Prof.dr. Alexander Serebrenik
November 4, 2022

INAUGURAL LECTURE

Social Software Engineering

TU/e
EINDHOVEN UNIVERSITY OF TECHNOLOGY

DEPARTMENT OF MATHEMATICS AND COMPUTER SCIENCE

PROF.DR. ALEXANDER SEREBRENIK

Social Software Engineering

Presented on November 4, 2022
at Eindhoven University of Technology

Introduction

Software is being developed for people and by people. So, it is not surprising that differences between the people developing software are reflected in how software is developed and what the resulting software looks like. Moreover, it is rare that any complex software product is created by working alone. Good teamwork is the requisite input for a great product: two heads are better than one, many hands make light work, teamwork makes the dream work and hundreds of other idioms hail the benefits of teamwork. However, looking around at scientific collaborations or at daily life, each one of us can easily recognise that not every group of people can successfully work together. In fact, when managing the development of OS/360 in the 1960s, Fred Brooks observed that merely adding more programmers to a project falling behind schedule delayed the project even further. He described these experiences in his book 'The Mythical Man-Month', published in 1975, which is broadly hailed as one of the most important books in software engineering.

Fast forward to 2022, almost 50 years after 'The Mythical Man-Month'. Software has become even more present than in 1975. Indeed, just like oxygen is an invisible but essential element for all lifeforms, software is an essential yet invisible driving force of the present world: there's virtually no aspect of society that is not facilitated or mediated by software and every company is in fact a software company.[1]
Not only did software change; the ways in which it is being developed have also changed. While Brooks was referring to developers depending on their colleagues to jointly create a large programming system, developers today depend on software created by people on the other side of the globe, across continents and timezones, paid programmers and volunteers alike. Finally, society has changed: there is more attention to such topics as equality, discrimination and safety, implying new standards of acceptable behavior. As software influences society and is influenced by it, **we need to create better software to create a better society** and **we need to create a better software world to create better software**. I am using the word 'world' here in a very broad sense to encompass groups of different sizes and scales, companies and open-source communities involving professional developers and people developing software who do not see themselves as developers, such as research engineers or accountants working with Excel.

[1] Paraphrased from the European Software Manifesto.

So, how do we create a better software world?

By making it more inclusive.

So what is inclusion? Inclusion refers to an individual's perception that their unique contribution to the organization or project is appreciated and that their full participation is encouraged.[2] From the management literature, we know that inclusion involves satisfying two complementary needs: uniqueness and belonging.[3] On the one hand, a sense of belonging is important, but contributors that have to give up part of their unique experiences in order to belong do not really experience inclusion. For example, when we have interviewed LGBTIQ+ software developers, many of them have stressed the importance of bringing their whole selfs to work, i.e., being able to belong without sacrificing their gender identity or sexual orientation. Similarly, older developers are being encouraged to "blend in" with the younger crowd, attempting to find this sense of belonging by updating their CVs, choice of dress and hairstyle or performing plastic surgery. On the other hand, recognition of someone's uniqueness without ensuring that people feel that they belong is not enough for inclusion. For example, we observed older developers working at places where most of their coworkers are much younger; they sometimes feel out of place in social situations but are valued in more technical ones.

So, the first question we need to address is understanding how software development is experienced by individuals from minoritized groups as opposed to those from the dominant group.

[2] Michàlle E. Mor Barak (2015) Inclusion is the Key to Diversity Management, but What is Inclusion?, Human Service Organizations: Management, Leadership & Governance, 39:2, 83-88, DOI: 10.1080/23303131.2015.1035599, paraphrased.

[3] Shore, L. M., Randel, A. E., Chung, B. G., Dean, M. A., Holcombe Ehrhart, K., & Singh, G. (2011). Inclusion and diversity in work groups: A review and model for future research. Journal of Management, 37, 1262–1289. DOI:10.1177/0149206310385943

Human aspects of software engineering

While the expression 'human aspects' can cover a multitude of subjects, we focus here on the aspects of software engineering as experienced by individuals as opposed to aspects related to collaboration and communication between developers. Studies on human aspects typically consider such subjects as the interdependence between the cognitive and emotional states of individual developers and the software systems that they are working on, interactions between developers and the systems that they use to create software, the personalities of software developers and the appropriateness of specific tasks to different personalities, mentoring, and career development. More recently, colleagues have also studied topics related to stress and burnout.

WHAT DO WE KNOW ABOUT THE EXPERIENCES OF DEVELOPERS FROM MINORITIZED GROUPS?

When it comes to a comparison of individuals from minoritized groups as opposed to those from the dominant group, the research so far has predominantly focused on gender and, in particular, on the binary interpretation of gender, comparing women and men. For example, Imtiaz et al. have observed that while women concentrate their work across fewer projects and organizations, men contribute to a higher number of projects and organizations,[4] that men and women follow different comprehension strategies when reading source code,[5] and that while men tend to switch more frequently between debugging strategies,[6] some end-user tools for debuggers may not fully support women's preferred debugging

[4] Imtiaz N, Middleton J, Chakraborty J, Robson N, Bai G, Murphy-Hill E (2019) Investigating the effects of gender bias on github. In: 2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE). IEEE, pp 700–711

[5] Zohreh Sharafi, Soh Z, Guéhéneuc Y-G, Antoniol G (2012) Women and men – different but equal: On the impact of identifier style on source code reading. In: 2012 20th IEEE International Conference on Program Comprehension (ICPC). IEEE, pp 27–36

[6] Cao J, Rector K, Park TH, Fleming SD, Burnett M, Wiedenbeck S (2010) A debugging perspective on end-user mashup programming. In: 2010 IEEE Symposium on Visual Languages and Human-Centric Computing. IEEE, pp 149–156

strategies.[7] In our previous work, we have observed that men engage for longer on Stack Overflow[8] and are similarly more likely to stay around for longer on GitHub.[9] At the same time, multiple studies report a lack of statistically significant gender differences between men and women in terms of e.g., the ability to learn new features when debugging[10] or individual productivity in OSS projects.[11] In our previous work, we have observed that the duration of engagement for women and men is comparable for mailing list-based Drupal and WordPress.[12]

Much less attention has been paid to studying the experiences of software developers from other minoritized groups. Topics related to national culture have been studied in the context of pull request submission and code reviews: GitHub contributors from countries with low human development indexes face more rejections than other contributors from countries with high human development indexes,[13] while contributors from Switzerland are more than two times as likely to see their pull requests being accepted than their peers from China.[14] Older developers are often stereotyped as more rigid and unfamiliar with the most recent technological advances,[15] even though there is not a strong correlation between age and technical knowledge in specific knowledge areas.[16] Neurodiverse developers have reported that team meetings and job interviews cause stress, as does interpreting communication through the multitude of channels that

[7] Subrahmaniyan N, Beckwith L, Grigoreanu V, Burnett M, Wiedenbeck S, Narayanan V, Bucht K, Drummond R, Fern X (2008) Testing vs. code inspection vs. what else? Male and female end users' debugging strategies. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. pp 617–626

[8] Vasilescu B, Capiluppi A, Serebrenik A (2014) Gender, representation and online participation: A quantitative study. Interact Comput. 26(5):488–511

