics previously proposed in literature (Tamburri et al. 2013a), i.e., community structure, geodispersion, longevity, engagement, formality, and cohesion. Second, based on the previous measurements, YOSHI associates a community pattern of organisational structure types (Tamburri et al. 2012, 2013a) matching the characteristics of the community.

On the one hand, a community pattern is associated with multiple types since different sub-communities of the target community work in a different way. On the other hand, knowing the pattern and the parameters behind it, leads to diagnosing and resolving type-specific problems using mitigation strategies from organisations research (Millen et al. 2002; Wenger 1998; Ala-Mutka 2009). For example, assume the Apache Spark community

[1] http://openhub.net/

[2] https://bitergia.com/

[3] https://www.quora.com/Is-SourceForge-dying-Why-or-why-not

features a pattern of three types, associated to three sub-communities—if there are types in the pattern with opposite characteristics (e.g., an informal community type, versus a formal community type), then there may exist organisational conflicts that need resolution. The proposed tool YOSHI would allow to diagnose such conditions and act upon them using measurable quantities. We made the proposed tool publicly available and open source on GitHub.[4]

1.1 Research Questions

To assess the validity of the tool and the extent to which open-source practitioners may benefit from its usage, we validate YOSHI by conducting an empirical investigation of 25 open-source software communities aiming at providing insights with respect to two main objectives, i.e., accuracy and usefulness of the tool. On the one hand, we aim at understanding the extent to which the tool can provide developers with meaningful metrics; on the other hand, we aim at verifying whether the patterns extracted by the tool actually provide a factual view of the community structure of a software system. Specifically, we answer the following research questions:

- **RQ$_1$.** *Does* YOSHI *correctly measure the community aspects characterising different software communities?*
- **RQ$_2$.** *Does* YOSHI *provide a correct indication of the community structure of a software system?*

These research questions analyse the extent to which the output of YOSHI is reliable, evaluating the validity of (i) the metrics computed to measure the community aspects characterising a software community and (ii) the indication about the community structure of a software system.

Evaluation results show that (i) the measures computed by YOSHI correctly characterise a software community associating a pattern which reflects the community sub-structures and their way of working and (ii) YOSHI is highly reliable when employed for understanding the structure of a community. Moreover, in the context of our analyses we also discover how different community design patterns correspond to different project quality parameters such as number of stars and number of forks.

We conclude that: (1) YOSHI reliably predicts community patterns, thus allowing further studies as well as the reuse of theories and measurable quantities from organisations and social-networks research; (2) YOSHI effectively eases finding correlations between community types and community-related metrics of an open-source community.

Summarising, this paper offers three major contributions beyond the state of the art:

1. **YOSHI, a novel automated tool for open-source community design pattern detection**, which we built based on previously known community types – we made the tool publicly available and open source. The tool is designed to work jointly with web-portals such as OpenHub and Bitergia, and reuses insights and theories from organisations and social-networks research.

[4]The entire source code and running instructions are available online: https://github.com/maelstromdat/YOSHI

2. **Results achieved on a large-scale empirical study on 25 open source communities**, where we empirically evaluated the actual validity of the proposed tool as a decision support system for open source communities able to characterise their social aspects.
3. **A comprehensive replication package**, that is publicly available and contains all the data used to evaluate the tool (Tamburri et al. 2017).

1.2 Motivations

Measuring and tracking the organisational structure type and characteristics of an observable community is critical to achieve such quality for at least two reasons. First, the state of the art in organisations research, social networks analysis, management information systems and related disciplines provide many type-specific organisational problems that often recur in software engineering. For example, an extraordinary number of recurrent issues reported for overly formal organisational structures such as Formal Networks and Formal Groups (Fredrickson 1986), these issues vary from lack of motivation or trust across employees at all levels (Miles et al. 2015) to institutional isomorphism (Lai et al. 2006; DiMaggio and Powell 1983), to name a few. As a matter of fact, these factors are still reported as causes for several major software failures, e.g., in the context of global software development (Jiménez and Piattini 2008). Similarly, the lack of centralised management or leadership in Informal Networks leads to organisational stagnation (Jeppesen et al. 2011; Kim 2007)—this is suspected by many to be a cause behind open-source developer turnover (Homscheid and Schaarschmidt 2016; Li et al. 2012). Moreover, several other studies have addressed the relation between organisational structure types and characteristics with measurable software quality outcomes focusing on factors such as organisational fit (Nielsen 1995) or organisational culture difference (Siakas and Georgiadou 2002). We argue that the influence of the above organisational circumstances has seen little or no automated support in software engineering organisations as much as open-source forges - our research conjecture in the scope of this article is that automated, transparent means to measure and quantify these circumstances leads to avoiding some of the connected software friction (Avgeriou et al. 2016).

Second, software engineering research still lacks reference quality models for quantifiable organisational structures. Assuming that, as the state of the art in software engineering research has already shown (Nagappan et al. 2008; Bird et al. 2009; Nguyen et al. 2008; Pinzger et al. 2008), all software organisations and their qualities are inextricably and heavily related to software qualities, we advocate the use of organisational structure types and their measurable characteristics as means to research *community quality models*, that is, sets of metrics and stability thresholds to track software engineering organisational health. To the best of our knowledge, these instruments are still rudimentary (Jansen 2014), if not completely lacking. In pursuit of such quality models, our previous work also defined and partially evaluated a potential community quality model (Magnoni et al. 2017), systematically surveying software engineering literature as well as experienced practitioners. In the scope of this article we investigate if and how the state of the art in organisations research, as represented by known organisational structure types implemented in YOSHI can play a role in defining and predicting software community quality.

1.3 Structure of The Article

The remainder of this paper is organised as follows. Section 2 provides an overview of the background and theoretical foundations upon which YOSHI was built, as well as the research

objectives behind this article. Section 3 provides a detailed technical overview of YOSHI and the metrics it computes, while Section 4 reports the design and results of the empirical study conducted to evaluate its effectiveness. Section 5 discusses the main findings of our study and proposes new insights on the usefulness of YOSHI. Section 6 discusses the limitation of the tool as well as the threats that might have influenced the empirical study. In Section 7 we outline the related literature, before concluding the paper in Section 8.

2 Background and Research Statement

This section outlines the background in organisational structures, providing a general overview and definitions. Subsequently, the section discusses the background and general objectives of organisational structure quality research and how it relates to software engineering in general and our tool in particular. Given that the background section is dense with concepts and definitions not strictly part of software engineering research but interdisciplinary in nature, in the following we offer a nutshell summary—the interested reader can find full details in the remainder of the section.

A *software development community* is a specific type of social network upon which certain properties constantly hold (e.g., informal communication across electronic channels of open-source projects) (Tamburri et al. 2013a; Magnoni et al. 2017) across *community members*, that is, the set of people who interact in any way, shape, or form with the *practice* reflected by the community (e.g., a software product).

Across such development social networks and their many possible properties (e.g., informality, goals, membership selection, intercommunication protocols, etc.), communities can develop sub-optimal conditions which we previously defined as *community smells* (Tamburri et al. 2015; Palomba et al. 2018) in analogy to code smells—the analogy signifies that, on one hand, community smells identify unlikable circumstances (e.g., the lack of communication across different modules of a software system) but, on the other hand, these conditions do not necessarily stop or void the organisational behaviour across the community, rather, they prove detrimental and cause additional project cost in several possible ways (e.g., recurrent delays in communication, wrongful knowledge sharing).

Finally, with the term *project*, we identify the goal or shared practice that the community maintains as its central endeavour, e.g., the Apache Spark community holds the delivery of the Apache Spark product as its key *project*.

Background and Goals Digest. A community type is a social network where certain characteristics are constantly true, for example, an informal community is a social network where all interactions are *always* informal. Disciplines such as organisations research and social-networks analysis study community structures and types to measure and manage their salient characteristics to socially healthy and organisationally performant levels. YOSHI is a tool that applies that intelligence and knowledge to detect structural design patterns across open-source software engineering communities, and is able to identify nine types using their unique identifying characteristics. Our ultimate objective is using YOSHI and community patterns as instruments to assess open-source organisational quality.

2.1 Organisational Structures Explained

The literature in organisational structure research resides mostly in the following fields:

- Organisations research—in this field organisational structure types and characteristics represent more or less effective consequences of *organisational design*, i.e., the management activity of planning a strategic organisational agenda around a pre-specified organisational structure (Chatha 2003);
- Social-Network Analysis—in this field organisational structure types and characteristics represent measurable quantities that can augment social-networks from any context or domain (networks of people, communities of partners, networks of organisations, etc.) (Kilduff and Tsai 2003; Otte and Rousseau 2002);
- Cognitive Ergonomics—in this field organisational structure types represent models that allow reasoning on transactive-memory processes (Nevo and Wand 2005) (i.e., who knows what, where, etc.), information representation, as well as information exchange policies;

The following sections offer more precise definitions of organisational structures, their types and characteristics as well as outline their role in the context of this study.

2.1.1 Organisational Types and Their Characteristics

Several seminal works address organisational types in the state of the art of software engineering. For example, Mockus et al. (2002) investigate Mozilla and Apache, characterising quantitatively and qualitatively their organisational structure, but without explicitly associating a type (i.e. a set of social and organisational characteristics) from the state of the art. Conversely, for the benefit of software engineering research and practice, in our own previous work (Tamburri et al. 2013a) we strived to summarise the insights on organisational structures from the fields above as well as others, into common themes or *types* of structures. In layman terms, a structure type is a set of measurable or otherwise evident organisational characteristics (e.g., the presence of informal communication channels across an organisation). Based on how organisational characteristics influence the structure, the way of working in the structure can change radically. For example, the way of working in a Community of Practice (collocated, tightly knit, practice-focused) is different than that of a Formal Network (formal, distributed, protocol-based). Also, if characteristic X has its *highest* manifestation in a certain type, X can be used as an *identifying indicator* for that type, that is, the primary characteristic which is a necessary condition for its identification (Tamburri et al. 2013a). For example, Formality is a primary indicator for organisational structures with well-defined rules and regulations, typically dictated by corporate governance. More precisely:

$$\textit{Organisational Structure Type:}$$
$$\omega = [\delta(C_1)+, ..., +\delta(C_n)];$$

where ω represents the organisational structure type as a "sum", i.e., the combined effect of organisational and social characteristics (C_1,..., C_n). On the one hand, the characteristics themselves are heterogeneous, for example, some refer to the community's location (e.g., virtual, situated) and some refer to the closeness of community interactions (e.g., cohesion, informality). On the other hand, all these characteristics can be quantified by means of observability functions (δ), i.e., sensing functions which assign a Likert-scale value

based on the level of influence that each characteristic bears on the structure according to its members/participants. For example, an Informal Network type is strongly indicative of *informal communications* and might lead to *engaged members* (Tamburri et al. 2013a). Only informality is *necessary* for the identification of Informal Networks, and hence, a unique indicator for such types. If indeed in addition to informal communication a high degree of engagement has been observed, then we consider this highly-engaged version of Informal Networks as a distributed version of Informal Community. Fluctuation of engagement levels in this instance, during the evolution of the organisational structure, can reflect changes from Informal Community type to Informal Network or vice versa.

YOSHI Analysis Lens. YOSHI focuses on detecting community design patterns using the characteristics and types evident across an observable community, hence determining the *pattern* of types that the community exhibits across its organisational structure.

As an example of the equation above for IC see the following:

$$\begin{aligned} &\textit{Organisational Structure Type IC:} \\ &IC = [\textit{Informality(High)} + \textit{Engagement(High)}...]; \end{aligned}$$

Figure 1 visualises the point above, using the example pattern:

$$IN, WG = [\textit{Informality(High)} + \textit{Cohesion(High)}]; \tag{1}$$

in the example, a likely scenario reflects a set of globally dispersed software practitioners working over the same open-source product (e.g., a video-game) constitute an Informal Network which can show high cohesion (adding in the primary characteristic of Working

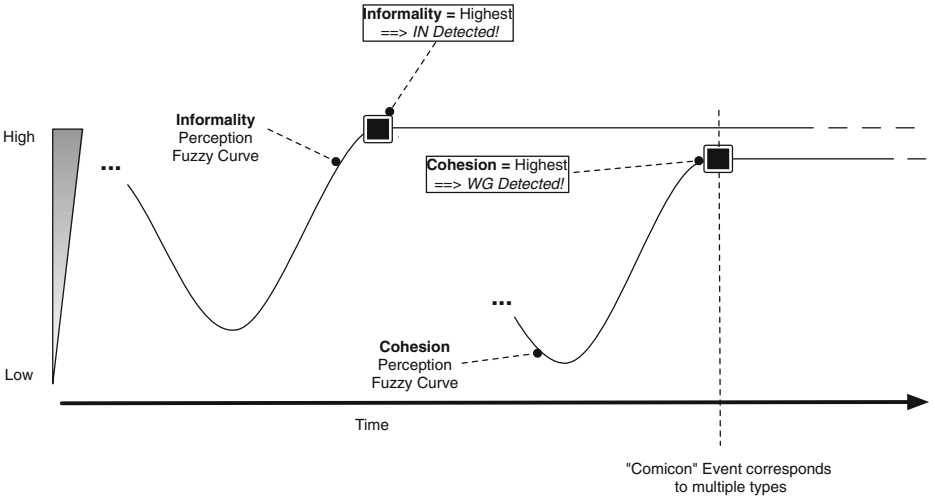


Fig. 1 An overview of the nature of organisational and social characteristics behind communities - our tool predicts community design patterns by evaluating the trend of the perception curves for primary community type indicators. The figure also reports (right-hand side) the "Comicon" event in correspondence to two identified types, from our example

Groups) when practitioners meet face-to-face (e.g., at comic conventions, "Comicons", or gaming tournaments). YOSHI would identify a single pattern including both types blended together for the "Comicon" community snapshot. Nevertheless, these two types may diverge into other types later on in the community lifetime, e.g., into a formal type during release. YOSHI currently returns types whose identifiers remain the highest and over a certain threshold for the entire duration of the observed 3-month snapshot. Consequently, considering Fig. 1 YOSHI would return a pattern constituted by both types only in correspondence of the point in time when both Informality and Cohesion are highest, and Informal Networks otherwise.

In summary, a single organisation can exhibit the traits of multiple types at once and even very different *or conflicting* types, over time—meaning that multiple, sometimes even conflicting characteristics, often blend into the same organisational structure. Addressing organisational conflict is a key objective of organisations and social-networks research (Jeppesen et al. 2011; Fredrickson 1986), and, thus, is a fundamental part of our motivation to support automated detection of community design patterns in open-source.

2.1.2 A Methodology to Discover Organisational Patterns in Software Engineering

In the recent past, a number of studies were aimed at understanding community types and their role in software engineering as well as at finding ways to use community types as reference patterns during software processes. Literature review reveals a total of more than 70 organisational and social structure characteristics (Tamburri et al. 2013a) to be measured for fully describing community structure types. Out of these characteristics a total of 13 characteristics were distilled, each individually reflecting a single type. In the scope of this paper, we focus on detecting community design patterns which feature the six characteristics that we were able to operationalise for open-source communities, namely, (1) community structure,[5] (2) formality, (3) engagement, (4) cohesion, (5) longevity and (6) geodispersion. These aforementioned characteristics were operationalised in YOSHI as an original contribution of this paper (see Section 3.3).

In fact, contrarily to literature in organisations research (Prandy 2000; Mislove et al. 2007; Ryynnen 2012) where organisational types and characteristics are studied *qualitatively*, as an original contribution of this paper, we measure the *quantitative manifestations* of community characteristics, namely, we use quantitative, automatically measurable indicators of the perception functions introduced previously. For example, to measure engagement, we evaluate together the amount, frequency, and kinds of contributions of an open-source community member with respect to its peers.

In our early exploratory experiments with community types and patterns while designing YOSHI automations, we observed that (1) different levels of the same characteristics correspond to different types and (2) measuring open-source communities reveals at least two co-existing types. From this early experimentation, we made two observations. First, YOSHI must be designed to detect *design pattern*s composed of recurrent community characteristics and their corrresponding types. Second, it is not sufficient to only measure the six characteristics above. Automated detection of organisational design patterns demands a way to identify the level of their highest manifestations above all remaining characteristics such that the most prominent community types can be revealed and distilled into a pattern.

[5]The first characteristic, structure, is a necessary pre-condition to all of them; in fact, all communities are social-networks (SNs) that exhibit a community structure across which certain characteristics remain constant.

Consequently, we engaged in and contributed to a highly active open-source community along a 15-month ethnographical study of its organisational structure (di Nitto et al. 2013), for the purpose of determining empirical thresholds to all our primary community indicators.

