12-9-2022

Comparing importance of knowledge and professional skill areas for engineering programming utilizing a two group Delphi survey

John F. Hutton
Mississippi State University, jfh232@msstate.edu

John F. Hutton
Mississippi State University, john.f.hutton@gmail.com

Comparing importance of knowledge and professional skill areas for engineering programming utilizing a two group Delphi survey

By

John F. Hutton

Approved by:

Jean Mohammadi-Aragh (Major Professor)
John E. Ball
Bryan Jones
Chaomin Luo
Jenny Du (Graduate Coordinator)
Jason M. Keith (Dean, Bagley College of Engineering)

A Dissertation
Submitted to the Faculty of
Mississippi State University
in Partial Fulfillment of the Requirements
for the Degree of Doctor of Philosophy
in Electrical and Computer Engineering
in the Department of Electrical and Computer Engineering.

Mississippi State, Mississippi

December 2022

Name: John F. Hutton

Date of Degree: December 9, 2022

Institution: Mississippi State University

Major Field: Electrical and Computer Engineering

Major Professor: Jean Mohammadi-Aragh

Title of Study: Comparing importance of knowledge and professional skill areas for engineering programming utilizing a two group Delphi survey

Pages in Study: 166

Candidate for Degree of Doctor of Philosophy

All engineering careers require some level of programming proficiency. However, beginning programming classes are challenging for many students. Difficulties have been well-documented and contribute to high drop-out rates which prevent students from pursuing engineering. While many approaches have been tried to improve the performance of students and reduce the dropout rate, continued work is needed. This research seeks to re-examine what items are critical for programming education and how those might inform what is taught in introductory programming classes (CS1). Following trends coming from accreditation and academic boards on the importance of professional skills, we desire to rank knowledge and professional skill areas in one list. While programming curricula focus almost exclusively on knowledge areas, integrating critical professional skill areas could provide students with a better high-level understanding of what engineering encompasses. Enhancing the current knowledge centric syllabi with critical professional skills should allow students to have better visibility into what an engineering job might be like at the earliest classes in the engineering degree. To define our list of important professional skills, we use a two-group, three-round Delphi survey to build consensus ranked lists of knowledge and professional skill areas from industry and academic

experts. Performing a gap analysis between the expert groups shows that industry experts focus more on professional skills then their academic counterparts. We use this resulting list to recommend ways to further integrate professional skills into engineering programming curriculum.

DEDICATION

Dedicated to my wife, Marlene Hutton who has been my most stalwart supporter, and encourager. I could have done none of this without her. Praise to God the Father and Christ my savior who always sustains me.

ACKNOWLEDGEMENTS

Microsoft and my managers supported my dissertation work through their continuing education benefit. Both their financial support as well as ongoing encouragements helped make this a reality.

I am deeply grateful to all my survey participants. My engineering friends and peers took time out of their busy days to complete all the surveys. They also endured some hiccups at the start as I was learning the survey system. My academic experts agreed to help based on a blind email, and most of them stuck around through all three rounds despite my timing being near the end of the spring semester and start of summer. Without all my experts' diligence I would not have been able to complete this work.

Thanks to each of my committee members for their help and support. They have been by me through many changes and several delays. Special thank to Prof. Jean Mohammadi-Aragh for her support of this remote student for the past four plus years. Being allowed to come alongside for her NSF grant work is an honor.

My wife, Marlene, has been enduring upwards of five years of me working full time and PhDing part time. She has also had the pleasure (or pain) of editing every word I have written throughout this process (though any remaining mistakes are solely mine).

TABLE OF CONTENTS

APPENDIX

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

1.1 Introduction

Engineering degrees, as well as engineering careers, depend on some proficiency with programming. While most engineers will take several programming classes in pursuit of their degree, as well as using programming to solve problems in some of their engineering classes, learning programming knowledge must be coupled with development of professional skills to be a truly effective engineer. While we would like to consider recommendations which apply at the engineering degree level, we are narrowing our research to engineering programming. As we consider the challenges, even at this narrowed scope, we begin our problem analysis by looking at introductory computer programming, often referenced as CS1.

