WIP: Better Understanding Software Engineering Practices and Tools in Engineering Education

Stephanos Matsumoto
Olin College of Engineering
Needham, MA, USA
smatsumoto@olin.edu

Michelle Jarvie-Eggart
Michigan Technological University
Houghton, MI, USA
mejarvie@mtu.edu

***Abstract*—In this work-in-progress research paper, we present the design and preliminary results of an ongoing thematic analysis of software engineering practices and tools (SEPTs), using Bloom's cognitive taxonomy to analyze the content of the Software Engineering Body of Knowledge (SWEBOK) Guide. Our work is motivated by the fact that while the software development process is becoming increasingly important in engineering, software engineering skills are hardly taught in most undergraduate engineering curricula, and existing descriptions of these skills are not easily accessible to educators in many engineering disciplines. We present results from the analysis of the Software Requirements knowledge area, which those outside of computing have characterized as especially important but not well understood. We identified 92 practices and tools in the knowledge area, and found some promise in our approach: our results help highlight key topics within the knowledge area where SEPTs can be especially helpful, such as requirements elicitation, analysis, and change process management, and provide further clarity on SEPTs from prior work that may be helpful to engineering educators. Though our current findings do not cover the extent to which our identified SEPTs are accessible to those outside of engineering, we are optimistic that our work helps a broader range of engineering educators identify SEPTs that might be underutilized in their engineering practice, and the depth to which they might teach these SEPTs in their curricula. Our work is thus an important first step in addressing the gap between how we teach students to develop software in undergraduate engineering programs and how practitioners in the field develop software.**

***Index Terms*—Computing skills, Engineering standards, Bloom's taxonomy, Professional skills, Document analysis**

I. Introduction

Developing software, that is, writing code that can be executed to accomplish a desired task, is a skill that is becoming a core literacy of the coming generation [1]. Even in engineering disciplines that are traditionally outside computing, we see that the software engineering process is becoming increasingly important [2] and collaborative in nature [3]. Despite this shift, undergraduate engineering education has largely been slow to adapt: though software engineering encompasses more than just programming [4], many undergraduate engineering programs heavily focus on programming over software engineering skills [5], even in computing-heavy fields such as software engineering [6]. As a result, there exists a gap between the *software engineering practices and tools (SEPTs)* taught in undergraduate engineering programs and the skills needed in modern engineering careers [6].

While addressing this gap will ultimately require that undergraduate engineering programs incorporate a wider range of SEPTs into their curricula, a critical hurdle towards this goal is that the way in which the existing literature describes and assesses SEPTs is not easily accessible to engineering instructors in fields outside of software. We primarily draw our understanding of what SEPTs are from systematizations of knowledge in software engineering [4], [7], [8]. As a result, SEPTs are often assessed as they might be in practitioners within the software engineering industry, that is, those whose primary work is the development of software [9]. Moreover, SEPTs are often communicated to an audience assumed to have experience in software engineering [8], leading to descriptions of SEPTs that educators in other engineering disciplines may struggle to integrate into their curricula.

In this paper, we present the design and preliminary results of an ongoing study that represents a first step towards making SEPTs easier to understand and assess among educators of a broader set of engineering disciplines. Specifically, we conduct a thematic analysis on the Software Engineering Body of Knowledge (SWEBOK) Guide [7], which represents a generally accepted body of knowledge for early-career software engineers created with input from practitioners in industry and academia. We use Bloom's cognitive taxonomy [10] as an analytical framework to identify and classify SEPTs; while this has been done before [11], our positionality (educators within engineering and not computer science, one with extensive software training and one without) allows us to communicate these SEPTs in a more accessible way while also validating prior efforts. Overall, our work aims to address two key research questions:

1. How does Bloom's taxonomy applied to the SWEBOK Guide validate, or add to, our understanding of SEPTs as described in prior work?
2. To what extent are the description and assessment of these SEPTs accessible to engineering educators outside of software-centric fields?

Our work in this paper addresses the first of these questions.