Table 1 provides an overview of the above results, briefly describing community types, their indicators, as well as highlighting the empirical thresholds elicited as part of our ethnographical research (di Nitto et al. 2013). The thresholds allow determining high or low values for community indicators, thus allowing identification.[6]

In what remains of this subsection, we provide an overview of the thresholds that we mention in Table 1. In particular, in previous work (di Nitto et al. 2013), we were interested in ways to measurably increase the awareness of open-source developers over known organisational and socio-technical characteristics of communities from organisations and social-networks research (see Table 1). For this reason, one of the co-authors of this study along with two master students started contributing to Apache Allura, an open source community building the infrastructure behind SourceForge, a widely known open-source forge. In this process of contribution, the following data was gathered for the first interaction by the three observers: (a) guidelines of contribution; (b) code of conduct across the community; (c) expected contribution. Moreover, for the rest of our 15-month involvement, every other interaction with the community was documented as follows: (a) type of interaction (direct/indirect); (b) involved actors (presence of communication intermediaries); (c) means of communication (e.g., formal/informal means); (d) perception of "tone" of communication (formal/informal); (e) location of the involved participants and organisations; (f) explicit/implicit guidelines for contribution in question; (g) previous members' relation with observers or amongst themselves; (h) delay in response. Finally, the following data was elaborated in a conclusive summary of the community: (a) skills profile of community members; (b) roles and responsibilities; (c) organisational structure sociogram (Kilduff and Tsai 2003).

Subsequently, we sought to associate a ground-truth set of community types and characteristics corresponding to the data thus obtained. Hence, at the end of the 15-month study, we asked 7 top-level contributors to Allura their perceived values over the characteristics from Table 1 and their perceived open-source community type(s), if any. Through this process, Allura was determined to be a *Formal Network* type blended with a *Network of Practice*—this empirically defines two thresholds for the two primary characteristics that manifest for those types: (1) *Formality* - the highest primary characteristics reflecting formality in Allura would define our Formality threshold; (2) *Geodispersion* - the average geographical and cultural distance between Allura members would define our Geodispersion threshold.

Concerning the remaining characteristics, we analysed our data on developer interactions. First, we observed *Informality* manifesting itself among the four core Allura maintainers. Focusing on the interactions among the four developers in question, we isolated their commonalities (e.g., they all shared previous relations on other projects, they all shared at least three background skills, etc.) and evaluated thresholds for resulting factors.

Similarly, we observed that *Engagement* and *Cohesion* of Allura developers were very high when the community was closing in on a release of its platform. Consequently, we measured Cohesion (represented by the well known social-network analysis metric (Kilduff

[6]The interested reader can find detailed information and full characterisation of each type in our previous work (Tamburri et al. 2013a,b, 2016)

Table 1 Organisational structure types, an overview from previous work

Name	Description	Indicator	Empirical Threshold
Communities of practice (CoP)	A CoP consists of collocated groups of people who share a concern, a set of problems, or a passion about a practice. Interactions are frequent, face-to-face, collaborative (to help each other) and constructive (to increase mutual knowledge). This set of social processes and conditions is called situatedness (Gallagher 2006). An example is the SRII community[a] which gathers multiple CoPs (corporate and academic) into a single one, meeting physically to informally exchange best practices in services science.	Situatedness	Global Distance $<$ 4926 Kilometers
Informal Networks (IN)	INs are loose networks of ties between individuals that happen to come informally in contact in the same context. Primary indicator is the high strength of informal member ties. Finally, IN do not use governance practices (Cross et al. 2005). An example in academia, is the informal and loosely coupled set of research communities around a single topic (e.g., computer science) is a world-wide informal network.	Informality	Formality Levels $<$ 0.1; Global Distance $>>$ 4926
Formal Networks (FN)	FNs rigorously select and prescribe memberships, which are created and acknowledged by FN management. Direction is carried out according to corporate strategy and its mission is to follow this strategy (Tamburri et al. 2013a). An example in software engineering is the OMG (Object Management Group): it is a formal network, since the interaction dynamics and status of the members (i.e. the organizations which are part of OMG) are formal; also, the meeting participants (i.e. the people that corporations send as representatives) are acknowledged formally by their corporate sponsors.	Formality	Formality Levels $>$ 20; Global Distance $>>$ 4926
Informal Communities (IC)	ICs reflect sets of people part of highly-dispersed organisation, with a common interest, often closely dependent on their practice. They interact informally across unbound distances, frequently over a common history or culture (e.g. shared ideas, experience etc). The main difference they have with all communities (with the exception of NoPs) is that their localisation is necessarily dispersed (e.g., contrarily to INs where networked interactions can also be in the same timezone or physical location) so that the community can reach a wider audience (Tamburri et al. 2013a). Loosely-affiliated political movements (such as green-peace) are examples of ICs: their members disseminate their vision (based on a common idea, which is the goal of the IC).	Engagement	Engagement Levels $>$ 3.5
Networks of Practice (NoP)	A NoP is a networked system of communication and collaboration that connects CoPs (which are localised). In principle anyone can join it without selection of candidates (e.g. Open-Source forges are an instance of NoP). NoPs have the highest geodispersion. An unspoken requirement is expected IT literacy (Ruikar et al. 2009). For example, previous literature (Bird et al. 2009) discusses Socio-technical Networks in software engineering using the exact terms with which NoPs are defined in literature.	Geodispersion	Global Distance $>>$ 4926

Table 1 (continued)

Name	Description	Indicator	Empirical Threshold
Workgroups (WG)	WG are made of technical experts whose goals span a specific business area. WGs are always accompanied by a number of organisational sponsors and are expected to generate tangible assets and benefits (i.e., Return-On-Investment). Fundamental attributes of WGs are collocation and the highest cohesion of their members (e.g., long-time collaborators). For example, in software engineering, the IFIP WG 2.10 on software architecture[b] is obviously a WG, since its effort is planned and steady, with highly cohesive action of its members, as well as focused on pursuing the benefits of certain organisational sponsors (e.g. UNESCO for IFIP).	Cohesion	Cohesion Levels > 11; Global Distance < 4926 Kilometers
Project-Teams (PT)	PTs are fixed-term, problem-specific aggregations of people with complementary skills who work together to achieve a common purpose for which they are accountable. They are enforced by their organisation and follow specific strategies or organisational guidelines (e.g. time-to-market, effectiveness, low-cost, etc.). Their final goal is delivery of a product or service (Tamburri et al. 2013a).	Time-Boxed Longevity	Longevity < 93 Full-time Equivalent Man-days; Global Distance < 4926 Kilometers
Formal Groups (FG)	FGs are comprised of people which are explicitly grouped by corporations to act on (or by means of) them (e.g. governing employees or ease their job or practice by grouping them in areas of interest). Each group has a single organisational goal, called mission (governing boards are groups of executives whose mission is to devise and apply governance practices successfully). In comparison to Formal Networks, they seldom rely on networking technologies, on the contrary, they are local in nature and are less formal since there are no explicit governance protocols employed other than the grouping mechanism and the common goal. Examples of formal groups in software engineering are software taskforces, e.g. IEEE Open-Source Software Task Force[c].	Explicit Governance Structure	Formality Levels > 0.1 and <20; Global Distance < 4926 Kilometers
Social Networks (SN)	SNs represent the emergent network of social ties spontaneously arising between individuals who share, either willingly or not, a practice or common interest. Conversely, an unstructured network is (often by-design) not constrained by any design or structural tie (e.g., a common social practice) (Zich et al. 2008). SNs act as a gateway to communicating communities (Cross et al. 2005).	Community Structure	Structured Network = True

The four types not identified by YOSHI are omitted for the sake of space

[a] www.theSrii.org

[b] http://www.softwarearchitectureportal.org/

[c] http://ewh.ieee.org/cmte/psace/CAMS_taskforce/index.htm

and Tsai 2003)) and Engagement levels (represented by summing all possible contributions that members would make to the release of Allura and computing an average).

In the same study, to strengthen the validity of our thresholds, we measured and empirically evaluated the metrics and thresholds for an additional four communities hosted on SourceForge, seeking and successfully evaluating the agreement of those communities' members with our type predictions.

In the scope of this article, we sought to operationalise the metrics defined and evaluated in our previous work (di Nitto et al. 2013) offering three tool-specific contributions beyond previous work:

1. a tool designed for **large-scale use**: in our previous study the measurements and empirical analysis was conducted by hand, using crude statistical analysis and focused on distilling the type of four communities only, while in this article we focus on offering an automated tool designed for large scale use and using GitHub data. Moreover, the empirical evaluation in the scope of this article encompasses 25 randomly-sampled open-source communities.
2. a tool designed for **precision**: in order to be actionable, a type prediction needs to be accurate; in our previous study we used a single quantitative metric per every primary characteristic, while with YOSHI we provide between 1 and 3 non-overlapping metrics in the detection pattern of characteristics for which our prediction in previous work was imprecise. Moreover, we offer an evaluation of YOSHI precision using null-model analysis.
3. a tool intended for **further replication** and open-source release; our study of community design patterns in open-source reflects the fundamental research of open-source organisational structures and we want to encourage others to pursue the research path we are currently exploring. In this study we offer a completely free and open-source replication package to call for, and encourage verifiability.

As a result, the study reported in this article offers a more precise, scalable, replicable, and verifiable tool along with its empirical evaluation results.

2.2 Organisational Structure Quality

Despite the fact that previous work on open- and closed-source software communities does in fact offer relevant insights into the characteristics of the different organisational structure types, it is important to note that: (i) there is still a lack of tools that provide automatic identification of community characteristics and type; (ii) previous work has been mainly oriented toward industrial environments, thus missing a detailed analysis in the context of open-source teams, which are becoming ever more important for the development of both academic and industrial software (Raju 2007; Crowston et al. 2012).

Such an analysis is of paramount importance to highlight commonalities and differences among the different organisational structures in different development contexts, and to understand to what extent the management and evolution of open-source systems may benefit from the usage of community-related information. Moreover, some organisational types may work better than others for the purpose of software engineering and evolution; this line of inquiry reflects organisational structure quality and can be assisted by the use of tools such as YOSHI which map open-source communities onto known organisational types and characteristics and their *quality*.

The quality of an organisational structure generally refers to the organisational structure's fitness for purpose, i.e., the measurable degree to which the structure is fitting with its

objective (Espejo 1993; Afsar and Badir 2015; Oreja-Rodriguez and Yanes-Estevez 2006). In our domain of software engineering, a quality organisational structure refers to better software, which is of more sustainable and measurable technical qualities (Nielsen 1995). For example, the Jet-Propulsion Laboratory at NASA can be said to have a high-quality organisational structure since it produces and maintains software which is virtually error-free[7] through a combination of organisational as much as technical tools and approaches.

3 YOSHI: An Automatic Tool for Discovering Community Types

This section reports the implementation details behind YOSHI, as well as the details on the architecture and the functionalities currently implemented in the tool. As previously introduced in Section 2, all operationalisations and detection patterns follow the Goal-Question-Metric approach (Basili et al. 1994) and use empirically-defined thresholds from previous work (di Nitto et al. 2013).

3.1 The YOSHI Approach to Open-Source Community Design Patterns Detection: General Overview

YOSHI is a social-networks analysis tool specifically designed for detecting open-source community types. The tool focuses on determining the levels of the previously-mentioned identifying characteristics, and combines specific version-control and committer activity data implemented in an information retrieval component (see bottom of Fig. 5). For example, to determine how formal a community is, YOSHI looks at how many levels of control are assigned across repository contributors. Similarly, to evaluate engagement YOSHI looks both at the technical (e.g., commits, pull requests) and social or organisational (e.g., comments, new watchers) activities.

Once all characteristics are determined, YOSHI runs Algorithm 1 to determine the community type a given repository refers to. It is important to remark again that the tool allows to identify the existence of community types by looking at the existence of key community characteristics as well as their combination. For this reason, YOSHI *identifies a community design pattern featuring multiple types within a certain repository*; several possible scenarios may exemplify this, e.g., multiple sub-teams working as different community types or the community works with different types at different phases in its organisational activity.

To precisely distinguish the types existing in the observed organisation, YOSHI iteratively uses an algorithmic representation (see Section 3.2) of the decision-tree we previously evaluated in industry (Tamburri et al. 2013b). The decision-tree in question (reported in Fig. 2) encodes the set of relations (e.g., implication or mutual-exclusion) across primary indicators for community types from Table 1. This set of relations forms, by definition, a partial-order

[7] https://www.fastcompany.com/28121/they-write-right-stuff

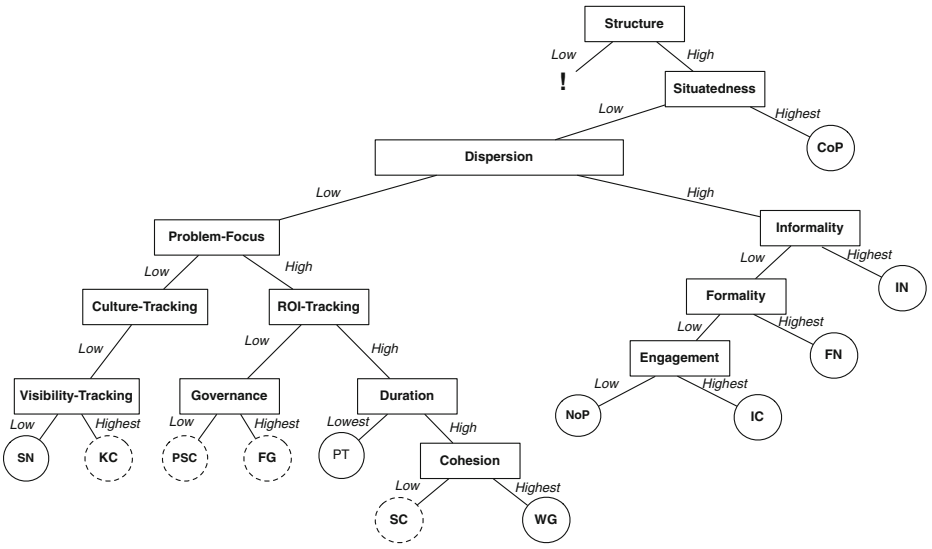


Fig. 2 A decision-tree for organisational structures - dotted nodes identify types not currently implemented in YOSHI

function, i.e., a function that associates an ordering or sequencing to the elements of a set. The decision-tree (see Fig. 2) is a representation of this partial-order function and is to be visited top-to-bottom (most generic to most specific type) and right-to-left (most collocated to most dispersed type).[8] YOSHI iterates on the decision-tree until no new community types are discovered over available data.

To exemplify the workings of the decision-tree, consider the tree-visit reflecting the identification of FNs in Fig. 3.

Finally, YOSHI is able to visualise the software development network and its statistics over a world map, reporting statistics in *.csv format—this feature is implemented in YOSHI 's own visualisation component.

YOSHI does not offer any insights over the technical qualities of the artefacts worked on by open-source communities under observation (e.g., software architecture, code, etc.), since these aspects are covered by several other state-of-the-art tools, e.g., SonarQube, CAST Software, or Titan (Xiao et al. 2014).

The above approach, can be easily replicated, generalised or further specialised at will. For example, the key organisational and socio-technical characteristics from the state-of-the-art (Tamburri et al. 2013a) may be observed through other, possibly more precise means (e.g., Natural-Language Processing (Manning and Schütze 1999), Neuro-Linguistic Programming (Molzberger 1986)). Similarly, specific tools (or YOSHI forks) can be designed to address a more precise identification of one or two specific community types, e.g., focusing on Communities and Networks of Practice.

[8] All relations and decision-tree functional demonstration by construction can be found online at http://tinyurl.com/mzojyp2

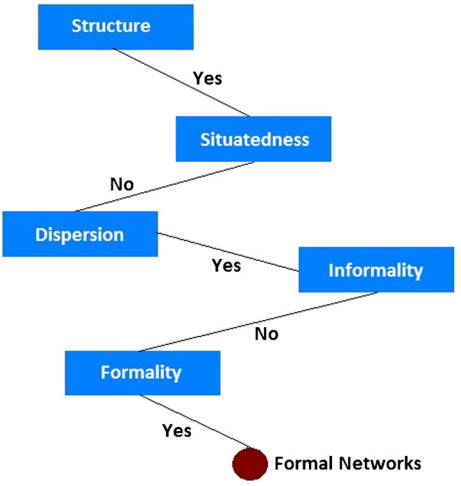


Fig. 3 A decision-tree for organisational structures - YOSHI 's visit to identify FNs

3.2 YOSHI : Algorithmic Representation

Algorithm 1 shows YOSHI's measurement function `measure()` as applied to observable open-source communities. To extract community types from observable data, Algorithm 1 is executed as follows.