[9] Qiu HS, Nolte A, Brown A, Serebrenik A, Vasilescu B (2019) Going farther together: The impact of social capital on sustained participation in open source. In: 2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE). IEEE, pp 688–699

[10] Beckwith L, Inman D, Rector K, Burnett M (2007) On to the real world: Gender and self-efficacy in Excel. In: IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC 2007). IEEE, pp 119–126

[11] Bosu A, Sultana KZ (2019) Diversity and inclusion in open source software (OSS) projects: Where do we stand? In: 2019 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM), pp 1–11. IEEE

[12] Vasilescu B, Capiluppi A, Serebrenik A (2014) Gender, representation and online participation: A quantitative study. Interact Comput. 26(5):488–511

[13] Furtado L, Cartaxo B, Treude C, Pinto G (2020) How successful are open source contributions from countries with different levels of human development? IEEE Software

[14] Rastogi A, Nagappan N, Gousios G, van der Hoek Andre (2018) Relationship between geographical location and evaluation of developer contributions in GitHub. In: Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, pp 1–8

[15] Baltes S, Park G, Serebrenik A (2020) Is 40 the new 60? How popular media portrays the employability of older software developers. IEEE Software 37(6):26–31

[16] Morrison P, Murphy-Hill E (2013) Is programming knowledge related to age? In: MSR 2013

developers use.[17] In fact, many neurodiverse developers do not disclose their diagnosis despite their desire for some form of accommodation. Finally, developers with visual disabilities have often reported struggling with inadequate tool support, such as screen readers.[18]

It is well known, however, that different diversity aspects do not operate in isolation. In fact, Kimberlé Crenshaw introduced the concept of intersectionality, arguing that diversity aspects are not mutually exclusive but intersecting,[19] implying that one should be acutely aware of different challenges experienced by people at the intersection of multiple diversity aspects. For example, the experiences of Black women differ from those of Black men and of non-Black women: for example, "a smaller percentage of Black women reported being introduced to CS by a family member or a friend (17% and 3%, respectively) than was the case for non-Black women (24% and 10%, respectively) and Black men (21% and 9%, respectively)."[20] Moreover, Black women do not necessarily know whether their negative experiences should be attributed to their gender or their race.[21] In a recently completed master's thesis, similar observations have been made for older women, who are sometimes unsure of whether the negative experiences were because of their gender or their age. An early study of transgender women who develop software has identified three important themes, namely control of identity disclosure, ability to obtain economically stable work and autonomy to disengage or re-engage - all of these have been enabled by voluntary remote work, but it is not known to what extent these themes are recognised by cisgender women and transgender individuals of other genders.[22]

[17] Morris MR, Begel A, Wiedermann B. (2015) Understanding the challenges faced by neurodiverse software engineering employees: Towards a more inclusive and productive technical workforce. In: Proceedings of the 17th International ACM SIGACCESS Conference on Computers & Accessibility. pp 173–184

[18] Mealin S, Murphy-Hill E (2012) An exploratory study of blind software developers. In: 2012 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC). IEEE, pp 71–74

[19] K. W. Crenshaw, "Race, reform, and retrenchment: Transformation and legitimation in antidiscrimination law," Harvard Law Rev., vol. 101, no. 7, pp. 1331–1387, 1988. DOI: 10.2307/1341398.

[20] Ross M, Hazari Z, Sonnert G, Sadler P (2020) The intersection of being black and being a woman: Examining the effect of social computing relationships on computer science career choice. ACM Trans Comput Education (TOCE) 20(2):1–15

[21] Thomas JO, Joseph N, Williams A, Burge J et al (2018) Speaking truth to power: Exploring the intersectional experiences of black women in computing. In: 2018 Research on Equity and Sustained Participation in Engineering, Computing, and Technology (RESPECT). IEEE, pp 1–8

[22] Ford D, Milewicz R, Serebrenik A (2019) How remote work can foster a more inclusive environment for transgender developers. In: 2019 IEEE/ACM 2nd International Workshop on Gender Equality in Software Engineering (GE). IEEE, pp 9–12

WHAT DO WE NEED TO UNDERSTAND IN THE COMING YEARS?

This calls for a concerted effort to reach out to different groups of developers. We should be aware that demographic differences influence differences in experiences and we should no longer assume that the opinions of young, Caucasian, US-based, abled, straight men from the middle class can be generalized to other developers.

Indeed, according to the Evans Corporation, India is expected to overtake the US as the largest developer population center by 2024, with Latin America experiencing the second strongest growth. This means, at the very least, that interview and survey-based studies claiming some form of global insight into software engineering phenomena should not be limited to English and should consider, for example, Spanish, Portuguese, Chinese and Japanese. Unfortunately, with the notable exception of the 'Pandemic Programming' study,[23] multilingual surveys are not common. Furthermore, it is not uncommon to equate the geographic location of the place of work with the national culture, failing to take into account phenomena related to migration, multiculturality and colonialism. This calls for a more careful reflection on the country and nationality-related data that we collect and use when studying diversity and inclusion in the software engineering context.

We also need to expand the scope of our studies to people developing software who do not necessarily consider themselves software developers. First of all, there are people developing software in a very different context, e.g., computational scientists or kids learning how to program. They do not necessarily consider themselves software developers and indeed do not necessarily have the same goals. For example, computational scientists are more likely to be interested in the results of the computation than in the source code that has been created to perform the computation and may completely overlook the software engineering aspects we tend to take for granted, such as testing, documentation and version control. In fact, I recently talked to a colleague from a different department and discovered that the entire research group uses a software system for their work but every single person has a slightly different version of it. New students "inherit" a version from their predecessors and extend it in their own ways. Ultimately, the

[23] Paul Ralph, Sebastian Baltes, Gianisa Adisaputri, Richard Torkar, Vladimir Kovalenko, Marcos Kalinowski, Nicole Novielli, Shin Yoo, Xavier Devroey, Xin Tan, Minghui Zhou, Burak Turhan, Rashina Hoda, Hideaki Hata, Gregorio Robles, Amin Milani Fard, Rana Alkadhi: Pandemic Programming. Empir. Softw. Eng. 25(6): 4927-4961 (2020)

landscape of currently available versions is highly complex - nobody is even sure where all of the versions are, let alone how they are related to each other! Another colleague has tried to organize pizza parties and documentation hackathons to no avail: PhD students feel the pain of working with undocumented code and do not feel eager to document it themselves. Another very different group of people developing software but not considering themselves software engineers are individuals without formal college or university training. In fact, there are two engineering titles in the Netherlands: 'ir.' for engineers trained at universities of technology and 'ing.' for engineers trained at universities of applied sciences. This kind of separation of different kinds of developers is not limited to the Netherlands. Recently, I interviewed a US-based developer who indicated that they only had bootcamp training and that their university-trained peers repeatedly made a point that they do not deserve the title of an engineer.