II. Theoretical Framework

Bloom's cognitive taxonomy (often referred to simply as *Bloom's taxonomy*) has seen longstanding, widespread use as a way to classify learning objectives, assessments, and activities

in education, with six levels roughly ordered from least to greatest proficiency: (1) **Knowledge**, (2) **Comprehension**, (3) **Application**, (4) **Analysis**, (5) **Synthesis**, and (6) **Evaluation** [10]. The taxonomy was later revised to (1) **Remember**, (2) **Understand**, (3) **Apply**, (4) **Analyze**, (5) **Evaluate**, and (6) **Create** [12], sometimes called *Bloom's revised taxonomy*.

This paper uses the earlier (non-revised) taxonomy for several key reasons. Much of the existing literature on SEPTs and the SWEBOK Guide use the older taxonomy [11], making it easier to compare our findings with those of prior work. Additionally, the Software Engineering Competency Model (SWECOM), which systematically defines SEPTs based on the SWEBOK Guide and other resources, uses an alternate taxonomy that also contains the verb Create, but specifically for the creation of new methods and tools [8]. Thus, the use of the earlier taxonomy also avoids some confusion for work based on the SWECOM. Finally, since much of the software engineering process involves creation of some kind, we found that the use of the older taxonomy was more helpful in distinguishing the different ways in which SEPTs were used.

Bloom's taxonomy has been used to analyze parts of the SWEBOK Guide before, helping to characterize software engineers' proficiency in different knowledge areas over time [9], [13] or clarify objectives in specific knowledge areas [14]. The taxonomy has also been used to categorize existing research in software engineering education, using the knowledge areas of the SWEBOK Guide as an organizing framework [11]. Prior work applying Bloom's taxonomy to the SWEBOK Guide pointed out a need to identify practices and tools and made preliminary attempts to do so, but did so by bringing in additional theoretical frameworks beyond Bloom's taxonomy [15]. We only use Bloom's taxonomy to identify SEPTs in the SWEBOK Guide, as the levels from Application upward can be used to differentiate concrete actions or tools from factual recall or conceptual understanding.

III. Methodology

Our study is a qualitative document analysis on the SWEBOK Guide. Below, we describe our detailed study design.

A. Object of Study

Our object of study is Version 3 of the SWEBOK Guide [7], which as of the time of writing is the most recently published version; although a Version 4 is under development, it has not been finalized. The guide traces its roots back to early efforts to classify and organize software engineering knowledge [16], and is seen today as an overview of a generally accepted body of knowledge for practicing software engineers, compiled with extensive input from both academia and industry [7], [13]. It is also used as a basis for the more skills-focused SWECOM [8] and the undergraduate-focused Software Engineering Education Knowledge (SEEK) within curricular guidelines for software engineering [4], thus representing a viable set of SEPTs that we might incorporate into undergraduate engineering education in the future.

Version 3 of the SWEBOK Guide is divided into 15 knowledge areas, of which 9 are specifically relevant to software engineering: (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**. The other areas cover concepts that might be reasonably found in other engineering disciplines (e.g., project management, professional practice, economics, or mathematical foundations). These knowledge areas are further divided into topics and subtopics that outline more detailed knowledge.

Though previous work has aimed to assess the use of knowledge from these areas [6], the methods for doing so have used only the titles of knowledge areas and topics, which can be vague: for example, the Software Requirements area has a topic titled Practical Considerations, which refers specifically to managing changes to software requirements or the process used to work with them, as well as to analyzing the sources and effects of specific requirements [7]. The content of these knowledge areas and topics (rather than just their titles) provide a starting framework we can use to organize the SEPTs identified in this study into a taxonomic hierarchy.

B. Positionality

Both authors are engineering educators at institutions of higher learning in the United States, and almost exclusively teach undergraduate students within engineering. Neither author teaches within a computer science or software engineering program, though both authors do teach programming within an engineering context. One of us (Matsumoto) also has extensive experience in computing and primarily teaches software courses, while the other (Jarvie-Eggart) has extensive experience in environmental engineering.