- YOSHI establishes that there is in fact a high degree of community structure: $\textbf{\textit{measure}}(structure) == high$;
- YOSHI measures the indicators for the remaining five community characteristics: $m[] \leftarrow \textbf{measure}(GEO, LON, ENG, For, COH)$;
- YOSHI ascertains that characteristics are not null: $\textbf{Assume}(m! = \emptyset)$;
- YOSHI applies empirical thresholds (di Nitto et al. 2013) and returns a certain community type if and only if its identifier has been found as "Highest": $T_x \leftarrow \textbf{True} \iff \textbf{Value}(m_x) = Highest \wedge \textbf{Attribute}(m_x) = T_{identifier}$;

The 5 characteristics (besides community structure) computed by YOSHI (GEO, LON, ENG, For, COH in Algorithm 1) are, intuitively: (GEO) geodispersion; (LON) longevity; (ENG) engagement; (For) formality; (COH) cohesion. The characteristics are operationalised in the tool as detailed in the following subsections.

Algorithm 1 YOSHI algorithm for community type detection using thresholds from previous work.

if measure$(structure) == high$ **then**
  $m[] \leftarrow \textbf{measure}(GEO, LON, ENG, For, COH)$
  if $m! = \emptyset$ **then** $Comm_{type} \leftarrow [T_1 ... T_n]$
    where:
    forall$x = 1..8$:
    $T_x \leftarrow \textbf{True} \iff \textbf{Value}(m_x) = Highest \wedge \textbf{Attribute}(m_x) = T_{identifier}$
  end if
end if
return $Comm_{type}$

3.2.1 Community Structure

As operationalised within YOSHI, this characteristic represents the ability to distinguish a non-trivial organisational and social structure within the observed set of people working on a project. Establishing this characteristic is paramount to identify any community type, since, by definition, organisational structures are sets of constant properties acting across social networks exhibiting community structure (Tamburri et al. 2013a; Newman and Girvan 2004; Newman 2003). The success of open-source projects crucially depends on the voluntary contributions of a sufficiently large community of users. Apart from the size of the community, *Structure* can be identified by looking at the evolution of structural features (e.g., connection density) of collaborations between community members. To analyse the social structure of communities, we collected data regarding user collaborations using API requests to each analysed repository. A definition of "community" in the context of social networks analysis is a subnetwork whose intra-community edges are denser than the inter-community edges (Kilduff and Tsai 2003). YOSHI computes a network of nodes representing community members and edges representing any particular social or organisational interaction between any two members.

Detection Strategy. Two nodes are connected if at least one of the following conditions holds:

1. Common projects: two community members have at least one common repository to which they are contributing, except for the currently analysed repository;
2. List of followers: between the considered community members exists either a "is following" or "follows" relation;
3. Pull request interaction: we consider the connection between the pull request author and other community members that are participating on the pull request.

Consequently, a structure exists if at least one of the above parameters is met within the standard analysis window of 3 months considered by YOSHI: the 3-month window is a common practice in organisations research (Traag et al. 2013) to observe organisational activity. *Structure* is therefore a binary characteristic, to be possibly refined into a ratio or degree in the scope of future work around the tool.

3.2.2 Community Geodispersion

As operationalised within YOSHI, this characteristic represents the cultural and geographical dispersion between community members. Establishing this characteristic is key to identifying either a *Network of Practice* (high geodispersion) or a *Community of Practice* (geodispersion low or none). For geographical dispersion (`GeoDispersion` class) YOSHI retrieves community members' specified location form their own profile and uses it to compute the median and standard deviation of the distance between them and to create a geographical distribution map (Li et al. 2010) and, for cultural dispersion, YOSHI computes (`CulturalDispersion` class) Hofstede cultural distance metrics (Hofstede et al. 2010) and their standard deviation.

Detection Strategy. Because the location values do not provide geographical coordinates, YOSHI first uses location values to calculate Hofstede metrics and their standard deviation per-member and then exploits the `Google Geocoding` API to convert member addresses into geographic coordinates to compute the spherical distance between any two community members. Figure 4 shows an example geographical distribution map. The red dots on Fig. 4 represent developers which are currently members of the targeted development community. As defined in our previous work (di Nitto et al. 2013), *Geodispersion* is computed as the sum of standard deviations of geographical and cultural difference (i.e., the sum of standard deviation of geographical distance across members and of all Hofstede metrics between all pairs of members across the community). From a mathematical perspective, YOSHI averages the variances of the two quantities and then takes the square root to get the average standard deviation; the tool also provides an option to choose for the Wolfram Alpha compute engine as an alternative to local JVM computations.

3.2.3 Community Longevity

As operationalised within YOSHI, this characteristic represents the committer's longevity as a member of the observed community. Establishing this characteristic is essential to identifying *Project Teams and Problem-Solving Communities* (low or time-bound longevity) or *Workgroups* (high longevity). Committer longevity is a measure of how long one author remains part of the community.

Detection Strategy. YOSHI iterates through the list of commits and extracts: (a) committing member; (b) creation date. These are then used for updating the data from `commiterStartDate` and `commiterStopDate` for every member. After all commits have been preprocessed, YOSHI computes the *Longevity* value for each member as the difference between the two previously determined values.

Fig. 4 A geographical distribution map in YOSHI

3.2.4 Community Engagement

As operationalised within YOSHI, this characteristic represents the participation levels across the analysed community, intended as the amount of time the member is actively participating with community-related actions. Establishing this characteristic is essential to identifying *Informal Communities or Informal Networks* (high engagement). Also, evaluating this characteristic is essential for several community health reasons, for example, the case study presented by Kujala et al. (2005) shows that developer engagement in their software projects is key in successful project development and has positive effects on user satisfaction.

Detection Strategy. To establish engagement, YOSHI computes the following data about each community member:

1. Total number of pull-request comments;
2. Monthly distribution of total posted pull/commit comments—YOSHI extracts pull-request and commit comments posted by community members;
3. Number and list of repository active members (i.e., the members who committed at least once in the last 30 days)—YOSHI uses attributes values to measure the number of commit events initiated by users;
4. Repository watcher members, i.e., third-parties who receive notifications of the activity across the community;
5. Subscriptions, i.e., third-parties who get digests of commit activity across the community;
6. Distribution of commits for each user;
7. Distribution of collaborations on files—YOSHI examines the development activities of repository contributors to see if they were working together on common issues opened in the standard GitHub issue-tracker.

Finally, *Engagement* levels across the community are established as the member medians of the measurements above.

As an illustrative example, we focus on the value which indicates how tightly project members collaborate on repository artefacts, that is, the number of community members that commit on common repository artefacts. YOSHI uses the `ContentsService` and `DataService` GitHub API classes to retrieve the repository file structure and associated commits. YOSHI then uses the `CommitService` GitHub API class that provides the `pageCommits` method for retrieving the history of commits for each file. In summary, YOSHI extracts authors for each commit and adds them to the set of file contributors. The result of these preprocessing operations is a `HashMap` which stores the sets of contributors for each repository artefact. This map allows us to determine the number of community members that commit on common repository artefacts. Each entry from this collection represents the set of connections that a repository user has established by common collaboration on repository items.

3.2.5 Community Formality

As operationalised within YOSHI, this characteristic represents the level of control (access privileges, milestones scheduling and regularity of contribution) exercised or self-imposed

on the community. Establishing this characteristic is essential to identifying *Formal Groups or Formal Networks* (high formality). Also, evaluating this is essential for several reasons. For example, as reported by Crowston et al. (2012), open-source communities' approach to project milestones does not follow a common pattern. The results show that some projects have quite informal release schedules, following the pattern of releasing early and releasing often, whilst in other projects releases are more informal and come at an irregular rate (Glance 2004). Depending on the formality type, different governance support schemes might apply (e.g., formal community types as opposed to informal ones (Tamburri et al. 2013a)).

Detection Strategy. YOSHI uses three measurements for formality:

1. Membership types—in general, Contributor and Collaborator are typical GitHub membership types and therefore, the levels of control assigned to repository members are +1 for contributors and +2 for collaborators (which have more privileges);
2. Project milestones—we used the Milestones GitHub API to extract the list of milestones associated to a project;
3. Project lifetime—this value is computed using the earliest and latest commits for a project, which are obtained using the API class `CommitService`.

Formality is then determined as the mean membership type divided by the milestones per project-lifetime ratio.

The division above is mathematically grounded as follows. Since GitHub only allows contributor and collaborator as membership types, YOSHI associates a 1 to a contributor membership type and a 0 to collaborator membership type. Hence, the average number of contributors (i.e., the 1's) divided by the amount of work they have been able to carry out gives an indication of how well-structured their collaboration, co-location, co-operation works and hence, an indication of the *formality*. Conversely, the more external collaborators (i.e., the 0's) there are, the less formal (i.e., closer to 0 formality) the structure will be.

For the sake of completeness, we only elaborate on how YOSHI detects member contributions and evaluates membership type, as well as defining the contributor & collaborator subset. First, for each community member, YOSHI retrieves the set of repositories to which he/she has contributed. One of the following actions is considered a contribution: (a) a commit to a project's default branch or the gh-pages branch; (b) opening/closing an issue; (c) proposing a pull request. Second, YOSHI defines the set of repository members as the union of repository collaborators and repository contributors. The `CollaboratorService` GitHub API class allows retrieving the sets of collaborator-users for each of the considered repositories, whilst the `RepositoryService` GitHub API class that provides us with set of contributor-users.

3.2.6 Community Cohesion

As operationalised within YOSHI, this characteristic represents the level to which there are tight social or organisational or technical relations among members of the community. It is worth noting that we inherit the term "community cohesion", its definition, and rationale from the state of the art in working-groups (Moody and White 2003; Hung and Gatica-Perez 2010; Giraldo and Passino 2016); the definition in question includes a strong connotation

of community cohesion (Moody and White 2003; Hung and Gatica-Perez 2010) which is associated to low cognitive distance among members and hence, high expertise overlap (Nooteboom et al. 2006).

Detection Strategy. For each community member YOSHI determines the number of followed or following users, which are also members of the same community and share significant expertise overlap (i.e., common programming language skills). Using the `UserService` API class YOSHI extracts the list of users that a repository member is following and the list of users that are following his activity. From these two lists YOSHI excludes the users that were not members of the currently analysed repository. Applying the same preprocessing operations for all repository members, the result is a `Map<User, Integer>` representing a mapping between users and their number of team-followers. *Cohesion* is computed as the community median of the composite measurements above.

For the sake of completeness, we elaborate on how YOSHI detects common skills. In general, GitHub user profile attributes do not include their technical skills. In substitution, YOSHI uses the `RepositoryService` API class to retrieve the repositories to which a user has made contributions. Repository entities include the main programming language attribute value which allows us to compute a set of programming languages from the list of repositories. For each repository, we obtain a map `Map<Contributor, Set<String>>` representing the mapping between repository members and a crude list of their acquired technical skills. These values are used as a basis for determining followers with common skills. Using this data for each repository member we compute the list of projects to which they have contributed and determine the number of projects to which members of the current project community have collaborated. Finally, the `WordFrequency` class in YOSHI uses the content of messages exchanged between community members including the commit messages and pull requests messages to determine the most frequently used words and categorise them into skills using a taxonomy of software engineering skills of our own design (Tamburri and Casale 2017).

3.3 YOSHI—Architecture

Figure 5 shows a basic view of YOSHI's software architecture using a basic input-output control flow diagram (Bass et al. 1998). YOSHI has a modular architecture arranged in three components.

First, an information retrieval component (bottom part of Fig. 5) is responsible for retrieving data with which YOSHI can perform its functions. The component automates the retrieval of data from public repositories of projects hosted on Github, using GET requests from GitHub Archive to obtain publicly available data. The retrieval component extracts data from two data sources: source code management systems and issue trackers. First, GitHub Archive records the public GitHub timeline, archives it, and makes it easily accessible for further analysis; the archive dataset can be accessed via Google BigQuery. This data is used to compute attributes' values related to the software development process and study targeted open-source software development communities.

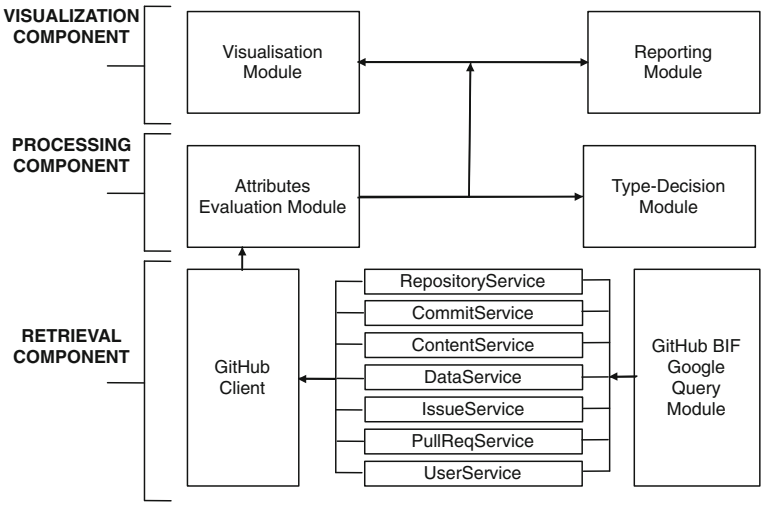


Fig. 5 YOSHI high-level architecture

Second, the processing component is responsible for evaluating metrics using data available from the retrieval component and to enable community detection (see Algorithm 1). The component uses: (a) Gephi—a Java library which provides useful and efficient network visualisation and exploration techniques; (b) Google Geocoding API—used for converting the addresses of repositories members into geographic coordinates, which is used to calculate distances; (c) a direct way to access services via an HTTP request; (d) an implementation of Algorithm 1.

Third, the visualisation component uses data computed by the processing component to create graphical representations of community members' geographical distribution. This component is able to export images and Comma-Separated Values (CSV) files for the produced representations. Finally, the current implementation of YOSHI also supports reporting of computed characteristics, their composing metrics, their values and the resulting community design patterns. Furthermore, YOSHI was designed in a modular, service-based architecture, so as to be easily extended with third-party tools (e.g., sentiment analysis (Novielli et al. 2014; Jongeling et al. 2017), natural-language processing (Arnaoudova et al. 2015)).

4 Evaluation

4.1 Study Design

The *goal* of the study is to evaluate the ability of YOSHI to discriminate the different community types in open source, with the *purpose* of understanding to what extent the proposed method can be adopted to analyse the social relationships occurring among the developers of a software system. To achieve this goal, we explore two main research questions aimed at (i) evaluating the accuracy of the measurements provided by YOSHI and (ii) evaluating the potential usefulness of the tool in practice:

– **RQ**$_1$. *Does* YOSHI *correctly measure the community aspects characterising different software communities?*

Table 2 Characteristics of the Software Projects Considered in the Study, as extracted from GitHub on April 2017 - Domain Taxonomy tailored from literature

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

– **RQ$_2$**. *Does* YOSHI *provide a correct indication of the community structure of a software system?*

The *context* of the study consists of 25 open source software communities coming from the GitHub repository, sampled according to guidelines from the state of the art (Falessi et al. 2017) and refining our results using best-practice sampling criteria (Kalliamvakou et al. 2016). Table 2 reports the characteristics of the subject projects[9] in terms of (i) their size measured as number of public releases issued and number of commits performed over their history, (ii) number of contributors who committed at least once to the repository, and (iii) their application domain according to the taxonomy proposed by Borges et al. (2016). To select this dataset, we first ranked the GitHub projects based on number of commits, to control for project activity; in this respect, a fixed boundary of no less than 100 commits was adopted. Then, projects were filtered based on number of members (at least 10) and number of LOCs (at least 50k): in this way, we allowed the selection of non-trivial communities

[9]Characteristics extracted on April 2017

that have to deal with large codebases. Moreover, we also based our selection on diversity: in cases where two projects had the same scope, we randomly excluded one of them in order to pick a population that was as different as possible (note that the domain might still be the same, as it refers to the general objective of a certain project (Borges et al. 2016)). Finally, we have manually inspected the selected projects and made sure that all of them are real projects (rather than student projects, assignments, etc.), as suggested by recent work (Munaiah et al. 2017). The specific query employed for the selection of the subject projects was done on April 2017 and can be found in our on-line appendix (Tamburri et al. 2017).

