How Do Scientists Develop Scientific Software? An External Replication

Gustavo Pinto
UFPA
Belém, Brazil
gpinto@ufpa.br

Igor Wiese
UTFPR
Campo Mourão, Brazil
igorf@utfpr.edu.br

Luiz Felipe Dias
USP
São Paulo, Brazil
luizdias@alunos.utfpr.edu.br

***Abstract*—Although the goal of scientists is to do science, not to develop software, many scientists have extended their roles to include software development to their skills. However, since scientists have different background, it remains unclear how do they perceive software engineering practices or how do they acquire software engineering knowledge. In this paper we conducted an external replication of one influential 10 years paper about how scientists develop and use scientific software. In particular, we employed the same method (an on-line questionnaire) in a different population (R developers). When analyzing the more than 1,574 responses received, enriched with data gathered from their GitHub repositories, we correlated our findings with the original study. We found that the results were consistent in many ways, including: (1) scientists that develop software work mostly alone, (2) they decide themselves what they want to work on next, and (3) most of what they learnt came from self-study, rather than a formal education. However, we also uncover new facts, such as: some of the "pain points" regarding software development are not related to technical activities (e.g., interruptions, lack of collaborators, and lack of a reward system play a role). Our replication can help researchers, practitioners, and educators to better focus their efforts on topics that are important to the scientific community that develops software.**

I. Introduction

Data are the cornerstone data science. To study and analyze such data, in the big data era, (data) scientists have to rely on robust yet tailored software tools to help them uncover new facts or refute established beliefs, providing guidance for leaders to take action. However, due to the myriad of specific problems that scientists usually work on [1], it is not uncommon the need to create their own tooling, since tool (either proprietary or open source) tailored to deal with their particular needs might not even exist.

Although the goal of scientists is to do science, not create software [2], scientists have no other option other than develop software to help them to conduct their work. However, scientists often come from diverse backgrounds and frequently do not have much, if any, formal training in computer science, in general, or software engineering, in particular [3]. As a consequence, the resulting scientific software usually serves only the purpose of validating an idea; i.e., it does not exhibit maturity, nor the breadth of scope necessary for use in real software development.

To shed some initial light on how scientists develop software, the work conducted by Hannay and colleagues [3] presented a comprehensive overview of the state of the practice of scientific software development. This study surveyed about 2,000 scientists and, among the findings, the authors observed that the lack of software engineering knowledge is one of the main pain points when developing scientific software. This study was published at the Software Engineering for Computational Science and Engineering (SECSE) [4], a workshop held in conjunction with ICSE'2009.

Why this replication is needed? As of 2017, the paper of Hannay *et al* [3] became very influential for the scientific community, in general, and for scientists that develop software, in particular. Unfortunately, after about 10 years of its initial publication, no replication was conducted. 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 on-line learning platforms) and unique challenges of scientific research (e.g., the frequent and unforeseen changes in requirements and the need for both highly specialized domain knowledge and programming expertise [5]), little is known about the current state-of-the-practice of scientific software development.

The goal of this paper is to update the understanding about how scientists develop scientific software. To achieve this goal, we present an external replication of the original study[1]. In the context of Software Engineering, a replication consists of repeating an original experiment by reusing its materials, experiment design, or analysis procedures [6]. A replication can be either internal (conducted by the same research group) or external (by other groups in other context) [7].

Although original paper was focused on both developing and using scientific software, in this replication we focused only on how developers develop software (we discuss our rationale for not including the questions related to how they use software in Section III-A). Similar to the original study, we collected scientists' perception through an on-line survey (details about our methodology at Section IV). We reused the list of questions of the original study (the original authors gently made their survey available). Because of our focus on how scientists develop scientific software, we removed the majority of the survey's questions related to how scientists scientific use software.

[1]Throughout this paper, we refer to the study of Hannay *et al* [3], as the "original study".

The target of our survey is R scientists that developed and published at least one software package at CRAN (The Comprehensive R Archive Network), the most well-known R package manager (details on why R developers at Section II). To avoid R developers that do not consider themselves are scientists, in our invitation email we asked them to not answer the questionnaire. Our survey received 1,574 responses from R developers all over the world. Still, 603 respondents left the GitHub address of the projects they maintain. We take advantage of information from these repositories to complement the findings from our survey.

II. R and CRAN

R is a multi-paradigm, dynamic typed programming language introduced in 1993. It is open-source, released under GPL license. CRAN is the most well-known package manager that manages and indexes R packages. CRAN distributes both the source and binary code of packages published on it. As of October 2017, CRAN contains over 11K R software packages for all kind of specialized purposes. Anyone interested in submitting her R package to CRAN should follow the guidelines available at: `https://cran.r-project.org/submit.html`. The CRAN team carefully analyzes each submission, looking for issues related to (1) the software license/copyright, (2) the package dependencies, or (3) the organization of the binary and source code packages [8].

For this study, we consider any developer that has published at least one R package through CRAN as a scientist. We chose to focus only on the R programming language because it this programming language was curated by an statistician for statistical computations, therefore, anyone interested in data analysis (a domain well-explored by data scientists) is a suitable R user. Still, different than other traditional programming languages, the R programming language is well-used in a variety of fields, including Statistics, Economics, Biology, Psychology, and many others, for research purposes. The software engineering field is particularly well-supported, with textbooks [9], general guidelines, and workshops[2]. As a consequence, about 1% of the articles indexed by Elsevier's Scopus cites R or one of its packages [10].

III. The Original Study

According to Carver [7], at a minimum, a replication paper should report the following information about the original study: Research questions, Design, Participants, Artifacts, context variable, and summary of the results. We discuss each one of them next.

A. Research Questions

The original paper had the following research questions:

RQ1. How did scientists learn what they know about developing/using scientific software?
RQ2. When did scientists learn what they know about developing/using scientific software?
RQ3. How important is developing scientific software to scientists?
RQ4. How much of their working time do scientists spend on developing scientific software?
RQ5. Do scientists spend more time developing/using scientific software than in the past?
RQ6. On what scale of hardware do scientists develop/use scientific software?
RQ7. What are the sizes of the user communities of scientific software?
RQ8. How familiar are scientists with standard concepts of software engineering?
RQ9. Does program size, time spent on programming, or team size influence scientists' opinions about the importance of good software development practices?

