Naming the Pain in Developing Scientific Software

Igor Wiese, *Federal University of Technology - Paraná*

Ivanilton Polato, *Federal University of Technology - Paraná*

Gustavo Pinto, *Federal University of Pará, Brazil*

ABSTRACT. The scientific software community is eagerly embracing software development practices and tools. However, the lack of Computer Science background, in general, and Software Engineering training, in particular, pose a toll on scientists that need to develop software. Although some of the problems that scientific software developers face when developing scientific software are already known in the literature, we believe that a comprehensive characterization of this complex landscape is still missing. In this paper we build a taxonomy of 2,110 problems (the so called "pains") reported by 1,577 scientific software developers. These problems are grouped into three major axes: technical-related, social-related, and scientific-related problems. Our report helps to better understand the needs and challenges of this so far not fully understood community.

KEYWORDS. Scientific Software Development; Pain points; Survey.

1. Introduction

In the big data era, in which samples of data can easily scale to many orders of magnitude, manual data analysis is rather unfeasible. Sophisticated data analysis, therefore, depends on chains of computer programs that generate, clean up, augment, plot, and run statistical models on data. The need of specialized software tooling is imperative to scientists conduct their work. However, efficient, scalable, and reliable software solutions for scientific research are challenged by (1) the frequent and unforeseen changes in requirements, (2) the need for both highly specialized domain knowledge and programming expertise, and (3) the extremely complex patterns of communication on perpetually novel architectures[1]. In addition, such specialized tooling, if existing, is often hidden in research laboratories and universities. Research prototypes that usually serve only the purpose of validating an idea are not uncommon as well.

Although the goal of scientists is to do science, not to develop software[1,2,3], due to the lack of specialized tools, many scientists have no option other than acquiring software development skills and develop their own software. However, many challenges might be hidden in this activity. For instance, since most scientific software packages begin as small projects intended for internal use by a research team, one challenge resides when moving from a one-person project into a production-ready software intended for general

use[2]. Another challenge is related to the need of getting acquainted with the software engineering arsenal (e.g., process, practices, techniques and tools) required to build software.

While previous work shed some light on how scientists develop software[3], given the fundamental changes in the software development practices in the last few years (e.g., the introduction of social coding websites and the prevalence of online learning platforms), little is known about the current state-of-the-practice of scientific software development. In a recent replication study[4], we deployed an online survey to 1,577 R Developers that have published at least one R package. We used this population of R Developers as a proxy for scientific software developers. As we shall see in Section 2, our respondents work on a myriad of research fields, including chemistry, physics, and biology. In our questionnaire, we asked them questions about how they learn developing scientific software, or how familiar are them with standard concepts of software engineering. The complete list of questions as well as the answers are available online at: https://github.com/utfpr/NamingThePain.

One question that emerged from our survey that was not fully explored is related to the main problems (or the so called "pains") that scientific software developers face when developing scientific software. Although our study shed some initial light along this direction (e.g., we found that cross-platform compatibility, poor software documentation, and lack of formal reward system play a role), we argue that this question deserves an in-depth investigation, in particular, due to the unique challenges that pertain to this community, such as the lack of software development expertise, the inherent complexity of the domain, and the (current) intrinsic presence of social platforms.

Inspired by the naming the pain initiative[5], we conducted a comprehensive qualitative study built upon selected results from our previous survey[4]. This paper goes beyond the existing literature[3,6,7] by (1) providing an extensive categorization of the problems (e.g., some problems are known but the big picture was unknown) and, more importantly, (2) quantifying the phenomenon (e.g., it is not clear which technical problems are the most frequent ones). Our taxonomy is presented throughout Section 3. After describing the pains, we discuss potential remedies for them at Section 4. We also relate this work to relevant related work at Section 5.

2. Who is the Scientific Software Developer?

In a previous work we conducted a survey with 1,577 scientists that develop R packages[4]. This sample corresponds to 25% of the R developers that have at least one software package published at CRAN. Although one might argue that not all R developers develop scientific software, in our invitation email, we kindly ask the participants that do not consider themselves as scientists not to answer the questionnaire.