To answer our first research question, we evaluated whether the metrics computed by our approach actually represent valid community measurements: indeed, a necessary condition to provide automated support for community steering and governance is that YOSHI delivers reliable insights into key community characteristics and type. To this aim, it is *necessary and sufficient* that the metrics coded within YOSHI satisfy the representation condition (Fenton 1991), given that the decision-tree algorithm within YOSHI only represents a partial-order function among said community characteristics identified by the metrics. According to Fenton (1991), the representation condition for a metric holds "*if and only if a measurement mapping maps the entities into numbers, and empirical relations into numerical relations in such a way that the empirical relations are preserved by the numerical relations*". This means that, for instance, paraphrasing from Fenton (1991): "if we have an intuitive understanding that A is taller than B, then also the measurement mapping M must give that M(A) > M (B). The other way around, if M(A) > M (B) then it must be that A is intuitively understood to be taller than B". In our work, we ran YOSHI on the subject systems in our dataset and then, for each metric computed by the approach, we evaluated the representation condition using the guidelines provided by Fenton (1991).

In order to answer $\mathbf{RQ}_2$, we conducted a validation aimed at verifying the quality of the community structure extracted by YOSHI. As explained in Section 3.2.1, this is the main characteristic that leads to the identification of a community type, and its validation provides insights into the meaningfulness of the operations performed by our tool (di Nitto et al. 2013). To evaluate this aspect, we firstly extracted the information about the actual community structure of the communities considered. As ground truth we exploited the OpenHub community,[10] which reports data about different aspects of software communities, including a community structure modularity indicator comprised in the set $\{LOW, MEDIUM, HIGH\}$. It is worth noting that such data is **not** computed automatically but rather manually retrieved by the owners of OpenHub without the usage of proxy metrics, as directly reported in the OpenHub blog (Sands 2018) as well as previous literature in the field (Chełkowski et al. 2016; Druskat 2016). While we cannot speculate on how the owners of OpenHub manually classify communities based on their community structure, it is important to note that this data is constantly validated by the community around the platform, thus allowing us to be confident about its actual validity.

In the second place, we compared the social interactions detected by YOSHI with the ones identified by a *null-model* (Cohen 1988), i.e., a network which matches the original network in some of its topological features, but which does not display community structure. Using this strategy, we were able to verify whether the graph built by YOSHI actually displays a community structure or whether it is no better than a randomly created graph. More specifically, in our context we compared our approach with the null-model proposed by Newman

[10] https://www.openhub.net/

and Girvan (2004), i.e., a randomized version of the original graph, where edges are rewired at random, under the constraint that the expected degree of each vertex matches the degree of the vertex in the original graph. The comparison was made in terms of *modularity* coefficients (Newman 2006), i.e., an indicator that measures how well a network can be divided into clearly defined subsets of nodes. The higher the value of the metric the higher the community structure is supposed to be. Similarly, the lower the value, the lower the estimated community structure.

It is important to note that to adequately compare the modularity structure output by both YOSHI and the null model with the actual community structure of a certain community, we needed to transform the numeric indexes in a nominal scale comprised in the set $\{LOW, MEDIUM, HIGH\}$. To this aim, we followed the guidelines by Newman (2006): the community structure is *low* when $modularity < 0.30$, medium when $0.30 \leq modularity < 0.41$, and high when $modularity \geq 0.41$. Thus, if the modularity coefficient estimated by one (or both) of the experimented approaches is in accordance with the actual community structure modularity, then the approach correctly provides the indication.

To statistically verify the results, we applied the paired Mann-Whitney test (Conover 1998) comparing the distribution of the modularity coefficients computed by our approach with the ones computed by a randomly created one over the 25 considered systems. This is a non-parametric test used to evaluate the null hypothesis stating that it is equally likely that a randomly selected value from one sample will be less than or greater than a randomly selected value from a second sample. The results are intended as statistically significant at $\alpha = 0.05$. Note that in this case we relied on the numeric modularity values because we were interested in evaluating whether the indexed outputs by YOSHI were statistically different from those extracted by the random model.

While the analysis of how YOSHI performs when compared with a null model might provide insightful hints on the value of the information extracted by the proposed approach, it is also important to note that such information should effectively assist the members of a certain community. In other words, the fact that YOSHI provides better information than a random model does not directly imply that it is actually useful for community members. Thus, we needed an additional validation that was designed to gather opinions on the quality of the information provided by YOSHI from the members of the considered communities.

To this aim, we contacted the developers of the 25 open source communities considered: we limited our analyses to those developers having the highest number of commits (i.e., the most active ones), as they might have a more comprehensive knowledge of the development activities within the community and, therefore, a better overview of the underlying community structure that allow us to gather authoritative responses. On the contrary, we filtered out developers having a low number of commits and that are likely not so much involved in the community. Therefore, we contacted via e-mail the developers having a number of commits higher than the third quartile of all commits performed on each of the considered systems, i.e., those contributing the most to each repository, and we asked them to comment about the community structure that was in place in the specific time period analyzed in our paper: to ease the task, we provided them with a spreadsheet containing three tabs: (i) the first reporting detailed information on the commits performed on the repository; (ii) the second with the developers taking part in the development process in the considered time window, and (iii) the list of all communications between developers during the time period. In this way, the developers could better remember the project status in the period, and provide us with more careful observations of the community structure taking place in that period. To further

ease the task, we allowed developers to give us open answers, i.e., we did not provide them with fixed check-boxes reporting the possible community structures. We decided to go for this solution as developers might be not aware of the formal definition of the underlying community structure of their project.

When analyzing the developers' answers, we proceeded with a manual match of their opinions to the automatic community structure assigned by YOSHI to the community a certain developer corresponded to. To avoid any kind of confirmation bias, we recruited two independent external developers having more than 5 years of programming experience (from now on, we refer to them as the *inspectors*) and asked them to independently perform such a mapping. Specifically, we provided the two inspectors with the developers' answers and a list composed of the community types extractable using YOSHI. The task was to analyze each answer and tag it with one or more community types. For instance, if a developer replied by saying that *"in the considered time period, all communications passed from the mediation of one member, who had the role of disseminating it to other developers"*, the inspectors mapped this answer to the definition of formal community. This process required approximately 1.5 hours. At the end, we first computed the inter-rater agreement between the two inspectors using the Krippendorff's alpha Kr_α (Krippendorff 2004). Agreement measures to 0.90, considerably higher than the 0.80 standard reference score for Kr_α (Antoine et al. 2014) . In cases of disagreement, the inspectors opened a discussion in order to find a joint solution. In the second place, we verified how many times YOSHI was able to properly identify the community structure perceived by the developers of the considered project.

It is worth remarking that all the data and scripts used to evaluate YOSHI are publicly available in the form of a replication package in our on-line appendix (Tamburri et al. 2017).

4.2 Study Results

Before discussing the results for the two research questions formulated in the previous section, Table 3 reports for each system in the dataset the associated community types as inferred by YOSHI. All the projects exhibit characteristics attributable to more than one community type. Interestingly, about 60% of the communities have been typified as formal groups (FG) or formal networks (FN). This contrasts our expectation that open source projects are generally considered poorly formal, because of the contributions originating from volunteers. Yet, this evidence suggests to confirm theories reporting that a formal organisation is often needed to coordinate the activities among developers of a projects (Kraut and Streeter 1995; Elkins and Keller 2003). This finding is also confirmed by anecdotal evidence we found around several of the communities in our sample. For instance, the Netty project provides a formal guide[11] newcomers have to follow to be considered part of the community, thus establishing a formal structure and specific rules that community members must adhere to. More in line with the typical notion associated with open-source communities is the evident presence of Networks of Practice (around 40% of our sample) followed by working-groups, which, by their tightly-knit nature, are associated with collocated communities such as Modernizr (a tight community project with devoted maintainers: https://modernizr.com/).

[11] https://netty.io/community.html

Table 3 Community types inferred by YOSHI for the considered software projects

Project	Community Type(s)
Netty	IC, FN, FG
Android	IC, FN, FG
Arduino	IC, FN, FG
Bootstrap	IN, NOP
Boto	IC, IN
Bundler	NOP, FG
Cloud9	IC, FN, FG
Composer	IC, FN, FG
Cucumber	IN, IC, NOP
Ember-JS	FN, FG, WG
Gollum	IC, FN, FG
Hammer	IN, NOP
BoilerPlate	IN,NOP
Heroku	NOP, IN, FG
Modernizr	IN, NOP, WG
Mongoid	FN, FG, WG
Monodroid	IC, IN, FG
PDF-JS	IN, NOP
Scrapy	FN, FG, WG
RefineryCMS	FG, WG
Salt	FN, FG, WG
Scikit-Learn	NOP, IN, FG
SimpleCV	IC, NOP, IN, FG
Hawkthorne	IC, IN, FG
SocketRocket	NOP, IN, FG

As explained in Section 3, more community types can be associated to a single project

4.2.1 Does YOSHI correctly measure the community aspects characterising different software communities?

The first step in establishing if a metric provides a correct value is the satisfaction of its representation condition (Fenton 1991). Essentially, the representation condition states that a valid metric *must prove* that the observed empirical relation between the measured attributes is preserved by their associated numerical relations. To provide an objective validation for YOSHI metrics we analysed them individually, evaluating whether their representation condition hold by means of our own visual inspection. In particular, for each metric considered by YOSHI we performed the following steps:

1. We compute the values of the metric m for two communities C_i and C_j in our dataset;
2. If $m(C_i) > m(C_j)$, then we verified that the metric value computed on the community C_i was actually higher than the metric value computed on C_j.

In this paper, we focus on showing the evaluation of the representation condition of the *Geodispersion*, *Engagement* and *Formality* metrics. The proofs for remaining metrics refer to literature, that is, Longevity and Cohesion are well-established social-networks analysis metrics (Egghe and Rousseau 2003; Tikhonov 2016; Kozdoba and Mannor 2015).

Fig. 6 A geodistribution map for Twitter Bootstrap, the representation condition holds

Geodispersion Let us consider that the members of the community $C1$ are more dispersed (e.g., according to manual visual inspection) than the members of the community $C2$ if the average geo-dispersion across all the members of $C1$ is higher than the geo-dispersion across the members of $C2$. Considering the function average distance AD which maps average distances between community members, we can say that $C1$ is more dispersed than $C2$ if and only if:

$$AD(C1) > AD(C2) \tag{2}$$

where the average distance AD is computed as follows:

Let A be a member of the community, and let M_a be the average for geographical and cultural distances between A and the rest of the community members. The distance between two members is defined by the spherical distance between the geographical coordinates of the two users, determined using the same measuring unit, i.e., kilometers. Similarly, the cultural distance can be obtained following the Hofstede metrics (Hofstede et al. 2010). This operation is then applied for each member of the community, obtaining a set $M = \{M_a, M_b, ..., M_i, ..., M_n\}$ composed of the average geographical and cultural distances between each member i and the rest of the community. Thus, the average distance AD between community members is given by the average of the $M_i \in M$.

For sake of clarity, let's consider a practical case where the geo-dispersion of Twitter bootstrap[12] and dotcloud[13] are compared. As reported on-line on the websites of projects, we know that (a) Twitter bootstrap has contributors from all over the world; (b) dotcloud has contributors that are mostly grouped around the company offices in California (USA). Figures 6 and 7 present the actual geographical distribution of members collaborating on the Twitter bootstrap and dotcloud projects, respectively, as evaluated by YOSHI. The red circles represent the location of community members.[14]

From a visual inspection, it is clear that the representation condition of the metric holds. From a numerical one, the geo-dispersion among the Twitter bootstrap community members computed by YOSHI is 6,221, while the one of dotcloud community members is 380.

[12] https://github.com/twbs/bootstrap

[13] https://github.com/dotcloud/gitosis-on-dotcloud

[14] edges representing connections between users are not included in this figure, but a connection graph is available as part of the structural analysis

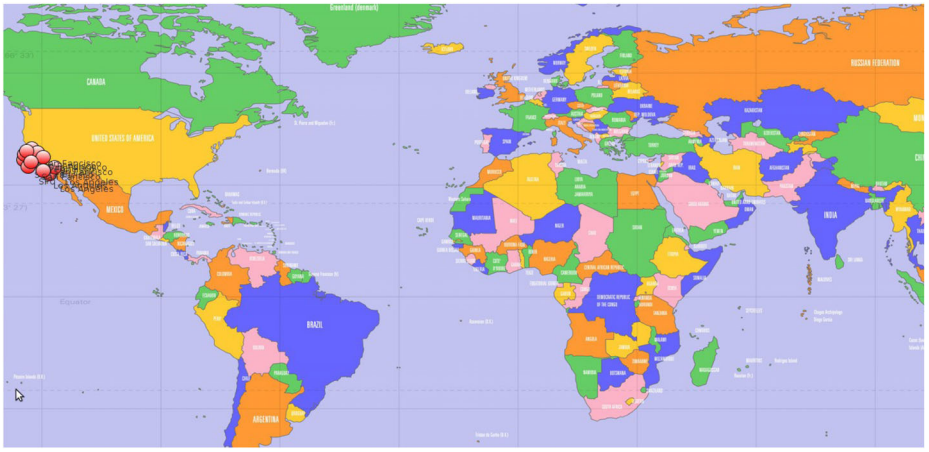


Fig. 7 A geodistribution map for Dotcloud

Based on the data computed by the application we visually confirmed that $AD(bootstrap) >> AD(dotcloud)$, which means that members of the Twitter bootstrap community are more dispersed than the members of the dotcloud project. Thus, we conclude that the representation condition for the *Geodispersion* metric is proven.

Engagement For the ENG property, consider, for the sake of simplicity, that community C1 displays a higher degree of member engagement than community C2 if the number of project subscriptions (i.e., item 5 from the enumeration in 3.2.4, but any other item may have been used just as well) made by the members of C1 is higher than the number of project subscriptions within community C2. Considering the function number of project subscriptions (PS), we can say that C1 has a higher degree of member engagement if PS(C1) $>$ PS(C2).

To evaluate the representation condition for engagement, it is sufficient to prove that the mechanism which YOSHI uses to determine PS is *consistent*, meaning that it determines a higher PS if the actual PS is effectively higher. YOSHI determines the list of projects-watch subscriptions using the specific GitHub API made for it and uses it to compute the number of project subscriptions. As an example, consider Arduino and Hammer-js projects.

Inspecting visually the project-watch number for both, the numbers for the Hammer-js project is 5438 while the number for Arduino is 2071. An analysis provided by YOSHI uses the same numbers and therefore would yield the same result. Note that the remaining points 1 to 7 in the enumeration at Sec. 3.2.4 are added in means, which mathematically does not alter in any way the representation condition for the metric in question.

Formality Formality is determined as the mean membership type in GitHub (+1 for contributors and +2 for collaborators) divided by the milestones per project-lifetime ratio. More formally, $\frac{\bar{a}}{M_L}$, where $\bar{a}$ is the mean membership type in GitHub and M_L is the milestones per project-lifetime ratio (i.e., the amounts of dates of the total project lifetime which are considered as milestone delivery dates or releases). This quantity increases at the increase of both metric quantities involved (which are both always positive and always different than 0, or the project would have no reason to exist). For the sake of simplicity we focus on showing that, by increasing $\bar{a}$ for a certain project C1 which is higher than another project

C2, the quantity reflected by the equation above is proportionally higher. For example, consider that interactions between the members of community Android are more formal than the interactions between the members of community Cloud9 (we know this because of the strict collaboration and participation guidelines behind Android); this means that their relative value $\bar{a}$ must reflect as follows: C1 > C2. According to YOSHI measurements, Android community members collaborate at 0.65 formality and the equivalent value for the Cloud9 project members is 0.43.

Applying a similar analysis to the rest of our dataset, we observed that more formal projects include core developers (i.e., collaborators) in projects that are hosted by the organization in which they are actively involved, meaning that they dedicate to that project no less than 20% of their working-hours. For example, core developers that contribute to the Pdf-js project (and that are in fact Mozilla employees) — this further reinforces the more formal nature of such communities.

Summary Stemming from the above demonstrations, we can answer our research question in a positive manner since the community characteristic measurement coded in YOSHI are correctly measured by the proposed approach.

> **Summary of RQ$_1$.** The representation condition is valid for all the metrics computed by YOSHI. Therefore, the proposed approach correctly measures the different organisational aspects characterising a software community.

4.2.2 Does YOSHI provide a meaningful view of the community structure of a software system?

Table 4 reports for each project (i) the actual value for its community structure, and (ii) the modularity coefficients achieved by YOSHI and by the randomly created null models over the 25 considered systems. As it is possible to see, in 100% the cases the nominal value associated to the coefficients computed by our approach are in accord to the actual community structure: this means that YOSHI provided correct indications over all the 25 subject systems. On the other hand, the baseline adequately estimated the modularity of the structure only in six cases, thus being often not able to provide meaningful results. As a consequence, we can claim that our approach not only can potentially accurately assists in understanding the structure of a community, but it is also able to perform better than the baseline. This result is also supported by the statistical tests. Indeed, when comparing the two distributions using the Mann-Whitney paired test, we found that the differences are statistically significant ($\rho < 0.001$); moreover, the magnitude of such differences is large (δ=0.74), according to the results achieved when running the Cliff's δ effect size test (Romano et al. 2006).

