Work in Progress: Identifying Software Engineering Practices and Tools Among Students and Practitioners in Non-Computing Engineering Disciplines

Stephanos Matsumoto, Olin College of Engineering

Stephanos (Steve) Matsumoto is an Assistant Professor of Computer Science and Engineering at the Olin College of Engineering. His research interests are in computing education, particularly in how to incorporate better software engineering practices when teaching computing in undergraduate STEM courses.

Dr. Michelle E Jarvie-Eggart P.E., Michigan Technological University

Dr. Jarvie-Eggart is a registered professional engineer with over a decade of experience as an environmental engineer. She is an Assistant Professor of Engineering Fundamentals at Michigan Technological University. Her research interests include technology adoption, problem based and service learning, and sustainability.

Work-in-Progress: Identifying Software Engineering Practices and Tools Among Students and Practitioners in Non-Computing Engineering Disciplines

Abstract

Despite the increasing importance of working with and developing software in numerous engineering fields, engineering education today largely focuses on programming, rather than software engineering practices and tools (SEPTs), that is, the tools and techniques for designing, implementing, and maintaining software over time. As a result, the productivity or reliability of engineering work involving software can be hampered by problems that could have been avoided with the use of modern best practices in software engineering. Despite a history of research on SEPTs in computing fields (e.g., computer science and software engineering) and computational science fields (e.g., computational physics and bioinformatics), the use of SEPTs in engineering fields is not well understood.

To address this problem, in this paper, we present ongoing work investigating how practitioners and undergraduate students in non-computing engineering disciplines understand and use SEPTs. Specifically, we present the preliminary design of a qualitative study, including a survey instrument to assess familiarity with software engineering terminology and use of SEPTs. Our survey is based on the Software Engineering Body of Knowledge (SWEBOK) Guide, which outlines a generally accepted, standard body of knowledge expected of practicing early-career software engineers. We design the survey to be accessible even to those unfamiliar with the specific software engineering terminology used in the SWEBOK Guide. In addition to the survey itself, we describe our planned approach to conduct a thematic analysis of participants' responses, using the taxonomy of the SWEBOK Guide as an analytical framework.

We hope that our study will help illuminate the landscape of how different engineering disciplines understand and develop software. While we intend for our survey to be used in studying engineers in non-computing fields, we anticipate that the results of our study will inform the development of further research to investigate SEPT use in engineering in a discipline-specific or discipline-agnostic manner. In the broader context, we expect that these insights will help us more identify and teach key SEPTs in undergraduate engineering education, and thereby help future engineers write and maintain software more effectively, whatever their discipline.

Introduction

Understanding and writing software is becoming increasingly important knowledge and practice in modern engineering (and for that matter, in all STEM fields), a trend noted both by students [1] and practitioners [2]. The processes by which scientists and engineers develop software has become more complex, involving many collaborators [3] and close coupling with other parts of the engineering design process [4]. Despite this increasing importance, the treatment of software in undergraduate STEM education largely consists of *programming*, that is, implementing software for achieving a specific task, rather than *software engineering practices and tools*

(SEPTs), that is, tools and techniques used in the overall process of designing, implementing, and maintaining software. This gap between teaching programming and SEPTs seems to exist both in computing-centric STEM disciplines such as software engineering [5] and computing-adjacent disciplines such as computational physics or bioinformatics [6].

While previous research has proposed interventions that are promising for improving some SEPTs in STEM fields, these interventions often assume a disciplinary context that is far too broad or too narrow to understand their effect within a specific engineering discipline. For example, the Carpentries program (formerly Software Carpentry) aims to teach basic SEPTs to scientists and engineers [7]. However, the impact of programs like these on engineers are not well-understood: studies usually measure self-reported confidence in workshop topics rather than actual SEPT usage, and reported results lump all participants together rather than considering discipline-specific outcomes [8, 9]. Other proposed interventions measure outcomes only for students studying computing-centric disciplines such as computer science or software engineering [10]. Rather than developing interventions in a broad STEM context or in a narrow computing context, it is important that we more deeply understand SEPT use in their disciplinary contexts. This understanding will help us identify important SEPTs to teach, potential barriers to their adoption, and ways to embed these SEPTs in authentic practice, maximizing the impact of these disciplinary-specific interventions.