C. Analytical Approach

We conducted a thematic analysis on the SWEBOK Guide, using deductive coding [17] to identify SEPTs as described in the SWEBOK Guide. Deductive codes were derived from the four upper levels of Bloom's taxonomy (Application, Analysis, Synthesis, and Evaluation) and applied to the knowledge areas, topics, and subtopics of the SWEBOK Guide. Codes were applied at the level of one or more sentences in the text, and multiple codes could be applied to the same text. Definitions and examples of our codes are shown in Table I.

We (the authors) coded all of the data independently. To ensure intercoder agreement, we held coding resolution meetings to explain our coding decisions and, if we disagreed, discussed our interpretations of the data and adjusted our coding decisions as necessary. For the data we presented here, we reached complete interpretive convergence (i.e., we mutually agreed on every coding decision presented).

We then identified specific SEPTs from the coded text and organized them based on the knowledge area, topic and/or subtopic of the SWEBOK Guide in which the text originally appeared, with one exception: many knowledge areas have a

TABLE I
DEFINITIONS AND EXAMPLES FOR CODES DERIVED FROM BLOOM'S TAXONOMY, AS USED IN THE ANALYSIS FOR THIS STUDY.

Code	Operationalization	Example from SWEBOK Guide (Practice Name)
Application (Ap)	Practice that directly uses knowledge of a definition or concept	"Requirements can be classified on... whether the requirement is functional or nonfunctional." (Requirements classification)
Analysis (An)	Practice that breaks information or concepts down, relates different components to each other, or generates evidence to support generalization	"A requirement should be traceable backward to the requirements and stakeholders that motivated it... Conversely, a requirement should be traceable forward into the requirements and design entities that satisfy it... and into the test case that verifies it." (Requirements tracing)
Synthesis (S)	Practice that joins diverse pieces of information to draw patterns or create new meaning	"... the requirements specialist needs to mediate between the domain of the stakeholder and that of software engineering." (Mediation among process actors)
Evaluation (E)	Practice that uses some criteria to make and/or defend an opinion	"... necessary, not only as a means to filter important requirements, but also in order to resolve conflicts and plan for staged deliveries, which means making complex decisions that require detailed domain knowledge and good estimation skills. (Requirements prioritization)

topic listing tools in that area, and while they generally include all of the tools mentioned elsewhere in the knowledge area, in some instances they do not. For those tools, we placed them in the tools topic of that knowledge area, indicating so in the knowledge area it was derived from. In some sections, the same SEPT was mentioned and identically coded multiple times; these SEPTs were presented only once in their respective sections. Some SEPTs appear in multiple sections (e.g., in the introduction of a topic and specific subtopics); these were listed multiple times.

IV. PRELIMINARY RESULTS

Here, we present preliminary results from our ongoing analysis: specifically, SEPTs in Software Requirements, a knowledge area perceived by STEM practitioners outside of computing as being especially important, but not well understood [18], as well as one in which little analysis with Bloom's taxonomy has been done [11].

A. Overview of Results

The frequency of use of each code (coding count) is shown in Table II, grouped by the eight topics and 29 subtopics of the Software Requirements knowledge area. Topics or subtopics marked *General* are not included in this count, but rather indicate SEPTs identified from the introductory text of the knowledge area or topic. In total, we identified 92 SEPTs in the knowledge area. Though there is some overlap among SEPTs as we describe below, we see a relatively large number of SEPTs that are generally accepted[1] in the software engineering discipline, even in a single knowledge area.

