FACETS
a multidisciplinary open access science journal

PERSPECTIVE

A graduate student perspective on overcoming barriers to interacting with open-source software

Oihane Cereceda[a]*, and Danielle E.A. Quinn[b]*

[a]Faculty of Engineering and Applied Science, Memorial University of Newfoundland, St. John's, NL A1C 5S7, Canada; [b]Faculty of Science, Memorial University of Newfoundland, St. John's, NL A1C 5S7, Canada

*danielle.quinn@mun.ca; occ356@mun.ca

Abstract

Computational methods, coding, and software are important tools for conducting research. In both academic and industry data analytics, open-source software (OSS) has gained massive popularity. Collaborative source code allows students to interact with researchers, code developers, and users from a variety of disciplines. Based on the authors' experiences as graduate students and coding instructors, this paper provides a unique overview of the obstacles that graduate students face in obtaining the knowledge and skills required to complete their research and in transitioning from an OSS user to a contributor: psychological, practical, and cultural barriers and challenges specific to graduate students including cognitive load in graduate school, the importance of a knowledgeable mentor, seeking help from both the online and local communities, and the ongoing campaign to recognize software as research output in career and degree progression. Specific and practical steps are recommended to provide a foundation for graduate students, supervisors, administrators, and members of the OSS community to help overcome these obstacles. In conclusion, the objective of these recommendations is to describe a possible framework that individuals from across the scientific community can adapt to their needs and facilitate a sustainable feedback loop between graduate students and OSS.

Key words: open-source software, graduate students, computational research, academic culture

Citation: Cereceda O and Quinn DEA. 2020. A graduate student perspective on overcoming barriers to interacting with open-source software. FACETS 5: 289–303. doi:10.1139/facets-2019-0020

Handling Editor: Debra Clendinneng

Received: May 2, 2019

Accepted: November 18, 2019

Published: May 7, 2020

Published by: Canadian Science Publishing

Introduction

From flexible programming languages to task-specific software, computational tools contribute significantly to research and development across both academia and industry (Prabhu et al. 2011; Nangia and Katz 2017). In a 2017 survey of 209 members of the United States National Postdoctoral Association, 95% of respondents said they use coding or software in their work, with 66% stating that their research would be impractical without the use of these tools (Nangia and Katz 2017). Research in almost any discipline today acknowledges the role and importance of computational tools, with software as an increasingly common research product. In a broader perspective, programming requires understanding the underlying processes, the application and interpretation of solutions, and the implementation of the code to accomplish specific tasks.

Simultaneously, the job market places high demand on computational proficiency; 20% of the 26 million examined job postings in the United States in 2015 required programming or software skills (Burning Glass Technologies 2016). No longer limited to traditional technical positions, these

skills are essential and marketable across disciplines including biology (Dreyfuss 2017; Carey and Papin 2018), business (Daisyme 2019), engineering (Teles dos Santos et al. 2018), and arts and design (Burning Glass Technologies 2016). With increasingly accessible technology, research, development, and innovation evolve in parallel to advances in computational techniques. Thus, engagement and involvement of both end-users and developers represents a valuable feedback loop.

Increasingly, the computational tools used in industry and research fall under the category of open-source software (OSS), defined as software that not only provides users with access to the underlying source code, but also complies with additional criteria including (a) free redistribution, (b) allowing of modification and distribution of derived works, and (c) having no discrimination against persons, groups, or fields of endeavors (Open Source Initiative 2007b). This flexible, no-cost solution for using, developing, and distributing code and software offers an ideal platform for the development of new technologies and collaboration across and within diverse industries and academic disciplines.

By tapping into the collective talent of OSS users and contributors, companies and researchers can generate better products and services. During the 2014 Linux Collaboration Summit, Jim Zemlin, the Executive Director of the Linux Foundation, recognized that OSS is changing the way companies carry out business by linking industries and sharing resources (Vaughan-Nichols 2014). This means that companies that use OSS platforms tend to integrate open source into their main resources; in recent years, corporations including Microsoft, Amazon, and Google have released millions of lines of open-source code and increasingly participate in open-source projects and communities (Amazon 2019b; Google 2019b; Microsoft 2019a). Others, like Red Hat, focus exclusively on their role as contributors and developers of OSS solutions (Red Hat 2019). Hecht and Clark (2018) referred to open-source programs in large companies as a "best practice" and predicted a tripling of the number of companies with such programs by 2020. In both industry and academic data analytics, open-source scripting languages, including Python and R, have gained massive popularity (Geiger et al. 2018), with obvious trends in OSS within-discipline as well. For example, in ecology a major shift has occurred from the proprietary software SAS, commonly used between 1990 and 2008, to the OSS R. By 2013, R was used three times more often than any other analysis software (Touchon and McCoy 2016).

In academia, the shift to OSS as the major analysis tool signifies that graduate students must overcome the challenge of learning how to engage effectively with OSS and the OSS community. This challenging and often nontraditional endeavor offers significant potential benefits to their own research and career. The ability to interact with open and collaborative source code offers students the opportunity to share their ideas locally and globally. Thus, students can develop novel computational resources specific to their field of study; collaborate and network with researchers, code developers, and users from a variety of disciplines; and self-promote by developing these same skills and their contributions to the software (Schweik 2004; Stamelos 2011). Although specialists typically review research papers, individuals from the OSS community, often with a variety of specialty backgrounds, normally evaluate OSS contributions. These diverse perspectives help to improve the clarity, reproducibility, and correctness of the code (Ghosh et al. 2012). Errors can be corrected through this feedback process, ultimately validating the methodology, increasing the quality of the research using the software, and effectively supporting traditional research products such as peer-reviewed papers or dissertations.