Among the 1,577 respondents, 88% are male, 45% have between 30–40 years (24% have between 18–30, 4% have over 60), and 49% are in Europe (34% in North America, 6% in Asia, 6% in South America, 4% in Oceania, 0.5% in Africa, and 0.5% in Central America). Regarding their academic degrees, 80% are working towards (or have already received) their PhD (15% have a master's degree). Moreover, 64% of the respondents are academic researchers (e.g., professors or postdocs), 20% are software engineers, 15% are

graduate students, 10% are lecturers, 9% are industrial research scientists (whereas another 8% are government research scientists). They work in a variety of fields: Agriculture (0.6%), Biology (8.2%), Biochemistry (0.2%), Bioinformatics (2.1%), Biostatistics (3.7%), Chemistry (1.7%), Computer Science (8.6%), Data Science (1.6%), Ecology (7.7%), Economy (4.6%), Education (1.0%), Electrical Engineering (0.6%), Engineering (1.7%), Environmental Sciences (1.2%), Genetics (2.1%), Geography (3.9%), Linguistics (1.0%), Mathematics (6.6%), MBA (0.1%), Mechanical Engineering (0.3%), Medical (4.4%), Neuro and Cognitive Science (0.7%), Physics (3.0%), Political Science (1.6%), Psychology (3.8%), Science (0.5%), Sociology (1.3%), Statistics (23.7%), Zoology (0.6%), and Other (1.6%).

When it comes to the presence in social coding platforms, we found that 45% of them have GitHub profiles. According to popularity metrics, their packages are not very popular; the median values for the number of contributors, the number of stars, and the number of forks is, respectively, 2, 2, and 1. This does not mean, however, that they are not used. Indeed, most of these packages could be downloaded and installed through CRAN (R package manager)[7].

3. Naming the pain

In our survey, we asked our participants questions about their education, their work habits, their needs and challenges. In this paper, we focus our attention in one question that was not properly explored: "*What are the three most pressing problems, challenges, issues, irritations, or other "pain points" you encounter when developing scientific software for your own research or for others?*". This was an optional open question, and as the question title might indicate, more than one problem could be reported per user. A total of 1,121 respondents answered this question, and a total of 2,110 pains were reported.

We coded all responses to identify patterns (e.g., similarities and differences) that describe the phenomenon under study. The coding process was conducted by two authors that analyzed each response individually. The analysis was followed by conflict resolution meetings to reach consensus. We used coding guidelines provided by Merriam[8] and Seaman[9] to code, categorize, and synthesize data. After this procedure, we observed that the main categories of problems are related to three broad groups: (1) technical-related problems, (2) social-related problems, and (3) scientific-related problems. We provide discussions accordingly. Throughout the following sections, we chose quoted the most representative answers. Still, although some quotes have typos, we decide to keep them in order to make the answers more trustworthy.