FROM UNDERSTANDING TO SUPPORT

Understanding the experiences and needs of developers from minoritized groups is, of course, the first step towards providing better support, either through better software engineering tools or through better software engineering processes. For example, based on a series of well-chosen studies from psychology, Margaret Burnett and Anita Sarma have created GenderMag,[24] a process enabling software practitioners to find gender-inclusivity 'bugs' in their software, i.e., aspects that might hinder the use of the software by people of specific genders. What is important is that GenderMag does not stereotype people by claiming that "all women behave like this" or "all men behave like that" but identifies gender-related differences with respect to five facets: motivations to use software, information processing styles, computer self-efficacy, attitudes toward risk, and style of learning new technologies. For example, while about 2/3 of men and 1/3 of women were motivated by exploring next-generation technology, more than 40% of women and merely 10% of men did not enjoy exploring next-generation technology. Based on these facets, GenderMag has created several personas, such as Abi and Tim. Abi embodies facets that are more common among women and Tim among men. So, Abi does not enjoy exploring next-generation technology while Tim does. By putting themselves in Abi's shoes, developers can imagine how people sharing Abi's facets will interact with their software and find bugs. GenderMag has been successfully applied in many companies and for different kinds of software, from

[24] Margaret M. Burnett, Anicia Peters, Charles Hill, Noha Elarief: Finding Gender-Inclusiveness Software Issues with GenderMag: A Field Investigation. CHI 2016: 2586-2598

educational software to conference websites. Building on GenderMag and the meta-approach known as InclusiveMag[25], we need to create similar approaches supporting other diversity aspects, whether this is age, socio-economic background, ethnicity or sexual orientation.

In addition to tools, developers might be supported through better software development processes, such as mentoring. Mentoring is a process in which a more experienced or more knowledgeable person (a mentor) helps to guide a less experienced or less knowledgeable person (a mentee). Workplace mentoring is commonly associated with benefits for both the mentees, e.g., career satisfaction, and the mentors, e.g., organizational power.[26] However, different individuals might have very different needs related to mentoring. For example, LGBTIQ+ are often specifically looking for other LGBTIQ+ for mentorship,[27] but non-'out' mentees might be reluctant to be mentored by 'out' mentors, fearing being seen as LGBTIQ+. Moreover, while LGBTIQ+ mentees with LGBTIQ+ mentors reported more job satisfaction, involvement and psychosocial support than those with heterosexual mentors, those with heterosexual mentors received more promotions.[28] From their side, mentors might experience unwelcomeness or hostility[29] and keep their sexual orientations private.[30] While these observations are likely to hold for any domain, the climate in the engineering disciplines is especially problematic for LGBTIQ+[31] due to heteronormative cultural norms that are common in engineering disciplines[32] and the limited attention to engineering from the broader LGBTIQ+ community.[33] Compared to other engineering disciplines, computer science is seen as less LGBTIQ+-friendly than e.g., biological

[25] Margaret Burnett: From GenderMag to InclusiveMag: A Journey for University IT. SIGUCCS 2021: 1-2

[26] A. Ramaswami, G. F. Dreher (2010). The Benefits Associated with Workplace Mentoring Relationships. The Blackwell Handbook of Mentoring: A Multiple Perspectives Approach.

[27] G.M. Russell, S.G. Horne, Finding equilibrium: Mentoring, sexual orientation, and gender identity. Prof. Psych. Res. Pract. 40, 194–200 (2009)

[28] M.R. Hebl, J. Lin, S. Tonidandel, J. Knight (2003). Super models: The impact of like-mentors for homosexual employees. Poster Society for Industrial and Organizational Psychology annual conf.

[29] E.V. Patridge, R.S. Barthelemy, S.R. Rankin, Factors impacting the academic climate for LGBQ STEM faculty. J Women Minor Sci En. 20, 75–98 (2014).

[30] D. Bilimoria, A.J. Stewart, "Don't Ask, Don't Tell": The academic climate for lesbian, gay, bisexual, and transgender faculty in science and engineering. NWSA J. 21, 85–103 (2009).

[31] E.A. Cech, M.V.; Pham. Queer in STEM Organizations: Workplace Disadvantages for LGBT Employees in STEM Related Federal Agencies. Soc. Sci. 2017, 6, 12.

[32] J.L. Linley, K.A. Renn, M.R. Woodford (2018) Examining the Ecological Systems of LGBTQ STEM Majors. J Women Minor Sci Eng, 24(1):1-16.

[33] K.F. Trenshaw, A. Hetrick, R.F. Oswald, S.L. Vostral, M.C. Loui (2013). Lesbian, gay, bisexual, and transgender students in engineering: Climate and perceptions. Frontier Education Conf. 1238–1240.

and chemical engineering but more friendly than e.g., mechanical engineering.[34] Hence, we need to design mentoring approaches geared towards software developers from minoritized groups such as LGBTIQ+.

Summarizing our research view in the area of human aspects of software engineering, we need to gain a more profound understanding of the experiences and needs of people from the understudied groups who create software and should design technological and organizational solutions that support their work and match their needs.

However, as explained at the beginning, software developers often work in teams. Hence, we consider the collaborative aspects of software engineering in the complementary lines of work.

[34] E.A. Cech, T.J. Waidzunas (2011) Navigating the heteronormativity of engineering: the experiences of lesbian, gay, and bisexual students, Engineering Studies, 3:1, 1-24

Collaborative aspects of software engineering

Team collaboration has many facets and software teams are no exception. In this section, we focus on the following separate but interrelated elements addressing the following questions: how are software engineering teams composed? How do they emerge? How do team members communicate and collaborate? How can this collaboration be facilitated or hindered by automatic tools such as continuous integration solutions or bots? And finally, how do different team compositions and ways of collaboration affect the software systems that the teams produce and how are they affected by them? From the inclusion perspective, the first question is related to diversity, i.e., how the team is composed in terms of such attributes as gender, age, ethnicity, nationality or sexual orientation. The second question is related to inclusion, as the sense of belonging and recognition of uniqueness both manifest themselves through communication. The third question is related to the hybrid nature of modern software teams: while software developers have been using tools such as compilers and IDEs for ages, developers nowadays no longer merely use tools but collaborate with them as equal partners in the software development process: GitHub CoPilot[35] generates source code based on natural language descriptions, SAPFIX fixes bugs,[36] and Dependabot updates dependencies[37]- all of these tasks were manually performed by developers in the past. Finally, the last question embeds teamwork in the socio-technical context by explicitly referring to the systems that developers are working on.

DIVERSITY IN SOFTWARE ENGINEERING TEAMS

Similarly to studies of individual developers, diversity in software teams has been mostly considered through the gender lens. Studies highlight that gender-

[35] https://github.com/features/copilot/
[36] Alexandru Marginean, Johannes Bader, Satish Chandra, Mark Harman, Yue Jia, Ke Mao, Alexander Mols, Andrew Scott: SapFix: automated end-to-end repair at scale. ICSE (SEIP) 2019: 269-278
[37] https://github.com/dependabot

diverse teams perform better[38] and are more productive,[39] more effective and more coordinated[40] than non-gender-diverse ones. Moreover, as women tend to take mediating roles,[41] software development teams are less likely to develop sub-optimal communication patterns known as community smells in the presence of women.[42] At the same time, nationality diversity has a negative impact on community engagement[43] and quality of work.[44] Similarly, studies of cultural diversity suggest that high cultural diversity negatively affects team communication[45] and that, in particular, forming sustainable software development teams might be more challenging in cultures with high levels of inequality in the hierarchical structure of software development organizations.[46] These results might appear independent but in practice they are not: as the percentage of women in computing broadly differs among countries, many gender-diverse teams in countries with a low percentage of women in computing involve female immigrants. Anecdotal evidence suggests that this might be the case in e.g., the Netherlands. In India, gender interacts with age/family status as Indian familial norms pressure women into computing as a field and, at the same time, encourage them to leave this upon marriage, implying that gender-diverse teams are more likely to be age-diverse as well.[47]

