The Secret Life of Software Communities: What we know and What we Don't know

Gemma Catolino
Delft University of Technology
g.catolino@tudelft.nl

Fabio Palomba
University of Salerno
fpalomba@unisa.it

Damian A. Tamburri
Eindhoven University of Technology and Jheronimus Academy of Data Science (JADS)
d.a.tamburri@tue.nl

Abstract

Communities of software practice are increasingly playing a central role in the development, operation, maintenance, and evolution of good-quality software, as well as DevOps pipelines, lean Organizations and Global Software Development. However, the structures and characteristics behind such communities are still unknown. For this reason, in this paper, we tried to explore the organizational secret of communities, trying to offer a few practical extracts of (1) what we know and is known, (2) what we know to be unknown, and (3) what we know to be tentatively discoverable in the near future from an empirical research point of view. Moreover, the paper provides a number of recommendations for practitioners to help and be helped in their community endeavors.

Index terms— Community Types; DevOps; Road Map

1 Introduction

Communities are the foundation and backbone of organized societies and just as much as our software-driven civil societies are dense with communities, the software makers are also organized into communities of practice (*e.g.*, in open-source), of intent (*e.g.*, in standardization groups), of purpose (*e.g.*, as part of agile movements). Recent studies showed how the community's health can reflect the quality of the software produced [1,2], for this reason the research community has tried to deeper analyze and characterize the structure of software communities (*i.e.*, defined as social units of size, dense strongly-typed and diverse interactions across diverse roles and fluid characteristics) [3].

In this paper, we discuss about a few basic facts and observations about the secret life of software communities which we were able to empirically establish over previous research studies as well as our own practice as part of software communities themselves.

We outline the roundup to encourage further research into understanding and harnessing the (still) very much secret life of open-source communities, as well as their structural, socio-technical, and health characteristics. Furthermore, we aim to call for help from the practitioners themselves in supporting their own software community activity and therefore, we provide practitioners with practical recommendations or calls for help defined with the intent of engaging their interest around the matter and/or allowing further understanding their needs from a research perspective, such that we as a research community can aid them in a better fashion. Moreover, we provide a brief road map for further research along this intriguing emerging topic.

In: D. Di Nucci, C. De Roover (eds.): Proceedings of the 18th Belgium-Netherlands Software Evolution Workshop, Brussels, Belgium, 28-11-2019, published at http://ceur-ws.org

2 Practical Recommendation

2.1 FACT 1. Software Communities Have Types

Several community types have been studied in organizational structures and social networks research - Figure 1 offers an overview of these types extracted from a systematic studies on the matter [3, 4]. In Layman's terms, a community type is a series of characteristics, a pattern, which remain constantly true across the social network underlying that community. What is still understood fairly little in the state of the art of software engineering research is that, on the one hand, software communities not surprisingly exhibit the same types already known in literature, but, on the other hand, the role of community types and characteristics for the benefit (or fallacy) of software code qualities as well as software processes remains mostly unknown.

What we don't know: (a) the influence of community types over software qualities as well as the qualities of software processes; (b) algorithms and measurements to precisely pinpoint type shifts across the community's life-cycle; (c) design patterns for community structures which are contiguous with design patterns in underlying software architectures.

What should practitioners do: (1) follow community tracking and measurement initiatives such as OpenHub[1] or Bitergia[2] to gather insights over their own community and act upon it - this ensures that progress in community management, measurement, and steering practices are harnessed; (2) engage in community quality and health initiatives such as CHAOSS[3] - this ensures that initiatives struggling to crack the code of sustainable software communities are well fed with engaged practitioners; (3) manifest their own organizational and socio-technical issues as much as the code-quality issues - this ensures that social software engineering [5] researchers have both quantities and qualities of the right data and evidence to study.