Motivation, context, and scope

Despite these clear benefits, graduate students face obstacles at every stage of the learning process that current standard practices in academia do not address. To the best of the authors' knowledge, the literature currently lacks any clear synthesis of these issues from the graduate student perspective.

This article is motivated by this notable gap, as well as by the experiences of the authors as Doctoral students in Computer Engineering and Biology, respectively, and as coding instructors with a combined 10 years of experience teaching open-source scientific programming to approximately 550 graduate students globally (as of late 2019). Thus, the points discussed herein are done so based largely on the shared experiences and personal observations of the authors and the diverse group of students they have interacted with and talked to about this topic.

This paper is intended for those in any discipline that involves some degree of interaction with OSS, including both scientific (e.g., computer science, ecology) and nonscientific fields (e.g., library studies, business). Written from a graduate student perspective, this discussion is intended for individuals in any research-based postgraduate degree program, those who closely work with these individuals (e.g., supervisors, researchers, administrators), and members of the OSS community; it is broadly accessible for those with both limited and well-developed computational knowledge. Best practices of coding and OSS use are not discussed herein; these topics have been discussed in detail by Wilson et al. (2014) and Jiménez et al. (2017). This paper neither discusses the existing differences in academic culture or research progression across countries or institutions, nor speaks to specific programming software or languages.

This article first provides a unique, in-depth discussion of the many barriers that graduate students face when learning, using, developing, and contributing to OSS in the context of their graduate research, including cognitive obstacles and cultural shifts. Following this, it proposes a series of practical guidelines for graduate students, supervisors, administrators, and OSS community members to help reduce and possibly eliminate these barriers. If maintained over time, these proposed actions aim to establish and normalize practices at all levels of academia and OSS that may facilitate a sustainable framework for the continued development of OSS in research while also boosting student success and confidence.

Barriers to learning computational skills

Many barriers exist, and not every graduate student will face them all. Indeed, a wide variety of factors shape and influence everyone's experience including field of study, institution, supervisor, peer support, project specifics, and prior knowledge. The impact of and response to individual barriers are not independent but rather interactive and cumulative; the academic and OSS communities must recognize likely population-level consequences of these barriers that limit progress across domains.

Inconsistencies in training

When students enter graduate school, the discovery that their research will require or improve significantly with data cleaning and other computational skills may cause significant uncertainty. Although most consider programming and the ability to use software programs "hard skills"—teachable abilities that can be defined and measured—inconsistencies and gaps in the training provided to students exist across and within disciplines. Of 55 geography departments of US institutions examined in 2017, 80% offered a programming course, but <11% of the undergraduate degrees offered by these departments required the completion of any programming course (Bowlick et al. 2017). Touchon and McCoy (2016) reported that only 25% of 154 ecological doctoral programs examined required that students take a biostatistics course, which is the primary way that most ecologists apply OSS to research. This reflects an ongoing issue; over a decade ago, in 2008, a workshop jointly organized by the International Union of Biochemistry and Molecular Biology and the Federation of Asian and Oceanian Biochemists and Molecular Biologists had already identified the absence of a standard means of measuring or classifying the computational skills of biology graduates as a major hindrance to the development of bioinformatics graduate education (Tan et al. 2009). As recently as 2017,

computational training has been identified as an unmet need that continues to be underprioritized; in the case of surveyed biologists, this lack of training was seen as the most important limiting factor in their ability to use the data generated by their research (Barone et al. 2017). Today, engineering programs usually require that students complete an introductory programming course, but the links between programming and research are often unclear. While both the breadth and depth of pre- and early-graduate school computing experience and instruction varies among specific disciplines, institutions, and research labs, many new graduate students lack the necessary computing knowledge and skills required for their research project and nor are there opportunities available to gain these skills through traditional or standardized ways (Katz et al. 2018).

Skill assessment

A significant barrier to graduate students at this early stage of their academic career is the need to assess their own programming skills and the skills needed to accomplish their research goals without having appropriate background on the topic or a means by which to undertake an assessment. The complexities surrounding the ignorance of one's own ignorance was discussed by Dunning (2011); the well-known Dunning-Kruger Effect states that poor performers in a domain are largely unaware of their own deficient skills or knowledge. This burden weighs heavily on graduate students struggling to assess their programming skills and deciding how much time and effort they need to develop these skills. Here, a knowledgeable and objective supervisor[1] is essential for providing guidance to their students about how to plan and conduct research (Alharbi and Jacobsen 2016). However, in the context of determining current and required computational skills of students, the success of this guidance is likely to depend on a supervisor's own computational skills and knowledge, which may be limited or outdated as a result of the relatively recent surge in the diversity and use of OSS across disciplines. Developing computational skills, whether out of necessity or in supplement, delays initial research progress. Graduate students must adhere to institutional deadlines and typically depend on limited funding that may require additional responsibilities such as teaching. These time-consuming demands ultimately add burden and stress to a valuable training process. Whether students lack awareness of the training process, or appreciate and embrace the challenges they face, committing the time to develop these skills add risk. As will be discussed in more detail herein, the pursuit of these skills introduces additional barriers to students, including cognitive obstacles during the learning process, a lack of guidance and resources, and technical difficulties linked to gaps in general knowledge and experience in using computers.