Since the original study was aimed at both understanding how scientists develop and use software, the RQ1–6 provided such guidance. In this replication paper, we decided not to replicate the research questions related to "using software". As of 2017, we believe that the majority of scientists use software for research purposes, and the way they use software is greatly influenced by the kind of research they are conducting (as already found by the authors). Therefore, due to our diverse sample, and the plethora of scientific software readily available nowadays, we believe that the questions related to how scientists use software could be better answered by field studies or ethnographic studies; we postponed this replication to another study.

We also removed RQ6, which is focused on hardware platforms. We removed this research question because, as of 2017, there is a number of hardware platforms that were not conceived (from smartphones, smartwatches, and smartglasses, to personal computers, mainframes, and drones) when the original study was published. Therefore, the understanding of these platforms and therefore the comparison with the original study, considering this complex landscape, was left for future work.

Finally, instead of asking the respondents their perception about the size of the user community (RQ7), we asked scientists to mention where they maintain the source code of their software. Therefore, we mined these repositories to gather data to answer this RQ. When possible, we also enrich the answers to the other research questions with data from these repositories.

B. Design

The original study conducted an on-line survey. They asked 36 questions (9 open questions). The survey was advertised through mailing lists, bulletin boards, word of mouth, and with advertisements in both the on-line and print editions of American Scientist magazine. Due to the nature of these channels, the authors were not able to estimate how many users received the survey invitation (and therefore calculate the response rate). The authors did not mention how long did they wait for the answers.

[2]https://github.com/Derek-Jones/ESEUR-workshop

C. Participants

The original study was answered by 1,972 participants. Among them, most of the respondents came from Europe and North America (725 and 715 respondents, respectively), followed by Australia/New Zealand and Asia (66 and 57 respondents, respectively). South America, Central America, and Africa had below 50 respondents each. About two-thirds of the respondents stated that their highest academic degree is a Ph.D. (or equivalent). Moreover, about 50% of the respondents are academic researchers (professors, post-docs, or similar).

D. Artifacts

Since the original study conducted an on-line survey, the main artifact used was the questionnaire. As we aforementioned, the original authors shared the questionnaire with us (without the responses, however). Only minor modifications were made. We asked 28 questions.

E. Context variables

Drawing general conclusions from empirical studies is difficult because any process depends on a potentially large number of relevant context variables [11]. The original study employed an on-line survey, advertised through different channels (details at Section III-B). We employed the same method, but targeting a more focused community: scientific developers that published at least one R package on the CRAN repository. As a consequence, the demographics from both studies are different in terms of location, number of respondents, software engineering background. Similar to the original study, we also cannot assume that the results generalize to other software-intensive communities.

F. Summary of the results

The summary of the results (from both the original and the replications study) are available at Table I.

IV. The Replication Survey

Similarly to Hannay *et al.* [3], in order to investigate how scientists develop scientific software, we conducted an on-line survey. Our target population are scientists that were the main author of at least one R software package. Therefore, for the purposes of this study, any developer who authored an R package was considered a scientist. We search for R packages in CRAN, the most well-known package manager for R-based packages.

A. Subjects

As of November 2017, in CRAN there are over 11k packages registered. We downloaded all these packages and extracted metadata related to author's information. We found that ∼5k R scientists authored more than one package. To those scientists, we decided not to to send email twice. We took this conscious precaution in an attempt to avoid jeopardizing the scientists view of this research and possible future researches since repetitive unsolicited e-mails can be viewed as spam. Our subjects comprehend a list of 6,381 scientists with valid email addresses.

B. Design

Our survey was based on the recommendations of Kitchenham *et al.* [12], we followed the phases prescribed: planning, creating the questionnaire, defining the target audience, evaluating, conducting the survey, and analyzing the results. We set up the survey as an on-line questionnaire (it can be found at: `https://goo.gl/aE2Xcq`). Before sending the actual survey, we conducted a pilot with 50 scientists. During the period of one week, we received 15 responses (30% of response rate). The feedback obtained from the respondents helped us to better clarify some questions (*i.e.,* when we asked about where they maintain their R packages, some respondents mentioned the CRAN website, which we were already aware of. We improved this question to invite respondents to refer to external repositories, such as GitHub or BitBucket) and to improve some closed options (*i.e.,* when we asked how they decide what to do next, two respondents mentioned that they "think independently", which was not initially covered. We made this option available). After these modifications, the initial responses were removed from the questionnaire, and we deployed the actual survey. We did not include scientists already included in the pilot survey. Still, 75 emails were not sent due to technical problems. During a period of about 30 days, we obtained 1,574 responses (25% response rate). For both surveys, participation was voluntary and the estimated time to complete each survey was 10-15 minutes.

C. Questions

Our survey had 28 questions (none were required, 6 were open). Some of the questions covered in the survey include:

— What is your gender? Choices: {male, female, other}
— Please list the highest academic degree you have received or are working toward it. Choices: {BS, MS, PhD}
— What is the subject of this degree?
— What is the URL which your R package is maintained? (e.g., GitHub)?
— What percentage of you total working time in a week do you spend developing software? Choices: {0%, 10% ... 100%}
— How important is developing software yourself for your own research? Choices: {Very important, Important, etc}
— How important was each of the following for helping you learn what you know about developing software? [self-study, from peers, at an education institution, at work] Choices: {Very important, Important, etc}
— Rank the time spend in the following activities (Planning, Reading or reviewing code, Coding and debugging, Quality assurance, Packaging software, and Documenting software) when you are developing software. Choices: {Never, Seldom, etc}
— How do you decide what to do next when you are developing software? Choices: {My supervisor tells, High-priority bugs, etc}
— What are the three most pressing problems, challenges, issues, irritations, or other "pain points" you encounter when developing scientific software?

The complete set of questions, as well as the actual survey, its responses, and the scripts used to gather additional data from the repositories, are available at the companion website: `https://github.com/fronchetti/SANER-2017`.

V. Results

In this section we discuss the results of our main study.