2.2 FACT 2. Types Influence Software Qualities

We conducted several observations/analyses to exploratively figure out the boundaries around the influence that software community types may be playing over the qualities for software production and its maintenance. Figure 2, for example, shows the influence of several types over the rate of re-opened issues investigated over 25 open-source communities sampled according to community participants age, size, gender diversity, community programming language, type of product, geographical dispersion, and activity, the last one measured as the mean activity stemming from Bitergia and OpenHub. The Box-plot in Figure 2 was generated by operationalizing a series of metrics to measure essential characteristics from Figure 1 and match them with corresponding types as well as their mean proneness to re-opening issues, that is, the mean ratio of re-opened issues over 6 months worth of releases in the projects lifetime. The plot shows that something is in fact going on - in this case, the data shows that Formal-Networks (second bar from the left) may be exhibiting an organizational behavior which tends to re-open issues more often than other types. Conversely, the opposite seems true for Informal-Communities (third bar from the left).

What we don't know: (a) a general quality model of what software qualities are influenced by which community structure qualities; (b) which community structure characteristics play a role for software communities; (c) how to quantify what is the additional cost or socio-technical debt connected to sub-optimal characteristics; (d) how to measure a type and use those measurements as devices for improved governance or organizational structure agility.

What should practitioners do: (1) open issues on issue-trackers not only for software code but also for perceived community structure issues - they are as much impactful as they are dangerous and can even lead to breaking the internet (see the NPM incident[4]); (2) report software community accidents on your version control system as much as you do on StackOverflow - researchers need to study both to come up with proper empirically established ground truths as well as practical outputs to support your work.

2.3 FACT 3. Types narrow with Practitioners' Experience

Similarly to results in Figure 2, we report that the number of types intermixed in the same community types, that is, the number of attributes diversity across a community reduces as the practitioners' age (or their experience, skills, community participation). In the same study that led us to prepare the plot in Figure 2, we reported an inverse correlation of ~ -0.40 ($p\text{-}value << 0.05$) between the mean developers experience with the community (i.e., the total time they spent working in the community) across our projects sample and the amount of community types and characteristics that manifest explicitly across the community.

What we don't know: (a) how to quantify mentorship and experience as assets across software communities and how to reward both in a proper fashion; (b) how to infer community evangelists and use them as thought leaders for their software communities; (c)

[1] http://openhub.net/ [2] https://bitergia.com/
[3] http://chaoss.community/
[4] https://tinyurl.com/yaferj3b

Name Features	CoP - Community of Practice	IN - Informal Network	FN - Formal Network	IC - Informal Community	NoP - Network of Practice	WG - Working Group	PT - Project Team	FG - Formal Group	PSC - Problem-Solving Community	LC - Learning Community	SN - Social Network Site	KC - Knowledge Community	SC - Strategic Community
Community Structure	high	high	high	high	high	high	high	high	high	high	high	high	high
Situatedness	high	low	low	low	low	high	high	high	low	low	low	low	low
Dispersion	low	high	high	high	high	low	low	low	high	low	low	high	high
Problem-Solving	low	low	low	low	low	high	high	high	high	low	low	low	high
Informality	high	high	low	low	high	high	high	high	high	high	low	high	low
Formality	low	low	high	low	low	high	high	high	low	low	low	high	high
Engagement	high	high	low	high	low	high	high	low	high	high	low	low	low
Cohesion	low	low	high	low	low	high	high	low	low	low	low	low	low
Duration	high	high	high	high	high	high	low	high	high	high	low	low	high
ROI-Tracking	low	low	low	low	low	low	high	low	low	low	low	low	high
Governance	low	low	high	low	low	high	high	high	low	low	low	low	high
Culture	low	low	low	low	low	high	low	low	high	high	low	low	low
Visibility	low	low	low	low	low	low	low	low	low	low	low	high	low

Figure 1: An Overview of Community Types from the State of the Arts [3]