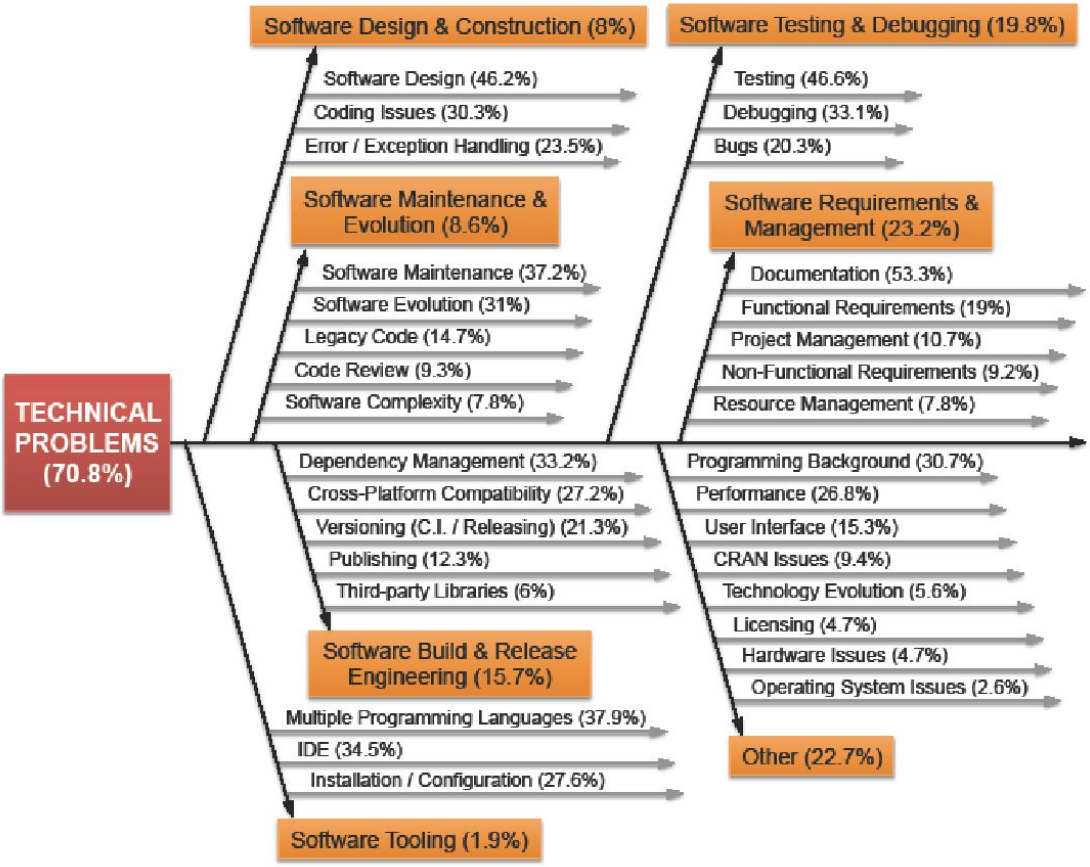
3.1 Technical-related problems

Technical-related problems are by far the most common problems that our scientific software developers face. As Figure 1 shows, technical-related problems represent more than 70% of the overall set of problems. The percentages represent the share of respondents that mentioned each pain.

According to our respondents, **Software Requirements and Management** is the software engineering discipline that most hurts scientific developers (they account for

23% of the technical problems reported). Problems related to the evolution of **Functional Requirements** (e.g., "*The objective changes after having worked on it for months*") was cited by many respondents. Moreover, in spite of its importance, we perceived that writing a good **Documentation** also poses a major issue. For instance, one respondent mentioned that "*I work with biologists that have little knowledge of programming. Therefore, the software must be easy to use. Write a clear documentation is always a challenge*". Documentation issues are 8.7% of the total problems raised by the respondents, considering the technical, social and scientific-related problems group.

FIGURE 1. Technical-related problems



Scientific developers also face a hard time when conducting **Software Testing and Debugging** activities, which represents more than 14% of the total pains presented. This is particularly relevant due to the intrinsic non-deterministic nature of exploring research questions, as one respondent mentioned: "*It's frequently difficult to test scientific software, since you might not even know in advance what the answer should be*". As a consequence, software development best practices such as Test-Driven Development are much harder to be introduced. Despite this grin scenario, a well-tested scientific software seems to be crucial for their scientific work, as one respondent revealed: "*testing have to be rigorous to publish the work in a scientific journal*". Overall, problems related to

Software Requirements & Management and **Software Testing & Debugging** account for more than 30% of the total problems reported by our respondents.

Participants in our survey also pointed several issues in the **Software Building and Release Engineering** category. In particular, **Dependency Management** appeared as one third of the pains in this group. Developers complain about the complexity to keep up with package evolution ("*Software become obsolete quickly in research. Managing dependencies on other software can be difficult due to poor packaging*"; "*Hard-to-install dependencies, obscure packages that are related to what one is doing but were last updated many years ago and now won't just work out-of-the-box*"). These issues affect directly the software development, and although out of the developers' control, broken packages and dependencies bring frustration to the process. Such issues are closely related to the poor use of version control and the absence of proper documentation during development.