A. Characterizing the R Community

When analyzing the quantitative data from the survey, we observed that 88% of our respondents are male, 45% have between 30–40 years (24% have between 18–30, 4% have over 60), 49% are located in Europe (34% in North America, 0.5% in Africa). About their academic degrees, 80% are working towards (or have already received) their PhD (15% have a master degree). Moreover, 64% of the respondents are Academic Researchers (e.g., professor, post-docs, etc), 20% are software engineers, 15% are graduate students, 10% are teachers, 9% are industrial research scientists (whereas another 8% are government research scientists). Our respondents are from a diverse set of research fields, including Mathematical Statistics, Chemistry, Forestry, Biophysics, and and Computer Science. As a consequence of this diversity, they are working on myriad of different problems, including Genomics, Cognitive Neuroscience, Microbial Ecology, Biogeography, and Computer Vision.

Finally, 603 respondents left the GitHub URL of their repositories. We take advantage of this information and enriched the survey responses with data from the repositories. To draw a perspective about the size of the R packages our respondents maintain, Figure 1 shows the distribution of the number of lines of code, computed by a tool called `cloc`[3]. As we can see, although R is the main programming language for all studied projects, our respondents often employ other programming languages. In our survey, our respondents mentioned that when they not developing scientific software using R, they use C/C++ (55%), Python (51%), or Shell Script (31%).

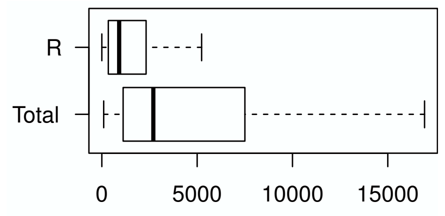

Fig. 1. The distribution of the number of lines of code

Discussion: Similar to the original study, 50% of the respondents were academic researchers. However, the percentage of graduate students and government scientist was higher in the original study: 25% and 16%, respectively (in contrast to 14% and 8% in our study). Besides R, C/C++ and Python are also well-used. In terms of projects' size, the R packages studies have a median size of 1,609 lines of code, when considering all programming languages (min: 1, 1st quartile: 602, 3rd quartile: 4,457, max 214,356); whereas a median size of 909 lines of code, when considering only R (min: 0, 1st quartile: 341, 3rd Quartile: 2,321, max: 49,234).

[3]https://github.com/AlDanial/cloc

B. Training and Education

In this group of questions we provide answers to **RQ1** and **RQ2**. We first asked respondents (Q16) how important was each of the following when learning about software development (the examples under parenthesis were provided to the respondents): Informally by self-study (i.e., on your own), Informally from peers (e.g., at school, university, or work), Formally at an education institution (e.g., by taking a course), Formally at work (e.g., by talking a training course). Figure 3 shows the results.

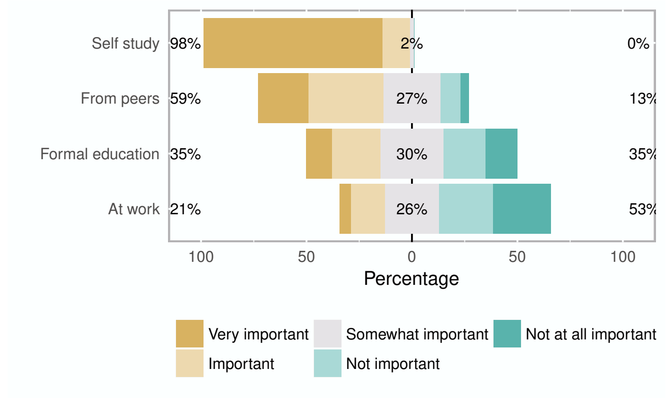

Fig. 2. How important was the way you acquire software development knowledge?

Discussion: In the original study, 97% of the respondents stated that self-study was important or very important for developing software. A similar behavior was found in our study: 99% of our respondents agree that self-study was important or very important. However, when comparing learning from peers or from formal education, we found a remarkable agreement: in both studies, 60% and 35% of the respondents, respectively, agree that they are important or very important. In contrast, only 13% of the respondents of the original study believed that formal training at work was import or very important. In our study, about 22% of the respondents shared this belief.

To answer **RQ2**, we asked respondents (Q19) how important was each of the following periods for helping them learn about software development: During high school, During undergraduate studies, During graduate studies, During professional work more than 15 years ago, During professional work 11-15 years ago, During professional work 6-10 years ago, During professional work 5 years ago. Figure 3 shows the results.

Discussion: Grad school was considered much more important than undergrad or high school, for acquiring software development knowledge: 72% of the respondents believe that graduate school was important or very important to learn what they know about software development; 61% of them considered High school (61%) as not important or not important at all. These results are also in sharp agreement with the results of Hannay *et al* [3]. Interestingly, there is a growth

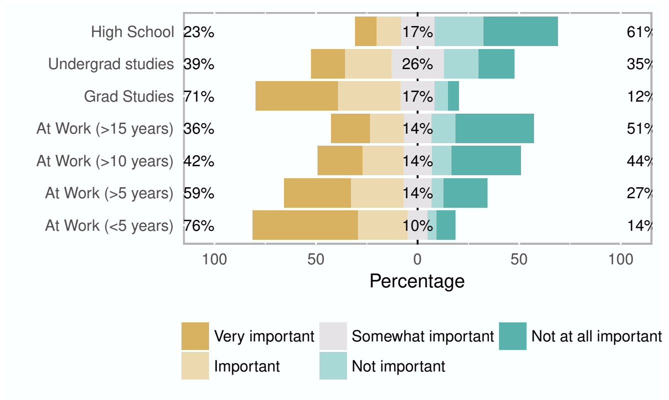

Fig. 3. How important was the way you acquire software development knowledge?

of importance of software development learning at work over the years. Only 36% of the respondents considered the period of 15+ years ago as important or very important for acquiring software engineering knowledge; 76% shared this impression, when considering only the last five years of work.

C. Importance and Frequency of Developing Scientific Software

To provide answers to **RQ3**, we asked our respondents how important is developing software for their own research (Q13) and for the research of others (Q14). Figure 4 shows the results.