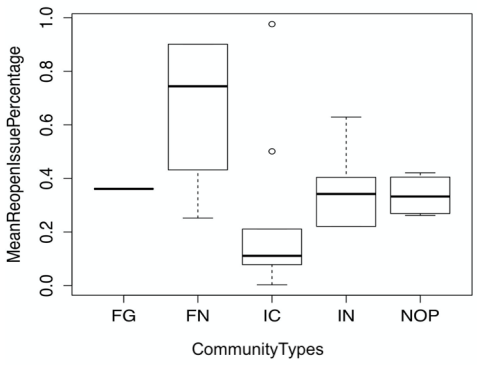

Figure 2: The influence of community types over reopened software issues

what skills and characteristics matter more than others in software communities and which skills grow with age and which others do not, as well as how to foster the growth and nourishment of skills that cannot grow with time (e.g., knowledge communication)

What should practitioners do: (1) prepare codes of conducts [6] which reflect also the mentorship, governance structure, experience and reward management as well as the code contribution policies or good-behavior across the community; (2) make the software code reflect the code of conduct, namely, use comments in software code to reprimand or exercise social sanctioning [4, 7] wherefore codes of conducts are violated.

2.4 FACT 4. Communities with more formal characteristics may result more often into abandon-ware

We observed that 82% of the abandonware communities in our sample exhibited a formal type in the last 6 months of their life. This is in line with previous research and observations from Crowston *et al.* [8] where informality is signaled as an essential software community health parameter.

What we don't know: (1) whether it is in fact true if formal characteristics and types lead to abandon-ware — empirical software engineering needs to be instrumented to actually assess and establish this link; (2) what 'formal' means, in the software engineering world and in terms of socio-technical and organizational relations — typically in software engineering a formal formulation involves proofs, foundational theories, but from the realm of social-networks analysis and organizations research, the word and concept of "formality" assumes a sensibly different meaning; (3) the measured role of "informal" as opposed to formal — how can one foster informal? Is Informal beneficial? These and similar research questions are still out there for the taking;

What should practitioners do: we don't know, yet.

2.5 FACT 5. Communities can be healthy and sustainable

Many studies in the literature identify healthy values for several community characteristics, *e.g.*, socio-technical congruence [9] or community structure verification [10]. These indicators, stemming from top studies, lead to argue that software communities, like other

thriving communities in our societal structures, can be turned healthy and made sustainable with appropriate and dedicated efforts — sites such as Bitergia, OpenHub, as well as initiatives and research projects such as OSSMeter [5] are fundamental assets to drive the societal challenge of making software communities aware and sensible to their health, as sustainable communities should be. In this respect, the state of the art in organizations research, social networks analysis as well as emerging disciplines such as sustainable community development [8, 11] can aid but even typical, structured approaches used in software engineering such as formalization [12], *e.g.*, to better understand and measure the socio-organizational dynamics playing a role across software communities. Although some scholars may see this endeavor with some healthy skepticism and assign less value to it in comparison to other areas of software engineering, such as software testing or maintenance, it is important to note that these aspects have shown to be of major importance as it can be seen from the research literature [13] or [14].

What we don't know: (a) a systematically-generated, general quality model for software communities with supporting evaluation and analytic tools and automatons; (b) a practical implementation of sustainability in software communities which are currently managed in a rather trial-and-error fashion, with due exceptions (e.g., the Apache Software Foundation); (c) the dimensions, qualities, and metrics of software community sustainability, to be used jointly with a quality model for performing, long-lived, high-quality software communities.

What should practitioners do: we don't know, yet.