Interesting is the case of the Scrapy community, which has a high modularity as indicated by the OpenHub data. YOSHI is able to provide a correct indication, since the coefficient computed is equal to 0.45, while the baseline estimates the modularity of the community structure as low. Looking more in depth into the characteristics of this project, we found that this is the one in our dataset having the higher level of interaction among the members, and this is clearly visible looking at the number of pull requests of the project (1,472, of which 154 are still open), and the number of average comments per pull request (5.4). In this case, the detection pattern used by YOSHI is quite effective in the identification of the

Table 4 Modularity Coefficients achieved by the Experimented Approaches - indexes refer to April 2017

Project	Actual Community Structure	Yoshi	Null Model
Android	LOW	0.27 (low)	0.44 (high)
Arduino	LOW	0.25 (low)	0.33 (medium)
BoilerPlate	MEDIUM	0.31 (medium)	0.13 (low)
Bootstrap	LOW	0.23 (low)	0.28 (low)
Boto	LOW	0.28 (low)	0.29 (low)
Bundler	MEDIUM	0.34 (medium)	0.31 (medium)
Cloud9	MEDIUM	0.35 (medium)	0.23 (low)
Composer	HIGH	0.42 (high)	0.25 (low)
Cucumber	MEDIUM	0.37 (medium)	0.33 (medium)
Ember-JS	MEDIUM	0.38 (medium)	0.15 (low)
Gollum	HIGH	0.44 (high)	0.21 (low)
Hammer	MEDIUM	0.37 (medium)	0.26 (low)
Heroku	HIGH	0.49 (high)	0.37 (medium)
Modernizr	HIGH	0.42 (high)	0.19 (low)
Mongoid	MEDIUM	0.33 (medium)	0.24 (low)
Monodroid	HIGH	0.45 (high)	0.22 (low)
Netty	LOW	0.24 (low)	0.46 (high)
PDF-JS	MEDIUM	0.31 (medium)	0.33 (medium)
RefineryCMS	MEDIUM	0.39 (medium)	0.19 (low)
Salt	HIGH	0.43 (high)	0.21 (low)
Scikit-Learn	HIGH	0.44 (high)	0.18 (low)
Scrapy	HIGH	0.45 (high)	0.15 (low)
SimpleCV	MEDIUM	0.36 (medium)	0.31 (medium)
SocketRocket	HIGH	0.48 (high)	0.21 (low)
Hawkthorne	HIGH	0.42 (high)	0.25 (low)

structure, since it mainly relies on the information captured by pull requests. Conversely, the randomly created baseline wrongly approximated the relationship between the members of the community, thus providing an unreliable result.

The lower modularity coefficient (0.23) was assigned by Yoshi to the Bootstrap community. Manually investigating this case, we found evidence of the low structure behind this project. For instance, of the 389 contributors involved in it only a small subset of them heavily participate in the activities of the community. Indeed, there are only five *core* committers. Moreover, in most of the cases (i) pull requests are reviewed and commented by these five developers and (ii) discussions about bugs and improvements on the issue tracker only involve them. As a result, our approach correctly marked this community as having a low structure, being able to provide an accurate indication.

The results for the other projects are consistent with the mentioned cases. As a consequence, we can claim that the measures applied by our approach can be potentially effective when employed to understand the underlying structure of the community. As explained before, to further verify this claim we directly involved the most active members of the considered communities, asking them to verify the information provided by our approach. Overall, we obtained 36 answers (1.44 answers per project) out of the 95 invitations sent:

therefore, the response rate was 38%, that is almost twice than what has been achieved by previous papers (e.g., Palomba et al. (2015), Palomba et al. (2017), and Vasilescu et al. (2015b)). The response rate was likely pretty satisfactory because of the methodology used to contact developers (direct e-mails): indeed, as done in previous work (Silva et al. 2016), this strategy is generally a good one to obtain quick and effective answers from developers.

Looking at the actual results, we found that 92% of the times there was a correspondence between the YOSHI output and what reported by developers, meaning that 33 community members fully agreed with the output community structure given by our approach. We believe that this result further reinforces the quantitative findings: indeed, not only is YOSHI able to mine developers' communication and coordination information to discover the corresponding community structure, but it also provides data that reflects the developers' perception of the community. An interesting example regards the RefineryCMS project, where a developer reported:

> *"We were dislocated across several countries and for a long while we had communication issues because of that. As a solution, we then decided to start being more selective when accepting members in the community and all the communications passed through our mailing list. This had some benefits, and indeed from that moment we were able to do things in a more cohesive and natural manner."*

This answer clearly refers to the definitions of Formal Networks and Workgroups, as it indicates the presence of both member selection strategies and formal communications—which are typical of FN—but also some degree of cohesiveness between team members, that highlights the presence of a workgroup. In the cases where the developers' answers were not in line with the output of YOSHI, we discovered that it was due to false positive cases in which a Formal Network was interpreted as a Formal Group. Nevertheless, also in those cases we can argue that our approach successfully identified the formality of the communities, thus potentially providing good hints.

Summary of RQ_2. In 100% of the cases YOSHI correctly estimated the modularity of the community structures analysed. Moreover, YOSHI's results are much more accurate than the experimented baseline and, more importantly, 92% of the developers confirmed the correctness of the output presented. Thus, we conclude that our approach can factually support the activities of project managers in the understanding of the community structure behind a software project.

5 Discussion and Further Insights

The results of our research questions highlight some relevant findings, as they showed how the proposed automated solution is able to properly identify and characterise community design patterns in open source. Besides that, we also found that the members of the considered communities actually agree with the output of YOSHI: this is, likely, one of the most important outcomes of our analyses. Indeed, it seems that developers can exploit our approach to monitor and gather information about the status of the community, thus possibly taking informed decisions on how to improve communications and/or coordinations as

well as reasoning on how a certain community structure influences external qualities of the developed project.

To reinforce our claims, in this section we discuss a key use case scenario, namely how YOSHI can help practitioners when assessing the health status of a community. In particular, we computed four repository-related metrics for the 25 communities considered in the study. They include (i) the mean number of commits per month, (ii) the mean number of contributors per month, (iii) the number of stars, and (iv) the number and forks the repository has on Github. It is important to note that these metrics were defined by Crowston and Howison (2006) and Jansen (2014) as meaningful indicators of the health of open source software ecosystems. For this reason, we were interested in understanding if the community patterns proposed by YOSHI can somehow indicate a more or less healthy condition of the repository, thus allowing community shepherds to monitor how healthy the underlying community is and plan preventive actions, where needed. Of course, it is important to note that all the observations made in this section may be a reflection of other factors, i.e., the health status of a community can be not only (or not at all) influenced by the presence of a certain community pattern. Nevertheless, our goal is to simply outline some potential scenarios where the use of YOSHI can assist practitioners when taking decisions on how to evolve the community.

We graphically report these further analyses by means of violin plots (Hintze and Nelson 1998) depicting the health metrics distributions for each community pattern inferred by YOSHI. Besides showing how the data is distributed, this graphical representation also shows the probability density of the data at different values allowing a more detailed overview of the differences among the different community types taken into account. In addition to the analysis of the metrics computed on the exploited dataset, we also provide practical examples aimed at explaining in which situations the usage of YOSHI can provide benefits within a software community. As our tool identified more than one community type for each considered project, an analysis of the community types in isolation would not have provided insights due to the high overlap between the data points. For this reason, we decided to analyse the behavior of the four most frequent patterns (i.e., co-occurring community types) coming from Table 3, i.e., $\{IC, FN, FG\}$ (6 projects), $\{FN, FG, WG\}$ (4 projects), $\{IN, NOP\}$ (4 projects), and $\{NOP, IN, FG\}$ (3 projects).

Figure 8 reports the results of these additional analyses. In the first place, we found relevant differences in terms of both number of commits and contributors per month among the frequently co-occurring community types. In particular, the results revealed that communities having a strong level of formality (typified by YOSHI as both FN and FG) have lower turnover (i.e., a constant number of contributors). This result somehow confirms the mediatory governance role of formal participatory guidelines typical of that community type (Antunes et al. 1995). An interesting case regards the Arduino project: it was typified by YOSHI as a mixture of formal network, formal group and informal communities. Such characteristics allow the project to be more continuous in terms of both contributors and commits, as reported in Fig. 8. However, looking deeply into such community through the last-month statistics—reported by the Openhub repository[15]—we discovered that the current situation reports that the number of commits is going to decrease while the number of team members is increasing: this might mean that the introduction of new members within the community might change the nature of socio-technical relationships among the team members, possibly leading to a new community type in the near future. Nevertheless,

[15] https://www.openhub.net/p/arduino

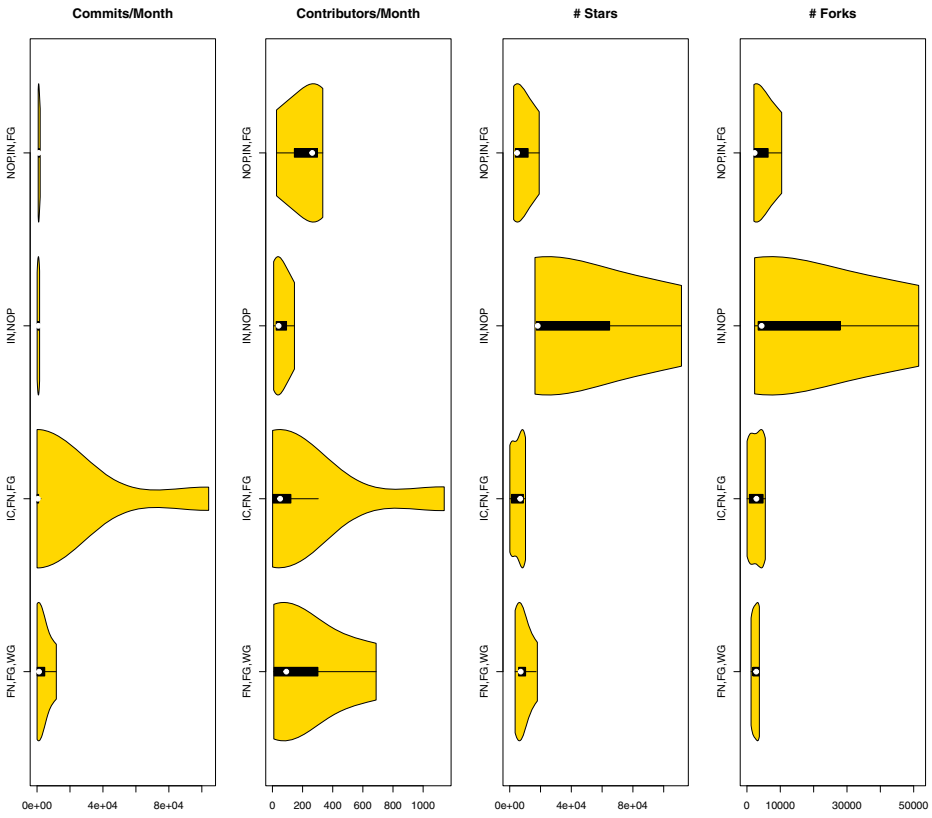


Fig. 8 Violin Plots reporting the repository health metrics for each community type inferred by YOSHI

such a change within a community is something that a community shepherd might desire or not, even because different community design patterns might be associated with some negative manifestations or lead to different characteristics: for instance, as reported in Fig. 8 a change towards an informal community might decrease the overall contributors continuity and increasing the number of forks of the project. To account and manage such compromises, a tool like the one proposed in this paper can be adopted by community shepherds during their decision making process.

At the same time, the usage of YOSHI can be useful for community shepherds to take decisions with respect to their own or their community's *organisational reference*. For example, consider a situation in which a certain community is undergoing organisational distress (e.g., after adopting new tools for their own community work or changing process model); in this scenario, community shepherds need to make decisions aimed at reorganizing the overall structure of the community. In the same scenario, YOSHI acts as a community monitoring system and can be used in such a circumstance to understand what are the characteristics of an external community that the community shepherd wants to replicate in its own context. For instance, the Salt project[16]—typified by our tool as a mixture of formal

[16] http://www.saltproject.org

group, formal network, and working group—has a high contributor continuity and is well-known to be successfully used by several other open- and closed-source organizations. Thus, it might represent the example-to-follow for another community whose shepherd might try to transplant community-related characteristics in her own context. In this case, YOSHI might provide this community shepherd with useful information aimed at replicating some external practices in her community.

When considering the remaining two repository health metrics, i.e., number of stars and forks achieved by the repository, the violin plots in Fig. 8 revealed important, almost specular differences due to the different nature of the community types. On the one hand, the relatively low number of stargazers and forkers of formal groups may be connected to their rather closed and static organisational structure, an established negative pattern typical of formal organisations (Zhu et al. 2012). Known mitigations for that pattern reflect simplified organisations, with increased outreach activities (e.g., increased participation to mentoring forums such as StackOverflow or YCombinator). Interestingly, on the other hand we also found that the number of stars and forks scores are higher for informal and open, highly diverse and geographically distributed types such as Informal Networks and Networks of Practice. This empirically confirms the conjecture by Crowston and Howison (2006) that *informality is a key health target for open-source*, since it is in fact informality which drives the engagement and popularity of open-source communities to success and organisational sustainability—this very same observation, however, is in contrast to the evident community types discovered by YOSHI (see Table 3). This suggests that developers, forges, and community platforms such as GitHub should be further supported in achieving and maintaining healthy values of informality across their communities—YOSHI can serve as a basis toolkit to diagnose this condition and aid in its resolution.

In conclusion, we argue that the usage of YOSHI can aid in understanding and shepherding the key characteristics of open source communities such as contributor-continuity and popularity.

6 Tool Limitations and Threats to Validity

In this section we discuss the main limitations of the tool proposed, as well as the factors that might have influenced the findings achieved in our case study.

6.1 Tool Limitations

Although a relevant contribution in its own accord, YOSHI shows many limitations that inhibit its effectiveness and usability.

For YOSHI's ability to provide computer-assisted governance support to open-source communities, our first concern is generalisability. In our analysis we presented an evaluation of 25 projects consisting of popular open-source software projects hosted on Github. Based on this limited scope, our results might not easily generalise to other domains (e.g., closed-source). Second, our analysis relies on the validity of metrics detection and thresholds values identification provided in previous work (di Nitto et al. 2013), hence suffering from potential *construct validity* (Wohlin et al. 2000). Third, finally, in its current version YOSHI is limited to establishing the presence of 8 our of 13 possible community types from organizations and social networks research literature (Tamburri et al. 2013a). Although these types were found (di Nitto et al. 2013) to be the most compatible with open-source organisational forms, this poses a limitation since YOSHI is not able to ascertain the presence of

observed types while discriminating the presence of the remaining five types. This means that YOSHI is only partially effective in types that intermix characteristics from unsupported communities - we recognise this implementation problem as the major technical limitation of the tool and are working to address the limitation for its next version.

For YOSHI's ability to support researchers in finding correlations analysing open-source communities, YOSHI currently relies solely on Github's publicly available data sources, including its own issue-tracking system. Some projects (especially closed-source ones) use a different issue tracking system such as Jira or Bugzilla and we should add support for collecting data from more sources of data. Finally, it might become necessary to extend the framework such that it can be accessed by a web browser. This feature would allow: (1) users to define their own sets of projects that they want to analyse further; (2) add support for more complex visualisation features.

Moreover, as we mentioned, YOSHI is currently limited to supporting the detection of 9 out of 13 possible community types emerging in open-source. Although in previous work (di Nitto et al. 2013) we showed that the community types implemented by YOSHI are the most recurrent in open-source, but this does not mean that other features do not play a role. For example, YOSHI does not allow measurement of artefacts and/or software architecture visibility (intended as the ability for people to quickly retrieve information about community artefacts) in open-source. Conversely, the Apache Software Foundation fosters the creation of strong and fully engaged communities wherefore the organisational visibility and tracking mechanisms need to be made clear and are controlled monthly by a specific ASF authority and tracked via compiled and templated reports (Severance 2012). Similarly, there is a wealth of organisational and socio-technical characteristics from the state of the art in organisations research and social-networks analysis (e.g., community de-coupling, reciprocity levels, etc.) that YOSHI cannot currently take into account and an equal number of interesting principles (the organisational self-similarity principle and the consequent institutional isomorphism (Lai et al. 2006)) to be further explored. We are currently in the process of operationalising these features for further empirical assessment in the scope of software engineering, to determine their mediating role, if any.