CS1 is one of the fundamental courses engineering students take early in their college career. Programming can be difficult to learn [1] and some percentage of students fail or drop out [2]. Motivations for dropping out are complex [3]. One area of current research is trying to assess what the difficulties are in both teaching and learning CS1 material [4]. Another orthogonal area of research is searching for ways to improve or enhance the curriculum. Examples here are adding methods like peer instruction [5], [6], adding gaming to the programming content [7], and including automation in assistive programming tools and assessments grading [8]–[11]. Research continues because we do not yet have a complete understanding of how to improve programming instruction that can translate to the variety of programming classes, teachers, and students. This

dissertation implements a survey with supporting methods and analysis, to discover possible focus areas that could enhance student motivation and improve performance in engineering programming courses.

1.2 Brief history of CS1 course content

To study what content areas a CS1 course covers, we start at a high level by considering what has changed in CS1 curricula over the past twenty years. Pears et. al., in their survey of literature from 2007 [12], studied papers on CS1 courses and generalized these papers into four areas: curricula, pedagogy, language choice, and tools. Forward to 2019 where a similar paper by Becker et. al. [13] showed how paper topics have expanded to encompass eight categories. While they had the same starting four topics as the 2007 study, they sub-divided the group called "curricula" into two separate groups: CS1 design, structure, and approach; and CS1 content. In addition, they added three previously unclassified groups: collaborative approaches, learning and assessment, and students. It may not surprise anyone today that the category "students" had the largest number of papers in the 2010s. Some of the sub-divisions of the "students" group include: teaching CS1 to non-majors; student retention; gender, diversity, inclusion and accessibility; and predicting and measuring success. We see an expansion across these 20 years broadening the study of what goes into CS1 material. Considering this expanded focus where the student is key, we believe there are untapped areas for ideas of continuing to improve the CS1 course as well as engineering programming courses generally.

Looking beyond literature reviews into curriculum content, Becker and Fitzpatrick [14], reviewed 234 CS1 syllabi from 207 institutions and evaluated all of the learning outcomes to create a list of 54 key concepts. Of these, 52 were knowledge-based items like testing and debugging, writing programs, and if/then statements. The two remaining concepts were problem

solving, and teamwork and communication. These two concepts are not knowledge areas; they are *professional skills*. These skills are learned, but they have a much broader application than a knowledge-based item like “writing programs”. “Problem solving” is critical for a programming course, but it is also critical for almost every engineering course. We will define professional skills more clearly later in this chapter. We believe these skills represent the tip of the iceberg when it comes to a full set of professional skills that are important to both programming and engineering.

1.3 Defining knowledge and skills

Before we proceed to look deeper into the area knowledge and skill areas in the next chapter, a brief definition of knowledge and skills is needed.

1.3.1 Knowledge areas

The need for knowledge in programming is irrefutable. Programming knowledge includes needed facts and information about computers, programming languages, and programming concepts. As mentioned from the Becker syllabi review, the authors found 52 different categories of knowledge that show up in a cross-section of CS1 course curricula. The knowledge differential between a beginning programmer and an expert programmer is substantial. CS1, as one of the first programming classes, lays the foundation of programming knowledge that helps move students toward gaining all the knowledge needed to become solid programmers.

The dictionary definition of knowledge is “acquaintance with facts, truths, or principles, as from study or investigation”. Engineering knowledge includes the concepts that are required

to solve engineering problems across the spectrum of engineering fields. In the Computer Curricula 2020 report:

> Knowledge is that "know-what" dimension of competency that can be understood as factual. An element of knowledge designates a core concept essential to a competency. [7 p125]

One challenge when teaching a CS1 class is selecting the correct knowledge areas to build the foundation of the students' current and future programming courses. Some specialized knowledge, like the syntax of one particular programming language, is required to write programs. This is needed to be able to teach broader concepts. While some time must be dedicated to this learning, deep mastery of syntax has limited application outside that particular language. Curriculum planners and teachers must strive to balance the specific knowledge required to perform the work of the class with the time spent focusing on broader foundational concepts. In his paper Learning to Program is Easy, Andrew Luxton-Reilly concludes: "Our current approach to teaching programming is to cover too much content too rapidly and expect students to be able to program at a higher level than they are capable of achieving at the end of an introductory programming course" [16]. We will revisit knowledge areas in both our review of literature and our methods chapter. Ultimately, we build a specific list of knowledge areas to support our survey goals.