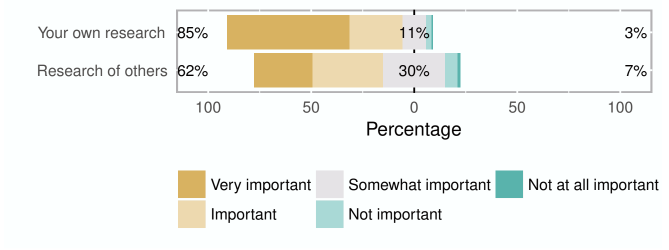

Fig. 4. On the importance of scientific software.

Discussion: In our study, 86% of the respondents believe that scientific software is important or very important to their own research (84% of the respondents of the original study shared the same perception). However, 63% of our respondents stated that developing software for the research of others is important or very important (in contrast to the 46% of the respondents of the original study).

Although software development is not the main activity of our respondents (as we saw in Section V-A, there are Chemistry, Forestry, Biophysics, and so on), when analyzing Q11, we found that our respondents are fairly active, when it comes to software development: 63% of them have developed software in the same day or in the day prior to answering our survey (18% in the same week). The majority of the respondents (98%) have developed software in the last five years (Q9).

To answer **RQ4**, we asked our respondents about their perception regarding the total time they devote for software development (Q10). We observed that, on average, our respondents spent 30% of their working hours developing software per week (min: 0%, 1st quartile: 10%, median: 20%, max: 100%, std dev: 23.73%). Figure 5 hows the distribution. Likewise, on average, the respondents of the original study spent less than 30% of their working hours developing software.

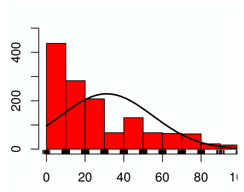

Fig. 5. Hours spent developing software per week (%)

Furthermore, to provide answers to **RQ5**, Figure 6 shows the perception of our respondents about the amount of time devoted developing software, when compared to the last year, the last 5 years, and the last 10 years (Q15).

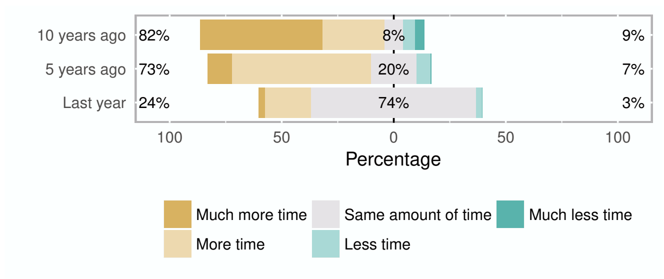

Fig. 6. How important our respondent think about the software engineering concepts?

Discussion: As we can see in this figure, 82% of the respondents believe that they spend “much more time” or “more time” developing software than 10 years ago; 72% when compared to the last five years. In the original study, 53% of the respondents state that they spend more or much more time than 10 years ago (which would be 1998, since the original study was published in 2008). Even though the two groups of respondents believe that they send on average 30% of their working time developing software (**RQ4**), they also believe that the amount of time devoted to software development is increasing over the decades.

To complement this finding, we investigated the commit activity of the studied projects over the years. Figure 7 shows the number of projects with activity (i.e., commits in 3-month time window) to the master branch, excluding merge commits. As we can see, the number of projects with activity is increasing over the years.

D. Teams and Communities

To answer **RQ7**, we asked how large is their development team (Q12). We found that our scientists work mostly alone (53% of the respondents develop software alone) or in small teams (another 42% develop software within a team of two to five). When mining the software repositories, we found that, on average, the set of scientific softwares studied have 3.44 source code contributors (min: 0, max: 36, standard deviation: 3.6).

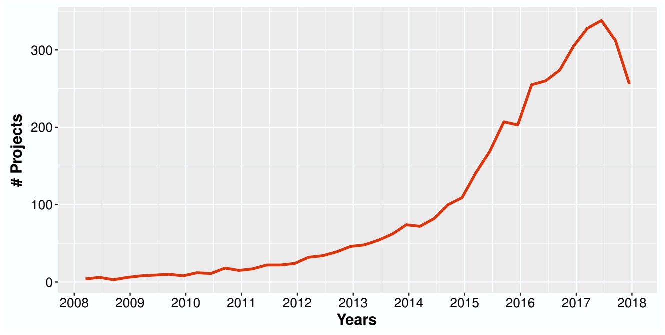

Fig. 7. The number of projects with commits to the master branch, excluding merge commits.

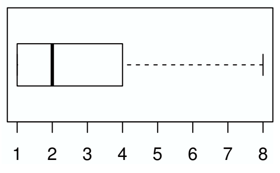

Fig. 8. The distribution of source code contributors on the analyzed projects (excluding outliers)

Interestingly, we found two projects with zero contributors. Investigating these projects, we found that they are read-only mirrors of the original CRAN R packages[4]. We discarded these projects for the remaining analysis. Figure 8 shows the distribution of source code contributors in the analyzed projects (removing outliers to ease visualization).

Still, in the original study, the respondents were asked about the size of the community around their scientific software. In this study, instead of asking the respondents about their perception (which could be inaccurate, due to the lack of ways to measure it), we mined popularity metrics, available on coding hosting websites, such as the number of stars and forks. These metrics describe how developers appreciate or are interested in a given open-source project [13] (therefore, they might work as a proxy for project popularity). Figure 9 shows the distribution of the number of stars and forks for the analyzed projects (excluding outliers to ease visualization).

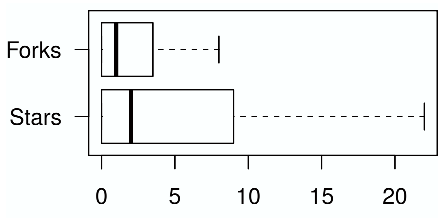

Fig. 9. The distribution of the number of stars and forks (excluding outliers)

As we can see, the median of stars is 2 (max: 723, 3rd quartile: 9) and the median of forks is 1 (max: 139, 3rd quartile: 3). In comparison, in the original study, 56% of the respondents believe that the scientific software they use have more than 5,000 users worldwide.

E. Time Spend on Software Engineering Activities

Figure 10 ranks the time spend in the following activities when developing software (Q18): Planning (*e.g.,* discussing or finding out what new functionality to implement in software, discussing or finding out how to structure or design the software in terms of modules, which procedures will do what, etc.), Reading or reviewing code, Coding or debugging (*e.g.,* writing and compiling C++ for a new module or procedure), Quality assurance (*e.g.,* creating tests, checking results), Packaging software (*e.g.,* updating Makefiles or creating packages for release), Documenting software.