[38] Gila AR, Jaafa J, Omar M, Tunio MZ (2014) Impact of personality and gender diversity on software development teams' performance. In: 2014 International Conference on Computer, Communications, and Control Technology (I4CT). IEEE, pp 261–265

[39] Vasilescu B, Posnett D, Ray B, van den Brand MGJ, Serebrenik A, evanbu P, Filkov V (2015) Gender and tenure diversity in GitHub teams. In: Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems, pp 3789–3798

[40] Marques M (2015) Software engineering education – Does gender matter in project results? – A Chilean case study. In: 2015 IEEE Frontiers in Education Conference (FIE). IEEE, pp 1–8

[41] Razavian M, Lago P (2015) Feminine expertise in architecting teams. IEEE Software 33(4):64–71

[42] Catolino G, Palomba F, Tamburri DA, Serebrenik A, Ferrucci F (2019) Gender diversity and women in software teams: How do they affect community smells? In: 2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Society (ICSE-SEIS), pp 11–20. IEEE

[43] Daniel S, Agarwal R, Stewart KJ (2013) The effects of diversity in global, distributed collectives: A study of open source project success. Inf. Syst. Res. 24(2):312–333

[44] Pieterse V, van Eekelen M (2018) Cultural diversity and the performance of student software engineering teams. In: Southern African Computer Lecturers Association (SACLA), vol 2018, p 120

[45] Casey V (2009) Leveraging or exploiting cultural difference? In: 2009 Fourth IEEE International Conference on Global Software Engineering, pp 8–17. IEEE

[46] Borchers G (2003) The software engineering impacts of cultural factors on multi-cultural software development teams. In: 25th International Conference on Software Engineering, 2003. Proceedings. IEEE, pp 540–545

[47] Divy Thakkar, Nithya Sambasivan, Purva Kulkarni, Pratap Kalenahalli Sudarshan, and Kentaro Toyama. 2018. The Unexpected Entry and Exodus of Women in Computing and HCI in India. In Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems (CHI '18). Association for Computing Machinery, New York, NY, USA, Paper 352, 1–12

The aforementioned interactions between different diversity aspects call for follow-up studies taking an intersectional perspective. As a first step in this direction, our recent study of community smells in open-source projects[48] has included variables related to gender diversity, diversity of national cultures and geographic distances. The results were surprising as the addition of variables related to national culture and geographic distances rendered some of the gender-related variables no longer statistically significant, while some of those related to the national culture remained so. While this was an early study and definitely needs to be replicated with different data, it highlights the importance of including variables related to different diversity aspects when performing statistical modeling of software engineering phenomena. Moreover, following the intersectional perspective, interactions between those variables should be explicitly taken into account, similarly to our previous study on gender and social capital.[49] We have found evidence that the attachment of women to open teams with regard to diversity of information (which is known to be associated with weak ties and hence the *bridging social of capital*) increases their chance of prolonged engagement; diversity of information interacts with gender.

COMMUNICATION IN SOFTWARE ENGINEERING TEAMS

Understanding communication in software engineering teams is essential for community managers and project leaders to gauge the atmosphere within their projects and communities. Moreover, developers themselves can use the insights in the communication to decide whether they would like to join the community or continue contributing to it.

[48] Stefano Lambiase, Gemma Catolino, Damian Andrew Tamburri, Alexander Serebrenik, Fabio Palomba, Filomena Ferrucci. Good Fences Make Good Neighbours? On the Impact of Cultural and Geographical Dispersion on Community Smells, 44th International Conference on Software Engineering, Software Engineering in Society, 2022

[49] Qiu HS, Nolte A, Brown A, Serebrenik A, Vasilescu B (2019) Going farther together: The impact of social capital on sustained participation in open source. In: 2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE). IEEE, pp 688–699

The aforementioned community smells are one of the examples of suboptimal communication patterns within software teams. Community smells[50] focus on the structural aspects of communication, i.e., who is talking to whom. For example, the Organizational Silo community smell refers to the presence of siloed areas of the community that do not communicate except through one or two of their respective members. However, community smells do not necessarily capture *how* developers communicate with each other. Moreover, while some smells are defined in reference to the tone of the conversation, e.g., the Lone Wolf smell that arises "when the development community presents unsanctioned or defiant contributors who carry out their work with little consideration of their peers, their decisions and communication," the tools proposed by the community to detect code smells[51][52] are not capable of taking the tone into account.

Conversation tone is related to sentiment and emotion expressed in communication between software developers. The detection of sentiment and emotion in such a context is a challenging task[53]; general purpose texts perform poorly on software engineering texts[54] and sentiment analysis tools designed and trained for software engineering data from one platform (e.g., Stack Overflow) underperform when applied to a different software engineering platform (e.g., Jira or GitHub issue comments).[55] However, the presence of multiple sentiment analysis tools targeting the software engineering domain (Senti4SD,[56] SentiStrength-

[50] Damian Andrew Tamburri, Philippe Kruchten, Patricia Lago, and Hans van Vliet. Social Debt in Software Engineering: Insights from Industry. Journal of Internet Services and Applications (2015).

[51] Damian Andrew Tamburri, Fabio Palomba, and Rick Kazman. Exploring Community Smells in Open-Source: An Automated Approach. IEEE Transactions on Software Engineering 47, 3 (2019), 630–652.

[52] Carlos Paradis, Rick Kazman. Design Choices in Building an MSR Tool: The Case of Kaiaulu. International Workshop on Mining Software Repositories for Software Architecture, 2021.

[53] Nicole Novielli, Fabio Calefato, Filippo Lanubile: The challenges of sentiment detection in the social programmer ecosystem. SSE@SIGSOFT FSE 2015: 33-40

[54] Robbert Jongeling, Proshanta Sarkar, Subhajit Datta, Alexander Serebrenik: On negative results when using sentiment analysis tools for software engineering research. Empir. Softw. Eng. 22(5): 2543-2584 (2017)

[55] Nicole Novielli, Fabio Calefato, Filippo Lanubile, Alexander Serebrenik: Assessment of off-the-shelf SE-specific sentiment analysis tools: An extended replication study. Empir. Softw. Eng. 26(4): 77 (2021)

[56] Fabio Calefato, Filippo Lanubile, Federico Maiorano, and Nicole Novielli. 2018. Sentiment polarity detection for software development. Empir. Softw. Eng. 23, 3 (2018), 1352–1382

SE[57], SentiCR,[58] SentiSW,[59] SentiSE,[60] SEntiMoji,[61] and SESSION[62]) and a recent systematic literature review[63] discussing the strengths and weaknesses of these tools (as well as concerns or limitations that SE researchers might need to take into account when applying or customizing these tools) have opened up the possibility for the integration of sentiment-related concerns when detecting suboptimal communication patterns.