The levels of Bloom's taxonomy among the SEPTs are relatively balanced overall within the knowledge area, though Analysis appears somewhat more frequently. Within topics, however, the levels of Bloom's taxonomy help indicate how SEPTs might be used. For example, the *Software Requirements Fundamentals* topic, which largely presents key definitions and concepts within the knowledge area, has Application-focused SEPTs that apply these definitions and concepts, such as "Express requirement for software quality" (Ap). The Requirements Elicitation and Requirements Analysis topics contain more than half of all Analysis codes (and almost half of all SEPT overall), such as "Determine requirement scope (i.e., affects which software components and to what extent)" (S, An). The *Practical Considerations* topic, which focuses on managing changes to requirements, has SEPTs focusing on Synthesis or Evaluation, such as "Create process to assess and approve requirements changes" (E, S, An).

Because the SWEBOK Guide focuses on describing *knowledge* rather than practices and tools, at times a topic or subtopic is only described very generally, or in a way that references a different knowledge area or topic within the guide. For example, in the subtopic *Change Management*:

> This topic describes the role of change management, the procedures that need to be in place, and the analysis that should be applied to proposed changes.

While the topic is more broadly about managing changes to requirements, specific practices around change management are not mentioned. We typically only inferred practices and tools that are explicitly mentioned in the text, and thus this SEPT was described as "Analyze proposed requirements changes" and coded as Analysis (An).

B. Overlap in SEPTs

Some SEPTs are mentioned multiple times within the knowledge area; these are counted for each topic/subtopic in which they are mentioned. For example, consider the definition of a software requirement:

> Software requirements express the needs and constraints placed on a software product that contribute to the solution of some real-world problem.

This definition appears (albeit with slightly different wording) both in the introductory text and in the subtopic *Definition of a Software Requirement*. In both, we describe this as the practice "Express a need or constraint on a software product" and code the SEPT as Application (Ap).

[1]Here, "generally accepted" means something that a software engineer could be reasonably expected to do in practice, but not necessarily something that is widely used.

TABLE II
NUMBER OF SEPTS (#) AND EACH CODE BY TOPIC AND SUBTOPIC.

Topic/Subtopic	#	Ap	An	S	E
General	1	1	0	0	0
SW Reqs. Fundamentals (Total)	11	8	4	4	2
Definition of a Software Req.	4	2	2	1	1
Product and Process Requirements	2	1	1	0	0
Functional/Nonfunctional Reqs.	2	2	0	0	1
Emergent Properties	1	1	0	1	0
Quantifiable Requirements	1	1	1	1	0
System Reqs. and Software Reqs.	1	1	0	1	0
Requirements Process (Total)	8	2	3	4	5
Process Models	4	2	2	1	3
Process Actors	2	0	0	2	1
Process Support and Management	1	0	1	1	0
Process Quality and Improvement	1	0	0	0	1
Requirements Elicitation (Total)	21	4	15	10	7
General	4	0	2	4	1
Requirements Sources	8	3	3	3	3
Elicitation Techniques	9	1	9	3	3
Requirements Analysis (Total)	21	8	11	6	5
General	1	0	1	0	0
Requirements Classification	7	3	5	1	2
Conceptual Modeling	5	2	1	2	0
Arch. Design and Reqs. Allocation	2	0	1	2	0
Requirements Negotiation	3	0	2	1	3
Formal Analysis	3	3	1	0	0
Requirements Specification (Total)	8	4	3	2	3
General	1	0	1	0	0
System Definition Document	1	1	0	1	0
System Requirements Specification	1	0	1	0	0
Software Reqs. Specification	5	3	1	1	3
Requirements Validation (Total)	5	2	2	0	5
General	1	1	0	0	1
Requirements Reviews	1	0	0	0	1
Prototyping	1	1	0	0	1
Model Validation	1	0	1	0	1
Acceptance Tests	1	0	1	0	1
Practical Considerations (Total)	14	2	6	8	7
General	2	0	0	2	2
Iterative Nature of Reqs. Process	6	1	2	5	4
Change Management	1	0	1	0	0
Requirements Attributes	2	0	1	0	1
Requirements Tracing	2	0	1	1	0
Measuring Requirements	1	1	1	0	0
Software Requirements Tools	3	3	0	0	0
Total	92	33	44	34	34

Some SEPTs essentially refer to the same practice, but in different topics/subtopics and for different purposes. For example, in *Requirements Classification*:

> The requirement priority. The higher the priority, the more essential the requirement is for meeting the overall goals of the software. Often classified on a fixed-point scale such as mandatory, highly desirable, desirable, or optional, the priority often has to be balanced against the cost of development and implementation.