[4]for instance, `https://github.com/cran/nearfar`

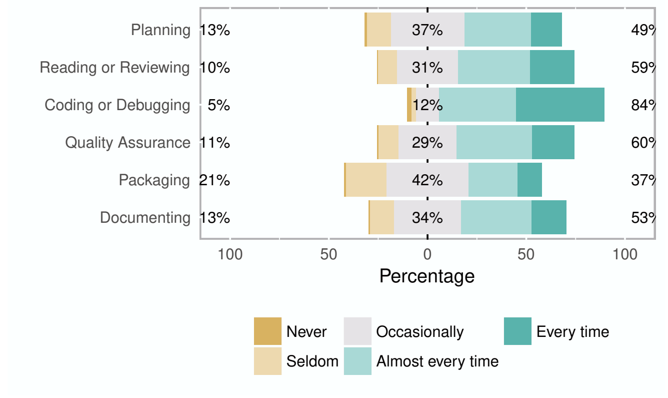

Fig. 10. Rank of the time spent in software engineering activities

Discussion: As we can see, 84% of our respondents develop or debug code every time or almost every time. Other time-intensive activities are: quality assurance (60% do every time or almost every time) and reading or reviewing code (59% do every time or almost every time). Planning and packaging software are the activities that our respondents spend the least time. Still, since the majority of the respondents work alone (Section V-D), it might come as no surprise that 64% of our respondents "decide themselves what they want to work on next" (Q19). Likewise, another 19% "look at yesterday's (the most recent) results and decide what to change in the program today". Only 9% consult colleagues, and 4% consult the software specification and choose the next piece of item to implement.

F. Importance of Software Engineering Practices

To answer **RQ8**, Figure 11 shows the results of Q20, which we asked how important they think are the following software engineering concepts: Software requirements (*e.g.,* eliciting, analyzing, specifying and prioritizing functional and non- functional requirements), Software design (*e.g.,* specifying architecture and detailed design using design by contract, design patterns, pseudo-code, or UML)), Software construction (*e.g.,* coding, compiling, defensive programming), Software verification (*e.g.,* correctness proofs, model-checking, static analysis, inspections), Software testing (*e.g.,* unit testing, integration testing, acceptance testing, regression testing, code coverage, convergence analysis), Software maintenance (*e.g.,* correcting a defect, porting to new platforms, refactoring), Software product management (*e.g.,* configuration management, version control, release planning), and Software project management (*e.g.,* cost/effort estimation, task planning, personnel allocation)

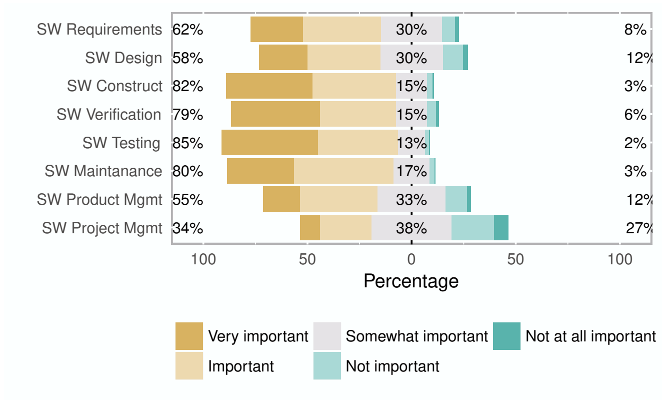

Fig. 11. The importance about the software engineering concepts

Similarly to Q20, in Q21 we asked how well our respondents think they understand the same software engineering concepts. Figure 12 shows the results.

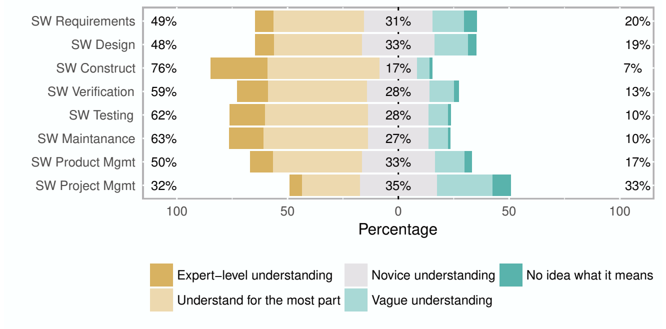

Fig. 12. How well our respondents understand the software engineering concepts?

Discussion: Our respondents classified the majority of the software engineering concepts as very important or important (Figure 11). For instance, Software testing was considered important or very important for 85% of the respondents. In contrast, only 34% of the respondents considered Software project management as important or very important. Moreover, we notice that Software construction is the practice that our respondents consider themselves as well-experienced (76% of them understand the most parts or are experts). Software project management, on the other hand, is the practice that our respondents have the least understanding (69% of the respondents have novice understanding, vague understanding, or have no idea about it). Generally speaking, when comparing these findings, we observed that the respondents judged they knew less about the concepts of what they believe are important. To confirm this observation, we used Chi-Square to test the hypothesis whether the levels of understanding (Figure 12) are independent from the levels of importance (Figure 11) assigned by scientists for each software engineering concept. We found a p-value less than 0.05 in all pair-wised tests. Thus, we reject the null hypothesis that the scientist knowledge depends from what they judge important. This behavior was different from the original study, when the respondents mentioned a high level of understanding to five software engineering practices ("software requirements", "software design", "software maintenance", "software product management", and "software project management") than they judge their level of importance.

To answer **RQ9**, in the original paper, authors employed a one-way Analysis of Variance (ANOVA) statistical test [14] to check if team size and project size influence the scientist's perceptions about the importance of software engineering concepts. To perform a similar analysis, we used data from Q12 (How large is your development team?) to describe the team. Using the same scale of the original study, we obtained the following distribution: 1 developer (768 respondents), 2–5 developers (585 respondents), 6–10 developers (15 respondents) and, more than 20 developers (10 respondents). Regarding project size, we used the already calculated lines of code (Section V-A). Using the same scale of the original study, we obtained the following distribution: up to 500 LOC (127 respondents), 501–5K LOC (325 respondents), 5K–50K LOC (126 respondents), 50K–500K LOC (6 respondents).