Furthermore, even in the absence of suboptimal communication patterns, conversations in software engineering teams or on software engineering platforms might turn toxic.[64] Toxicity is defined as "rude, disrespectful, or unreasonable language that is likely to make someone leave a discussion."[65] By definition, toxic behavior is an example of exclusionary behavior. Toxicity is related to negative sentiment and emotions such as anger and disgust but, at the same time, it is different to them e.g., "colors are horrible for […], just look at this shit" is both toxic and expresses negative sentiment, while the arrogant "Never heard about [standard]? A baseline for developers. Use Google" is undoubtedly toxic but does not express sentiment (both examples are due to Miller et al.[66]). Toxic communication between developers has been observed on such platforms as

[57] Md Rakibul Islam and Minhaz F. Zibran. 2018. SentiStrength-SE: Exploiting domain specificity for improved sentiment analysis in software engineering text. J. Syst. Softw. 145 (2018), 125–146

[58] Toufique Ahmed, Amiangshu Bosu, Anindya Iqbal, and Shahram Rahimi. 2017. SentiCR: A customized sentiment analysis tool for code review interactions. In: Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering, Grigore Rosu, Massimiliano Di Penta, and Tien N. Nguyen (Eds.). IEEE Computer Society, 106–111

[59] Jin Ding, Hailong Sun, Xu Wang, and Xudong Liu. 2018. Entity-level sentiment analysis of issue comments. In Proceedings of the 3rd International Workshop on Emotion Awareness in Software Engineering, Andrew Begel, Alexander Serebrenik, and Daniel Graziotin (Eds.). ACM, 7–13

[60] https://github.com/amiangshu/SentiSE

[61] Zhenpeng Chen, Yanbin Cao, Xuan Lu, Qiaozhu Mei, and Xuanzhe Liu. 2019. SEntiMoji: An emoji-powered learning approach for sentiment analysis in software engineering. In: Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. Association for Computing Machinery, 841–852

[62] Kexin Sun, Hui Gao, Hongyu Kuang, Xiaoxing Ma, Guoping Rong, Dong Shao, He Zhang: Exploiting the Unique Expression for Improved Sentiment Analysis in Software Engineering Text. ICPC 2021: 149-159

[63] Bin Lin, Nathan Cassee, Alexander Serebrenik, Gabriele Bavota, Nicole Novielli, Michele Lanza: Opinion Mining for Software Development: A Systematic Literature Review. ACM Trans. Softw. Eng. Methodol. 31(3): 38:1-38:41 (2022)

[64] Sophie Cohen: Contextualizing toxicity in open source: a qualitative study. ESEC/SIGSOFT FSE 2021: 1669-1671

[65] Originally proposed by Google's project Jigsaw, this definition was used widely by Google in their toxicity detection research.

[66] Courtney Miller, Sophie Cohen, Daniel Klug, Bogdan Vasilescu, Christian Kästner: "Did You Miss My Comment or What?" Understanding Toxicity in Open Source Discussions. ICSE 2022: 710-722

GitHub Issues,[67] GitHub Actions,[68] Gitter, Slack and Stack Overflow.[69] It has been hypothesized that toxic environments might burn out open-source developers.[70] Moreover, practitioners recognise the importance of detecting toxicity and creating bots that can respond to toxic language on GitHub; these include, for example, the sentiment-bot[71] or the safe-space bot.[72] Unfortunately, the effectiveness of these bots is not clear due to the ability of developers to circumvent the automatic detection of undesirable behavior and subjective and culturally-dependent notions of toxicity. For example, the extent to which profanity is actually toxic is controversial. Linus Torvalds has often argued that his use of expletives is part of his leadership style, referring to "management by perkele" as a cultural phenomenon in Finland. Moreover, Baruch and Jenkins have identified the relevance, and even the importance, of using non-conventional and sometimes uncivil language in the workplace.[73] This calls for more toxicity detection mechanisms that are capable of taking into account the project culture, e.g., project governance styles.[74]

Whether toxic or not, communication in software development teams might involve conflicts. Moreover, when conflicts emerge in online communities such as remote teams or open-source software projects, they usually result in a bigger mess compared to offline situations because people demonstrate more disinhibited behavior online, resulting in quickly escalating conflicts.[75] Not surprisingly, several studies have reported conflicts in open-source communities such as GNU Enterprise,[76] Netbeans,[77] Linux, Apache and Debian.[78] Moreover,

[67] Miller et al. op. cit.
[68] Hideaki Hata, Nicole Novielli, Sebastian Baltes, Raula Gaikovina Kula, Christoph Treude: GitHub Discussions: An exploratory study of early adoption. Empir. Softw. Eng. 27(1): 3 (2022)
[69] Jithin Cheriyan, Bastin Tony Roy Savarimuthu, Stephen Cranefield: Towards offensive language detection and reduction in four Software Engineering communities. EASE 2021: 254-259
[70] Raman, N., Cao, M., Tsvetkov, Y., Kästner, C., & Vasilescu, B. (2020). Stress and burnout in open source. Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: New Ideas and Emerging Results, 57–60
[71] https://probot.github.io/apps/sentiment-bot/
[72] https://github.com/charliegerard/safe-space
[73] Yehuda Baruch, Stuart Jenkins. Swearing at work and permissive leadership culture. When anti-social becomes social and incivility is acceptable. Leadership & Organization Development Journal Vol. 28 No. 6, 2007 pp. 492-507
[74] Adam Alami, Raul Pardo, Marisa Leavitt Cohn, Andrzej Wasowski. Pull Request Governance In Open Source Communities. IEEE Transactions on Software Engineering 2022.
[75] John Suler: The online inhibition effect. CyberPsychology and Behavior 7(3), 321-326, 2004
[76] Margaret S. Elliott, Walt Scacchi: Free software developers as an occupational community: resolving conflicts and fostering collaboration. GROUP 2003: 21-30
[77] Chris Jensen, Walt Scacchi: Collaboration, Leadership, Control, and Conflict Negotiation and the Netbeans.org Open Source Software Development Community. HICSS 2005
[78] Ruben van Wendel de Joode. Managing Conflicts in Open Source Communities.

two previous approaches proposed to detect toxicity[79] and pushback[80] have recently been successfully applied to automatically detect interpersonal conflict.[81] Going beyond the detection of conflict situations, we need to explore possible triggers for conflicts and identify successful mitigation and conflict handling strategies. Successful conflict handling is more likely when people are, for example, highly cognitively flexible, can achieve a balance between a focus on the self and on others, and can regulate their emotions.[82] As cognitive flexibility can be trained,[83] we need to design interventions equipping software developers with individualized behavioral strategies that they might consider using when handling conflicts.

Finally, in many projects and companies, communication within software development teams is expected to adhere to codes of conduct. Codes of conduct are common in OSS projects and particularly in influential GitHub projects.[84] However, the application of a code of conduct is not trivial in practice: a controversial moderatory decision can cause a backlash[85] and a normative discussion, i.e., on what kind of behavior is acceptable or not rather than on the handling of the individual case. Moreover, the validity and relevance of codes of conduct are undermined from two opposite directions. On the one hand, some projects have created mock codes of conduct welcoming conflict and disagreement[86] or suggesting that adults do not even need codes of conduct in the first place.[87] On the other hand, GitHub has made it extremely simple for projects to adopt codes of conduct, reducing the value of a code of conduct as a

[79] Naveen Raman, Minxuan Cao, Yulia Tsvetkov, Christian Kästner, and Bogdan Vasilescu. 2020. Stress and Burnout in Open Source: Toward Finding, Understanding, and Mitigating Unhealthy Interactions. In: International Conference on Software Engineering, New Ideas and Emerging Results (ICSE). ACM, 57–60.