There are existing survey instruments to measure SEPT use, but these are also tailored to a specific disciplinary context, often computing [11] or scientific computing [12]. Adapting these instruments to a disciplinary-specific context is difficult, largely for two reasons: (1) each discipline develops software within its own epistemological paradigm [13], and (2) the terminology used in one discipline may not match what is used within another [14]. Thus, to deeply understand SEPT use within a discipline, we must understand the ways in which practitioners of that discipline engage with and articulate the process of software development.

To address the above problem, we present in this paper a preliminary design for a qualitative study designed to answer the following research questions:

1. How do students and practitioners in non-computing-centric engineering disciplines understand generally accepted terminology in software engineering?
2. What generally accepted SEPTs do students and practitioners in non-computing-centric engineering disciplines use in their work?

For the purpose of our study, we define *generally accepted* as in the 2014 Software Engineering Body of Knowledge (SWEBOK) Guide, which describes knowledge that a software engineer is expected to have after several years of professional practice. To address our research questions, we design a qualitative survey instrument, with the SWEBOK Guide providing a guiding framework both for the design of the survey and for the analysis of the resulting data. Ultimately, we intend for the insights from our study to guide the creation of a quantitative survey instrument to answer on overarching research question: What software engineering practices and tools do undergraduate engineering students and practitioners currently use in their work?

In the remainder of this paper, we first present background on SEPTs and the SWEBOK Guide. We then describe our study design in detail. We conclude the paper with a discussion on potential

threats to validity and the anticipated impacts of our study.

Background and Related Work

In this section, we begin by describing relevant background and prior work. Specifically, we describe SEPTs in more detail, including how they differ from programming, their importance in engineering work, and previous efforts to teach or assess them. We then provide background on the SWEBOK Guide, including its purpose and high-level organization.

Software Engineering Practices and Tools

As we mention earlier in this paper, SEPTs and the field of software engineering are distinct from programming. Brian Randell reportedly described software engineering as "the multi-person development of multi-version programs" [15], and more recently, software engineering has been described as "programming integrated over time", that is, the process of evolving code as its requirements, users, or underlying technologies change [16].

Thus, while SEPTs are practices and tools that *may* be used while writing programs, their primary goal is to *support the process of designing, implementing, or maintaining software*. As an example, a SEPT may be the *practice* of interviewing key stakeholders to elicit software requirements, which involves no programming whatsoever. Or, a SEPT may be a *tool* to automatically generate test cases for a given software implementation, helping to catch bugs during the development process.

The term *SEPT* is not in common use; it is a term that we chose to distinguish our object of study from software engineering *knowledge*. As an example, the concept of a class in object-oriented programming would not be a SEPT, whereas a specific class design pattern would be, as it can be applied and implemented in order to solve some problem. One way to distinguish a SEPT from software engineering knowledge would be to apply Bloom's taxonomy (specifically, in the cognitive domain), with the Application, Analysis, Synthesis, and Evaluation levels referring to SEPTs as opposed to knowledge [17].

Importance of SEPTs

As computing becomes increasingly intertwined with the practice of other fields of engineering, the benefits of good SEPTs in software development becomes clearer, leading to demonstrable impacts on productivity. For example, software engineering research shows that good use of SEPTs can result in more accurate estimations of effort required [18], shorter development time [19], and more trustworthy and reproducible analysis [20, 21].

By contrast, scientific computing research shows that poor use of SEPTs can hinder productivity [22] or even invalidate previously published research results [23]. These examples highlight the importance of understanding SEPTs, as they have the potential to impact both the velocity and reliability of engineering work involving software.

Proposed Improvements in Teaching SEPTs

Proposed improvements to how we teach SEPTs to undergraduate engineering students are presented in a range of conferences and journals in education, including both computing and engineering education venues. Interventions from prior work apply practices such as code review and refactoring [24], pedagogical frameworks such as cognitive apprenticeship [25], and

partnerships such as those between full-time instructors and industry practitioners [26]. These largely target students, faculty, and practitioners in computing-centric fields, such as computer science and software engineering.