Finally, the **Other** category groups pains associated with, for instance, the lack of programming background (e.g., "*not having a software development background (i.e.: learning on the fly)*"), the algorithms' performance (e.g., "*optimizing code that is not running fast enough (pure engineering issue)*"), and the user experience (e.g., "*anticipating user behavior and designing [a good] user interface*"). Note that, problems regarding **Programming Background** and software **Performance** represent more than 9% of the total problems cited.

Although it is out of the scope of this work to understand why some problems are more frequently reported than other, this overwhelming presence of technical problems could be in part justified by the low number of respondents with Computer Science background (around 3%). More research is needed to understand the impact of the lack of CS background might have on how scientific developers face daily software engineering challenges.

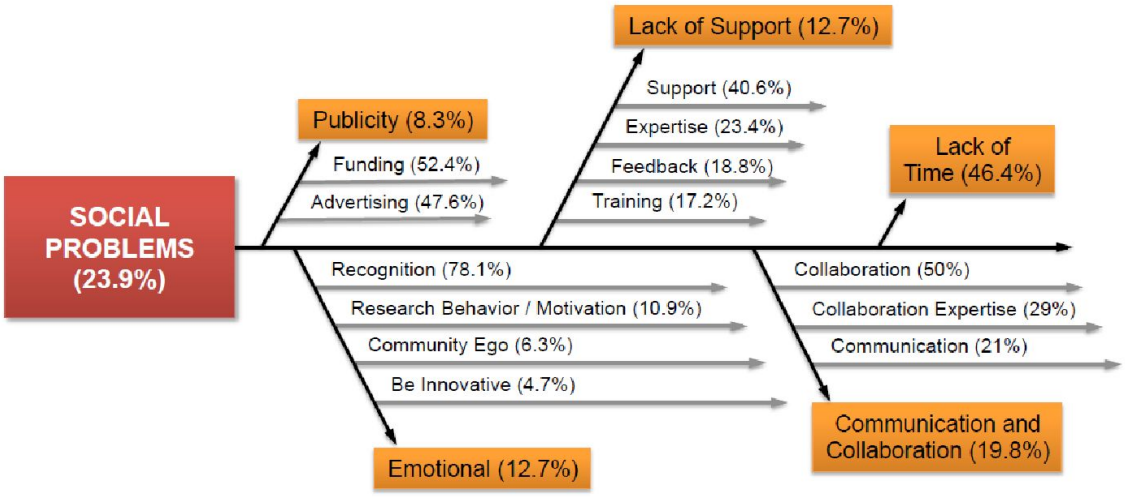
3.2 Social-related problems

Social-related problems are well-known from developers[10]. These problems are no exception in the scientific software development domain. As Figure 2 shows, social-related problems are responsible for 23.9% of the associated pains. First and foremost, **Lack of time** was the most reported pain (e.g., "*time to maintain all the packages that I've written and keep them up-to-date*"). Developers' lack of time represents more than 11% of the total problems presented in our survey. Similar to the **Lack of time**, pains related to **Publicity** were also mentioned. As an example, one respondent mentioned that "*publishing norms make it hard to get the same credit for writing a software package that you would for a publication. Software packages are often not cited by researchers using them*", even though initiatives exist, such as the Journal of Open Source Software. We also found that **Emotional** aspects, such as the lack of recognition, play a role. Our respondents shared the impression that "*the development of a scientific software is not recognized as a scientific achievement*", which might hinder the long-term contribution to scientific software projects.