Cognitive obstacles

Faced with the daunting task of learning to program, students often experience obstacles that span many learning environments and are well described by cognitive and educational psychologists (Murphy and Thomas 2008; Cutts et. al. 2010). Most of the graduate students the authors have interacted with report that they seek out coding as a tool to accomplish a task related to their research objectives, rather than specifically to learn to program. This is further evidenced by researchers spending significant amounts of time applying code written by others, rather than attempting to write original code (Nowogrodzki 2019); half of UK-based academics surveyed in 2014 reported that they do not write their own software (Smith et al. 2018). Students learn code while also trying to apply these new skills to solve additional problems such as unfamiliar statistical analyses or data transformation; all in addition to managing other research tasks. According to cognitive load theory, these extraneous pieces of information can overwhelm the limited capacity of a student's working memory, leading to problems in learning and retaining new information (Sweller et al. 2011). Students belonging to

[1]Here, supervisor refers to the faculty overseeing the graduate student project, and in various contexts may also be known as the "principal investigator" or the "thesis director".

underrepresented groups in computing may experience additional mental strain in the form of stereotype threat, a fear of conforming to negative stereotypes that may reduce situational performance (Steele and Quinn 1999).

Students require clear analysis objectives to focus their learning. Overwhelmed students often miss out on necessary discussions about the relative importance of syntax specifications, programming concepts, and tool utility in the context of their research and the degree of knowledge and skills needed for each. Without clear objectives and specific knowledge of these new tools, the task of "learning to code" may seem insurmountable to many students. Often students turn to popular online resources, but these resources can be problematic if consumed out of context and the messages that often accompany these resources further increase coding anxiety. For example, some resources claim that only passionate individuals willing to put in the years of effort needed to reach a "master" level should learn programming (Flombaum 2017), providing groundless guidelines about the time commitment required—"a minimum of 3 months [on a schedule of] 11 am to 11 pm" (Kusanagi 2016)—or emphasize that "programming requires being able to absorb a huge swath of ever-changing information" (Pratt 2017).

Struggling students may suffer from a fixed mindset, believing that their abilities or talents are predetermined or innate (Dweck 2007). Comments such as "programming is not for me" or "I'm just not good at coding" reveal this attitude. Unfortunately, blogs, discussion boards, and online articles often support these misconceptions (e.g., Flombaum 2017; Pratt 2017); graduate students are likely to stumble upon such sources of information in the early stages of the learning process when alternative resources are unavailable. A reluctance to take opportunities to learn and declining effort when faced with errors often accompanies a fixed mindset (Dweck 2007). Students often interpret coding errors as a reflection of their own abilities rather than recognizing their inevitability and framing them as valuable learning opportunities (Rodrigo et al. 2009; Brown et al. 2018; Carey and Papin 2018).

Knowledge constraints

A lack of available guidance and resources during the learning process often compounds psychological demotivators. In the earliest stages of learning, students often miss important conversations about tool availability and the advantages and limitations of each or miss the significance of these conversations (Bacharakas 2018). As a result, they may choose software suboptimal for addressing their research problem or supporting their long-term career goals. Initial and ongoing technical considerations such as software installation, operating system compatibility issues, version control, interface options (e.g., R Commander, RStudio, R Notebooks), software launching, directory navigation, and data formatting prompt set-up routines that introduce some degree of dependency. This is particularly problematic for students who use institutional laptops and lack the administrative permissions required to install and update software. The complexity of software installation and configuration can potentially lead to confusion or technical problems that can waste time and demotivate students, especially those with limited computer knowledge (List et al. 2017). These knowledge gaps create obstacles in many ways; for example, a student may have difficulty importing a data set without a general understanding of directory structure or if the file format does not conform to the analysis software requirements (Baker 2017).

After overcoming software installation and configuration, students may be eager to quickly solve complex research tasks, but they must first (*i*) learn the basics of coding, including vocabulary and grammar; (*ii*) be aware of how to apply code in a practical way; and (*iii*) understand how to approach these workflows. Students face a steep learning curve, and where they cannot access formal training through traditional course-based means, students must take responsibility in finding and using online resources, local peer-instruction groups (e.g., Dreyfuss 2017), or other learning opportunities such as workshops or tutorials (Baker 2017). In possible cases where supervisors lack familiarity with a tool,

cannot keep up with the current state of research computing, or simply lack awareness of the value of these skills, students might need to defend the time dedicated to this learning process because progress toward their specific research goals may appear slow.

Seeking help

As students gain foundational skills and begin tackling higher-level research tasks, they will face hurdles such as interpreting error messages, applying unfamiliar functions, and uncertainty about availability of prewritten code for completing common analysis tasks. Success depends upon finding solutions to these problems, but students may hesitate to ask for assistance. At this stage, many students consult notoriously problematic OSS documentation, unaware of its limitations. A survey of GitHub users showed that 93% of respondents reported problems with incomplete, outdated, or confusing OSS documentation (Geiger et al. 2018). Even well-documented contributions include technical language outside of the scope of student's knowledge, forcing them to seek further help online or locally.