That being said, there are also numerous efforts to improve the teaching of SEPTs in scientific computing. Greg Wilson, who created the initial Software Carpentry project [7], has made numerous recommendations to adopt of key SEPTs in the scientific computing community [20, 27–29]. Interventions in engineering fields (outside of computing) are rare.

The SWEBOK Guide

From the earliest uses of the term "software engineering", there have been calls to establish the field as an engineering profession [30], particularly by the two leading professional societies in computing, the Association for Computing Machinery (ACM) and the Computer Society of the Institute of Electrical and Electronics Engineers (IEEE). Efforts to establish software engineering as a profession have included a call to compile a standard set of knowledge that all practicing software engineers are expected to have [31]. The SWEBOK Guide is one work towards this goal, describing and organizing a generally accepted body of knowledge in professional software engineering [32], which refers to the knowledge that a practitioner would typically be expected to have after several years of work experience.

The SWEBOK Guide is maintained by the Professional and Educational Activities Board of the IEEE Computer Society. Its latest version (from 2014) identifies 15 key knowledge areas within the field of software engineering:

1. Software Requirements
2. Software Design
3. Software Construction
4. Software Testing
5. Software Maintenance
6. Software Configuration Management
7. Software Engineering Management
8. Software Engineering Process
9. Software Engineering Models and Methods
10. Software Quality
11. Software Engineering Professional Practice
12. Software Engineering Economics
13. Computing Foundations
14. Mathematical Foundations
15. Engineering Foundations

Assessing SEPT Use

As with teaching SEPTs, efforts to assess SEPTs among students and practitioners have focused mainly on computing-centric fields [5, 33], with some prior work focusing on specific software engineering practices such as handling technical debt in code [34] or tools such as those used for international collaboration [35]. There is also numerous work to assess various SEPTs in the scientific computing community [12, 22, 36–38].

Table 1: Section 1 of the survey developed by Garousi et al. Note 1: possible answers are 0 (Completely useless), 1 (Occasionally useful), 2 (Moderately useful), 3 (Very useful), and 4 (Critical to my work). Note 2: possible answers are 0 (Learned nothing at all), 1 (Learned the basics), 2 (Moderate working knowledge), 3 (Learned a lot), and 4 (Learned in depth; became expert).

	Importance/usage in your job	**How much you learned in your whole university education**
1. Software requirements	Note 1	Note 2
1.1. Software requirements fundamentals	Note 1	Note 2
1.2. Requirements process	Note 1	Note 2
1.3. Requirements elicitation	Note 1	Note 2
1.4. Requirements analysis	Note 1	Note 2
1.5. Requirements specification	Note 1	Note 2
1.6. Requirements validation	Note 1	Note 2
1.7. Practical considerations such as requirements traceability	Note 1	Note 2
1.8. Software requirements tools	Note 1	Note 2

The work of Garousi et al. specifically assesses SEPT use among recent graduates (i.e., relatively new software engineers) based on the the SWEBOK Guide [5]. The authors operationalize knowledge gaps between software engineering education and practice by assessing practitioners' self-reported measure of using a software tool and how much they covered that tool in their university education [11]. Examples of such questions are shown in Table 1.

The results of the above survey seem to indicate that among software engineers, the most commonly used areas were Software Design and Software Construction. Some areas with higher importance/usage are well-covered in undergraduate education (e.g., the above two areas and Software Requirements), while some have "knowledge gaps" and thus are not well covered in undergraduate programs (e.g., Software Maintenance, Software Configuration Management and Software Testing). These results largely line up with those of the previously mentioned surveys of the scientific computing community.

Study Design

In this section, we describe the design of our study. We begin by describing specific knowledge areas within the SWEBOK Guide in which we assess SEPTs. We then present our preliminary survey design and our plans for distributing the survey and analyzing the results.