In addition, YOSHI is also limited to reusing empirical thresholds evaluated in previous ethnomethodological research. However, the thresholds in question may be biased too much against the same community in which they were observed. Also, several studies indicate that the very use of clear-cut thresholds may not be as straightforward as we assume, at least for some of the dimensions we consider, e.g., geographical distance (Prikladnicki 2012). While we are indeed considering replicating the 15-month study that led to the definition of the thresholds, we found it impossible with our current means. Nevertheless, we are aware of the limitations and threats to validity of that previous study and of reusing thresholds. Consequently, we chose to: (a) design the tool to make the thresholds application and evaluation very highly-modularised into the YOSHI architecture such that substituting thresholds and detection patterns is a non-invasive improvement to YOSHI 's capabilities; (b) release a fully-fledged replication package to call for, and encourage replication of our work.

Finally, detection and identification of community types is itself still a matter of research in organisations and social networks literature. On one hand, we researched, evaluated, and replicated the evaluation of our detection Algorithm 1 multiple industrial organisations in previous research (Tamburri et al. 2015, 2016). Our experimentation in the scope of this article revealed also that there are types which are most common in open-source. For example, we noticed that YOSHI did not reveal any presence of Project-Teams nor Communities of Practice across our sample—this could either indicate an uncontrolled variable in our random sampling strategy or a specific flavour of communities which are more common in

open-source. In either case, more research is in order to further investigate this circumstance. Furthermore, there are other community structure approaches that may be used for community detection, e.g., those proposed by Lancichinetti et al. (2008) or Medus et al. (2005). Other approaches are focused more on using graph- and motif-analysis over organisational networks and we cannot be certain that our own community typing approach is "better" than others. Further research is needed into the community analyses in YOSHI , possibly through mixed-methods research (Di Penta and Tamburri 2017) triangulated with industrial studies.

6.2 Industry vs. Open-Source, insights from YOSHI

Reflecting on YOSHI design and features, as well as the decision-tree implemented in its core, we also operated a comparison of previous research results with respect to YOSHI 's findings in the scope of this article. More in particular, in our previous work (Tamburri et al. 2013b) we conjectured that software teams exhibit latent social community types that need uncovering to understand vital socio-technical community characteristics. In the same work, we confirmed, using the decision-tree at the root of YOSHI detection algorithm, the complex organisational and social structure nature by means of qualitative analysis in a single, large industrial case-study of distributed software development. Our industrial data analysis confirmed the presence of at least 3 types (FN,CoP,IN) blended together, two of which were revealed to be in latent conflict (FN and IN) by our own analysis. Our conclusions where that: (a) latent community types did play a role for better software and different types reflected different activity latency in our case-study object, i.e., different development, issue-solving, or task-allocation times; (b) organisational changes such as agile adoption changes the type of communities.

In the scope of the present article, YOSHI allowed us to clearly observe quantitative confirmation of finding (a) while also providing a basis for further confirming finding (b).

With respect to finding (a), YOSHI reveals several insights, such as that formal community design patterns seem to show higher expectancy of life while retaining the advantage of "trust"—these features could be a valuable asset especially for "liability of foreignness" organisational barriers, i.e., the inability of a company to enter a localised and closed market (Zimmermann 2008). Companies experiencing liability may enter or participate into trusted open-source communities rather than directly entering those closed markets. This issue is frequent and heavily impactful in software offshoring exercises such as the one we studied in our previous industrial work (Tamburri et al. 2013b).

With respect to finding (b), YOSHI has not been applied in instances wherefore open-source communities forcibly changed their structures, e.g., as part of *community forking*. This not withstanding, YOSHI does in fact provide the means to study phenomena such as community forks to further understand and characterise their complex organisational and socio-technical nature—this line of inquiry is currently under planning and is part of our future work.

6.3 Threats to Validity

The study conducted when evaluating YOSHI might have been influenced by a number of factors.

Threats to Construct Validity As for factors threatening the relation between theory and observation, in our context they are mainly concerned with the measurements we performed. Above all, the metrics on which the proposed approach relies, i.e., community structure,

geodispersion, longevity, engagement, formality, and cohesion, were validated with respect to their representation conditions ($\mathbf{RQ}_1$). Moreover, we further evaluated the community structure in terms of modularity coefficient ($\mathbf{RQ}_2$), comparing the results of YOSHI with the ones of a baseline approach and supporting the findings with appropriate statistical tests. Conversely, one of our operationalisations does indeed suffer heavily from this threat, namely the way we measure levels of informality across communities. On one hand, in organisations research informality is not the opposite of formality (Tamburri et al. 2013a). On the other hand, for the sake of its operationalisation, YOSHI actually assumes this to be the case and measures informality as {1 - formality}; although this operationalisation does offer values which are indicative of informality levels, it cannot currently match the actual definition from literature, which indicates more the degree of openness, reciprocity, and unmediated interaction across the community (Fuks et al. 2005). Furthermore, because YOSHI focuses on the collaboration structure existing across developers and on typing its informality, the tool currently ignores comments, their structure, contents, and possible contribution towards informality—this is currently a known limitation of the tool and must be further understood and addressed in future work. Further research must be invested to refine these measurements to achieve further precision. For example, further understanding the tone, contents, and structure of comments exerted by software developers and operators during their work might reveal ways in which formality can be identified and mediated more precisely.

Threats to Conclusion Validity A first threat in this category is related to the ground truth exploited in the context of $\mathbf{RQ}_2$: in this case, it is important to note that the OpenHub repository contains validated datasets. For this reason, we are confident about the validity of the data exploited and used to validate the modularity coefficients achieved by YOSHI and by the baseline approach. Nevertheless, we cannot exclude imprecisions; an in-depth analysis on the quality of the data coming from OpenHub would be desirable and part of our future research agenda.

Moreover, to further verify the conclusions we contacted developers of the analyzed communities and asked them to comment about the community structure that was in place in the specific time period analyzed in our paper. To gather authoritative responses, for each project we only contacted developers having a number of commits higher than the third quartile of the distribution of all the commits in the repository. To reach a higher response rate, we kept the survey short; therefore, we did not ask any question to developers other than those required to verify the community structure of the considered projects. For this reason, we do not have data on the participants' professional experience, role, etc. However, this does not represent a threat to our results: indeed, our goal was to verify that the community structure given by YOSHI was in line with the structure assigned by developers that actively worked for the considered projects. As such, the only requirement needed to be part of our study was the actual active participation to the project, independently from other factors like, for instance, the overall programming experience of the participants. By nature, our selection process guarantees the involvement of people that can be considered expert of the projects analyzed and that can actually provide authoritative opinions on the community structure of their projects. As part of the discussion, we computed well-known repository health metrics previously studied in literature (Crowston and Howison 2006; Jansen 2014), thus focusing only on measures actually able to provide an established overview of the status of a given repository.

Threats to External Validity The main issue concerned with the generalization of the results is the number of software communities analyzed in the study. While a set of 25

systems is not a statistically significant sample of the most active projects present in Github, it is important to remark that the main goal of our paper was to evaluate YOSHI and not that of studying properties of open-source systems on an ultra-large scale. Furthermore, the size of the dataset allowed us to study the involved projects closely, thus providing finer observations on the performance of our approach. For this reason, we believe that the dataset can be considered large enough for answering our research questions. Nevertheless, to make our findings as generalizable as possible we took into account a variety of communities having different characteristics, scope, size, and coming from different application domains. The set of key-attributes that are frequently associated with open-source communities and the attributes measuring the project quality can be further extended and applied on a larger number of projects for a better understanding of the relationship among software communities, their practices and the characteristics related to community types. We plan to extend our investigation on a larger set of communities.

7 Related Work

There are several works related to YOSHI, mainly residing in the general areas of open-source community governance studies and computer-aided open-source management. In this section, we overview relevant previous papers in these fields.

7.1 Governance and Community Aspects in Open-Source

First and foremost, works that strive to understanding and steering the specific coordination and governance models adopted in an open-source project are highly related to the proposed one. This is critical both for developers to assess whether to contribute or not to the project and for final users of the resulting application, since trust in the community can vary strongly according to the governance mechanism underlying the development. Garzarelli and Galoppini (2003) identified three main categories of projects:[17]

- **Corporate projects**, entirely developed within a single company and then released as open-source.
- **Voluntary projects**, which are supported by volunteers only, offering their efforts and resources without being remunerated.
- **Hybrid projects**, jointly developed by volunteers and employers working for the company which runs the project itself.

An example of voluntary project is represented by Debian,[18] a completely free operating system launched in 1993 by Ian Murdock. One of the most relevant characteristics of the organization model adopted by the Debian community consists in the adoption of the Debian Social Contract, a document listing the moral rules and the values at the basis of the project itself. The coordination mechanisms in place within the project are defined within another formal document, the Debian Constitution.[19] The governance structure is hierarchical

[17] A similar categorisation has been proposed by Robles et al. (2009).
[18] http://www.debian.org
[19] http://www.debian.org/devel/constitution.en.html

and includes different roles, such as the Project Leader, annually elected by developers, the Technical Committee, mainly responsible for technical issues or problems related to overlapping responsibilities, and developers, managing the packages they are in charge of Garzarelli and Galoppini (2003).

Code of conducts in open source have been also the object of the study by Tourani et al. (2017), who investigated role, scope and influence of codes of conduct in practice. Key findings of their work report that thousands of projects rely on code of conducts to manage the behavior of project's members and avoid unfriendly environments, and they use to stipulate contracts targeting all collaboration spaces of the community, trying to fix strict rules for collaborators.

Coelho and Valente (2017) investigated the causes behind the failure of modern software systems. The described a set of nine reasons, and interestingly five of them were related to team- or environment-related issues. For instance, a notable amount of them failed because of conflicts between the contributors.

With respect to these papers, it is important to remark that YOSHI does not directly identify governance models, specific coordination requirements, or code of conducts, but it captures a snapshot of open-source communities' current organisational and social layout by analysing their key characteristics. Appropriate governance models should be selected after (or evaluated with) the results achieved by YOSHI, and can be adopted to monitor how the overall community is evolving and whether the specific pattern is follows might create conflicts or coordination problems that potentially lead to more serious issues.

Even considering similar communities, it is still possible to identify differences in the governance practices they follow. Prattico (2012) has considered six communities supported by active open source foundations: Apache, Eclipse, GNOME, Plone, Python and SPI. Using computer-aided text analysis of each foundation's bylaws, Prattico noted that, although each foundation adopted different terms, it was possible to identify three common main power centers: the *members of the community*, the *board of directors* and the *chairman of the community*. For example, the chairman of the community can be named by the board of directors, as in the Eclipse foundation, or elected by the members, as in the Debian project. The board of directors is composed of people elected by the members. The board takes decisions about the piece of software it is in charge of. Also, different communities showed a different distribution of power. For example, in the Eclipse Software Foundation power is mostly managed by the chairman, while in the Apache Software Foundation the board of directors and the members exert the most power, with an inclination towards the board of directors. Given the above works, it becomes critical for software engineering theory and practice to learn as much as possible from open-source communities and their ways of coping with GSD (Global Software Development).

Also in this case, YOSHI does not directly uncover best-practices concerning community aspects in open-source but it does allow further reasoning such that best practices can be developed with additional study.

Similar works have been done by Onoue et al. (2014, 2016), who studied the population structures of open source systems: in particular, they first defined a method for predicting the survival of open source community members within a software project, and then investigated the pyramidal roles present in such projects. With respect to these works, our paper can be seen as complementary, as it proposes an approach to compute community-related metrics and understand what is the underlying structure of community, so that practitioners can take informed decisions on how to evolve it.

7.2 Computer-Supported Cooperative Work in Open-Source

Usually, communication within a distributed team, either open-source or closed-source, is supported by forges typically including mailing lists and Web-based tools like forums and wikis. Contributions are shared by means of Concurrent Versions Systems (CVSs) or Distributed Version Control Systems (DVCSs), like Subversion or Git, which provide versioning features, allowing to easily check or revert someone else's contributions. Moreover, tracking systems are used by the community itself and by external users to report bugs or other problems and to ask for the development of new features.

Several studies have been dedicated to identifying social and technical barriers for newcomers' participation in Open Source (Steinmacher et al. 2015; Mendez et al. 2018; Ford et al. 2016; Balali et al. 2018; Vasilescu et al. 2015a). Going beyond identification of the barriers, some tools are explicitly aimed at tackling one or more technical barriers often related to communication, e.g., by allowing practitioners to talk with remote colleagues in an easy and informal way. For example, the Jazz project, sponsored by IBM, added instant messaging features to the Eclipse Platform, together with other tools that show which files are currently being edited by whom (Cheng et al. 2003). Also, a number of tools focus on supporting activities such as distributed problem analysis, requirements elicitation and activity planning. For example, the tool MasePlanner is an Eclipse plug-in with features for simple agile planning in distributed contexts. Users can create story cards shown in a common virtual workspace. Cards can be organised and modified by project members to plan their activities.

Other tools aim at improving awareness by extracting information from forges. For example, tools proposed by the Libresoft group mine data extracted from code repositories, malinglists, discussions and tracking systems (Robles et al. 2011). The SeCold portal adopts mining techniques to build a shared knowledge base about several open-source projects, including explicit facts like code content and statements, as well as implicit data, such as the adopted license and the number of clones produced from a project (Keivanloo et al. 2012). In similar studies, mining techniques are used to extract patterns to represent and improve the decision process adopted in software development.

Finally, the ALERT[20] project uses ontologies to increase awareness by gathering and linking related information obtained from the different tools adopted by the community, including structured and unstructured sources (Stojanovic et al. 2011). ALERT is intended to use this information to build personalised, real-time and context-aware notifications, to detect duplicate bug reports and assigning bugs reports to be solved.

To the best of our knowledge, there is no computer-assisted tool support for identifying and measuring the community, social and organisational aspects behind open-source communities. Yet, studying these aspects is essential for capturing organisational and social best-practices from the widely discussed and studied open-source phenomenon.

8 Conclusions and Future Work

This paper introduces YOSHI, a tool for the automated organisational structure pattern detection across open-source communities. YOSHI gathers information about open-source

[20] http://www.alert-project.eu

communities and projects and executes necessary preprocessing operations to obtain information about community characteristics. Finally, YOSHI uses community characteristics found to infer a pattern of open-source community structures. YOSHI stores and reports results and offers a modular design, ready for further extension and analysis. The rest of this section recaps conclusions and future work beyond the scope of this article.

8.1 Lessons Learned

While testing YOSHI on 25 projects from GitHub, we observed that:

1. **YOSHI allows the reliable prediction of community structure patterns using software engineering data.** The experiment conducted to verify the accuracy of the tool revealed that it offers a valid set of metrics which satisfy the basic representation condition as defined in literature (Fenton 1991). Furthermore, YOSHI provides an accurate information with respect to the community structure according to our null-model analysis.
2. **YOSHI can be used to monitor and manage social debt within open-source communities.** The design pattern output by the proposed tool can be used by practitioners to control the quality status of social and organisational relationships among developers of a community: indeed, each community type has its own peculiarities and might reflect the presence of specific social debt items (Tamburri et al. 2015; Magnoni et al. 2017) within the community.
3. **YOSHI can be used to steer repository popularity by comparison.** Our analyses revealed that informal groups and networks tend to have a higher number of stargazers and forkers, thus having a higher likelihood to be reused by other projects. As a consequence, the pattern detection facility provided by the tool can be exploited to monitor the repository status and plan specific preventive actions aimed at increasing its reuse-proneness.

From the above features we can conclude that YOSHI : (a) allows reuse of type-specific community steering and adaptation best practices from literature; (b) yields deeper understanding of open-source communities' organisational and socio-technical nature; (c) offers a valuable basis for diagnosing open-source community structures and design patterns thereof.

8.2 Future Work and Outlook: Forming the Software Community Shepherd

Our prototyping and experimentation with YOSHI showed us that while sites like Bitergia and OpenHub may provide vital insights into open-source communities they do little to elicit and measure key community design pattern performances. Conversely, the community shepherd is a persona whose goal aims at: (a) measurably understanding the organisational structure requirements behind software, to encourage sustainability; (b) applying models, techniques, and approaches from organisations research to measurably improve the open-source community structure, to encourage continuous improvement; (c) making the organisational structure characteristics more transparent, open, and measurable to encourage further research.