3 Conclusions

As it turns out, the impact of communities in software development needs to be further investigated. As a matter of fact, our non-exhaustive list cannot convey enough that there is much more we don't know with respect to how much we do know; most especially, we don't know a lot about what should practitioners do to cater for their communities, striving for healthy types matching their organizational requirements over time and in a sustainable fashion. Our conclusion is not only that more work is needed to explore the aspects we non-exhaustively pointed at — rather, the most dire observation is that, in order to better support the (secret) life of software practitioners in their communities, practitioners themselves may be well encouraged to treat the community as a software-influencing artifact itself, thus, for example, opening issues if a community issue or smell does in fact manifest. An increased practitioner community consciousness will increase awareness over the problem, making it more explicit, measurable, and hence improvable by practitioners as well as for researchers. In the future it is our ultimate intention to further pursue the road map that emerges from the previous identified shortcomings, and, at the same time, work to improve software forges, collaborative coding environments, IDEs or other software equipment that practitioners may use to build, maintain, or work upon their code as part of a lively and healthy community and with more appropriate software engineering 'ergonomics', intended as the disciplines of software engineering which engage into designing or arranging software commons, communities, workplaces, work-products, and working systems so that they fit the people and goals around them [8].

[5] `http://ossmeter.org/`

References

[1] Fabio Palomba, Damian A. Tamburri, Alexander Serebrenik, Andy Zaidman, Francesca Arcelli Fontana, and Rocco Oliveto. How do community smells influence code smells? In *ICSE (Companion Volume)*, pages 240–241. ACM, 2018.

[2] Fabio Palomba, Damian Andrew Andrew Tamburri, Francesca Arcelli Fontana, Rocco Oliveto, Andy Zaidman, and Alexander Serebrenik. Beyond technical aspects: How do community smells influence the intensity of code smells? *IEEE transactions on software engineering*, 2018.

[3] Damian A Tamburri, Patricia Lago, and Hans van Vliet. Organizational social structures for software engineering. *ACM Computing Surveys (CSUR)*, 46(1):3, 2013.

[4] Damian A Tamburri, Patricia Lago, and Hans Van Vliet. Uncovering latent social communities in software development. *IEEE software*, 30(1):29–36, 2012.

[5] Will Tracz. Lord of the files: essays on the social aspects of software engineering by russel ovans. *ACM SIGSOFT Software Engineering Notes*, 36(6):31–31, 2011.

[6] Computing Machinery ACM. Acm code of ethics and professional conduct. *Code of Ethics*, 1992.

[7] Karl Sigmund, Christoph Hauert, Arne Traulsen, and Hannelore De Silva. Social control and the social contract: the emergence of sanctioning systems for collective action. *Dynamic Games and Applications*, 1(1):149–171, 2011.

[8] Richard T Watson, Marie-Claude Boudreau, and Adela J Chen. Information systems and environmentally sustainable development: Energy informatics and new directions for the is community. *MIS quarterly*, 34(1), 2010.

[9] Marcelo Cataldo, James D Herbsleb, and Kathleen M Carley. Socio-technical congruence: a framework for assessing the impact of technical and work dependencies on software development productivity. In *Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement*, pages 2–11. ACM, 2008.

[10] Mitchell Joblin, Wolfgang Mauerer, Sven Apel, Janet Siegmund, and Dirk Riehle. From developer networks to verified communities: a fine-grained approach. In *Proceedings of the 37th International Conference on Software Engineering-Volume 1*, pages 563–573. IEEE Press, 2015.

[11] Juan Lucena, Jen Schneider, and Jon A Leydens. Engineering and sustainable community development. *Synthesis Lectures on Engineers, Technology, and Society*, 5(1):1–230, 2010.

[12] Dieter Rombach and Frank Seelisch. Formalisms in software engineering: Myths versus empirical facts. In *IFIP Central and East European Conference on Software Engineering Techniques*, pages 13–25. Springer, 2007.

[13] Darja Šmite and Zane Galviņa. Socio-technical congruence sabotaged by a hidden onshore outsourcing relationship: lessons learned from an empirical study. In *International Conference on Product Focused Software Process Improvement*, pages 190–202. Springer, 2012.

[14] Johann Rost and Robert L Glass. *The dark side of software engineering: evil on computing projects*. John Wiley & Sons, 2011.