Knowledge Areas

Among the 15 knowledge areas of the SWEBOK Guide, the three areas not directly related to software engineering (Computing Foundations, Mathematical Foundations, and Engineering Foundations) are not assessed in the Garousi et al. survey [5]. These areas represent disciplinary expertise necessary to conduct some parts of the software engineering process, but do not cover practices and tools within the field of software engineering. We thus decided to leave these areas out of our survey instrument.

We further identified three additional knowledge areas that cover practices and tools that are general to engineering as a whole: Software Engineering Management, Software Engineering Professional Practice, and Software Engineering Economics. Though these knowledge areas are named in a way that seems specific to software engineering, their content includes topics such as project planning, risk management, legal issues, teamwork, and cash flow. Since these are not specific to the software development process, we also decided to leave these areas out of our survey instrument.

Thus, in the end, we were left with nine selected knowledge areas, numbered as 1–6 and 8–10 in the 2014 SWEBOK Guide: (1) Software Requirements, (2) Software Design, (3) Software Construction, (4) Software Testing, (5) Software Maintenance, (6) Software Configuration Management, (7) Software Engineering Process, (8) Software Engineering Models and Methods, and (9) Software Quality. Our survey is intended to assess participants' understanding of SEPT-related terminology and their SEPT usage in each of these nine selected areas.

Survey Design

Recall that our study intends to shed light on (1) how students and practitioners in non-computing engineering disciplines understand software engineering terminology as defined in the SWEBOK Guide, and (2) what SEPTs students and practitioners in non-computing engineering disciplines use in their work. While we considered fine-grained survey items that aimed to elicit different potential terms and SEPTs within each knowledge area, we found that the organization of the SWEBOK Guide made this difficult. As an example, consider the Software Requirement knowledge area shown in Table 1. Without referencing the SWEBOK Guide, it is difficult to know what SEPTs belong in subcategories such as Software Requirements Fundamentals or in Practical Considerations. We thus chose to provide open-ended survey questions at the *knowledge area* level, assessing both participants' understanding of terminology and their use of SEPTs.

In designing our survey questions, we followed guidelines by Kitchenham and Pfleeger for designing survey questions [39], as well as guidelines by Lenzner for wording survey questions in an accessible manner [40]. Our survey instrument consists of three parts, which we explain in greater detail below.

Terminology. Our survey begins with an assessment of participants' understanding of software engineering terminology. This section of the survey begins with the following prompt:

> This section of the survey asks about how you understand terms from the field of software engineering. To ensure that your response reflects your understanding, please do not look up the meaning of these terms until you have completed the survey.
>
> Consider each of the following topics *within the context of developing software*. What words, phrases, or concepts come to mind?

The survey then presents each of the nine selected knowledge areas. For each knowledge area, participants can provide a short, open-ended response.

SEPT Usage. The second section of our survey focuses on the generally accept SEPTs that participants use in their work. This section presents the following questions:

> This section of the survey asks about the approaches and tools you have used when developing software *in your work, and over the past 12 months*. In each question, the term "your software" refers to *any software* that meets these criteria.
>
> - What approaches or tools did you use to determine the goals and/or constraints on your software?
> - What approaches or tools did you use to organize your software into smaller components?
> - What approaches or tools did you use to actually implement your software?
> - What approaches or tools did you use to check that your software behaved correctly?
> - What approaches or tools did you use to adapt your software to new goals/constraints, technologies, or bugs?
> - What approaches or tools did you use to determine the hardware or other software required to use your software?
> - What approaches or tools did you use to determine the process for developing your software?
> - What approaches or tools did you use to communicate key aspects of your software or development process to stakeholders?
> - What approaches or tools did you use to check that your software actually met its goals/constraints?

These questions represent each of the nine knowledge areas above. Rather than simply listing the name of the knowledge area when asking about participants' SEPT use, we briefly describe the content of each knowledge area. This approach frees participants from needing to map their own practices to software engineering terminology, which they may or may not understand well, and enables them to describe their software development process in their own language.

