Social Debt Analytics for Improving the Management of Software Evolution Tasks

Fabio Palomba
Delft University of Technology
Eindhoven University of Technology
The Netherlands
f.palomba@tudelft.nl
f.palomba@tue.nl

Alexander Serebrenik
Eindhoven University of Technology
The Netherlands
a.serebrenik@tue.nl

Andy Zaidman
Delft University of Technology
The Netherlands
a.e.zaidman@tudelft.nl

***Abstract*—The success of software engineering projects is in a large part dependent on social and organization aspects of the development community. Indeed, it not only depends on the complexity of the product or the number of requirements to be implemented, but also on people, processes, and how they impact the technical side of software development. Social debt represents patterns across the organizational structure around a software system that may lead to additional unforeseen project costs. Condescending behavior, disgruntlement or rage quitting are just some examples of social issues that may occur among the developers of a software project. While the research community has recently investigated the underlying dynamics leading to the introduction of social debt (e.g., the so-called "community smells" which represent symptoms of the presence of social problems in a community), as well as how such debt can be payed off, there is still a noticeable lack of empirical evidence on how social debt impacts software maintenance and evolution. In this paper, we present our position on how social debt can impacts technical aspects of source code by presenting a road map toward a deeper understanding of such relationship.**

I. Research Problem and Motivation

Software engineering is clearly characterized by two main components, i.e., technical and social [39]. While the former is related to the processes aimed at producing sound technical products that meet the expected requirements, the latter has to do with the relationships involving organizations, developers, and stakeholders who are responsible for the definition of the technical product [12]. According to the Netherlands Knowledge and Innovation Agenda ICT 2016 - 2019, *"software and system complexity is not solely of technological nature but also defined by people and processes"*[1]. Indeed, the success of software engineering projects is strongly dependent on social and organization aspects of the development community [11].

In past and recent years the software evolution research community has actively focused on technical aspects of software projects, by (i) understanding the key factors making technical products easier to maintain [2], [4], [10], [29], [32], [44], [48] or (ii) devising techniques and tools to support developers during different evolutionary tasks [17], [25], [31], [36], [45]. For example, Cunningham [14] introduced the metaphor of *technical debt*, which refers to practices performed by developers that lead to the introduction of bad design solutions that will turn in an additional cost during software maintenance and evolution. One noticeable symptom of technical debt is the presence of *code smells* [19], i.e., bad programming practices applied by developers that lead the source code to be less maintainable and more change- and fault-prone [30].

On the other hand, software communities have been the subject of several empirical investigations as well [5], [9], [15], [33], [40], [41]: in particular, they have mainly focused on how such communities evolve [21], [27], [28], [47], while they only partially considered the relationships between community-related information and software maintenance and evolution of technical products. Indeed, such studies especially targeted the so-called *social debt*, i.e., unforeseen project cost connected to a "suboptimal" development community (i.e., both in structure and behavior) [42], [43]. For instance, Tamburri et al. [41], [43] defined a set of *community smells*, a set of socio-technical characteristics (e.g., high formality) and patterns (e.g., recurrent condescending behavior, or rage-quitting), which may lead to the emergence of social debt. One of the typical community smells they found is the *Organizational Silo Effect*, which arises when a software community presents siloed areas that essentially do not communicate, except through one or two of their respective members: as a consequence, the development activities might be delayed due to lack of communication between developers.

Besides the studies on social debt, a consistent number of empirical analyses have been carried out on the so-called *socio-technical congruence* [8], i.e., the alignment between coordination requirements extracted from technical dependencies among tasks and the actual coordination activities performed by the developers. While studies in this category had the intention to investigate the relationship between social and technical sides of software communities (e.g., studying how the collaboration among developers influence their productivity [16], [18]), we believe there is still a lack of evidence on their connection, i.e., how social-related issues occurring among the developers of a software project and indicated by the presence of social debt might impact the quality of

[1] https://www.4tu.nl/nirict/en/Research/knowledge-and-innovation-agenda-ict-2016-2020.pdf

both technical products and processes. In other words, it is still unclear whether factors like coordination issues, lack of communication, or structure of development community have an effect on the quality of source code (e.g., code smells [19]) or the quality of software processes (e.g., code review [2] and bug resolution [22] completion time). We believe that such relationship is of a paramount importance for enlarging the knowledge on how software systems evolve and how to properly support developers along with software evolution. While some exploratory studies in this direction have been conducted in the domain of requirements engineering for coordination by Cataldo et al. [8] and Kwan et al. [24], both these lines of inquiry and related research are still far from relating social and technical debt with the goal of comprehending their co-creation and evolution.