While searching for solutions online, students often visit StackOverflow (stackoverflow.com), a valuable community question-answering website where students can browse previously submitted questions and responses or submit their own questions. Community question websites, including StackOverflow, often offer community members who answer questions at high levels of understanding or use technical terminology outside the scope of student's knowledge, especially when the question is not placed in an appropriate context. Students provided with both a solution and explanation of the solution to a specific problem learn more effectively than those provided with only a solution, if the delivery of the explanation coincides with the level of the student's current understanding (Webb 1989). Question quality is important because high-quality questions can attract more users and result in high-quality responses (Ravi et al. 2014). However, students unaware of what makes a high-quality question may make errors in their inquiries, such as using ambiguous terminology, not clearly stating the problem, not contextualizing the problem, or not demonstrating what background research they have done to address the problem themselves, including ensuring that the website does not already answer their question. These oversights can reduce the number and quality of responses to their question and increase the chances of receiving a rude or condescending response from a frustrated community member. Such responses may discourage students from asking future questions or lead them to simply use solution code without gaining an understanding of why or how it works.

Interaction and involvement with online communities outside offers an invaluable resource to increase understanding and use of OSS, but it requires that students gain some understanding of the structure and culture of such communities. The online presence of each OSS community is unique in scope and form, but may include activity on various social media platforms such as Twitter, chat platforms such as Slack, and community sharing through tutorials or blog posts (Carey and Papin 2018; see Amazon 2019a; Google 2019a; Microsoft 2019b). Graduate students often lack awareness of these informal but powerful means of connecting with other users and developers of OSS. Similarly, students might be unaware of local resources such as student-led study groups, formalized help centers on campus, and relevant community meet-up groups. As a result, it is possible that students unknowingly disconnect from peers at the same institution or even within the same department who face similar challenges in learning and using particular software. Lack of availability of these local resources (both on and off campus) represents a significant gap that needs to be addressed at both institutional and community levels.

Barriers to contributing to OSS

Despite seemingly overwhelming challenges, many graduate students build strong programming skills. Some students could potentially transition from learners and users to developers and

contributors. A review of bioinformatics education at both the undergraduate and graduate levels recognized a need for better training of future professionals not only as consumers of computational tools, but also as producers of tools and resources (Magana et al. 2014). This new role includes documenting and sharing code on an accessible platform such as GitHub, receiving and providing feedback, and contributing to the OSS community in other ways, such as organizing resources, creating tutorials, teaching others, translating or improving existing documentation, and participating in discussions. These steps offer a particularly powerful means for graduate students to develop their ideas, work in a collaborative environment, continue to build their programming skills, network with peers and professionals, develop marketable skills like teamwork and interdisciplinary research, and achieve self-promotion through their technical contributions (Schweik 2004; Stamelos 2011). Unfortunately, newcomers, which includes graduate students, may rarely accomplish or even attempt these tasks due to a variety of barriers (Steinmacher et al. 2015).

Cultural obstacles

As noted by Smith et al. (2018), there is a lack of systems in place for rewarding software-based contributions to research. This is reflected in graduate schools and academic culture today through the lack of associated curricula and the perceived unimportance of such contributions. While students explicitly learn about the process of publishing research results as a traditional manuscript in a peer-reviewed journal, comparatively little discussion focuses on the role or importance of publishing code or software or software-related articles. However, traditional manuscripts increasingly include research code and data as supplemental components and some scientific journals now expect it. Although this change represents an important step toward reproducible and transparent science, many students, unaware of their options, might publish "software articles" as a placeholder before receiving credit for their work in traditional formats (Smith et al. 2018). Current publication practices may not acknowledge other code or software-based contributions as primary research products; for instance, these contributions may be omitted from calculations of author metrics on systems such as Google Scholar Citations and Mendeley (Martín-Martín et al. 2018).

Reward systems driving career progression such as tenure reviews and funding applications focus on traditional research contributions and subsequently measure productivity and success in terms of publications and citations of peer-reviewed research papers (Hey and Payne 2015). These metrics do not adequately capture the full range of contributions in modern computational research (Smith et al. 2018). Similarly, degree requirements and institutional expectations of dissertation structure also bias toward traditional research contributions and rarely recognize code—even well-documented, peer-reviewed code—as a comparable product. The computational portion of a student's work, which typically represents a major time commitment, valuable skill development, and potentially significant contributions to the discipline, may end up relegated to appendices or ignored altogether. Other means of contributing to the OSS community, such as creating tutorials or organizing existing resources are recognized as extracurricular volunteer work, at best, and play little or no role in degree progression. Thus, those students who devote time to publishing their code and contributing to OSS do so at the risk of delays in degree or career progress, and may face criticism from supervisors and peers despite the inherent value of these tasks, which may be sought by industry (Learner et al. 2006; Smith et al. 2018).

Other constraints

In addition to the challenges associated with academic culture, few graduate students are prepared for roles as contributors to OSS. A recent survey by Mozilla of university students who wanted to contribute to OSS but were unable to revealed that nearly 70% cited "not knowing where to start" as the primary reason they did not contribute (Bacharakas 2018). Graduate school curricula may not include

the practical skills needed for contributing, including where and how to share code, where and how to ask for feedback, and general knowledge about licensing and maintenance responsibilities. These knowledge gaps may be particularly daunting and complex in instances where graduate students have adapted another user's code for their own purposes, which is the case most often seen by the authors. Unfortunately, newcomers who join the OSS community with limited practical knowledge of contributing rarely become long-term contributors, in part because of a notable absence of mentoring in the OSS community (Mendez et al. 2018). Steinmacher et al. (2015) provided evidence that a lack of social interaction between newcomers and the OSS community may represent as significant a barrier to contributing as a lack of technical skills.