Demographics and Experience. In order to understand differences across engineering disciplines, we conclude our survey with questions on participants' discipline, status (as student or practitioner), and experience in developing software. This section contains the following questions (choices shown in parentheses):

> This section of the survey asks for basic information about your engineering discipline and level of experience, both in your chosen discipline and with software.
>
> - Are you a student or a practitioner (working in the field)? (Student, Practitioner)
> - What is your primary engineering discipline? (Mechanical, Electrical, Civil)
> - (Students only) What is your class status? (Freshman, Sophomore, Junior, Senior, Fifth-Year or later)
> - (Practitioners only) How many years of experience do you have in your field since completing your bachelor's degree?
> - In the past 12 months, how often have you designed, implemented, or maintained software/code for your work? (Never, A few times, Once a month, Once or twice a week, more than once a week)
> - In the past 12 months, how often have you designed, implemented, or maintained software/code *outside* of work? (Never, A few times, Once a month,

Once or twice a week, more than once a week)

For our initial survey, we limited our target participant population to students and practitioners in three engineering disciplines: mechanical, electrical and civil. According to the most recent ASEE Profiles of Engineering and Engineering Technology report, these are the three engineering disciplines outside of computer science in which the greatest number of bachelor's degrees were awarded, at a total of 58,900 out of 141,826, or about 41.5% [41]. In order to interpret participants' SEPT use in the context of their experience, we asked three additional questions: (1) class year (for students) or years of experience (for practitioners), (2) frequency of recent work with software., and (3) frequency of software development outside of work. While not directly tied to our research questions, these additional survey questions may help us in future work. For example, they may shed light on how differences in disciplinary or software experience correlate with those in SEPT use.

Institutional Context

Much of our study will take place at our respective institutions. Olin College is a small engineering college located in Needham, Massachusetts in the US. Under the Carnegie Classification, it is considered a very small (fewer than 1000 undergraduate students), exclusively undergraduate, special focus undergraduate program (on engineering and other technologies). It only offers undergraduate degrees in three programs: Engineering (with a specialized concentration, such as Design), Mechanical Engineering, and Electrical and Computer Engineering (ECE).

Michigan Technological University is a medium-sized public research university located in Houghton, Michigan in the US. Under the Carnegie Classification, it is considered to have a medium (3000–10000 undergraduate students), highly undergraduate (10–25% graduate students overall), professions-focused undergraduate program. It offers undergraduate degrees in a wide range of engineering disciplines, including Mechanical, Electrical, and Civil Engineering (which comprise our target participant population).

Survey Pilot and Distribution

We are in the process of conducting a pilot study of our survey among undergraduate students at our respective institutions. We are leveraging our institutions' alumni networks to recruit practitioners to pilot our survey as well. Our pilot studies will consist of small focus groups of around 2–3 participants from each participant group (students and practitioners, each from the three engineering disciplines, for a total of 12–18 pilot participants). We will collect and analyze pilot data to determine if the wording of the survey questions, particularly in the second section (assessing SEPT use), should be refined. To do this, we will ask each group two questions following the survey:

1. On what questions, if any, were you unsure about how to answer, and why?
2. What aspects of your software development practices or tools, if any, did you not get a chance to discuss in this survey?

We recognize that some practices and tools may not be in the SWEBOK Guide due to not being generally accepted within the field, or more recent than 2014 (when the SWEBOK Guide was last published). While our study will continue to use the nine knowledge areas above, we will use

these pilot responses to plan follow-up research investigating SEPTs not appearing in the SWEBOK Guide. Based on the pilot responses, we will make necessary refinements to the wording of our survey instrument's prompts and questions. We will then seek IRB approval of the refined survey, making additional changes to our study design as required by the IRB.

For distribution of the survey to students, we will use convenience sampling [42], distributing the survey among our respective institutions (Olin College and Michigan Technological University). For distribution of the survey to practitioners, we will leverage not only our institutions' alumni networks, but also contacts in our professional networks, including institutional colleagues and our respective professional societies, to identify feasible distribution channels. We will then distribute the survey through those channels, using snowball sampling [43] if necessary to find and recruit additional participants. We hope to collect data from at least 15 participants in each group (at least 90 participants overall). To incentivize participation, we will offer the chance of winning one of a set number of gift cards to those who complete the survey.