Still, our respondents perceived some **Communication and Collaboration** pains. One respondent summarized this pain as "*the absence of a collaborative scientific*

programming community (e.g., for hackathons, co-teaching/co-learning, etc.)", evidencing an ample opportunity for fostering the development of scientific software. Finally, we observed pains related to the **Lack of support**: for instance, one respondent said that there is a "*Lack of feedback because the user base is so small*". Along these lines, the complexity to use the software was also noted as a pain. Some respondents also mentioned that users do not always have the expertise required to install or use the software (e.g., "*Installation and compatibility of software parts and keeping up to date with their recent versions*"). This particular finding corroborates on the importance of writing good documentation for scientific software projects.

FIGURE 2. Social-related problems



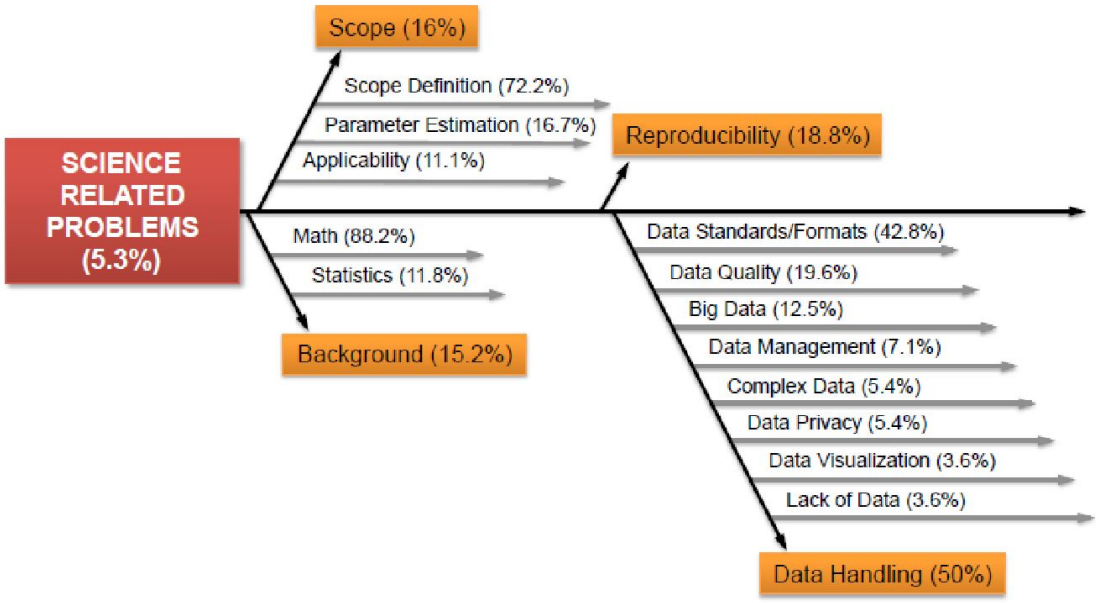
3.2 Scientific-related problems

Figure 3 presents the scientific-related problems. The problems mentioned in this category are responsible for 5.3% of the pains described by scientific developers. In the center of scientific-related problems there are **Data Handling** pains. For instance, one respondent mentioned that "*poorly implemented community data standards (overly complex) and/or arbitrary design standards*" makes the data handling process painful. Other **Data Handling** issues were found, such as Data quality, Data management, Big Data, and Data privacy. One of the respondents argued that "*dealing with incomplete data, finding optimal default parameters, handling large datasets*" and "*the difficulty to share data*" are also difficulties that scientific software developers suffer.

Moreover, although scientists have been advocating in favor of **Reproducibility** along the last decades, we observed that many respondents still perceive it as a common pain both on experiments and on source code. For instance, one respondent mentioned that "*colleagues write outrageous code and do not care about the reproducibility of their results*", which could introduce some threats in their research. Another pain point are related to the difficulty to define the correct **Scope** (e.g., "*figure out what exactly the*

scope of the software should be") and the mismatch between **Background** skills (e.g., "*mathematical methods to be implemented*") and coding skills that are needed to conduct scientific work.