1.3.2 Professional skills

While there is no doubt that knowledge is critical for a CS1 study, the idea of what professional skills are required is much less formalized. Professional skills appear to be as important, or even more important, than many of the knowledge areas currently found in CS1 curricula. Professional skills have been receiving significant attention recently as both a critical

and underappreciated part of engineering education. In her paper A hard stop to the term "soft skills", Berdanier states:

> In recent generations, these competencies that prioritize human interaction have been labeled as "soft skills" or "nontechnical skills," rhetorically separated from "hard" or "technical" engineering even though they are essential for engineers to thrive. [17]

Some skills, such as critical thinking and problem-solving, deal with how we utilize our mind. Other skills, such as teamwork and communication, focus more on how we express our thoughts and interact with others. We also classify as skills personal aptitudes or dispositions such as resilience, creativity, and persistence. In South Africa, research has focuses on "graduateness" which comprise as a set of professional skills that employers expect from someone who has graduated with a college degree. Many accreditation organizations are also including skills as part of their evolving recommendations and requirements. But there is still work to do. In the Becker CS1 curricula study, only two skill items were included in the 54 aggregated concepts. While knowledge is critical, we posit that several professional skills are equally important and should be integral to both engineering programming courses and engineering degree courses.

1.4 Should professional skills be added to engineering programming curricula?

We believe there should be a combination of knowledge and professional skills areas included in teaching/learning goals of a engineering programming classes. To make this argument, we need to look inside and outside the classroom experience. As engineering education has a primary focus on training engineers for industry jobs, we should understand expectations from industry. In addition, the experiences and opinions of teachers and academicians control the content and teaching of CS1 courses. Understanding their expectations and experiences is also mandatory.

1.4.1 Industry

What the student learns throughout their engineering degree program, prepares them for jobs in industry. Classes like CS1, which are completed early in the college experience, should fill a twofold purpose. First, the course should transfer knowledge that will combine with other courses to help make capable programmers and engineers. This is the focus of most CS1 courses today as we saw from the Becker and Fitzpatrick syllabi review [14]. The second purpose should be to help prospective engineers discover if they truly want to pursue engineering as a career. This is no easy task. In addition to knowledge areas, students need to understand what professional skills are necessary for engineering and programming. Learning specific knowledge, like programming language syntax, does not provide much insight into what an engineer does as part of an industry job. However, learning to work with a team, learning how to solve problems, or learning now to be creative might be much more indicative of what a future job in engineering would look like. The more job-like experiences students participate in, the better they can see themselves fitting into an industry setting. Academic jobs also rely on professional skills daily. Learning how to work on a team in industry is like learning how to work on a team of professors or a team of researchers.

Engineering jobs, like most jobs, involve a combination of knowledge, tasks, interactions, goals, people, time, and skills. It is rare that any engineer would spend 100% of their time on purely technical tasks. In many settings, engineers may have seasons where they only spend half of their time engaged in technical design work like programming. This means that even great engineering jobs may have up to 50% of their time engaged in non-technical tasks. These items range from meetings, one-on-one interactions with peers, giving presentations, mentoring, being mentored, working on budgets, figuring out program schedules, managing email, and many other

varied tasks. It would be extremely difficult to simulate all these items in any engineering course. This may be why internships and coops can be valuable to students [18]. Learning and practicing some of these professional skills may do more to help a student understand if they are interested in mastering engineering than simply mastering knowledge. Extending this concept throughout the entire program, as outlined in many accreditation board standards today, may be necessary to produce engineers that can face the challenges of the future.

1.4.2 Academia

Instructors who develop and teach engineering courses in general, and CS1 classes in specific, spend time carefully building a syllabus, preparing lessons, teaching classes, grading homework and tests, and striving to give individual help and attention to all their students. They have a vested interest in helping their students be successful. With so much knowledge that could be provided, it can feel like sacrifices must be made on what is included and what is excluded. Again, from the syllabi review of Becker and Fitzpatrick [14], most instructors focus on knowledge.