Analysis Plan

We will conduct thematic analysis on our survey responses. Specifically, we will use a deductive coding approach, mapping responses in each section of the survey to terms, concepts, practices, and tools in the SWEBOK Guide. We will code responses independently, holding regular coding meetings to discuss our analyses and resolve differences to ensure intercoder agreement. We expect this process to result in a taxonomy of SEPTs and relevant terminology, mapped to knowledge areas in the SWEBOK Guide. We will also present the use of the terms and SEPTs in our taxonomy by student/practitioner status and by engineering discipline.

Discussion

Below, we discuss our work in a broader context, including its expected impacts, how it may inform engineering education, and its limitations.

Limitations and Potential Threats to Validity

Because participants will voluntarily complete our survey, our analysis will reflect the SEPTs and understanding of terminology among a self-selected population. Additionally, our distribution channels for our survey instrument will rely heavily on potential participants at our respective institutions. While there are significant differences between our institutions, there are some common factors, including a mostly full-time, highly residential undergraduate student body, higher selectivity, a lower population of transfer students, and a large fraction of degrees awarded in professional fields. These factors may affect the degree to which our analysis is indeed representative of SEPT use and understanding in each of our participant populations.

As we note in the design of our survey pilot, we also anticipate that some SEPTs may not neatly map to the knowledge areas of the SWEBOK Guide. We acknowledge that the SWEBOK Guide represents a general consensus on software development knowledge among software engineers, but this is by no means the full extent of SEPTs. Indeed, we know from previous work that the software workflows of those who are not software engineering can look very different [3]. We emphasize that despite the results of our analysis, any differences in SEPT use that we identified may be completely warranted, and is not indicative of bad practices that need to be "corrected" in other engineering disciplines.

Expected Impacts

As far as we know, our work is the first to assess SEPTs specifically in engineering fields outside of computing. We hope that this work will shed light on the ways in which students and practitioners in these disciplines engage with software and the software development process. This understanding will in turn serve as a springboard for future work to develop more detailed insights on SEPTs in different engineering disciplines. Specifically, we hope to use the results of this study to develop a more comprehensive survey instrument on SEPTs and gain further insight into SEPT use in engineering disciplines beyond those studied here. We also hope to identify potential participants for follow-up interviews, in which we can gain richer data on the factors that may explain why certain SEPTs are or are not used in a field.

Applications to Engineering Education

With a deeper understanding of SEPT use in engineering, provided both by this survey and by follow-up work described above, we can identify opportunities to improve the way we teach SEPTs within different engineering disciplines. Previous work demonstrates that the effective use of SEPTs benefits productivity and the reliability of software, and as computing becomes an increasingly important part of engineering work, we believe that teaching these best practices will be critical to teaching engineering. Our insight on SEPT use will help set the stage for those innovations in engineering education.

Conclusions

In this paper, we described the preliminary design of a survey to assess SEPTs in students and practitioners of mechanical, electrical, and civil engineering. Our design focused on being accessible to those unfamiliar with software engineering technology, as well as on building atop professional standards in software engineering used in previous work. While this work in still in progress, we are optimistic about the potential of this work to shed much-needed insight on how engineering disciplines engage with software. We hope that this understanding allows us to make important improvements to how we teach software development in engineering education.

References

[1] J. Parham-Mocello, J. Garcia, and M. Sadler, "Engineering student perspectives of a new required programming course," in *IEEE Frontiers in Education Conference (FIE)*. IEEE, Oct. 2023.

[2] H. Jang, "Identifying 21st century STEM competencies using workplace data," *Journal of Science Education and Technology*, vol. 25, no. 2, pp. 284–301, Dec. 2015.

[3] D. Paine and C. P. Lee, ""Who has plots?": Contextualizing scientific software, practice, and visualizations," *Proceedings of the ACM on Human-Computer Interaction (PACMHCI)*, vol. 1, no. CSCW, pp. 1–21, Nov. 2017.

[4] H. Sporer, G. Macher, E. Armengaud, and C. Kreiner, "Incorporation of model-based system and software development environments," in *Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. IEEE, Aug. 2015, pp. 177–180.