[80] Carolyn D Egelman, Emerson Murphy-Hill, Elizabeth Kammer, Margaret Morrow Hodges, Collin Green, Ciera Jaspan, and James Lin. 2020. Predicting developers' negative feelings about code review. In: 2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE). IEEE, 174–185.

[81] Huilian Sophie Qiu, Bogdan Vasilescu, Christian Kästner, Carolyn D. Egelman, Ciera Jaspan, Emerson R. Murphy-Hill: Detecting Interpersonal Conflict in Issues and Code Review: Cross Pollinating Open- and Closed-Source Approaches. ICSE-SEIS 2022: 41-55.

[82] Debra Gilin Oore, Michael P. Leiter, & Diane E. LeBlanc (2015). Individual and organizational factors promoting successful responses to workplace conflict. Canadian Psychology/Psychologie canadienne, 56(3), 301.

[83] Laurie R. Weingart, Jeanne M. Brett, Mara Olekalns, & Phillip L. Smith (2007). Conflicting social motives in negotiating groups. Journal of Personality and Social Psychology, 93, 994 –1010.

[84] Parastou Tourani, Bram Adams, Alexander Serebrenik: Code of conduct in open source projects. SANER 2017: 24-33

[85] Renee Li, Pavitthra Pandurangan, Hana Frluckaj, Laura Dabbish: Code of Conduct Conversations in Open Source Software Projects on Github. Proc. ACM Hum. Comput. Interact. 5(CSCW1): 1-31 (2021)

[86] https://www.kernel.org/doc/html/v4.10/process/code-of-conflict.html

[87] https://github.com/domgetter/NCoC

signal of inclusion. Ultimately, the simplicity of adopting codes of conduct might harm developers from minoritized groups. Hence, it is not clear whether codes of conduct succeed in creating the welcoming and inclusive environments that they aim for or rather contribute to further disenfranchising developers from minoritized groups.

In summary, we need to design ways to provide a more careful understanding of communication within software engineering teams. This understanding can benefit both community managers or project leaders and software developers themselves. The understanding should be supported by automatic tools as many projects are too big for a single person to keep an eye on.

HUMANS, BOTS AND MORE

Developers have been using tools to develop software for a long time. The number, variety and popularity of automated software development techniques are constantly increasing: while in the past, automation was applied to facilitate such menial tasks as system building,[88] recent years have seen automated tools take over non-trivial tasks such as analyzing programs,[89] reporting and repairing bugs[9091] and team and task management.[92] By now, one could argue that software developers no longer merely *use* automatic tools but *collaborate* with them as equal partners in the software development process.
The impact of automation on communication between developers is not always easy to predict.[93] For example, continuous integration tools for building systems and testing them reduce the amount of communication, saving up to one review

[88] Martin Fowler, "Continuous integration," http://martinfowler.com/articles/continuousIntegration.html 2006
[89] Ivan Beschastnikh, Mircea F. Lungu, and Yanyan Zhuang. 2017. Accelerating software engineering research adoption with analysis bots. In: Proceedings of the 39th International Conference on Software Engineering: New Ideas and Emerging Results Track (ICSE-NIER '17). IEEE Press, 35-38
[90] Simon Urli, Zhongxing Yu, Lionel Seinturier, Martin Monperrus. How to Design a Program Repair Bot? Insights from the Repairnator Project. ICSE 2018 - 40th International Conference on Software Engineering, Track Software Engineering in Practice (SEIP), May 2018, pp.95-104
[91] Marieli Wessel, Bruno Mendes de Souza, Igor Steinmacher, Igor S. Wiese, Ivanilton Polato, Ana Paula Chaves, and Macro Aurelio Gerosa. The Power of Bots: Understanding Bots in OSS Projects. CSCW 2018.
[92] Bin Lin, Alexey Zagalsky, Margaret-Anne Storey, and Alexander Serebrenik. 2016. Why Developers Are Slacking Off: Understanding How Software Teams Use Slack. In: Proceedings of the 19th ACM Conference on Computer Supported Cooperative Work and Social Computing Companion, 333-336
[93] Peng, Z. & Ma, X. (2019). Exploring how software developers work with mention bot in GitHub. CCF Transactions on Pervasive Computing and Interaction, 1(3), 190-203.

comment per pull request.[94] While one comment per pull request does not look like much, one should remember that the average pull request for the set of projects that was analyzed had a mean number of three general comments and 1.5 review comments. Therefore, the reduction in the number of comments is, on average, quite substantial.

Moreover, more and more automation is being implemented by means of bots. Bots communicate directly with developers, asking them to sign agreements and update their dependencies or even detecting toxicity in their communication. Few studies have focused on what kind of bot behavior is seen as undesirable e.g., that bots should not produce too many unnecessary or repetitive messages[95] and that their behavior should be congruent.[96] This calls for a more detailed investigation into what kind of interactions with bots would be acceptable or unacceptable for developers. We expect that bot behavior which is acceptable for one developer is not necessarily acceptable for another developer. Indeed, different developers already have different expectations when it comes to what kind of automation deserves to be called a bot: while some of them expect bots to be autonomous, others expect bots to communicate through a chat-like interface and yet others expect bots to be 'smart', i.e., able to perform non-trivial tasks.[97] Specifically, we would like to understand the expectations and experiences of developers from minoritized groups as chatbots might reproduce and exaggerate the behavior of developers from the majority group and hence produce a hostile and exclusive climate.

Furthermore, early results show that if bot behavior triggers an emotional response from developers, developers use the laughter emoji to indicate that the bot behavior is inappropriate.[98] This line of research should be extended as emotional responses are not limited to emojis. We also need to understand whether interactions with bots are likely to trigger negative emotions and problematic behavior, e.g., verbal abuse towards the bot. The latter issue is important as more

[94] Nathan Cassee, Bogdan Vasilescu, Alexander Serebrenik: The Silent Helper: The Impact of Continuous Integration on Code Reviews. SANER 2020: 423-434

[95] Mairieli Santos Wessel, Ahmad Abdellatif, Igor Wiese, Tayana Conte, Emad Shihab, Marco Aurélio Gerosa, Igor Steinmacher: Bots for Pull Requests: The Good, the Bad, and the Promising. ICSE 2022: 274-286

[96] Juan Carlos Farah, Basile Spaenlehauer, Xinyang Lu, Sandy Ingram and Denis Gillet. An Exploratory Study of Reactions to Bot Comments on GitHub. 4th International Workshop on Bots in Software Engineering (BotSE 2022), Pittsburgh, PA, USA, May 9, 2022.