Would-be contributors face a new set of psychological challenges. Both students and well-established scientists worry that their code and documentation are not sufficiently polished, and that they will face judgement for poor variable names, misused technical terminology, or errors in their code (Barnes 2010). Although it is important that users validate their code, especially code that may have been reverse engineered or adapted, students may fear that errors in their code will poorly reflect the quality of their research. These concerns and overall lack of confidence decrease the likelihood they will share or publish their code even if the value of such a contribution is high (Barnes 2010; Bacharakas 2018). Graduate students are not taught and are unaware that unpolished code is normal and acceptable even among professional software developers, and code that accomplishes something others are also trying to accomplish represents a valuable contribution that should be shared and reviewed (Barnes 2010).

Recommendations

Barriers discourage graduate students at every stage of the learning process and hinder the advancement and adoption of both programming and OSS community involvement as interdisciplinary tools in academia. As these skills become increasingly important across and within disciplines, graduate students, supervisors, administrators, and OSS community members must address these barriers in a systematic and intentional way.

Appropriate and accurate skill assessment, both of self and of others, is a complex component of many of the barriers discussed herein. Recognizing this, the authors recommend ongoing communication among those interacting with OSS, which facilitates access to instructors and mentors, provides opportunities for objective feedback from varied perspectives, and exposes novices to the diversity of users, software, tools, and approaches available.

Regarding the other barriers discussed, the authors stress that a wide diversity of situations exists, and that not all of the recommendations provided below will be accessible, timely, and practical in every case. Instead, these items are meant as guidelines to introduce readers to a variety of approaches and actions that may improve the graduate student and (or) supervisor experience and contribute to the general advancement of OSS use in academia.

Recommendations for graduate students

Success in graduate school critically depends on maintaining motivation. Thus, graduate students must balance recognizing their limitations, identifying the knowledge and resource gaps they need to fill, and acknowledging their progress. To encourage a productive and motivating environment for themselves and their peers, graduate students should:

1. Talk to others about what computational skills will likely be required for conducting their research effectively and how long it will take to acquire those skills. Importantly, this includes communicating with senior graduate students who are likely to have recent experiences and

fresh insight. In some situations, graduate students may need to pitch the value of OSS to their supervisors or administrators, and providing real-world examples of how others are applying these skills to their research or career can be an effective approach.

2. Create a plan, including a learning strategy, a set of expectations, and a list of tasks to complete. Consider the extra time to develop (or learn how to use) tools and code and consider other technical difficulties when defining specific goals. Discuss the plan with peers or supervisor(s) and adjust it as needed to maintain a steady learning process. Concentrate on one task at a time if possible. A good starting point for this process is to identify and discuss the computational skills required for conducting their research and existing gaps in a student's knowledge as a way of framing the next steps of the learning process.

3. Ask for help when needed; seek out knowledgeable local and online resources that can provide feedback from the context of a project and as a fresh perspective.

4. Help build local communities by promoting the use of OSS and sharing existing open-source projects that may benefit other research teams. This may include introducing community-developed resources such as The Carpentries curricula through local networks and sharing their own code as a resource to novices.

5. Form or join local discussion groups on or off campus; individuals from a similar background and at similar career stages will likely comprise a student-led study group, making them well equipped to provide effective and timely support in a group setting because they unite in their problem solving processes and likely share similar language (Webb 1989). More diverse local resources such as programming clubs or hackathon events introduce a wide range of knowledge and experiences from which students can learn and benefit.

6. Follow the software community on social media to stay up-to-date with new software releases, updates, discussions, and interesting applications of OSS and create a direct contact with the online community. Students can strengthen and maintain these connections by contributing to ongoing discussions, assisting with tasks such as documentation, and sharing components of their own work or coding exercises publicly. These communities make valuable resources accessible to new users through groups like The Carpentries, R-Ladies, and Pearl Monks (Carey and Papin 2018).

7. Validate code by asking for feedback from peers and external researchers, and encourage other students to do the same by demonstrating the significance of OSS in research and academia. Submitting code for review to a software journal will not only consolidate it, but also contribute to the discipline as a scientific contribution.

8. Encourage informal or formal mentoring by pursuing opportunities to connect with experts outside of the primary research team and act as a mentor for junior graduate students. If possible, consider external programs such as Google Summer of Code to gain OSS experience and skills in a more formal setting (Google 2019c).

Recommendations for supervisors

Supervisors represent the primary resource for guidance and mentorship for graduate students and can potentially greatly influence their skills, knowledge, and attitude toward OSS. To help students avoid or overcome barriers to OSS, supervisors should:

1. Stay up-to-date on current and available computational tools by attending seminars and workshops and keep an open mind to changing computational approaches.

2. Acknowledge the limitations of their personal expertise; ask administrators to provide appropriate training opportunities to faculty and (or) identify alternative resources for their students where necessary.

3. Demonstrate and instil a growth mindset among graduate students and colleagues through encouragement, praise, and error-framing.

4. Help students connect with a variety of resources to continue to develop their skill sets, including through peers, local groups, technical resources (tutorials and text books), and online communities.

5. Encourage interdisciplinary collaboration (e.g., organize seminars at the same institution within similar fields but different backgrounds), and discuss the importance of transferable skills and diverse perspectives in the context of computational research.