In this paper, we present our position about the relationship between social and technical aspects of source code by describing our road map toward its proper management.

II. On the Relationship between Social and Technical Debt

The first envisioned step toward the exploration of the connection between social and technical debt is to assess whether such relationship actually exists and, as a consequence, what its strength is. In particular, we hypothesize that community-related issues can be the cause of introducing technical debt. To verify this hypothesis, we believe that future research effort should be devoted to design a number of empirical investigations aimed at characterizing the interaction between the two aspects, including but not limited to:

1) *How social debt impact the presence of technical debt.* As briefly explained in the previous section, community smells indicate frequent negative patterns over a software community organization that might result in additional project costs. We suppose that such project costs can be basically due to (i) pure social-related issues raising between developers and (ii) technical issues arising in the source code. While the causes behind additional costs due to the presence of social issues within communities [5], [9], [15], [34], an important challenge for the research community is that to assess the extent to which additional technical costs are due to social debt. Specifically, we believe that future research should focus on the relationship between social debt, e.g., measured in terms of community smells [41], and technical aspects such as (i) presence of code smells and (ii) introduction of defects. In the first case, a clear real-case scenario might involve the presence of sub-communities that do not communicate with each other (e.g., in presence of a *Organizational Silo* smell) that, as a consequence, are not able to come up with a correct way to modularize the different modules of the systems they are developing, thus possibly introducing architectural or code smells like *Promiscuous Package* or *Blob* [7], [19]. In the second case, it is reasonable to think that the presence of issues within a community might lead developers to misunderstand requirements implementation or design choices to apply, thus possibly introducing bugs. For instance, Kwan et al. [24] suggested that social debt not only impacts software build success, but also may contribute to worsen program comprehension.
2) *How different software community types are affected by technical debt.* As clearly visible looking at the modern software development practices, current systems heavily rely on open-source software (OSS) [13], [35]. Indeed, over the past years OSS has moved from an academic curiosity to a mainstream focus within software communities [13]. Despite their high popularity, open-source development communities rarely rely on governance insights from organizations research and/or tracking their organizational status using social networks analysis (SNA) [23], e.g., to evaluate the current salient social and organizational characteristics describing their community. Such a lack of structured information might lead to two undesirable consequences. On the one hand, the implosion of the community: for example, an increasing number of reflections and empirical evidence over abandonware [26], [37] or even failure of entire open-source forges indicates the need to understand and better support the organizational and social aspects of open-source communities [20], [38]. On the other hand, the introduction of specific technical debt based on the underlying community type of a software project: for instance, communities characterized by the presence of formal vs informal communications might suffer of different technical issues due to the different nature of developers' interactions. Thus arises the need to support clients of OSS in the evaluation of the risks associated to the usage of such open-source code in their systems. Therefore, an important challenge is represented by a deeper analysis of (i) how to automatically identify the key characteristics of different community types so that they can be classified and (ii) to what extent different community types exhibit different technical debt such as code smells or bugs.

To face the challenges described so far, a mixture of quantitative and qualitative analyses is required. Indeed, on the one the assessment of the relationship between social and technical debt needs to be performed by means of correlation, statistical, and predictive analyses. On the other hand, surveys or semi-structured interviews with developers might reveal additional insights on such a relationship, possibly explaining (i) the motivations behind it and (ii) effective ways to reorganize software communities to remove social debt.

III. On the Impact of Social Debt on Technical Tasks Completion Effort

Once we have assessed the actual relationship between social and technical debt of software systems, the subsequent step that we envision is to determine the impact of community-related aspects on different evolutionary tasks performed by

developers during software maintenance. In particular, we believe that poor communication between developers increases the time needed to complete technical tasks such as code review, issue resolution, etc. This set of empirical investigations include:

1) *How social debt impacts code review and bug resolution time.* The modern assessment of software quality during maintenance and evolution is typically guided by modern code reviews [2]: In a typical code review, one or more developers dress the role of reviewers and verify that the proposed source code changes meet the quality requirements with respect to readability, maintainability, and the absence of defects, before being deployed into production. In this context, social issues occurring within the development community might influence this process in two different ways: (i) how source code is reviewed, i.e., the quality of a review, and (ii) the time needed to verify that the changes applied meet the requirements. Indeed, following the socio-technical argumentations proposed so far [8], [24], problems in the technical development may be a reflection of issues occurring at the community-level. At the same time, another critical software engineering process during maintenance and evolution is related to the way developers fix bugs [1]: on the one hand, we believe that the research community should focus on investigations aimed at understanding (i) how the presence of community-related issues impacts the ability to diagnose bugs and (ii) whether the time needed to resolve a bug is higher when the reviewer is involved in social debt.