[97] Linda Erlenhov, Francisco Gomes de Oliveira Neto, Philipp Leitner: An empirical study of bots in software development: characteristics and challenges from a practitioner's perspective. ESEC/SIGSOFT FSE 2020: 445-455

[98] Farah et al. Op. cit.

human-like bot personas might make bots more acceptable for developers but, at the same time, are also more likely to trigger verbal abuse.[99]

Finally, given the previous discussion of communication in software engineering teams, more research is needed to understand how bots can better support the detection of community smells, inappropriate or toxic behavior or violations of codes of conduct.

HOW DO TEAM COMPOSITIONS AND WAYS OF COLLABORATION AFFECT THE SOFTWARE SYSTEMS THAT TEAMS PRODUCE AND HOW ARE THEY AFFECTED BY THEM?

We have already mentioned that previous research has shown that gender-diverse teams perform better[100] and are more productive,[101] more effective and more coordinated[102] than non-gender-diverse ones and are less likely to exhibit community smells.[103] Palomba et al. have also related community smells to code smells, i.e., the suboptimal organization of communication to the suboptimal organization of the source code.[104] However, code smells are not the only code quality aspects that have been studied in the software evolution literature: further source code variables should be considered to obtain more refined insights into the relation between diversity and inclusion variables, variables of software processes (such as productivity or communication) and variables of source code. In our previous work, we have investigated variables related to readability[105] and

[99] Merel Keijsers, Christoph Bartneck, Friederike Eyssel. What's to bullying a bot? Correlates between chatbot humanlikeness and abuse. Interaction Studies, Volume 22, Issue 1, Sept 2021, p. 55-80

[100] Gila AR, Jaafa J, Omar M, Tunio MZ (2014) Impact of personality and gender diversity on software development teams' performance. In: 2014 International Conference on Computer, Communications, and Control Technology (I4CT). IEEE, pp 261–265

[101] Vasilescu B, Posnett D, Ray B, van den Brand MGJ, Serebrenik A, Devanbu P, Filkov V (2015) Gender and tenure diversity in GitHub teams. In: Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems, pp 3789–3798

[102] Marques M (2015) Software engineering education – Does gender matter in project results? – A Chilean case study. In: 2015 IEEE Frontiers in Education Conference (FIE). IEEE, pp 1–8

[103] Catolino G, Palomba F, Tamburri DA, Serebrenik A, Ferrucci F (2019) Gender diversity and women in software teams: How do they affect community smells? In: 2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Society (ICSE-SEIS), pp 11–20. IEEE

[104] Fabio Palomba, Damian Andrew Tamburri, Francesca Arcelli Fontana, Rocco Oliveto, Andy Zaidman, Alexander Serebrenik: Beyond Technical Aspects: How Do Community Smells Influence the Intensity of Code Smells? IEEE Trans. Software Eng. 47(1): 108-129 (2021)

[105] Valentina Piantadosi, Fabiana Fierro, Simone Scalabrino, Alexander Serebrenik, Rocco Oliveto: How does code readability change during software evolution? Empir. Softw. Eng. 25(6): 5374-5412 (2020)

self-admitted technical debt (SATD).[106] In particular, SATD (the indication that the code is not ready yet in the source code comments) shares the intuition of suboptimal organization with community smells and code smells. This is why we expect specific community smells to foster the introduction of SATD or to reduce the likelihood of the elimination of SATD.[107] Similarly, women have more difficulty comprehending code with single-letter identifiers than men[108], suggesting that readability might be more important for gender-diverse teams.

Team composition and collaboration do not only affect software but are also affected by it. For example, research software engineering projects are usually carried out by teams of researchers of different levels of seniority, from master's students to academic staff. However, none of these researchers are usually trained as software engineers; they are motivated by the results that can be obtained by using software rather than by the software itself and many of them leave the project merely a couple of months after joining it (bachelor's and master's students). This calls for the different organization of a team than in the case of traditional software engineering teams that are expected to be sustained for longer periods of time. Another example is provided by teams working at the National Aeronautics and Space Administration (NASA) in the United States of America, which build and operate planetary robotic spacecraft. These teams are conservative in their attitude towards technological innovation due to high reliability requirements[109] and consider their older and more experienced members as very valuable members of the team, respecting their understanding of what has worked in the past.[110] The latter forms a sharp contrast with software engineering in general, where experience does not seem to be valued to the same degree as being fast and keeping up with the latest trends[111] and older developers are considered less adaptive to change[112] and more expensive due to higher salaries.[113]

[106] Gianmarco Fucci, Nathan Cassee, Fiorella Zampetti, Nicole Novielli, Alexander Serebrenik, Massimiliano Di Penta: Waiting around or job half-done? Sentiment in self-admitted technical debt. MSR 2021: 403-414

[107] Fiorella Zampetti, Alexander Serebrenik, Massimiliano Di Penta: Was self-admitted technical debt removal a real removal?: an in-depth perspective. MSR 2018: 526-536

[108] Dawn J. Lawrie, Christopher Morrell, Henry Feild, David W. Binkley: Effective identifier names for comprehension and memory. Innov. Syst. Softw. Eng. 3(4): 303-318 (2007)

[109] Karl E Weick. 1987. Organizational culture as a source of high reliability. California Management Review 29, 2 (1987), 112–127.

[110] Weick, Op. cit.

[111] Sebastian Baltes, Stephan Diehl: Towards a theory of software development expertise. ESEC/SIGSOFT FSE 2018: 187-200

[112] Sebastian Baltes, George Park, Alexander Serebrenik: Is 40 the New 60? How Popular Media Portrays the Employability of Older Software Developers. IEEE Softw. 37(6): 26-31 (2020)

[113] A. Xia and B. H. Kleiner (2001). Discrimination in the computer industry. Equal Opportunities International, 20(5/6/7):117–120

FROM UNDERSTANDING TO SUPPORT

Similarly to the discussion of the human aspects of software engineering, the ultimate goal of our research on collaborative aspects is to move from understanding to support, i.e., to translate our insights obtained to guidelines or tools for projects interested in creating, maintaining or improving collaboration within their communities.

In this context, I would like to mention two very different projects that have recently been started, offering us an opportunity to bring our insights to practice. Together with Anita Sarma and her team from Oregon State University, we are working on creating a dashboard for the project management committees of several projects operating under the aegis of the Apache Software Foundation. The intention is to allow the project managers to get insights into the ways that their communities collaborate and communicate and pinpoint community smells, as well as issues and opportunities related to diversity and inclusion.

The second project is a collaboration with Maarten Hornikx. Maarten Hornikx is a professor of Acoustics in the Department of the Built Environment. In a recently granted project, Maarten's team is going to develop software to implement advanced acoustic techniques, while we are going to use this project as a living lab: an opportunity to contribute to create an inclusive and sustainable software community.

Challenges

Studying human and collaborative aspects is fraught with challenges.

First of all, social software engineering research makes extensive use of empirical methods and hence of empirical data. Social software engineering data comes in many different shapes and varies in quality. Indeed, software engineering data ranges from source code to natural language texts and from tutorial videos to biometric data. Even when one focuses solely on natural language texts, there is a wide variety of those texts: from source code comments reflecting SATD to questions on Stack Overflow platforms and from bugs reported on JIRA to code review comments on GitHub. As developers communicate differently on different platforms, e.g., express emotion differently on different platforms, tools designed for one of the platforms underperform when applied to data from another platform.[114] Moreover, as developers tend to be active on multiple platforms at the same time, understanding their communication and collaboration patterns requires combining data from different sources. Such a combination brings with it technical,[115] legal (e.g., due to the GDPR) and ethical (e.g., due to risks of deanonymization and outing) challenges. Legal and ethical challenges are particularly important when studying the experiences of minoritized groups as the diversity aspects identifying them, such as racial or ethnic origin or sexual orientation, are deemed to be sensitive in terms of the GDPR and involuntary disclosure can harm the individuals involved.