Discussion: Regarding **RQ9**, in the original study, the authors found that large projects and large development teams might increase the perceived importance of some software engineering practices, including software construction and software maintenance. However, the authors also reported that there is no consistent trend of association that correlates an increase of project or team size to perceived importance of software engineering concepts. In our study, we could not reject the null hypothesis that project and team size influences the respondents' perceptions about the software engineering concepts.

G. *Naming the Pain on Developing Scientific Software*

Finally, in Q22, we asked what are the three most pressing problems, challenges, issues, irritations, or other "pain points" the respondents encounter when developing scientific software. A total of 1,062 respondents answered this questions. We adopted a straightforward data-driven thematic approach to analyze this question: We summarized each pain in a theme, and then read through the themes multiple times to find cross-cutting sets of common themes. Two authors analyzed the responses. To avoid bias, the themes procedure was done independently, followed by conflict resolution meetings.

Among the findings, we observed that some pain points are well-known from the software engineering practice, such as **Cross-platform compatibility** [15] (e.g., "*porting from mac to windows*"), **Poor software documentation** [16] (e.g., "*Poorly documented software and data packages and APIs (in particular, incomplete function argument or data field definitions)*"), or to **Interruptions while coding** [17] (e.g., "*Teach others not to interrupt my work when I'm focused on the programming*"). Some other pain points have been discussed in the open-source literature, such as **Lack of time** [18], **Scope bloat** (e.g., "*the goal to add features continuing to expand*"), or **Lack of user feedback** (e.g., "*No idea how the end user will ultimately be using the software I write.*").

However, we also observed some pain points intrinsically related to the scientific community, such as **Mismatch between coding skills and subject-matter skills** (e.g., "*I am mathematician, not an expert in software.*"), **Lack of formal reward system** (e.g., "*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*"), or, similarly, **Hard to collaborate on software projects**, since "*[scientists] are concerned with how credit will be allocated*". As a consequence, some respondents mentioned that **aloneness** plays a role (e.g., "*I work alone too much. I'd rather be part of a software development team*".) As we pointed out in Section V-D, 53% of the respondents work alone.

VI. Comparing the Results

According to Carver [7], one of the main values of a replication is the comparison of its results with the results of the original study. Table I summarizes the main results of both studies.

Finally, we also provide a comparison in terms of (1) consistent results and (2) differences in results.

Consistent results. Some of the findings that were in line with the original study include: (1) 99% of our respondents (97% of the original study) considered that self-study was important or very important; (2) Both studies suggested that High school and over the last 15 years at work as not important or not important at all; (3) in both studies, there is the perception that the time spend developing software is increasing over the decades (although both studies — conducted 10 years apart from each other — also agree that, on average, the respondents spend 30% of their working time developing software).

Differences in results. Some of the findings that were different from the original study include: (1) the authors of the original study found that developers that work in large projects or in large development teams might increase the importance of software engineering practices (we could not support this hypothesis); (2) 56% of the respondents of the original study believe that the scientific software they use have more than 5,000 users worldwide (when mining data from the repositories, we found that, at maximum, the analyzed projects have 723 stars and 139 forks); (3) developing software for the research of other was considered important or very important for 63% of the respondents in our study (46% in the original study).

VII. Threats to Validity

Since we are not the authors of the original study, we may have incorrectly employed the method or misunderstood the results reported, therefore, our comparison might lead to wrong conclusions. To mitigate this threat, we got in touch with the original authors, which kindly shared the same survey they employed. Although we added new questions to the survey (e.g., Q8 asked for the GitHub address of their projects), we kept the same questions' titles and options of the original study. Only small issues (e.g., typos) were fixed. Still, before sending the actual survey, we conducted a pilot one. We fixed the issues raised accordingly (Section IV-B for details).

One might argue that not all R developers are indeed scientists. We concur. To mitigate this threat, in our invitation email we kindly asked R developers that do not consider themselves as scientists not to answer the questionnaire. We also asked them not to share the questionnaire with their peers, since some of them might not be scientists as well. Still, as already discussed in Section II, since data is the cornerstone of any scientific work, we chose to focus on the CRAN community because the R programming language is well-employed in data-driven disciplines[5][6], including Biology, Sociology, and Chemistry.

Ultimately, our results only apply to scientists that develop R packages. Although our respondents came from a variety of fields and work on many different problems, our results do not cover other scientific development communities (e.g., scidevelopers of LaTeX macros). Finally, scientists from certain regions were not well represented (e.g., as discussed in Section V-A, we received less than 50 answers from respondents in Africa). Therefore, it is unclear how our results transfer to scientists in these places.

VIII. Related Work

In this section we describe the studies overlapping with the scope of our work.

Surveys about scientific software development. In a survey with 60 scientific developers, Nguyen-Hoan *et al.* [19] found that version control systems (VCSs) and integrated development environments (IDEs) were used by around 50% of developers. Similar to our study, the authors reported that testing and verification activities could be more widely used, such as peer review and integration testing. The majority of developers were Engineerings (non-software) with programming experience between 6 to 15 years, and with background on C, C++, and Perl. Only seven were R developers. Prabhu *et al.* [20] surveyed 114 randomly selected researchers from diverse fields of science and engineering at Princeton University. For programming, the majority of researchers used Fortran and Matlab (only 14% used R). On average, scientists estimate that 35% of their research time is spent in programming/developing software (30% was found in our study)Despite of the effort placed on acquiring programming skills, most scientists are not satisfied with the performance of their programs; they also believe that performance improvements might significantly improve their research. Our work differ from the previous ones due to our focus on how scientists develop scientific software; Prabhu *et al.* [20] addressed the practices researchers follow to enhance computational performance, while Nguyen-Hoan *et al.* [19] was focused on understanding what tools scientists use for software development, how do they document code, and how do they employ testing and verification practices.

[5]https://spectrum.ieee.org/static/interactive-the-top-programming-languages-2017

[6]http://r4stats.com/articles/popularity/

TABLE I
SUMMARY OF THE RESULTS OF BOTH STUDIES. THE ANSWERS TO THE RQS FOR THE ORIGINAL STUDY ARE VERBATIM QUOTES.