We labeled this practice as "Assign a priority value to a requirement" (coded as E, An). Elsewhere, in *Requirements Negotiation*:

> Requirements prioritization is necessary, not only as a means to filter important requirements, but also in order to resolve conflicts and plan for staged deliveries, which means making complex decisions that require detailed domain knowledge and good estimation skills.

We labeled this practice as "Prioritize requirements to resolve conflicts" (E). Later in that section, further text describes the prioritization process in more detail, which we labeled as "Analyze relative values and costs of requirements to determine requirement priority" (E, An). These distinctions can be useful to educators, providing insight into an appropriate level of depth to cover a SEPT, based on the use and topic.

C. Comparison with Prior Work

In comparison to prior work, our results generally provide greater clarity and detail about SEPTs. For example, in *Requirements Elicitation*, the SWECOM lists four activities [8]:

- Identifies stakeholders for elicitation of requirements.
- Engages stakeholders in elicitation of requirements.
- Uses appropriate methods to capture requirements.
- Negotiates conflicts among stakeholders during elicitation.

The SEEK, which provides guidance for undergraduate software engineering curricula, lists two areas [4], at the Bloom levels of Comprehension and Application, respectively:

- Elicitation sources (e.g., stakeholders, domain experts, and operational and organization environments)
- Elicitation techniques (e.g., interviews, questionnaires/surveys, prototypes, use cases, observation, and participatory techniques)

By contrast, the SEPTs we identified are shown in Table III. In comparison to both the SWECOM and SEEK, our results provide a more detailed description of elicitation and stakeholder engagement methods, including techniques such as posing specific scenarios and facilitating meetings to refine requirements with stakeholders. We also identify a greater set of requirements sources, including business rules, organizational structure, and the application domain, (though the SEEK provides some of this detail). This more detailed list of SEPTs provides engineering educators with concrete practices that they may choose to cover in their courses.

V. DISCUSSION AND IMPLICATIONS

Based on our preliminary results, analyzing the SWEBOK Guide using Bloom's taxonomy helps both to validate prior efforts such as the SWECOM and SEEK, while also providing more concrete detail on generally accepted SEPTs that engineering educators may adopt in their teaching, even outside of software-centric disciplines. Our results also shed some light on how these SEPTs may be used in the software development process, as well as the types of skills that instructors may

TABLE III
SEPTs WITHIN THE REQUIREMENTS ELICITATION TOPIC.

General
Build understanding of the problem by collecting requirements (S, An)
Effectively communicate among different stakeholders (S, An)
Form project scope and prioritize deliverables according to customer business needs (E, S)
Scale/extend list of requirements over time (S)
Requirements Sources
Identify/evaluate potential requirements sources (E, An)
Assess value (relative to priority) and cost of overall software goals (E, An)
Conduct a feasibility study to assess value and cost of goals (E)
Collect and/or structure relevant concepts in the application domain (S)
Identify, represent, and manage various stakeholder viewpoints (S, An)
Apply business rules to generate requirements (Ap)
Analyze, and derive requirements from, software operational environment (An, Ap)
Adjust requirements based on organizational structure, culture, politics (S, Ap)
Elicitation Techniques
Collect information and formulate requirements from requirements sources (S, An)
Analyze elicited information and relevance to requirements (An)
Apply techniques to make tacit business/technical requirements visible (An, Ap)
Interview stakeholders (An)
Pose scenarios (e.g., "what if", "how is this done") to elicit requirements (An)
Create and present prototype to stakeholders to elicit/clarify requirements (E, An)
Facilitate meetings to brainstorm and refine requirements information (E, S, An)
Observe stakeholders in their typical environment (An)
Elicit user stories and create acceptance procedures to formulate high-level requirements (E, S, An)

expect students to develop in topics within a knowledge area, such as requirements analysis.