The second challenge is related to the adaptation of existing theories and techniques to the software engineering domain. While organizational psychology has provided many foundational theories that might be relevant to social software engineering, they have to be refined and operationalized to address the peculiarities of the software engineering domain, such as online work, male dominance in the field or the fact that modern software development transcends the boundaries of companies and countries.

[114] Nicole Novielli, Fabio Calefato, Filippo Lanubile, Alexander Serebrenik: Assessment of off-the-shelf SE-specific sentiment analysis tools: An extended replication study. Empir. Softw. Eng. 26(4): 77 (2021)
[115] Erik Kouters, Bogdan Vasilescu, Alexander Serebrenik, Mark G. J. van den Brand: Who's who in Gnome: Using LSA to merge software repository identities. ICSM 2012: 592-595

The third challenge is related to the contextualisation of the findings. Context is often king in computing[116], [117] and differences between software engineering products, processes and individuals involved in the call for contextualization. We have mentioned several examples of such differences above, e.g., differences between research software engineering and traditional software engineering or differences between teams in high-reliability organizations such as NASA as opposed to regular software engineering teams. Such contextualisation can be obtained through qualitative analysis techniques like grounded theory building,[118] ethnography[119] and participatory action research.[120] While those techniques have been applied in software engineering, their application is particularly challenging as they have been designed in a very different context (social sciences) and their application requires the researchers to bridge the conceptual gap between social sciences and software engineering.

Contextualisation is also related to the problem of transferring insights across different contexts: from the lab setting to the field and from open-source communities to closed-source context. We have recently confirmed the insights from the biometric study of emotions conducted in the lab[121] through a follow-up study of industrial software developers in their workplace (field),[122] but more such studies are needed.

The last group of challenges is related to integrating social software engineering in education. In adherence to the ideas of challenge-based learning, our bachelor's students often work in teams. However, these teams are usually composed by students themselves, sometimes by teachers, and inclusion is not usually taken

[116] Carmine Vassallo, Sebastiano Panichella, Fabio Palomba, Sebastian Proksch, Andy Zaidman, Harald C. Gall: Context is king: The developer perspective on the usage of static analysis tools. SANER 2018: 38-49

[117] Robert M. Davison, Maris G. Martinsons: Context is king! Considering particularism in research design and reporting. J. Inf. Technol. 31(3): 241-249 (2016) See also follow-up texts to this article: Cathy Urquhart: Response to Davison and Martinsons: Context is king! Yes and no - it's still all about theory (building). J. Inf. Technol. 31(3): 254-256 (2016); Walter D. Fernández: Commentary on Davison and Martinsons' 'Context is King! Considering particularism in research design and reporting'. J. Inf. Technol. 31(3): 265-266 (2016) as well as Zhi (Aaron) Cheng, Angelika Dimoka, Paul A. Pavlou: Context may be King, but generalizability is the Emperor! J. Inf. Technol. 31(3): 257-264 (2016)

[118] Rashina Hoda Socio-Technical Grounded Theory for Software Engineering. IEEE Transactions on Software Engineering 2021.

[119] Helen Sharp, Yvonne Dittrich, Cleidson R. B. de Souza: The Role of Ethnographic Studies in Empirical Software Engineering. IEEE Trans. Software Eng. 42(8): 786-804 (2016)

[120] Miroslaw Staron. Action Research in Software Engineering. Springer Verlag, 978-3-030-32610-4

[121] Daniela Girardi, Nicole Novielli, Davide Fucci, Filippo Lanubile: Recognizing developers' emotions while programming. ICSE 2020: 666-677

[122] Daniela Girardi, Filippo Lanubile, Nicole Novielli, Alexander Serebrenik. Emotions and Perceived Paroductivity of Software Developers at the Workplace IEEE Transactions on Software Engineering, 2021

into consideration. The only way for teachers to understand how the team is communicating and collaborating is through individual interviews and anonymous peer reviews. However, these methods are known to be subjective; moreover, women might feel less confident rating their peers than men,[123] rendering their voices less heard. More careful ways of assessing team collaboration and communication can limit the impact of subjectivity and lack of confidence on the assessment results.

[123] Fitzpatrick, C. (1999). 'Students as evaluators in practicum: Examining peer/self assessment and self-efficacy', Presented at the National Conference of the Association for Counselor Education and Supervision, New Orleans as cited in Wen, L. M., & Tsai, C. C. (2006). University students' perceptions of and attitudes toward (online) peer assessment. Higher Education, 51(1), 27–44.

Conclusions

As software is omnipresent in our society and both influences it and is influenced by it, it is our societal and scientific responsibility to contribute to better software for a better society. For me, 'better' means 'more inclusive', 'more welcoming' and 'fairer' and the way that I believe one can create such software is by creating a more inclusive environment for software developers themselves. We need to understand the experiences of developers from minoritized groups and practices of collaboration and communication. Based on this understanding, we can design mechanisms to support individual developers and their teams.

I have spoken.

Curriculum Vitae

Prof. Alexander Serebrenik was appointed as full professor of Social Software Engineering in the Department of Mathematics and Computer Science at Eindhoven University of Technology (TU/e) on July 1, 2020.

Alexander Serebrenik (1975) received his bachelor's (1995) and master's degrees (1999) from the Hebrew University, Jerusalem, Israel, followed by a PhD (2003) from the Katholieke Universiteit Leuven, Belgium (2003). After a postdoctoral stay at École Polytechnique, France, he joined Eindhoven University of Technology as an assistant (2004) and associate professor (2013). His research goal is to facilitate the evolution of software by taking into account the social aspects of software development. His work tends to involve theories and methods both from within computer science (e.g., the theory of socio-technical coordination; methods from natural language processing and machine learning) and from outside of computer science (e.g., organizational psychology). The underlying idea of his work is empiricism, i.e., that solutions to software engineering challenges should be grounded in observation and experimentation and require a combination of social and technical perspectives. Alexander has co-authored a book, 'Evolving Software Systems' (Springer Verlag, 2014), and more than 200 scientific papers and articles. He is actively involved in the organization of scientific conferences and is a member of the editorial board of several journals. He has won multiple best paper and distinguished reviewer awards.

Colophon

Production
Communication Expertise Center

Cover photography
Bart van Overbeeke Photography, Eindhoven

Design
Grefo Prepress, Eindhoven

Digital version:
www.tue.nl/lectures/

Visiting address
Building 1, Auditorium
Groene Loper, Eindhoven
The Netherlands

Navigation
De Zaale, Eindhoven

Postal address
PO Box 513
5600 MB Eindhoven
The Netherlands
Tel. +31 (0)40 247 9111
www.tue.nl/map

TU/e
EINDHOVEN
UNIVERSITY OF
TECHNOLOGY