	Original Study	**Replication Study**
Method	on-line Survey	on-line Survey
Sample Size	1,972 participants	1,571 participants
Findings	**RQ1:** "96.9% of the respondents state that informal self study is important or very important for developing scientific software (60.1% state that informal learning from peers is important or very important). Only 34.4% state that formal education at an educational institution is important or very important."	**RQ1:**99% of the respondents agree that self-study was important or very important (60% suggested that learning from peers is important or very important). However, 35% of the respondents believe that formal education is important or very important.
	RQ2: "The importance of graduate studies is clearly greater than undergraduate studies. This, in turn, is clearly greater than that of high school studies. Formal training at work was considered as important or very important for only 13.1%."	**RQ2:** Similarly, most of the respondents believe that graduate studies (72%) and their last five years at work (76%) were important or very important to learn what they know about software development. Our respondents also perceived High school (61%) and over the last 15 years at work (51%) as not important or not important at all.
	RQ3: "84.3% of the responses state that developing scientific software is important or very important for their own research. 46.4% state that developing scientific software is important or very important for the research of others."	**RQ3:** 86% of the respondents believe that scientific software is important or very important to their own research. However, 63% of our respondents stated that developing software for the research of others is important or very important.
	RQ4: "On average, scientist spend approximately 30% of their work time developing scientific software."	**RQ4:** On average, our respondents spent 30% of their working hours developing software per week (min: 0%, 1st quartile: 10%, median: 20%, max: 100%, std dev: 23.73%).
	RQ5: "53.5% of the respondents state that they spend more or much more time developing scientific software than they did 10 years ago (44.7% spend more or much more time than they did 5 years ago and 14.5% spend more or much more time than they did 1 year ago."	**RQ5:** 82% of the respondents believe that they spend "much more time" or "more time" developing software than 10 years ago; 72% when compared to the last five years. Over the last decade, there is also an increase in activity, in terms of commits, in the studied projects.
	RQ7: "Scientific software is either used by a very large number of people (more than 5,000 users) or by a very small number of people (less than three)."	**RQ7:** Our respondents work mostly alone (53% of the respondents develop software alone) or in small teams (another 42% develop software within a team of two to five). At maximum, we found 723 stars and 139 forks in the analyzed projects
	RQ8: "The level of importance that scientists assigned to a software engineering concepts is mostly consistent with their understanding of this concept. Except for "software testing" and "software verification" scientists assigned a higher level of importance to these concepts than they judged understand."	**RQ8:** Generally speaking, we observed that the respondents judged they knew less about the concepts of what they believe are important. A Chi-Square test confirmed that the scientist knowledge depends from what they judge important.
	RQ9: "There is no consistent trend of association that links an increase of project or team size to perceived importance of software engineering concepts."	**RQ9:** After performed the ANOVA test, we could not reject the null hypothesis that project and team size influences the respondent's perceptions about the software engineering concepts, once all p-values were higher than 0.05.

Perceptions about software engineering training, practice, and importance. Software design (e.g., data structures and object oriented programming) and software engineering methods (e.g., software maintenance and software testing) are at the core of software engineering practice [21], although many studies suggest that software engineers only master them at their jobs [21], [22], [23]. Recent approaches leverage open-source software [24] or even gamification [25] inside the classroom to foster software engineering learning. Regarding scientists training and practice, Segal [26] conducted several field studies with scientists that develop software. According to Segal, scientists develop scientific software in a very iterative and incremental way. Requirements emerge, as the understanding of both software and science evolves. There is no explicit phase of requirements or evaluation. Testing is not often employed. Segal also published other studies comparing how scientific developers and software engineers develop software [27], [28]. For example, Segal said that scientists are not used to address software development activities such as maintainability, modifiability, or portability. Software engineers, on the other hand, expend a great effort in addressing such issues.

Replications in software engineering. Da Silva *et al.* [29] conducted a systematical review about replication on Software Engineering. The authors reported that 133 replications were performed between 1994 and 2010, based on 72 original studies. Software requirements, software construction, and software quality concentrated over 55% of the replications, while software design, configuration management, and software tools and methods were the topics with the smallest number of replications. According to authors, the number of replications has grown in the last few years, but the absolute number of replications is still small. Some of the recent replications focus on either propose methodologies for conducting a replication (e.g., [30], [7]) or, indeed, replicate an experiment (e.g., [31], [31]). Pfahl *et al.* [32] conducted an external replication to re-evaluated the learning effectiveness of using a process simulation model to teach software project management to students. Results suggest that students that used the simulation model gained a better understanding about typical behavior patterns of software development projects. Wesselen [31] replicated an empirical study about how Personal Software Process (PSP) influence the performance of an individual engineer. Both the original study and the replication one were conducted using data reported from the students that participated of the PSP course. The studies differ in term of the programming languages used, the class sizes, and the level of experience of the students. The results from the replication confirm the ones from the original study: PSP methods can help engineers, for instance, to decrease defects occurrence.

IX. Conclusion

More often than never, scientists have to rely on software in order to conduct their work. However, due to the innumerable research topics that scientists with diverse backgrounds work on, it is at least hard to find an appropriated tool (either proprietary or open source) tailored to deal with scientists' needs. As a consequence, scientists themselves have to get their hands dirty to create such tools. In this replication, we shed additional light on how scientists develop software. Among the findings, we found that scientists work mostly alone, they decide themselves what they want to work on next, most of what they learnt came from self-study, rather than a formal education, although their main activity is not software development, they develop software on regular basis. Admittedly, they concur on the importance of the software engineering practices; nevertheless, they knew less about the concepts of what they believe are important.

Acknowledgments

We thank our respondents and the reviewers. This work is supported by CNPq #406308/2016-0; PROPESP/UFPA; Fundacao Araucaria/UTFPR and FAPESP #2015/24527-3).

References

[1] P. D. Batista, M. G. Campiteli, and O. Kinouchi, "Is it possible to compare researchers with different scientific interests?" *Scientometrics*, vol. 68, no. 1, pp. 179–189, 2006.

[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.*, vol. 25, no. 4, pp. 29–36, Jul. 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 *Proceedings of the 2009 ICSE Workshop on Software Engineering for Computational Science and Engineering*, ser. SECSE '09, 2009, pp. 1–8.