[5] V. Garousi, G. Giray, and E. Tüzün, "Understanding the knowledge gaps of software engineers: An empirical analysis based on SWEBOK," *ACM Transactions on Computing Education (TOCE)*, vol. 20, no. 1, pp. 1–33, Feb. 2020.

[6] J. Carver, D. Heaton, L. Hochstein, and R. Bartlett, "Self-perceptions about software engineering: A survey of scientists and engineers," *Computing in Science and Engineering (CiSE)*, vol. 15, no. 1, pp. 7–11, Jan. 2013.

[7] G. Wilson, "Software Carpentry: Getting scientists to write better code by making them more productive," *Computing in Science & Engineering (CiSE)*, vol. 8, no. 6, pp. 66–69, Nov. 2006.

[8] A. Simperler and G. Wilson, "Software Carpentry – get more done in less time," arXiv:1506.02575, Jun. 2015.

[9] B. K. Weaver, "The efficacy and usefulness of Software Carpentry training: A follow-up cohort study," Master's thesis, The University of Queensland, 2019.

[10] A. Berg, S. Osnes, and R. Glassey, "If in doubt, try three: Developing better version control commit behavior with first year students," in *ACM Technical Symposium on Computer Science Education (SIGCSE)*, Feb. 2022, pp. 362–368.

[11] V. Garousi, G. Giray, and E. Tüzün, "Survey of the skills important to software professionals based on SWEBOK," https://zenodo.org/records/546594 (last accessed April 27, 2024), Apr. 2017.

[12] J. C. Carver, N. Weber, K. Ram, S. Gesing, and D. S. Katz, "A survey of the state of the practice for research software in the United States," *PeerJ Computer Science*, vol. 8, no. e963, May 2022.

[13] T. Storer, "Bridging the chasm: A survey of software engineering practice in scientific programming," *ACM Computing Surveys*, vol. 50, no. 4, pp. 1–32, Nov. 2017.

[14] S. Faulk, E. Loh, M. L. V. D. Vanter, S. Squires, and L. G. Votta, "Scientific computing's productivity gridlock: How software engineering can help," *Computing in Science and Engineering (CiSE)*, vol. 11, no. 6, pp. 30–39, Nov. 2009.

[15] D. L. Parnas, *Dependable and Historic Computing*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, Oct. 2011, vol. 6875, ch. Software Engineering: Multi-person Development of Multi-version Programs, pp. 413–427.

[16] T. Winters, T. Manshreck, and H. Wright, *Software Engineering at Google: Lessons Learned from Programming Over Time*. O'Reilly Media, Mar. 2020.

[17] M. D. Engelhart, W. H. Hill, E. J. Furst, and D. R. Krathwohl, *Taxonomy of educational objectives: The classification of educational goals*, B. S. Bloom, Ed. New York and London: Longman, 1956.

[18] T. K. Abdel-Hamid, K. Sengupta, and D. Ronan, "Software project control: an experimental investigation of judgment with fallible information," *IEEE Transactions on Software Engineering*, vol. 19, no. 6, pp. 603–612, Jun. 1993.

[19] N. Forsgren, J. Humble, and G. Kim, *Accelerate: Building and Scaling High Performing Technology Organizations*. IT Revolution Press, Mar. 2018.

[20] G. Wilson, J. Bryan, K. Cranston, J. Kitzes, L. Nederbragt, and T. K. Teal, "Good enough practices in scientific computing," *PLOS Computational Biology*, vol. 13, no. 6, Jun. 2017.

[21] G. Lee, S. Bacon, I. Bush, L. Fortunato, D. Gavaghan, T. Lestang, C. Morton, M. Robinson, P. Rocca-Serra, S.-A. Sansone, and H. Webb, "Barely sufficient practices in scientific computing," *Patterns*, vol. 2, no. 2, p. 100206, Feb. 2021.

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

[23] G. Miller, "A scientist's nightmare: Software problem leads to five retractions," *Science*, vol. 314, no. 5807, pp. 1856–1857, Dec. 2006.