While knowledge is required, we believe most professors understand the benefit and necessity of professional skills. Even if not expressly called out in their course goals, they include teaching professional skills, explicitly or implicitly, that they believe will be helpful to their students. As we have seen from our 2007 and 2019 survey of literature papers, research has been moving towards student needs as a key component of what ought to be taught. As we look to our industry experts to discover their ranked list of knowledge and professional skills needed for a programming job, we look to our academic experts with the same questions. How teachers answer these questions reflects what they would naturally strive to highlight in a class. If

professional skills rank highly among academic experts, there should be more work forming these into measurable goals for courses like CS1.

1.4.3 Understanding gaps between industry and academia

After understanding the rankings from industry and academic experts, a logical question is, "Do academic experts and industry experts agree on what knowledge and professional skills should be taught in engineering programming courses?" If they completely agree, then we have large common ground on what is, and should be, taught. However, if we have any gap between the two expert groups, that gap highlights a prime area to consider as fruitful areas for change.

This dissertation attempts to build an ordered list of what knowledge and professional skill areas are important for programming by surveying both industry and academic experts. This is expected to be somewhat difficult. In industry, there is a diversity of individual programmers as well as a breadth of programming positions. In academia there are many schools of thought on what should be taught in class, as well as what pedagogical techniques are best. In addition, every instructor has their own ideas and opinions around what they have found that works.

In addition to the challenge of gathering subjective data, we further desire to take this data and find a way to arrange it in a ranked list. This is another daunting task. If multiple individuals are asked to rank a list, every individual may have a different ranking. We need some way to allow our experts to build consensus. A method is needed which can help search for group consensus among a large potential list of knowledge and skills. We believe the Delphi Technique is the right tool for this task. This tool, originally developed in the 1960s, has been used specifically for the task of consensus-building among groups of experts. We consider the history of the Delphi survey in our next chapter and outline the method we utilize in chapter three.

1.5 Problem statement

Engineering classes should both teach the subject and help students decide whether they are well suited to an engineering degree and career. For courses like CS1, are there changes to the curriculum which would improve the student's ability to do this? CS1 is also the starting class which leads into other programming classes and content. The programming component of an engineering degree is also only a minor part of the entire collection of courses which make up an engineering degree. Our survey focuses on what is needed to be successful as a new hire. From this list, we can consider what items might be useful at both the engineering level, engineering programming sub-level, and, finally, at the CS1 level. Bloom's Taxonomy, in almost all its iterations, focuses on proving the necessary time for students to master the knowledge they are learning [19]. This means the most critical items generally need the most exposure. Whatever items are at the top of our list would be likely to fit in the CS1 course so they could be re-emphasized several times throughout the engineering degree.

We need a rank-ordered list of the knowledge and professional skill areas. We believe doing a carefully constructed Delphi survey will allow us to assemble this list. Including both academic and industry expert groups will allow us to compare and contrast these two lists to identify any knowledge and professional skill area gaps. This brings us to our primary hypothesis.

- H1: Academic experts and industry experts will have one or more gaps regarding critical knowledge and professional skill areas required for programming in an industry engineering position.

If we had complete agreement between academics and industry, it would be straightforward to assemble one list. Recent work by Groeneveld indicates that there is a skills gap between non-technical skills needed for programming versus what was taught in

undergraduate classes [20]. There were some limitations to this study. First, the study focused only on non-technical skills without including knowledge. While this is valuable, it has limited application for our purpose. In addition, while the study touched 11 countries and 21 companies and universities, all the interviews were done in Dutch. Expanding the coverage into English should help confirm the generalizability of their results. Finally, their single group combined industry and academic experts into one group. As we will highlight in our methods chapter, separating the two groups and doing a gap analysis should give more clarity to the results. Still, the results of this study initially confirm our hypothesis.

For our research, we break the problem down into three primary research questions.

- RQ1: According to industry experts, what are the most important knowledge and professional skills to consider for an industry programmer?
- RQ2: According to academic experts, what are the most important knowledge and professional skills to consider for an industry programmer?
- RQ3: What is the gap between industry and academic experts in their answers to these questions?