6. Demonstrate the importance of contributing to OSS by being an active community member and help students to get involved in local or online projects. Praise students for their software contributions and discuss the short- and long-term benefits of this work to their research and future career.

7. Encourage students to publish code, technical documentation, software, and software articles, defend these contributions as research contributions, and discuss how to reward or appropriately recognize these products.

Recommendations for administrators

Administrators can provide resources, prioritize institutional objectives, and influence academic culture. To support students and faculty interacting with OSS in their research, administrators should:

1. Regularly survey the needs of graduate students by evaluating their current research projects, asking for feedback, and considering short- and long-term changes to curricula that will help fulfill these needs.

2. Explore ways to integrate necessary skill development activities in the graduate school timeline, including seminar series, additional or modified coursework requirements, or analytical workshops. Facilitate the students' participation including time and funding allowances as needed.

3. Cultivate long-term expertise by motivating training opportunities for faculty and staff; incentivize participation and recognize local OSS ambassadors, mentors, and contributors.

4. Support the operation and sustainability of centralized resources on campus and encourage their use. These resources should address gaps in research skills, offer a platform for sharing knowledge, and be available on official online platforms as well as include an in-person component. This need also includes hardware and software resources for students to interact with the platforms they need to effectively conduct research.

5. Support the local OSS community by integrating graduate students, faculty, and other researchers.

6. Offer students and faculty guidance on the process and systems around publishing software, and encourage them to collaborate on OSS projects in addition to more traditional means of contributing to research.

Recommendations for members of the OSS community

As noted by the Open Source Initiative (2007a), "OSS and other collaborative projects benefit through, and because of, community". OSS community members can play an important role in supporting and mentoring new collaborators, including graduate students. When working with graduate students, OSS community members should:

1. Reach out to the academic community and encourage students to work with and contribute to OSS by highlighting how they can benefit from OSS to reach research and long-term career goals.

2. Continue to improve traditional and nontraditional documentation efforts, such as sensible error messages, to assist novice users with troubleshooting and code testing.

3. Structure online discussions from low to high complexity to make the content more accessible to all users. Provide solutions along with their explanation when answering online questions and encourage all users to follow that practice.

4. Help develop and support platforms where students can publish their code and software and be cited appropriately. These platforms should include resources around feedback, licensing, and citation processes.

5. Offer newcomers constructive feedback about the applicability and implementation of their code and other contributions.

6. Take on mentorship roles by being approachable, supportive, and accessible. Demonstrate and share best practices of interacting with OSS and the OSS community and promote knowledge transfer in the form of blogs, discussions, and meetups.

Conclusions

OSS offers an important tool for conducting and producing research in both academia and industry. Graduate students represent an impactful group that can potentially contribute in multiple ways. However, in the context of OSS, graduate students often face challenging barriers that delay or reduce the transferability of their scientific contributions.

This manuscript framed the problem of developing software and conducting research in graduate school, a hurdle often missed by the scientific community. The barriers are not confined to the coding learning process; students also must abstract the application, develop software tools, and validate their work. From a graduate student's perspective, psychological, cultural and practical barriers include insufficient documentation, disconnection from the OSS online community, and a lack of awareness of the resources. Despite discouraging barriers that may lead a student to quit, a team effort can turn around a poor situation. Considering barriers faced by many graduate students, the recommended practices listed above for each member in the academic and OSS community to implement and help the graduate student experience creates a positive culture. Increased communication with and exposure to the OSS community will help graduate students recognize the value of their efforts and contribute to OSS at different levels.

On a positive note, many now recognize the general challenges around OSS, and freely available resources can help. However, the resources per se have costs because they are maintained voluntarily by a group of developers and maintainers with no other interest than maintain useful software. As heavy users of OSS, graduate students represent a perfect fit for the long-term sustainability of OSS, which is currently at risk. Technology catalyzes changes in academia and simultaneously, the OSS community now demands changes in how others understand OSS. This relationship can be improved through graduate student contributions, noting the key role of students in OSS, academia, and research. The barriers, although sometimes disheartening, can be overcome and doing so should be considered as a means for a more important end: the continuance of OSS ecosystems.

Author contributions

OC and DEAQ conceived and designed the study. OC and DEAQ contributed resources. OC and DEAQ drafted or revised the manuscript.

Competing interests

The authors have declared that no competing interests exist.

Data availability statement

All relevant data are within the paper.

References

Alharbi H, and Jacobsen M. 2016. Educational development for quality graduate supervision. Papers on Postsecondary Learning and Teaching, 1: 41–46.

Amazon. 2019a. AWS Open Source (@AWSOpen) [online]: Available from twitter.com/awsopen.

Amazon. 2019b. Open Source at Amazon [online]: Available from amzn.github.io/.

Bacharakas C. 2018. Cracking the Code—how Mozilla is helping university students contribute to Open Source. Medium [online]: Available from medium.com/mozilla-open-innovation/cracking-the-code-how-mozilla-is-helping-university-students-contribute-to-open-source-25fa630d8c5c.