Nevertheless, we recognize some limitations of our approach: our object of study is based on a software-centric perspective of practice that may not match the context in which other engineering disciplines develop software, and the use of Bloom's taxonomy levels may provide overly ambitious expectations for what students in these disciplines should learn in terms of SEPTs. We also do not yet know if our list of SEPTs will indeed be more accessible to engineering educators, as outlined in RQ2.

Theoretical and methodological triangulation may help address some of these limitations. For example, a more open-ended study using surveys or interviews to collect SEPTs from practicing engineers of different disciplines would provide a view of SEPTs that better reflects actual use; this is especially relevant for practices and tools that have emerged since the publication of the latest SWEBOK Guide. Dittrich uses philosophical argumentation to explore the relationship between methods and practice in software development [19], which may help additionally clarify and classify SEPTs.

While our results are promising, more work remains to be done to get clearer answers to our research questions. Future work as described above may complement our approach, helping us better assess the SEPTs that undergraduate engineering students use in their work. In turn, this will help us identify important opportunities to better teach SEPTs in undergraduate engineering curricula and better understand the impacts of those improvements to teaching.

REFERENCES

[1] A. Vee, *Coding Literacy: How Computer Programming Is Changing Writing*. The MIT Press, Jul. 2017.

[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] Joint Task Force on Computing Curricula, IEEE Computer Society, Association for Computing Machinery, "Software engineering 2014: Curriculum guidelines for undergraduate degree programs in software engineering," *Computing Curricula*, Feb. 2015.

[5] P. Wong and B. Pejcinovic, "Teaching MATLAB and C programming in first-year electrical engineering courses using a data acquisition device," in *ASEE Annual Conference and Exposition*, Seattle, WA, Jun. 2015.

[6] 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.

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

[8] IEEE Computer Society, "Software engineering competency model," 2014.

[9] P. Bourque, L. Buglione, A. Abran, and A. April, "Bloom's taxonomy levels for three software engineer profiles," in *IEEE International Workshop on Software Technology and Engineering Practice (STEP)*, Sep. 2003, pp. 123–129.

[10] M. D. Engelhart, W. H. Hill, E. J. Furst, and D. R. Krathwohl, *Taxonomy of educational objectives: The classification of educational goals, Handbook 1: Cognitive domain*, B. S. Bloom, Ed. New York and London: Longman, 1956.

[11] R. Britto and M. Usman, "Bloom's taxonomy in software engineering education: A systematic mapping study," in *IEEE Frontiers in Education Conference (FIE)*, Oct. 2015.

[12] L. W. Anderson and D. R. Krathwohl, Eds., *Taxonomy for Learning, Teaching, and Assessing, A: A Revision of Bloom's Taxonomy of Educational Objectives, Complete Edition*, 1st ed. Pearson, Dec. 2000.

[13] P. Bourque and R. Dupuis, Eds., *Guide to the Software Engineering Body of Knowledge, 2004 Version*. IEEE, 2004.

[14] F. Robert, A. Abran, and P. Bourque, "A technical review of the Software Construction knowledge area in the SWEBOK Guide," in *IEEE International Workshop on Software Technology and Engineering Practice (STEP)*, Oct. 2002, pp. 36–42.

[15] M. Azuma, F. Coallier, and J. Garbajosa, "How to apply the bloom taxonomy to software engineering," in *IEEE International Workshop on Software Technology and Engineering Practice (STEP)*, Sep. 2003, pp. 117–122.

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

[17] J. Saldaña, *The coding manual for qualitative researchers*, 3rd ed. SAGE, 2016.

[18] 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)*, Mar. 2018, pp. 582–591.

[19] Y. Dittrich, "What does it mean to use a method? towards a practice theory for software engineering," *Information and Software Technology*, vol. 70, pp. 220–231, Feb. 2016.