To answer the first two research questions, we will conduct two separate Delphi surveys. Each will focus on building two group consensus ranked lists from the industry and academic experts. Once we have the two results, we will evaluate question three by analyzing the gap between the two expert groups. From this data, we hope to be able to propose an answer to our final research question.

- RQ4: Is there knowledge or a set of skills which should be emphasized or deemphasized in a CS1 curriculum which could give students a better ability to know whether engineering is a degree they want to pursue?

The remainder of this dissertation is organized as follows: Chapter 2 is a review of literature that outlines current research on computational thinking that leads into an expanded review of what we call knowledge areas. In our research on professional skills, we review

several areas to build a broader definition of professional skill areas. The chapter also reviews the method and history of Delphi surveys. Finally, we summarize several pedagogical techniques to inform any final proposals for curricular changes.

Chapter 3 details the specific methods used in this dissertation. We construct a classification framework to allow a straightforward method to distill open-ended answers to individual knowledge and professional skill areas. We also outline the specifics of our Delphi Survey. This includes discussion of our initial open-ended questions, which are the foundation of any Delphi survey. We also describe our addition of classification questions as one of the ways we can understand any data in the event we do not have complete consensus among each of the groups.

Chapter 4 presents results from our industry group of experts, and Chapter 5 details similar statistics from our academic group.

Chapters 6 concludes with discussions, recommendations, and future work.

CHAPTER II

REVIEW OF LITERATURE

2.1 Current state of CS1 courses

While arguments have been made for adjusting the total knowledge content of CS1 courses in light of achievable outcomes [21], [22], the literature review by Medeiros [23] calls out problem solving, background knowledge, and better tools as key factors for improving the teaching of programming. If our industry/instructor gap is weighted towards skills, we must do a deeper dive into the current research around non-technical skills. What are they? How do they rank? This is not a brand new or novel branch of research. Many papers on skills in recent years still focus on programming skill [24], [25] instead of the non-technical skills that have engineering applications outside of programming. South Africa, however, has conducted significant research into this area over the past ten years. In addition to this work, both the joint committee on computing curricula published by ASM/IEEE-CS, and the Accreditation Board for Engineering and Technology (ABET) have started calling out skills in their recent publications. With examples from all these sources, we will begin to build a deeper list of what skills we expect to arise when we start surveying our Industry and Academic experts.

2.2 Research on knowledge

Knowledge is a broad subject. Even trying to specialize around knowledge needed for programming still covers a large area of ground. Computational thinking is a concept that

abstracts how computers and computer programs execute into a model for problem solving. This can be indispensable knowledge for learning about the art of programming.

2.2.1 Computational thinking

In her seminal article from 2006, Janette Wing defined the educational aspect of computational thinking as involving "solving problems, designing systems, and understanding human behavior, by drawing on the concepts fundamental to computer science" [26]. This work has led to the concept of computational thinking (CT) applied at every level of the educational spectrum. At the elementary level, there is a child-friendly programming environment called Scratch which "enables young people to create their own interactive stories, games, and simulations, and then share those creations in an online community with other young programmers from around the world" [27]. Barr et al ask the questions "how can we make CT accessible" and "why is CT important" [28]. Their conclusion is that computer technology has entered every field and CT helps students learn how to leverage computers to solve daily problems. In the book Computational Thinking Education, multiple authors detail what CT is, discuss how to assess competency, and provide many specific K-12 examples. They end with educational policy and implementation recommendations which "indicate the importance of good policies and good planning in facilitating everyone in learning to think computationally [29].

However, there is not universal understanding or adoption of CT in education. While organizations such as CSTA, Computing at School, and ISTE have sought to clearly define computational thinking, Denning in his article "Remaining Trouble Spots with Computational Thinking" believes the definition remain vague [30]. He also advises teachers "use competency-based skill assessments to measure student progress" while being "wary of the claim of universal value". In their review of literature, Shute and Sun believe that an agreed-upon definition is lacking,

which they attribute to the immaturity of the field. Their identification of "six main facets: decomposition, abstraction, algorithm design, debugging, iteration, and generalization" [31], however, seems to be a reasonable summary of several key concepts in CT. Tedre and Denning, in their review, strive to both clarify claims that are exaggerated while highlighting "risks looming over CT" [32]. Finally, Angeli and Giannakos, in their short article about the issues and challenges of computational thinking education made this note:

> While it is well accepted in the literature that CT involves a number of skills, like problem decomposition (breaking down complex problems to simpler ones), developing algorithms (step-by-step solutions to problems), and abstraction, there is still limited evidence around the several issues and challenges someone needs to be aware of in order to design appropriate learning experiences for CT competences. [33]

We can see that the general field of CT for general educational application is not completely clear or settled.