Baker M. 2017. Scientific computing: code alert. Nature, 541(7638): 563–565. DOI: 10.1038/nj7638-563a

Barnes N. 2010. Publish your computer code: it is good enough. Nature, 467(7317): 753. PMID: 20944687 DOI: 10.1038/467753a

Barone L, Williams J, and Micklos D. 2017. Unmet needs for analyzing biological big data: a survey of 704 NSF principal investigators. PLoS Computational Biology, 13(10): e1005755. PMID: 29049281 DOI: 10.1371/journal.pcbi.1005755

Bowlick FJ, Goldberg DW, and Bednarz SW. 2017. Computer science and programming courses in geography departments in the United States. The Professional Geographer, 69(1): 138–150. DOI: 10.1080/00330124.2016.1184984

Brown AW, Kaiser KA, and Allison DB. 2018. Issues with data and analyses: errors, underlying themes, and potential solutions. Proceedings of the National Academy of Sciences of the USA, 115(11): 2563–2570. PMID: 29531079 DOI: 10.1073/pnas.1708279115

Burning Glass Technologies. 2016. Beyond point and click: the expanding demand for coding skills [online]: Available from academy.oracle.com/pages/Beyond_Point_Click_final.pdf.

Carey MA, and Papin JA. 2018. Ten simple rules for biologists learning to program. PLoS Computational Biology, 14(1): e1005871. PMID: 29300745 DOI: 10.1371/journal.pcbi.1005871

Cutts Q, Cutts E, Draper S, O'Donnell P, and Saffrey P. 2010. Manipulating mindset to positively influence introductory programming performance. *In* Proceedings of the 41st ACM Technical Symposium on Computer Science Education. ACM, New York, New York. pp. 431–435. DOI: 10.1145/1734263.1734409

Daisyme P. 2019. Should entrepreneurs know programming basics? Due [online]: Available from due.com/blog/should-entrepreneurs-know-programming-basics/.

Dreyfuss E. 2017. Want to make it as a biologist? Better learn to code. Wired [online]: Available from wired.com/2017/03/biologists-teaching-code-survive/.

Dunning D. 2011. The Dunning-Kruger Effect: on being ignorant of one's own ignorance. Advances in Experimental Social Psychology, 44: 247–296. DOI: 10.1016/B978-0-12-385522-0.00005-6

Dweck CS. 2007. The perils and promises of praise. Educational Leadership, 65(2): 34–39.

Flombaum A. 2017. Is it true that programming is not for everyone? Posted to: Quora [online]: Available from quora.com/Is-it-true-that-programming-is-not-for-everyone.

Geiger RS, Varoquaux N, Mezel-Cabasse C, and Holdgraf C. 2018. The types, roles, and practices of documentation in data analytics open source software libraries. Computer Supported Cooperative Work (CSCW), 27(3–6): 767–802. DOI: 10.1007/s10606-018-9333-1

Ghosh SS, Klein A, Avants B, and Milman KJ. 2012. Learning from open source software projects to improve scientific review. Frontiers in Computational Neuroscience, 6(18): 1–18. PMID: 22529798 DOI: 10.3389/fncom.2012.00018

Google. 2019a. Google GitHub [online]: Available from github.com/google.

Google. 2019b. Google open source [online]: Available from opensource.google.com/.

Google. 2019c. Google summer of code [online]: Available from summerofcode.withgoogle.com/.

Hecht L, and Clark L. 2018. Survey: open source programs are a best practice among large companies. The New Stack [online]: Available from thenewstack.io/survey-open-source-programs-are-a-best-practice-among-large-companies.

Hey T, and Payne MC. 2015. Open science decoded. Nature Physics, 11(5): 367–369. DOI: 10.1038/nphys3313

Jiménez RC, Kuzak M, Alhamdoosh M, Barker M, Batut B, Borg M, et al. 2017. Four simple recommendations to encourage best practices in research software. F1000Research, 6: 876. PMID: 28751965 DOI: 10.12688/f1000research.11407.1

Katz DS, McInnes LC, Bernholdt DE, Mayes AC, Hong NPC, Duckles J, et al. 2018. Community organizations: changing the culture in which research software is developed and sustained. arXiv:1811.08473v2. DOI: 10.1109/MCSE.2018.2883051

Kusanagi M. 2016. How quickly can a person with no coding ability expect to learn enough to get work coding? Posted to: Quora [online]: Available from quora.com/How-quickly-can-a-person-with-no-coding-ability-expect-to-learn-enough-to-get-work-coding.

Learner J, Pathak PA, and Tirole J. 2006. The dynamics of open-source contributors. American Economic Review, 96(2): 114–118. DOI: 10.1257/000282806777211874

List M, Ebert P, and Albrecht F. 2017. Ten simple rules for developing usable software in computational biology. PLoS Computational Biology, 13(1): e1005265. PMID: 28056032 DOI: 10.1371/journal.pcbi.1005265

Magana AJ, Taleyarkhan M, Reivera Alvarado D, Kane M, Springer J, and Chase K. 2014. A survey of scholarly literature describing the field of bioinformatics education and bioinformatics educational research. CBE—Life Sciences Education, 13: 607–623. PMID: 25452484 DOI: 10.1187/cbe.13-10-0193

Martín-Martín A, Orduna-Malea E, and Delgado López-Cózar E. 2018. Author-level metrics in the new academic profile platforms: the online behaviour of the bibliometrics community. Journal of Informetrics, 12(2): 494–509. DOI: 10.1016/j.joi.2018.04.001

Mendez C, Padala HS, Steine-Hanson Z, Hilderbrand C, Horvath A, Hill C, et al. 2018. Open source barriers to entry, revisited: a sociotechnical perspective. *In* ICSE '18: Proceedings of the 40th International Conference on Software Engineering. pp. 1004–1015. DOI: 10.1145/3180155.3180241

Microsoft. 2019a. Microsoft open source [online]: Available from opensource.microsoft.com/.