2) *How social debt impacts maintenance effort estimation models.* Since its birth, software effort estimation has represented the *Holy Grail* of software engineering [6], since it supports project managers in the correct estimation of effort and costs needed to develop a software system. The problem is still more tricky when it comes to the maintenance and evolution: indeed, this phase is responsible for more than 80% of the total cost of software [3]. While some researchers proposed the use of product and process metrics for estimating the effort estimation of future maintenance tasks [46], following the conjectures reported previously in this section we believe that a promising challenge is to exploit social debt as additional factors to correctly estimate the costs of maintenance and evolution tasks.

IV. Improving Software Evolution Tools by Exploiting Social Debt

Based on the results achieved from the studies described in Sections II and III, several software evolution tools might be improved by means of social debt analysis. For instance, it would be possible to study solutions aimed at (i) detecting social debt in software communities (e.g., by means of social network analysis techniques), (ii) prioritizing technical debt based on the co-occurrence of social debt, (iii) improving code review and bug resolution tools by exploiting social debt when recommending the developers that should perform such tasks, and (iv) improving maintenance effort estimation capabilities adding social debt information as additional predictors. The results of such line of research are intended to be useful for both managers, interested in solving potential social issues in a community, and developers, interested in estimating the effort—social and technical—required for maintaining the source code. Finally, the research might set a research agenda for socio-technical factors in software development.

V. Conclusion

Social debt is a manifestation of issues occurring within a software development team. For instance, community smells reflect sub-optimal organizational and socio-technical characteristics or patterns in the organizational structure of the software community. Much in the same way, technical debt represents sub-optimal solutions applied by programmers during the development of technical artifacts. Recent empirical studies provided some insights into a possible relationships between them that, however, has not been shown yet.

In this paper, we highlight a possible outline of future research on the relationship between social and technical debt arising in software systems, by describing a set of empirical studies that might lead to a more comprehensive understanding of the phenomenon. Starting from the connection between symptoms of the presence of social and technical debt, i.e., community and code smells, important challenges for the software evolution research community include the understanding of (i) how different software development community types are affected by social and technical debt, (ii) how different software development practices, such as code review, bug resolution, or effort estimation might be impacted by issues occurring at community-level, and (iii) whether and to what extent current software quality evaluation techniques and tools can be improved by exploiting social debt information.

We believe that such challenges would help in the deeper understanding of how software systems evolve and how community-related factors impact this process.

VI. Acknowledgment

Fabio Palomba is funded by the 4TU-NIRICT project “Social Aspects of Software Quality”. Any opinions, findings, and conclusions expressed herein are the authors’ and do not necessarily reflect those of the sponsors.

References

[1] J. Anvik, L. Hiew, and G. C. Murphy. Who should fix this bug? In *Proceedings of the 28th international conference on Software engineering*, pages 361–370. ACM, 2006.

[2] A. Bacchelli and C. Bird. Expectations, outcomes, and challenges of modern code review. In *Proceedings of the 2013 international conference on software engineering*, pages 712–721. IEEE Press, 2013.

[3] R. D. Banker, S. M. Datar, C. F. Kemerer, and D. Zweig. Software complexity and maintenance costs. *Communications of the ACM*, 36(11):81–95, 1993.

[4] P. Bhattacharya and I. Neamtiu. Fine-grained incremental learning and multi-feature tossing graphs to improve bug triaging. In *Software Maintenance (ICSM), 2010 IEEE International Conference on*, pages 1–10. IEEE, 2010.

[5] C. Bird, N. Nagappan, H. Gall, B. Murphy, and P. Devanbu. Putting it all together: Using socio-technical networks to predict failures. In *Proceedings of the 2009 20th International Symposium on Software Reliability Engineering*, ISSRE '09, pages 109–119, Washington, DC, USA, 2009. IEEE Computer Society.

[6] L. C. Briand and I. Wieczorek. Resource estimation in software engineering. *Encyclopedia of software engineering*, 2002.