Further experimentation is needed along the above research path. In that respect, YOSHI offers ample opportunity for improvement and further work. For example, YOSHI can be extended to include techniques such as more elaborate data-mining or sentiment

analysis (Novielli et al. 2014). This refines YOSHI even further in supporting governance and management across open-source communities.

In addition, YOSHI may be combined with tools that detect the technical qualities of open-source communities with the aim of eliciting and evaluating a full-fledged community quality model complementing well-known software product quality models (Ferenc et al. 2014)—we started researching along this path (Magnoni et al. 2017) but we concluded that we barely scratched the surface of a vast array of possibilities that require further research.

Moreover, further experimentation is needed to establish which patterns elicited by YOSHI actually correspond to which community smells and in which technical conditions. This analysis may reveal open-source organisational patterns which are best fitting with specific domains, or products.

Finally, YOSHI should be extended to cope with closed-source organizations as well. This extension entails an additional set of metrics to be devised to integrate remaining community patterns from Tamburri et al. (2013a, b). Also, this extension calls for an additional round of validation in closed-source software projects. In the future we plan to address the tool's technical limitations, while providing more ample empirical evaluation, possibly over multiple software forges other than GitHub.

Acknowledgments The authors would like to thank the students Alexandra Leta and Martin Anev that helped with the realisation of YOSHI. Also, the authors would like to thank the many practitioners that aided us in achieving YOSHI's evaluation in practice. Palomba gratefully acknowledge the support of the Swiss National Science Foundation through the SNF Project No. PP00P2_170529. Last but not least, the authors would like to thank the anonymous reviewers for their invaluable comments and contributions to this explorative and frontier research.

References

Afsar B, Badir YF (2015) The impacts of person-organisation fit and perceived organisational support on innovative work behaviour: the mediating effects of knowledge sharing behaviour. IJISCM 7(4):263–285. http://dblp.uni-trier.de/db/journals/ijiscm/ijiscm7.html#AfsarB15

Ala-Mutka K (2009) Review of lifelong learning in online communities. http://is.jrc.ec.europaeu/pages/EAP/documents/IPTSreportDraft230309_000.pdf

Antoine JY, Villaneau J, Lefeuvre A (2014) Weighted Krippendorff's alpha is a more reliable metrics for multi-coders ordinal annotations: experimental studies on emotion, opinion and coreference annotation. In: Bouma G, Parmentier Y (eds) Proceedings of the 14th conference of the European chapter of the association for computational linguistics. The Association for Computer Linguistics, EACL, pp 550–559. http://dblp.uni-trier.de/db/conf/eacl/eacl2014.html#AntoineVL14

Antunes P, Guimarães N, Segovia J, Cardeñosa J (1995) Beyond formal processes: augmenting workflow with group interaction techniques. In: Proceedings of the conference on organizational computing systems, COOCS. ACM, Milpitas, pp 1–9. https://doi.org/10.1145/224019.224020

Arnaoudova V, Haiduc S, Marcus A, Antoniol G (2015) The use of text retrieval and natural language processing in software engineering. In: Proceedings of the 37th international conference on software

engineering - vol 2, ICSE '15. IEEE Press, Piscataway, pp 949–950. http://dl.acm.org/citation.cfm?id=2819009.2819224
Avgeriou P, Kruchten P, Nord RL, Ozkaya I, Seaman CB (2016) Reducing friction in software development. IEEE Soft 33(1):66–73. http://dblp.uni-trier.de/db/journals/software/software33.html#AvgeriouKNOS16
Balali S, Steinmacher I, Annamalai U, Sarma A, Gerosa MA (2018) Newcomers' barriers... is that all? an analysis of mentors' and newcomers' barriers in OSS projects. Comp Support Coop W 27(3-6):679–714
Basili VR, Caldiera G, Rombach DH (1994) The goal question metric approach, vol I. Wiley, New York, pp 213–223
Bass L, Clements P, Kazman R (1998) Software architecture in practice. Addison Wesley, Boston
Bird C, Nagappan N, Gall H, Murphy B, Devanbu P (2009) Putting it all together: Using socio-technical networks to predict failures. In: Proceedings of the 2009 20th international symposium on software reliability engineering. IEEE Computer Society, Washington, pp 109–119, https://doi.org/10.1109/ISSRE.2009.17. ISSRE '09
Borges H, Hora A, Valente MT (2016) Understanding the factors that impact the popularity of github repositories. In: IEEE international conference on software maintenance and evolution. IEEE, pp 334–344
Capiluppi A, Lago P, Morisio M, e Informatica D (2003) Characteristics of open source projects. In: 2003 Proceedings seventh European conference on software maintenance and reengineering, vol 1, no 17, pp 317–327
Capra E, Francalanci C, Merlo F (2008) An empirical study on the relationship between software design quality, development effort and governance in open source projects. IEEE Trans Softw Eng 2(13):112–142. https://doi.org/10.1109/TSE.2008.68, in press
Chatha KA (2003) Multi-process modelling approach to complex organisation design. PhD thesis, Loughborough University
Chełkowski T, Gloor P, Jemielniak D (2016) Inequalities in open source software development: analysis of contributor?s commits in apache software foundation projects. PloS One 11(4):e0152,976
Cheng LT, Hupfer S, Ross S, Patterson J (2003) Jazzing up eclipse with collaborative tools. In: Proceedings of the 2003 OOPSLA workshop on eclipse technology eXchange, eclipse '03. ACM, New York, pp 5–49, https://doi.org/10.1145/965660.965670
Coelho J, Valente MT (2017) Why modern open source projects fail. In: Proceedings of the 2017 11th joint meeting on foundations of software engineering. ACM, New York, pp 186–196
Cohen J (1988) Statistical power analysis for the behavioral sciences, 2nd edn. Lawrence Earlbaum Associates, Mahwah
Conover WJ (1998) Practical nonparametric statistics, 3rd edn. Wiley, Hoboken, pp 07030–5774
Cross R, Liedtka J, Weiss L (2005) A practical guide to social networks. Harv Bus Rev 10(41):124–32
Crowston K, Howison J (2006) Assessing the health of open source communities. IEEE Comput 39(5):89–91. http://dblp.uni-trier.de/db/journals/computer/computer39.html#CrowstonH06
Crowston K, Wei K, Howison J, Wiggins A (2012) Free/libre open-source software development: what we know and what we do not know. ACM Comput Surv 44(2):7
Di Penta M, Tamburri DA (2017) Combining quantitative and qualitative studies in empirical software engineering research. In: Uchitel S, Orso A, Robillard MP (eds) Proceedings of the international conference on software engineering (ICSE Companion Volume), ACM. ACM Press, New York, pp 499–500. http://dblp.uni-trier.de/db/conf/icse/icse2017c.html#PentaT17
DiMaggio PJ, Powell WW (1983) The iron cage revisited: institutional isomorphism and collective rationality in organizational fields. Am Sociol Rev 48(2):147–160. https://doi.org/10.2307/2095101. http://www.jstor.org/stable/2095101?origin=crossref
Druskat S (2016) A proposal for the measurement and documentation of research software sustainability in interactive metadata repositories. arXiv:160804529
Egghe L, Rousseau R (2003) A measure for the cohesion of weighted networks. JASIST 54(3):193–202. http://dblp.uni-trier.de/db/journals/jasis/jasis54.html#EggheR03
Elkins T, Keller RT (2003) Leadership in research and development organizations: a literature review and conceptual framework. Leadersh Q 14(4-5):587–606
Espejo R (ed) (1993) Organisational fitness. Frankfurt am Main [u.a.], Campus-Verl.
Falessi D, Smith W, Serebrenik A (2017) Stress: A semi-automated, fully replicable approach for project selection. IEEE, ESEM, pp 151–156. http://dblp.uni-trier.de/db/conf/esem/esem2017.html#FalessiSS17
Fenton NE (1991) Software metrics - a rigorous approach. Chapman and Hall, UK
Ferenc R, Hegedüs P, Gyimóthy T (2014) Software product quality models. In: Evolving software systems. Springer, Berlin, pp 65–100
Ford D, Smith J, Guo PJ, Parnin C (2016) Paradise unplugged: identifying barriers for female participation on stack overflow. In: FSE, pp 846–857

Fredrickson JW (1986) The strategic decision process and organizational structure. The Academy of Mgmt Rev 11(2):280–297. https://doi.org/10.5465/AMR.1986.4283101
Fuks H, Raposo AB, Gerosa MA (2005) Applying the 3c model to groupware development. Int J Cooperative Inf Syst 14(2):299–328
Gallagher S (2006) Introduction: The arts and sciences of the situated body. Janus Head 9(2):1–2
Gamalielsson J, Lundell B (2013) Sustainability of open source software communities beyond a fork: how and why has the libreoffice project evolved? J Syst Softw 3(11):128–145. https://doi.org/10.1016/j.jss.2013.11.1077
Garzarelli G, Galoppini R (2003) Capability coordination in modular organization: voluntary fs/oss production and the case of debian gnu/linux. Industrial Organization 0312005, EconWPA. http://ideas.repec.org/p/wpa/wuwpio/0312005.html
Giraldo LF, Passino KM (2016) Dynamic task performance, cohesion, and communications in human groups. IEEE Trans Cyber 46(10):2207–2219
Glance DG (2004) Release criteria for the linux kernel. First Monday 9(4):4–5
Hata H, Todo T, Onoue S, Matsumoto K (2015) Characteristics of sustainable oss projects: A theoretical and empirical study. In: Proceedings of the 8th international workshop on cooperative and human aspects of software engineering, CHASE '15. IEEE Press, Piscataway, pp 15–21. http://dl.acm.org/citation.cfm?id=2819321.2819325
Hintze JL, Nelson RD (1998) Violin plots: a box plot-density trace synergism. Am Stat 52(2):181–184. https://doi.org/10.1080/00031305.1998.10480559. http://amstat.tandfonline.com/doi/abs/10.1080/00031305.1998.10480559
Hofstede G, Hofstede G, Minkov M (2010) Cultures and organizations: software of the mind, 3rd edn, McGraw-Hill Companies, Incorporated, IBM Inc. http://books.google.it/books?id=o4OqTgV3V00C
Homscheid D, Schaarschmidt M (2016) Between organization and community: investigating turnover intention factors of firm-sponsored open source software developers. In: Proceedings of the 8th international ACM web science conference. ACM Press, Piscataway, pp 326–337
Hung H, Gatica-Perez D (2010) Estimating cohesion in small groups using audio-visual nonverbal behavior. IEEE Trans Multimed 12(6):563–575
Jansen S (2014) Measuring the health of open source software ecosystems: beyond the scope of project health. Inf Softw Technol 56(11):1508–1519. http://dblp.uni-trier.de/db/journals/infsof/infsof56.html#Jansen14
Jeppesen HJ, Jnsson T, Shevlin M (2011) Employee attitudes to the distribution of organizational influence: who should have the most influence on which issues? Econ Ind Democr 32(1):69–86. https://doi.org/10.1177/0143831X10372432. http://eid.sagepub.com/content/32/1/69
Jiménez M, Piattini M (2008) Problems and solutions in distributed software development: a systematic review. In: Berkling, K, Joseph, M, Meyer, B, Nordio, M (eds) Second international conference on software engineering approaches for offshore and outsourced development SEAFOOD 2008, Zurich, Switzerland, July 2-3, 2008, lecture notes in business information processing, vol 16. Revised Papers, Springer, http://dblp.uni-trier.de/rec/bib/conf/seafood/JimenezP08, pp 107–125
Jongeling R, Sarkar P, Datta S, Serebrenik A (2017) On negative results when using sentiment analysis tools for software engineering research. Empir Softw Eng 22(5):2543–2584. https://doi.org/10.1007/s10664-016-9493-x
Kalliamvakou E, Gousios G, Blincoe K, Singer L, German DM, Damian DE (2016) An in-depth study of the promises and perils of mining github. Empir Softw Eng 21(5):2035–2071. http://dblp.uni-trier.de/db/journals/ese/ese21.html#KalliamvakouGBS16
Keivanloo I, Forbes C, Hmood A, Erfani M, Neal C, Peristerakis G, Rilling J (2012) A linked data platform for mining software repositories. In: In: 2012 9th IEEE working conference on mining software repositories (MSR), vol 3, no 6, pp 32–35. https://doi.org/10.1109/MSR.2012.6224296
Kilduff M, Tsai W (2003) Social networks and organizations. Sage Publications Ltd. http://www.amazon.com/Social-Networks-Organizations-Martin-Kilduff/dp/0761969578/ref=si3_rdr_bb_product/102-5868296-6616105
Kim H (2007) A multilevel study of antecedents and a mediator of employee-organization relationships. J Public Relat Res 19(2):167–197. https://doi.org/10.1080/10627260701290695
Kozdoba M, Mannor S (2015) Community detection via measure space embedding. In: Cortes C, Lawrence ND, Lee DD, Sugiyama M, Garnett R (eds) Advances in neural information processing systems 28: annual conference on neural information processing systems (NIPS), pp 2890–2898. http://dblp.uni-trier.de/db/conf/nips/nips2015.html#KozdobaM15
Kraut RE, Streeter LA (1995) Coordination in software development. Commun ACM 38(3):69–81
Krippendorff K (2004) Content analysis: an introduction to its methodology, 2nd edn. Sage Publications

Kujala S, Kauppinen M, Lehtola L, Kojo T (2005) The role of user involvement in requirements quality and project success. In: Proceedings of the 13th IEEE international conference on requirements engineering, RE '05. IEEE Computer Society, Washington, pp 75–84. https://doi.org/10.1109/RE.2005.72

Lai K, Wong CWY, Cheng TCE (2006) Institutional isomorphism and the adoption of information technology for supply chain management. Comput Ind 57(1):93–98. http://dblp.uni-trier.de/db/journals/cii/cii57.html#LaiWC06

Lancichinetti A, Fortunato S, Kertesz J (2008) Detecting the overlapping and hierarchical community structure of complex networks. http://arxiv.org/abs/0802.1218, Comment: 20 pages, 8 figures. Final version published on New Journal of Physics

Li W, Yang C, Yang C (2010) An active crawler for discovering geospatial web services and their distribution pattern - a case study of ogc web map service. Int J Geogr Inf Sci 24(8):1127–1147. http://dblp.uni-trier.de/db/journals/gis/gis24.html#LiYY10

Li Y, Tan CH, Teo HH (2012) Leadership characteristics and developers' motivation in open source software development. Inf Manag 49(5):257–267. http://dblp.uni-trier.de/db/journals/iam/iam49.html#LiTT12

Magnoni S, Tamburri DA, Di Nitto E, Kazman R (2017) Analyzing quality models for software communities. Communications of the ACM -: Under Review

Manning C, Schütze H (1999) Foundations of statistical natural language processing. MIT Press, Cambridge

Medus A, Acuña G, Dorso CO (2005) Detection of community structures in networks via global optimization. Physica A: Stat Mech Its Appl 358(2-4):593–604. http://www.sciencedirect.com/science/article/B6TVG-4G9PW36-3/1/b3321e67c43a26b2c87ddbe0579878a6

Mendez C, Padala HS, Steine-Hanson Z, Hilderbrand C, Horvath A, Hill C, Simpson L, Patil N, Sarma A, Burnett M (2018) Open source barriers to entry, revisited: a sociotechnical perspective. In: ICSE, pp 1004–1015

Miles M, Gilmore A, Harrigan P, Lewis G, Sethna Z (2015) Exploring entrepreneurial marketing. J Strateg Mark 23(2):94–111. https://doi.org/10.1080/0965254X.2014.914069

Millen DR, Fontaine MA, Muller MJ (2002) Understanding the benefit and costs of communities of practice. Commun ACM 45(4):69–73. https://doi.org/10.1145/505248.505276. http://portal.acm.org/citation.cfm?id=505276

Mislove A, Marcon M, Gummadi KP, Druschel P, Bhattacharjee B (2007) Measurement and analysis of online social networks. In: Proceedings of the 7th ACM SIGCOMM conference on internet measurement, IMC '07. ACM, New York, pp 29–42, https://doi.org/10.1145/1298306.1298311

Mockus A, Fielding RT, Herbsleb JD (2002) Two case studies of open source software development: apache and Mozilla. ACM Trans Softw Eng Methodol 11(3):309–346. https://doi.org/10.1145/567793.567795

Molzberger P (1986) Analyzing mental representation by means of nlp (neuro linguistic programming). In: Becker JD, Eisele I (eds) Proceedings of the workshop on parallel processing: logic, organization, and technology (WOPPLOT), Springer, Springer, NL, Lecture Notes in Computer Science, vol 253, pp 120–135. http://dblp.uni-trier.de/db/conf/wopplot/wopplot1986.html#Molzberger86