FIGURE 3. Scientific-related problems



4. Remedying the pain

The problems reported in the last sections would require an extensive, collaborative, and long term effort to remedy them. Since finding solutions to these problems is not the focus of this paper, we dedicate this section to inform the reader about significant work done along these lines.

Generally speaking, there are two guidelines that might help scientific developers to remedy those pains[11,12]. First, Wilson and colleagues[11] described a set of 24 recommendations linked to advanced practices at Software Carpentry (http://software-carpentry.org). Among the recommendations, there are (1) adopting a version control system, (2) turning bugs into test cases, or (3) making code style and formatting consistent. The authors suggest that the effort of implementing such recommendations is almost immediately offset by the productivity gains in productivity of the programmers involved. However, most of these recommendations are related to **Technical problems**; more attention should be given to **Social problems** and **Scientific problems**. Moreover, some well-known software development best practices do not properly fit in the scientific software domain. This is particularly the case of techniques such as test coverage and continuous integration, which require the program under development to have deterministic outputs. Other best practices such as code reviews, pair programming, or performance tuning are also challenging to translate to the scientific domain.

More recently, Wilson et al.[12] outlined a series of practices for scientific computing based on their collective experience. In this work the authors included a specific section

for improving their scientific collaboration. Among the strategies, the authors suggest to (1) create a "TODO-LIST" using an issue tracker system, (2) create an overview description about the project, or (3) highlight the importance of publicize the scientific project to attract new collaborators. These suggestions might be useful to remedy **Social problems** such as the ones grouped under Communication and Collaboration. Regarding the **Lack of Time** pain, Wilson and colleagues[11,12] suggest that improving scientists productivity — through training or following the set of recommendations — can alleviate this pain, since the research can become more reproducible and, consequently, more easily used by others.

Despite these important recommendations, according to our respondents, there are still gaps that ought to be better investigated. For instance, since Handling Data is the most common **Scientific problem**, one might expect collaborative programming environments (for non-experienced programmers) that could easily import/export/visualize data becoming widespread (which unfortunately is not yet the case). Similarly, since Reproducibility is still a recurring pain, existing collaborative programming environments could provide a scaffold for reproducibility, for instance, exporting build files in a human readable fashion or using state to the practice tools such as docker.

5. Relating the Findings

Although related work sheds an initial light on software development pains that affect scientists, they (1) do not provide a comprehensive categorization of the problems that scientific developers face (e.g., even though some problems are known, the big picture is unknown) and (2) do not quantify the phenomenon (e.g., it is not clear which technical problems were the most frequent ones). To better place this work in relation with the replication previously published[4] and the original study conducted in 2009[3], the main results of both analyses are summarized at Table 1.

As we can see, this work not only revisited traditional issues such as reproducibility, but also piled evidence of previous findings (e.g., major challenges with software testing were discussed by both Hannay[3] and Pinto[4]), and still highlighted additional pain points that were previously unknown in the scientifics software development arena (e.g., emotional issues such as Ego and Recognition). Moreover, both social and scientific pain points (which together correspond to about 30% of the named pains) were not well addressed previously. Based on our findings, we expect more research on the social side of scientific software development.

TABLE 1. Relating the Findings

Problems	This Paper	Replication paper[5]	Original paper[3]
Technical problems	A total of seven groups of problems (with 32 sub-problems) were found. Software Requirements & Management were the most common ones.	Four technical problems were presented: the "Cross-platform compatibility", "Poor software documentation", "Scope bloat", "Mismatch	Only software testing and verification were mentioned as challenging for scientific software developers.

	Documentation, in particular, was perceived as the most painful software development activity. Software testing and debugging next. Software build and release engineering also.	between coding skills and subject-matter skills". The lack of technical skills was also observed.	
Social problems	A total of five groups of problems (with 13 sub-problems) were found. Problems with time, collaboration, and recognition appears more frequently. The lack of training, nevertheless, was not frequently reported.	Five social problems were presented: the "Hard to collaborate on software projects" pain, the "aloneness" pain, the "Interruptions while coding" pain, the "Lack of time", and the "Lack of user feedback.	No mention.
Scientific problems	A total of four groups of problems (with 13 sub-problems) were found. Problems with data (e.g., management, privacy, visualization, etc.) are by far the most common ones. Defining the scope as well as reproducibility issues are also common.	Only one scientific problem was mentioned: the "Lack of formal reward system" one.	No mention.