[4] J. C. Carver, "Se-cse 2009: The second international workshop on software engineering for computational science and engineering," in *2009 31st International Conference on Software Engineering - Companion Volume*, May 2009, pp. 484–485.

[5] ——, "Software engineering for science," *Computing in Science & Engineering*, vol. 18, no. 2, pp. 4–5, 2016.

[6] J. Miller, "Replicating software engineering experiments: A poisoned chalice or the holy grail," *Inf. Softw. Technol.*, vol. 47, no. 4, pp. 233–244, Mar. 2005.

[7] 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.

[8] H. Wickham, *R Packages*, 1st ed. O'Reilly Media, Inc., 2015.

[9] D. M. Jones, *Empirical Software Engineering using R*, 1st ed. Knowledge Software, Inc., 2017. [Online]. Available: http://www.knosof.co.uk/ESEUR/

[10] S. Tippmann, "Programming tools: Adventures with r." *Nature*, vol. 517, no. 7532, pp. 109–110, Jan. 2015.

[11] V. R. Basili, F. Shull, and F. Lanubile, "Building knowledge through families of experiments," *IEEE Transactions on Software Engineering*, vol. 25, no. 4, pp. 456–473, Jul 1999.

[12] B. A. Kitchenham and S. L. Pfleeger, *Personal Opinion Surveys*. London: Springer London, 2008, pp. 63–92.

[13] H. Borges, A. Hora, and M. T. Valente, "Predicting the popularity of github repositories," in *Proceedings of the The 12th International Conference on Predictive Models and Data Analytics in Software Engineering*, ser. PROMISE 2016, 2016, pp. 9:1–9:10.

[14] R. Christensen, *One-Way ANOVA*. New York, NY: Springer New York, 1987, pp. 57–69.

[15] J. Bishop and N. Horspool, "Cross-platform development: Software that lasts," *Computer*, vol. 39, no. 10, pp. 26–35, Oct 2006.

[16] T. C. Lethbridge, J. Singer, and A. Forward, "How software engineers use documentation: The state of the practice," *IEEE Softw.*, vol. 20, no. 6, pp. 35–39, Nov. 2003.

[17] M. Züger, C. S. Corley, A. N. Meyer, B. Li, T. Fritz, D. C. Shepherd, V. Augustine, P. Francis, N. A. Kraft, and W. Snipes, "Reducing interruptions at work: A large-scale field study of flowlight," in *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems, Denver, CO, USA, May 06-11, 2017.*, 2017, pp. 61–72.

[18] G. Pinto, I. Steinmacher, and M. A. Gerosa, "More common than you think: An in-depth study of casual contributors," in *IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering, SANER 2016, Suita, Osaka, Japan, March 14-18, 2016 - Volume 1*, 2016, pp. 112–123.

[19] L. Nguyen-Hoan, S. Flint, and R. Sankaranarayana, "A survey of scientific software development," in *Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, ser. ESEM '10, 2010, pp. 12:1–12:10.

[20] P. Prabhu, H. Kim, T. Oh, T. B. Jablin, N. P. Johnson, M. Zoufaly, A. Raman, F. Liu, D. Walker, Y. Zhang, S. Ghosh, D. I. August, J. Huang, and S. Beard, "A survey of the practice of computational science," in *2011 International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, Nov 2011, pp. 1–12.

[21] T. C. Lethbridge, "What knowledge is important to a software professional?" *Computer*, vol. 33, no. 5, pp. 44–50, May 2000.

[22] ——, "Priorities for the education and training of software engineers," *J. Syst. Softw.*, vol. 53, no. 1, pp. 53–71, Jul. 2000.

[23] ——, "A survey of the relevance of computer science and software engineering education," in *Proceedings of the 11th Conference on Software Engineering Education and Training*, ser. CSEET '98, 1998, pp. 0056–.

[24] G. Pinto, F. F. Filho, I. Steinmacher, and M. A. Gerosa, "Training software engineers using open-source software: The professors' perspective," in *30th IEEE Conference on Software Engineering Education and Training, CSEE&T 2017, Savannah, GA, USA, November 7-9, 2017*, 2017, pp. 117–121.

[25] M. R. D. A. Souza, K. F. Constantino, L. F. Veado, and E. M. L. Figueiredo, "Gamification in software engineering education: An empirical study," in *30th IEEE Conference on Software Engineering Education and Training, CSEE&T 2017, Savannah, GA, USA, November 7-9, 2017*, 2017, pp. 276–284.

[26] J. Segal, "Models of scientific software development," in *SECSE 08, First International Workshop on Software Engineering in Computational Science and Engineering*, May 2008, workshop co-located with ICSE 08 http://icse08.upb.de/. [Online]. Available: http://oro.open.ac.uk/17673/

[27] ——, "When software engineers met research scientists: A case study," *Empirical Software Engineering*, vol. 10, no. 4, pp. 517–536, Oct 2005.

[28] ——, "Some challenges facing software engineers developing software for scientists," in *2009 ICSE Workshop on Software Engineering for Computational Science and Engineering*, May 2009, pp. 9–14.

[29] F. Q. B. da Silva, M. Suassuna, A. C. C. França, A. M. Grubb, T. B. Gouveia, C. V. F. Monteiro, and I. E. dos Santos, "Replication of empirical studies in software engineering research: a systematic mapping study," *Empirical Software Engineering*, vol. 19, no. 3, pp. 501–557, Jun 2014.

[30] N. Juristo and S. Vegas, "Using differences among replications of software engineering experiments to gain knowledge," in *Proceedings of the 2009 3rd International Symposium on Empirical Software Engineering and Measurement*, ser. ESEM '09, 2009, pp. 356–366.

[31] A. Wesslén, "A replicated empirical study of the impact of the methods in the psp on individual engineers," *Empirical Software Engineering*, vol. 5, no. 2, pp. 93–123, Jun 2000.

[32] D. Pfahl, O. Laitenberger, J. Dorsch, and G. Ruhe, "An externally replicated experiment for evaluating the learning effectiveness of using simulations in software project management education," *Empirical Softw. Engg.*, vol. 8, no. 4, pp. 367–395, Dec. 2003.