Moody J, White DR (2003) Structural cohesion and embeddedness: a hierarchical concept of social groups. Am Sociol Rev 68:103–127

Munaiah N, Kroh S, Cabrey C, Nagappan M (2017) Curating github for engineered software projects. Empir Softw Eng 22(6):3219–3253

Nagappan N, Murphy B, Basili V (2008) The influence of organizational structure on software quality: an empirical case study. In: International conference on software engineering. IEEE, Leipzig, pp 521–530. https://doi.org/10.1145/1368088.1368160

Nevo D, Wand Y (2005) Organizational memory information systems: a transactive memory approach. Decis Support Syst 39(4):549–562. http://dblp.uni-trier.de/db/journals/dss/dss39.html#NevoW05

Newman M (2003) Fast algorithm for detecting community structure in networks. Phys Rev E 69:667–674

Newman MEJ (2006) Modularity and community structure in networks. Proc Natl Acad Sci 103:8577–8582. http://www.pnas.org/cgi/doi/10.1073/pnas.0601602103

Newman MEJ, Girvan M (2004) Finding and evaluating community structure in networks. Phys Rev E 69(026113):620–627

Nguyen T, Wolf T, Damian D (2008) Global software development and delay: does distance still matter? In: 2008 IEEE International Conference on Global Software Engineering, 2008 ICGSE, vol 8, pp 45–54. https://doi.org/10.1109/ICGSE.2008.39

Nielsen SH (1995) Software quality management and organisational fit. Australasian J Inf Systems 3(1):1449–1576

di Nitto E, Gatti S, Invernizzi S, Tamburri DA (2013) Supporting awareness in open-source forges. Journal of Software: Evolution and Process - under review 1(4):1–21. Available Online for Peer-Review Only: https://tinyurl.com/ya3nhsqs

Nooteboom B, Vanhaverbeke W, Duysters G, Gilsing VA, van den Oord A (2006) Optimal cognitive distance and absorptive capacity. Res Policy 36(7):1016–1034

Novielli N, Calefato F, Lanubile F (2014) Towards discovering the role of emotions in stack overflow, vol 2014. ACM, New York, pp 33–36. https://doi.org/10.1145/2661685.2661689

Onoue S, Hata H, Matsumoto K (2014) 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, ACM, p 34

Onoue S, Hata H, Monden A, Matsumoto K (2016) Investigating and projecting population structures in open source software projects: a case study of projects in github. IEICE Trans Inf Syst 99(5):1304–1315

Oreja-Rodriguez JR, Yanes-Estevez V (2006) Knowledge structures of organisational environments: study of perceived uncertainty. IJKL 2(1/2):41–57. http://dblp.uni-trier.de/db/journals/ijkl/ijkl2.html#Oreja-RodriguezY06

Otte E, Rousseau R (2002) Social network analysis: a powerful strategy, also for the information sciences. J Inf Sci 28(6):441–453

Palomba F, Bavota G, Di Penta M, Oliveto R, Poshyvanyk D, De Lucia A (2015) Mining version histories for detecting code smells. IEEE Trans Softw Eng 41(5):462–489

Palomba F, Panichella A, Zaidman A, Oliveto R, De Lucia A (2017) The scent of a smell: an extensive comparison between textual and structural smells. IEEE Transactions on Software Engineering

Palomba F, Tamburri DA, Serebrenik A, Zaidman A, Fontana FA, Oliveto R (2018) How do community smells influence code smells? In: ICSE (Companion Volume), ACM

Pinzger M, Nagappan N, Murphy B (2008) Can developer-module networks predict failures? In: Proceedings of the 16th ACM SIGSOFT international symposium on foundations of software engineering, SIGSOFT '08/FSE-16. ACM, New York, pp 2–12. https://doi.org/10.1145/1453101.1453105

Prandy K (2000) The social interaction approach to the measurement and analysis of social stratification. No. 19 in 09, SAGE

Prattico L (2012) Governance of open source software foundations: who holds the power? Technol Innov Manag Rev 1(12):37–42

Prikladnicki R (2012) Propinquity in global software engineering: examining perceived distance in globally distributed project teams. J Soft Maint 24(2):119–137. http://dblp.uni-trier.de/db/journals/smr/smr24.html#Prikladnicki12

Raju K (2007) Is the future of software development in open source? Proprietary vs open source software: a cross country analysis. Journal of Intellectual Property Rights 12(2):21–42

Robles G, Gonzalez-Barahona JM, Herraiz I (2009) Evolution of the core team of developers in libre software projects. In: 2009 6th IEEE international working conference on mining software repositories, pp 167–170. https://doi.org/10.1109/MSR.2009.5069497

Robles G, Gonzalez-Barahona JM, Izquierdo-Cortazar D, Herraiz I (2011) Tools and datasets for mining libre software repositories, vol 1. IGI Global, Hershey, PA, chap 2, pp 24–42. http://www.igi-global.com/book/multi-disciplinary-advancement-open-source/46171

Romano J, Kromrey JD, Skowronek J, Devine L (2006) Exploring methods for evaluating group differences on the NSSE and other surveys: are the t-test and Cohen's d indices the most appropriate choices? In: Ann. meeting, South Assoc Institutional Research, pp 1–51

Ruikar K, Koskela L, Sexton M (2009) Communities of practice in construction case study organisations: questions and insights. Constr Innov 9(4):434–448. http://proquest.umi.com/pqdweb?did=1920022811&Fmt=7&clientId=4574&RQT=309&VName=PQD

Ryynnen H (2012) A social network analysis of internal communication in a matrix organisation - the context of project business. IJBIS 11(3):324–342. http://dblp.uni-trier.de/db/journals/ijbis/ijbis11.html#Ryynanen12

Sadowski BM, Sadowski-Rasters G, Duysters G (2008) Transition of governance in a mature open software source community: evidence from the debian case. Inf Econ Policy 20(4):323–332. http://dblp.uni-trier.de/db/journals/iepol/iepol20.html#SadowskiSD08

Sands R (2018) Blob post on openhub organizational features. https://blog.openhub.net/2012/10/introducing-ohloh-organizations-a-new-view-on-foss/

Schweik CM (2013) Sustainability in open source software commons: lessons learned from an empirical study of sourceforge projects. Technol Innov Manag Rev 3:13–19. http://timreview.ca/article/645

Severance C (2012) The apache software foundation: Brian Behlendorf. IEEE Comput 45(10):8–9. http://dblp.uni-trier.de/db/journals/computer/computer45.html#Severance12h

Siakas KV, Georgiadou E (2002) Empirical measurement of the effects of cultural diversity on software quality management. Softw Qual J 10(2):169–180. http://dblp.uni-trier.de/db/journals/sqj/sqj10.html#SiakasG02

Silva D, Tsantalis N, Valente MT (2016) Why we refactor? confessions of github contributors. In: Proceedings of the 2016 24th ACM SIGSOFT international symposium on foundations of software engineering, ACM, pp 858–870

Steinmacher I, Conte T, Gerosa MA, Redmiles D (2015) Social barriers faced by newcomers placing their first contribution in open source software projects. In: CSCW, ACM, pp 1379–1392

Stojanovic L, Ortega F, Dueñas S, Cañas-Díaz L (2011) Alert: active support and real-time coordination based on event processing in open source software development. In: Software maintenance and reengineering 2011 (CSMR). IEEE, Oldenburg, pp 359–362, https://doi.org/10.1109/CSMR.2011.52. http://www.se.uni-oldenburg.de/csmr2011/

Tamburri D, Casale G (2017) Cognitive distance vs. research output in doctoral computing education: a case-study. IEEE Transactions on Education 4(1):under review

Tamburri DA, di Nitto E, Lago P, van Vliet H (2012) On the nature of the GSE organizational social structure: an empirical study. In: Proceedings of the 7th IEEE international conference on global software engineering, vol 1, no 12, pp 114–123

Tamburri DA, Lago P, van Vliet H (2013a) Organizational social structures for software engineering. ACM Comput Surv 46(1):3,1–3,35. https://doi.org/10.1145/2522968.2522971

Tamburri DA, Lago P, van Vliet H (2013b) Uncovering latent social communities in software development. IEEE Soft 30(1):29–36. https://doi.org/10.1109/MS.2012.170

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

Tamburri DA, Kazman R, Fahimi H (2016) The architect's role in community shepherding. IEEE Soft 33(6):70–79. http://dblp.uni-trier.de/db/journals/software/software33.html#TamburriKF16

Tamburri DA, Palomba F, Serebrenik A, Zaidman A (2017) Discovering community types in open-source: a systematic approach and its evaluation - online appendix. http://tinyurl.com/y8oo4vkg

Tikhonov M (2016) Community-level cohesion without cooperation. eLife 5

Tourani P, Adams B, Serebrenik A (2017) Code of conduct in open source projects. ACM, Piscataway, pp 24–33. https://doi.org/10.1109/SANER.2017.7884606

Traag VA, Krings G, Dooren PV (2013) Significant scales in community structure. Nature. arXiv:1306.3398:66–89, http://dblp.uni-trier.de/db/journals/corr/corr1306.html#TraagKD13

Tsirakidis P, Kóbler F, Krcmar H (2009) Identification of success and failure factors of two agile software development teams in an open source organization. In: International conference on global software engineering, IEEE, pp 295–296. http://dblp.uni-trier.de/db/conf/icgse/icgse2009.html#TsirakidisKK09

Tullio DD, Staples DS (2014) The governance and control of open source software projects. J Manag Inf Syst 30(3):49–80. http://dblp.uni-trier.de/db/journals/jmis/jmis30.html#TullioS14

Vasilescu B, Filkov V, Serebrenik A (2015a) Perceptions of diversity on GitHub: a user survey. In: CHASE, pp 50–56

Vasilescu B, Posnett D, Ray B, van den Brand MGJ, Serebrenik A, Devanbu PT, Filkov V (2015b) Gender and tenure diversity in github teams. In: Begole B, Kim J, Inkpen K, Woo W (eds) Proceedings of the 33rd annual ACM conference on human factors in computing systems, CHI 2015, Seoul, Republic of Korea, April 18-23, 2015, ACM, pp 3789–3798. https://doi.org/10.1145/2702123.2702549

Wenger E (1998) Communities of practice: learning, meaning, and identity. Cambridge University Press, Cambridge

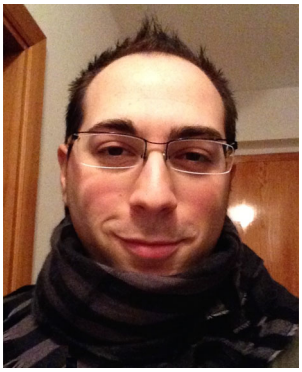
Wohlin C, Runeson P, Höst M, Ohlsson MC, Regnell B, Wesslén A (2000) Experimentation in software engineering: an introduction. Kluwer Academic Publishers, Norwell

Xiao L, Cai Y, Kazman R (2014) Design rule spaces: a new form of architecture insight. In: Jalote P, Briand LC, van der Hoek A (eds) Proceedings of the international conference on software engineering (ICSE). ACM, New York, pp 967–977. http://dblp.uni-trier.de/db/conf/icse/icse2014.html#XiaoCK14

Zhu H, Kraut R, Kittur A (2012) Organizing without formal organization: group identification, goal setting and social modeling in directing online production. In: Proceedings of the ACM 2012 conference on computer supported cooperative work. ACM Press, pp 935–944. http://dl.acm.org.proxy.lib.umich.edu/citation.cfm?id=2145344

Zich J, Kohayakawa Y, Rödl V, Sunderam V (2008) Jumpnet: improving connectivity and robustness in unstructured p2p networks by randomness. Internet Math 5(3):227–250. http://dblp.uni-trier.de/db/journals/im/im5.html#ZichKRS08

Zimmermann J (2008) Overcoming the inherent sources of liability of foreignness: measuring and compensating the disadvantage of being foreign. PhD thesis, Uni Augsburg



Damian A. Tamburri is an Assistant Professor at the Jheronimus Academy of Data Science, in s'Hertogenbosch, The Netherlands. He completed his Ph.D. at VU University Amsterdam, The Netherlands in March 2014 "cum laude and mention" one year in advance of a standard Ph.D. Contract. His research interests lie mainly in Complex Software Architectures (with a focus on Data-Intensive Architectures, Cloud & Microservices as well as Machine-Learning & Computational Intelligence Architectures), Complex Software Architecture Properties (with a focus on Privacy & Security), and Empirical Software Engineering (with a focus on Organisational, Social, and Societal aspects with Qualitative Methods, social-networks analysis as well as Machine-Learning). He has published over 80+ papers in either Journals such as the Transactions on Software Engineering (TSE) Journal, The ACM Computing Surveys (CSUR) Journal, the IEEE Software Magazine or top software engineering conferences (such as ICSE or FSE) and top software architecture conferences (such as ECSA or WICSA). He has been an active contributor and lead research in many EU FP6, FP7, and H2020 projects, such as S-Cube, MODAClouds, SeaClouds, DICE, ANITA, DossierCLOUD, ProTECT, and more. He is IEEE Software editorial board member, secretary of the TOSCA TC as well as secretary of the IFIP TC2, TC6, and TC8 WG on "Service-Oriented Computing".

Fabio Palomba is a Senior Research Associate at the University of Zurich, Switzerland. He received the PhD degree in Management & Information Technology from the University of Salerno, Italy, in 2017. His research interests include software maintenance and evolution, empirical software engineering, source code quality, change and defect prediction, and mining software repositories. He was also the recipient of two ACM/SIGSOFT and one IEEE/TCSE Distinguished Paper Awards at ASE'13, ICSE'15, and ICSME'17, respectively. Moreover, he was the recipient of a Best Paper Award Honorable Mention at CSCW'18 and a Best Tool Demo Paper Award at SANER'18. He serves and has served as a program committee member of various international conferences (e.g., MSR, ICPC, ICSME), and as referee for various international journals (e.g., TSE, TOSEM, JSS) in the fields of software engineering. Since 2016 he is Review Board Member of EMSE. He was the recipient of four Distinguished/Outstanding Reviewer Awards for his reviewing activities conducted for EMSE, IST (twice), and JSS. He co-organized the 2nd International Workshop on Machine Learning Techniques for Software Quality Evaluation (MaLTeSQuE 2018). He was also Guest Editor of special issues related to his research interests that will appear in EMSE and JSEP.

Alexander Serebrenik (PhD, K.U. Leuven, Belgium 2003; MSc, Hebrew University, Israel, 1999), senior member of IEEE, is an Associate Professor of software evolution at Eindhoven University of Technology. His research covers a wide range of topics, from source code analysis, to collaborative and human aspects of software engineering. He has co-authored a book "Evolving Software Systems" (Springer Verlag, 2014), and more than 100 scientific papers and articles. He has won Distinguished Paper awards at the International Conference on Software Engineering (2017) and the International Conference on the Quality of Information and Communications Technology (2014), as well as special contribution awards at ESEM 2018 and MSR 2017. He is the steering committee chair of the International Conference on Software Maintenance and Evolution. He is member of the editorial boards of Empirical Software Engineering (Spinger Verlag), Journal of Systems and Software (Elsevier), and Science of Computer Programming; he has also serves or served on the program committees of such software engineering conferences as ICSE, ESEC/FSE, ICSM(E), MSR, SANER and ICPC, winning several Distinguished Reviewer awards.

Andy Zaidman is an associate professor at the Delft University of Technology, The Netherlands. He obtained his MSc. (2002) and PhD degree (2006) in Computer Science from the University of Antwerp, Belgium. His main research interests are in software evolution, program comprehension, mining software repositories, software quality and software testing. He frequently serves on the program committees of conferences such as MSR, ICSM(E), ICPC, and SANER. He has been involved in the organisation of conferences such as WCRE (2008, 2009), VISSOFT (2014) and MSR (2018). He is an associate editor of the Journal of Systems and Software (JSS) and member of the steering committee of the Mining Software Repositories conference. In 2013 he was the laureate of a prestigious NWO Vidi research grant for his research into software testing.

Affiliations

Damian A. Tamburri[1] · Fabio Palomba[2] · Alexander Serebrenik[3] · Andy Zaidman[4]

Fabio Palomba
palomba@ifi.uzh.ch

Alexander Serebrenik
a.serebrenik@tue.nl

Andy Zaidman
zaidman@tudelft.nl

1 Jheronimus Academy of Data Science (JADS), Eindhoven University of Technology, s'Hertogenbosch, The Netherlands

2 University of Zürich, Zürich, Switzerland

3 Eindhoven University of Technology, Eindhoven, The Netherlands

4 Delft University of Technology, Delft, The Netherlands