[7] W. H. Brown, R. C. Malveau, H. W. S. McCormick, and T. J. Mowbray. *AntiPatterns: Refactoring Software, Architectures, and Projects in Crisis*. John Wiley & Sons, Inc., New York, NY, USA, 1st edition, 1998.

[8] M. Cataldo, J. D. Herbsleb, and K. M. Carley. Socio-technical congruence: a framework for assessing the impact of technical and work dependencies on software development productivity. In H. D. Rombach, S. G. Elbaum, and J. Mnch, editors, *Proceedings of the Int'l Symposium on Empirical Software Engineering and Measurement (ESEM)*, pages 2–11. ACM, 2008.

[9] M. Cataldo and S. Nambiar. The impact of geographic distribution and the nature of technical coupling on the quality of global software development projects. *Journal of Software: Evolution and Process*, 24(2):153–168, 2012.

[10] A. Chatzigeorgiou and A. Manakos. Investigating the evolution of code smells in object-oriented systems. *Innov. Syst. Softw. Eng.*, 10(1):3–18, Mar. 2014.

[11] M. Conway. How do committees invent. *Datamation*, 14(4):28–31, 1968.

[12] M. E. Conway. How do committees invent. *Datamation*, 14(4):28–31, 1968.

[13] K. Crowston, K. Wei, J. Howison, and A. Wiggins. Free/libre open-source software development: What we know and what we do not know. *ACM Computing Surveys (CSUR)*, 44(2):7, 2012.

[14] W. Cunningham. The wycash portfolio management system. *SIGPLAN OOPS Mess.*, 4(2):29–30, Dec. 1992.

[15] S. Datta, R. Sindhgatta, and B. Sengupta. Evolution of developer collaboration on the jazz platform: a study of a large scale agile project. In *Proceedings of the 4th India Software Engineering Conference*, ISEC '11, pages 21–30, New York, NY, USA, 2011. ACM.

[16] C. R. de Souza and D. F. Redmiles. On the roles of apis in the coordination of collaborative software development. *Computer Supported Cooperative Work (CSCW)*, 18(5-6):445, 2009.

[17] D. Di Nucci, F. Palomba, G. De Rosa, G. Bavota, R. Oliveto, and A. De Lucia. A developer centered bug prediction model. *IEEE Transactions on Software Engineering*, 2017.

[18] S. B. Fonseca, C. R. De Souza, and D. F. Redmiles. Exploring the relationship between dependencies and coordination to support global software development projects. In *Global Software Engineering, 2006. ICGSE'06. International Conference on*, pages 243–243. IEEE, 2006.

[19] M. Fowler. *Refactoring: Improving the Design of Existing Code*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.

[20] J. Gamalielsson and B. Lundell. Sustainability of open source software communities beyond a fork: How and why has the libreoffice project evolved? *Journal of Systems and Software*, 3(11):128–145, 2013.

[21] Y. Gao, V. Freeh, and G. Madey. Analysis and modeling of the open source software community. *NAACSOS, Pittsburgh*, 2003.

[22] G. Jeong, S. Kim, and T. Zimmermann. Improving bug triage with bug tossing graphs. In *Proceedings of the the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, pages 111–120. ACM, 2009.

[23] D. Knoke and S. Yang. *Social network analysis*, volume 154. Sage, 2008.

[24] I. Kwan, A. Schroter, and D. Damian. Does socio-technical congruence have an effect on software build success? a study of coordination in a software project. *IEEE Trans. Softw. Eng.*, 37(3):307–324, May 2011.

[25] M. Lanza. The evolution matrix: Recovering software evolution using software visualization techniques. In *Proceedings of the 4th international workshop on principles of software evolution*, pages 37–42. ACM, 2001.

[26] T. Mens, M. Claes, P. Grosjean, and A. Serebrenik. Studying evolving software ecosystems based on ecological models. In *Evolving Software Systems*, pages 297–326. Springer, 2014.

[27] T. Mens and M. Goeminne. Analysing the evolution of social aspects of open source software ecosystems. In *IWSECO@ ICSOB*, pages 1–14, 2011.

[28] K. Nakakoji, Y. Yamamoto, Y. Nishinaka, K. Kishida, and Y. Ye. Evolution patterns of open-source software systems and communities. In *Proceedings of the international workshop on Principles of software evolution*, pages 76–85. ACM, 2002.