Microsoft. 2019b. Microsoft open source blog [online]: Available from cloudblogs.microsoft.com/opensource/.

Murphy L, and Thomas L. 2008. Dangers of a fixed mindset: implications of self-theories research for computer science education. *In* Proceedings of the 13th Annual Conference on Innovation and Technology in Computer Science Education. pp. 271–275. DOI: 10.1145/1597849.1384344

Nangia U, and Katz DS. 2017. Surveying the US National Postdoctoral Association regarding software use and training in research. *In* Workshop on Sustainable Software for Science: Practice and Experiences. DOI: 10.6084/m9.figshare.5328442

Nowogrodzki A. 2019. How to support open-source software and stay sane. Nature, 571: 133–134. PMID: 31263262 DOI: 10.1038/d41586-019-02046-0

Open Source Initiative. 2007a. Community and collaboration [online]: Available from opensource.org/community.

Open Source Initiative. 2007b. The open source definition [online]: Available from opensource.org/osd.

Prabhu P, Jablin TB, Raman A, Zhang Y, Huang J, Kim H, et al. 2011. A survey of the practice of computational science. SC '11: State of the Practice Reports No. 19. pp. 1–12. DOI: 10.1145/2063348.2063374

Pratt C. 2017. Programming is not for everyone [online]: Available from cpratt.co/programming-is-not-for-everyone/.

Ravi S, Pang B, Rastogi V, and Kumar R. 2014. Great question! Question quality in community Q&A. *In* Proceedings of the Eighth International Association for the Advancement of Artificial Intelligence Conference on Web and Social Media [online]: Available from aaai.org/ocs/index.php/ICWSM/ICWSM14/paper/view/8080.

Red Hat. 2019. Creating better technology with open source [online]: Available from redhat.com/en/about/open-source#.

Rodrigo MMT, Baker RS, Jadud MC, Ammarra ACM, Dy T, Espejo-Lahoz MBV, et al. 2009. Affective and behavioral predictors of novice programmer achievement. *In* ITiCSE '09: Proceedings of the 14th Annual ACM SIGCSE Conference on Innovation and Technology in Computer Science Education. Vol. 41, No. 3, pp. 156–160. DOI: 10.1145/1595496.1562929

Schweik CM. 2004. The open-source paradigm and the production of scientific information: a future vision and implications for developing countries. *In* Open access and the public domain in digital data and information for science: Proceedings of an International Symposium.

Smith AM, Niemeyer KE, Katz DS, Barba LA, Githinji G, Gymrek M, et al. 2018. Journal of Open Source Software (JOSS): design and first-year review. PeerJ Computer Science, 4: e147. DOI: 10.7717/peerj-cs.147

Stamelos IG. 2011. Teaching software engineering with free/libre open source projects. *In* Multi-disciplinary advancement in open source software and processes. *Edited by* S Koch. IGI Global, Hershey, Pennsylvania. pp. 67–85. DOI: 10.4018/978-1-60960-513-1.ch005

Steele SJ, and Quinn DM. 1999. Stereotype threat and women's math performance. Journal of Experimental Social Psychology, 35: 4–28. DOI: 10.1006/jesp.1998.1373

Steinmacher I, Silva MAG, Gerosa MA, and Redmiles DF. 2015. A systematic literature review on the barriers faced by newcomers to open source software projects. Information and Software Technology, 59: 67–85. DOI: 10.1016/j.infsof.2014.11.001

Sweller J, Ayres P, and Kalyuga S. 2011. Intrinsic and extraneous cognitive load. *In* Cognitive load theory. *Edited by* J Sweller, P Ayres, and S Kalyuga. Springer, New York, New York. pp. 57–69. DOI: 10.1007/978-1-4419-8126-4

Tan TW, Lim SJ, Khan AM, and Ranganathan S. 2009. A proposed minimum skill set for university graduates to meet the informatics needs and challenges of the "-omics" era. BMC Genomics, 10(Suppl. 3): S36. PMID: 19958501 DOI: 10.1186/1471-2164-10-S3-S36

Teles dos Santos M, Vianna AS Jr, and Le Roux GAC. 2018. Programming skills in the industry 4.0: are chemical engineering students able to face new problems? Education for Chemical Engineers, 22: 69–76. DOI: 10.1016/j.ece.2018.01.002

Touchon JC, and McCoy MW. 2016. The mismatch between current statistical practice and doctoral training in ecology. Ecosphere, 7(8): e01394. DOI: 10.1002/ecs2.1394

Vaughan-Nichols SJ. 2014. It takes an open-source village to make commercial software. ZDNet [online]: Available from zdnet.com/article/it-takes-an-open-source-village-to-make-commercial-software/.

Webb NM. 1989. Peer interaction and learning in small groups. International Journal of Educational Research, 13(1): 21–39. DOI: 10.1016/0883-0355(89)90014-1

Wilson G, Aruliah DA, Brown CT, Hong NPC, Davis M, Guy RT, et al. 2014. Best practices for scientific computing. PLoS Biology, 12(1): e1001745. PMID: 24415924 DOI: 10.1371/journal.pbio.1001745