[24] S. K. Sripada and Y. R. Reddy, "Code comprehension activities in undergraduate software engineering course - a case study," in *Australasian Software Engineering Conference (ASWEC)*. IEEE, Sep. 2015, pp. 68–77.

[25] A. Shah, J. Yu, T. Tong, and A. G. Soosai Raj, "Working with large code bases: A cognitive apprenticeship approach to teaching software engineering," in *ACM Technical Symposium on Computer Science Education (SIGCSE TS)*, Mar. 2024.

[26] G. Kulczycki and S. Atkinson, "Why educators need to team with industry professionals in software development education," in *ASEE Annual Conference and Exposition*. ASEE Conferences, 2018. [Online]. Available: https://peer.asee.org/31243

[27] G. V. Wilson, "What should computer scientists teach to physical scientists and engineers?" *IEEE Computational Science and Engineering*, vol. 3, no. 2, pp. 46–65, 1996.

[28] G. Wilson, D. A. Aruliah, C. T. Brown, N. P. C. Hong, M. Davis, R. T. Guy, S. H. D. Haddock, K. D. Huff, I. M. Mitchell, M. D. Plumbley, B. Waugh, E. P. White, and P. Wilson, "Best practices for scientific computing," *PLoS Biology*, vol. 12, no. 1, Jan. 2014.

[29] M. Taschuk and G. Wilson, "Ten simple rules for making research software more robust," *PLOS Computational Biology*, vol. 13, no. 4, Apr. 2017.

[30] A. G. Oettinger, "President's letter to the ACM membership," *Communications of the ACM*, vol. 9, no. 8, pp. 545–546, Aug. 1966.

[31] S. McConnell, *After the Gold Rush: Creating a True Profession of Software Engineering*. Microsoft Press, 1999.

[32] P. Bourque and R. E. Fairley, Eds., *Guide to the Software Engineering Body of Knowledge, Version 3.0*. IEEE, 2014.

[33] A. Radermacher and G. Walia, "Gaps between industry expectations and the abilities of graduates," in *ACM Technical Symposium on Computer Science Education (SIGCSE)*. ACM, Mar. 2013, pp. 525–530.

[34] F. Gilson, M. Morales-Trujillo, and M. Mathews, "How junior developers deal with their technical debt?" in *International Conference on Technical Debt (TechDebt)*. ACM, Jun. 2020, pp. 51–61.

[35] J. Portillo-Rodríguez, A. Vizcaíno, M. Piattini, and S. Beecham, "Tools used in global software engineering: A systematic mapping review," *Information and Software Technology*, vol. 54, no. 7, pp. 663–685, Jul. 2012.

[36] J. E. Hannay, C. MacLeod, J. Singer, H. P. Langtangen, D. Pfahl, and G. Wilson, "How do scientists develop and use scientific software?" in *ICSE Workshop on Software Engineering for Computational Science and Engineering*, May 2009.

[37] L. Nguyen-Hoan, S. Flint, and R. Sankaranarayana, "A survey of scientific software development," in *ACM-IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, Sep. 2010.

[38] G. Pinto, I. Wiese, and L. F. Dias, "How do scientists develop scientific software? an external replication," in *IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, Mar. 2018, pp. 582–591.

[39] B. A. Kitchenham and S. L. Pfleeger, "Principles of survey research: part 3: constructing a survey instrument," *ACM SIGSOFT Software Engineering Notes*, vol. 27, no. 2, pp. 20–24, Mar. 2002.

[40] T. Lenzner, "Effects of survey question comprehensibility on response quality," *Field Methods*, vol. 24, no. 4, pp. 409–428, Sep. 2012.

[41] American Society for Engineering Education, *Profiles of Engineering and Engineering Technology, 2022*. Washington, DC: American Society for Engineering Education, 2023.

[42] R. Ferber, "Research by convenience," *Journal of Consumer Research*, vol. 4, no. 1, pp. 57–58, Jun. 1977.

[43] L. A. Goodman, "Snowball sampling," *The Annals of Mathematical Statistics*, vol. 32, no. 1, pp. 148–170, Mar. 1961.