References

1. J. C Carver. Software engineering for Science. Computing in Science & Engineering, 18(2):4–5, 2016.
2. V. R. Basili, J. C. Carver, D. Cruzes, L. M. Hochstein, J. K. Hollingsworth, F. Shull, and M. V. Zelkowitz. Understanding the high performance-computing community: A software engineer's perspective. IEEE Softw., 25(4):29–36, July 2008.
3. J. E. Hannay, C. MacLeod, J. Singer, H. P. Langtangen, D. Pfahl, and G. Wilson. How do scientists develop and use scientific software? In Workshop on Software Engineering for Computational Science and Engineering, SECSE '09, 2009.
4. G. Pinto, I. Wiese, and L. F. Dias. How do scientists develop scientific software? an external replication. In International Conference on Software Analysis, Evolution and Reengineering, SANER, 2018.
5. D. M. Fernandez, S. Wagner, M. Kalinowski, A. Schekelmann, A. Tuzcu, T. Conte, R. Spinola, and R. Prikladnicki. Naming the pain in requirements engineering: Comparing practices in brazil and germany. IEEE Software, 32(5):16–23, Sept 2015.
6. J. C Carver. Towards reporting guidelines for experimental replications: A proposal. In

Proceedings of the 1st International Workshop on Replication in Empirical Software Engineering Research (RESER), 2011.

7. R. Meloca, G. Pinto, L. Baiser, M. Mattos, I. Polato, I. Scaliante Wiese, and D. M. German. Understanding the usage, impact, and adoption of non-osi approved licenses. In International Conference on Mining Software Repositories, MSR, 2018
8. S.B. Merriam. Qualitative Research: A Guide to Design and Implementation. Higher and adult education series. John Wiley & Sons, 2009.
9. C. B. Seaman. Qualitative methods in empirical studies of software engineering. IEEE Transactions on Software Engineering, 25(4):557–572, Jul 1999.
10. D. A. Tamburri, P. Lago, and H. van Vliet. Organizational social structures for software engineering. ACM Computing Survey, 46(1):3:1–3:35, July 2013.
11. G. Wilson, D. A. Aruliah, C. T. Brown, N. P. Chue Hong, M. Davis, R. T. Guy, S. H. D. Haddock, K. D. Huff, I. M. Mitchell, M. D. Plumbley, B. Waugh, E. P. White, and P. Wilson. Best practices for scientific computing. PLOS Biology, 12(1):1–7, 01 2014.
12. G. Wilson, J. Bryan, K. Cranston, . Kitzes, L. Nederbragt, and T. K. Teal. Good enough practices in scientific computing. PLOS Computational Biology, 13(6):1–20, 06 2017.

About the authors

Igor Wiese is an associate professor in the Department of Computing at the Federal University of Technology -- Paraná, Brazil, where he researches Mining Software Repositories techniques, Human Aspects of Software Engineering and related topics. Wiese received a PhD in computer science from the University of São Paulo. Contact him at: igor@utfpr.edu.br.

Ivanilton Polato is an associate professor in the Department of Computing at the Federal University of Technology -- Paraná, Brazil. His research topics are related to Energy Consumption, Green Computing, Open Source Software and Software Repositories. Polato received a PhD in computer science from the University of São Paulo. Contact him at: ipolato@utfpr.edu.br.

Gustavo Pinto is an assistant professor of computer science at the Federal University of Pará, Brazil. His research focuses on the interactions between people and code, spanning the areas of software engineering and programming languages. Pinto received a PhD from Federal University of Pernambuco, Brazil. Contact him at: mail@gustavopinto.org