[29] W. Oizumi, A. Garcia, L. da Silva Sousa, B. Cafeo, and Y. Zhao. Code anomalies flock together: Exploring code anomaly agglomerations for locating design problems. In *Software Engineering (ICSE), 2016 IEEE/ACM 38th International Conference on*, pages 440–451. IEEE, 2016.

[30] F. Palomba, G. Bavota, M. Di Penta, F. Fasano, R. Oliveto, and A. De Lucia. On the diffuseness and the impact on maintainability of code smells: a large scale empirical investigation. *Empirical Software Engineering*, pages 1–34, 2017.

[31] F. Palomba, G. Bavota, M. D. Penta, R. Oliveto, D. Poshyvanyk, and A. D. Lucia. Mining version histories for detecting code smells. *IEEE Transactions on Software Engineering*, 41(5):462–489, May 2015.

[32] F. Palomba, A. Panichella, A. Zaidman, R. Oliveto, and A. De Lucia. The scent of a smell: An extensive comparison between textual and structural smells. *IEEE Transactions on Software Engineering*, 2017.

[33] M. Pinzger, N. Nagappan, and B. Murphy. Can developer-module networks predict failures? In *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*, SIGSOFT '08/FSE-16, pages 2–12, New York, NY, USA, 2008. ACM.

[34] L. Prattico. Governance of open source software foundations: Who holds the power? *Technology Innovation Management Review*, 1(12):37–42, 2012.

[35] K. Raju. Is the Future of Software Development in Open Source? Proprietary vs Open Source Software: A Cross Country Analysis. *Journal of Intellectual Property Rights, Vol. 12, No. 2, March 2007*, 12(2):21–42, 2007.

[36] R. Robbes and M. Lanza. A change-based approach to software evolution. *Electronic Notes in Theoretical Computer Science*, 166:93–109, 2007.

[37] G. Robles, J. M. González-Barahona, C. Cervigón, A. Capiluppi, and D. Izquierdo-Cortázar. Estimating development effort in free/open source software projects by mining software repositories: a case study of openstack. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, pages 222–231. ACM, 2014.

[38] C. M. Schweik. Sustainability in open source software commons: Lessons learned from an empirical study of sourceforge projects. *Technology Innovation Management Review*, 3:13–19, 01/2013 2013.

[39] I. Sommerville. *Software Engineering*. Addison-Wesley, Harlow, England, 9. edition, 2010.

[40] D. Surian, D. Lo, and E.-P. Lim. Mining collaboration patterns from a large developer network. In *WCRE*, pages 269–273, 2010.

[41] D. A. Tamburri, R. Kazman, and H. Fahimi. The architect's role in community shepherding. *IEEE Software*, 33(6):70–79, 2016.

[42] D. A. Tamburri, P. Kruchten, P. Lago, and H. van Vliet. What is social debt in software engineering? In *ICSE - CHASE Workshop Series*, pages 40–49, 2013.

[43] D. A. Tamburri, P. Kruchten, P. Lago, and H. van Vliet. Social debt in software engineering: insights from industry. *J. Internet Services and Applications*, 6(1):10:1–10:17, 2015.

[44] M. Tufano, F. Palomba, G. Bavota, R. Oliveto, M. D. Penta, A. D. Lucia, and D. Poshyvanyk. When and why your code starts to smell bad (and whether the smells go away). *IEEE Transactions on Software Engineering*, PP(99):1–1, 2017.

[45] M. White, M. Tufano, C. Vendome, and D. Poshyvanyk. Deep learning code fragments for code clone detection. In *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering*, pages 87–98. ACM, 2016.

[46] H. Wu, L. Shi, C. Chen, Q. Wang, and B. Boehm. Maintenance effort estimation for open source software: A systematic literature review. In *Software Maintenance and Evolution (ICSME), 2016 IEEE International Conference on*, pages 32–43. IEEE, 2016.

[47] J. Xu, Y. Gao, S. Christley, and G. Madey. A topological analysis of the open souce software development community. In *System Sciences, 2005. HICSS'05. Proceedings of the 38th Annual Hawaii International Conference on*, pages 198a–198a. IEEE, 2005.

[48] A. Zaidman, B. Van Rompaey, S. Demeyer, and A. Van Deursen. Mining software repositories to study co-evolution of production & test code. In *Software Testing, Verification, and Validation, 2008 1st International Conference on*, pages 220–229. IEEE, 2008.