While Computational Thinking may have some struggles as a general educational topic, what about application to programming and engineering? In this area, the direct linkage between computational thinking as a methodology to problem solve is much clearer. The scope initially called out by Janette Wing and the six facets extracted from literature by Shute and Sun provide some good fundamentals. Li pointed out that "programming was the most appropriate way for expressing CT" [34]. Gross et al argue that CT is a core capability for most engineers [35]. They link CT to recommendations by the National Academy of Engineering that "the essence of engineering—the iterative process of designing, predicting performance, building, and testing—should be taught from the earliest stages of the curriculum, including the first year" [36]. While the linkage between CT and teaching programming seems logical, there are still some struggles

on how we teach and assess CT in a programming class. Miller et al proposed that enabling computational thinking could be improved through creating thinking exercises [37]. Other recent work continues to clarify and identify useful assessment methodologies for measuring CT skills in an educational setting [38], [39]. Even in programming courses, a universal teaching of CT has not yet been accepted.

The recommendation of the NAE that engineers should be trained in the model of design, predict, build, test, is not directly a part of CT, but the concept of engineers being trained in this fundamental iterative process couples nicely with all the skill areas we have seen referenced in the definition of critical thinking. We appreciate the power of this model. Computational thinking is one way of wrapping several knowledge areas rooted in computer science along with some professional skills such as problem-solving and creative thinking.

For this investigation, however, computational thinking is not broad enough to cover the gamut of responses we may receive from our expert groups. We must expand our research to search for all the likely areas that will be brought up. We do expect that several of the principles contained in computational thinking will end up in our final lists.

2.2.2 Additional research on knowledge

Finding ways to categorize and group knowledge can be a daunting task. For this paper, we limit our study to programming knowledge. While we found no systematic analyses of programming knowledge, several papers include a list of knowledge content areas as part of their specific topic.

Qian et al researched misconceptions in introductory programming at the student and teacher level. In their student-based review of literature [40], their survey grouped misconceptions into three knowledge areas as well as seven likely causes. They then reviewed

existing strategies and tools that could address some of these problems. When they surveyed teachers [41], they evaluated the importance of PCK (pedagogical content knowledge) when it came to teacher confidence. They found 37 content areas across five topics where students struggled. The five topic areas were variables, data structures, loops, functions, and object-oriented programming.

When Schulte and Bennedsen preformed a similar review of introductory programming [42], they compiled a list of 28 topics that represented their base assumption of content that should be covered. Of this list, they had two items which might not be considered general programming knowledge concepts. The first was the “integrated development environment” (IDE) which covers items like the editor, compiler, file organization, and debugging. While these topics all have some general application, most of this information pertains to running the specific tool selected for the class to write programs. The second item was “ethics”. There was not an expanded definition for what this topic covers. Both of these areas ended up ranking very low in both importance and difficulty.

In their syllabi review, Becker and Fitzpatrick reviewed 234 CS1 syllabi from 207 institutions [14]. From their analysis of learning outcomes, they identified 52 knowledge areas and 2 professional skill areas. The following table has the top seven most identified areas out of the 234 parsed syllabi.

Table 2.1 Top areas from Becker's syllabi analysis

Becker 2019	# of Results
Writing programs	112
Testing & Debugging code	110
Control Structures & logic (if/else etc)	107
Problem Solving (and computational thinking)	106
Arrays, Lists, dictionaries, vectors, sets	93
Variables, assignment, arithmetic expressions, declarations, data types	91
Basic OOP	89

For all programming courses including CS1, writing programs as well as testing and debugging sound like what a student would need to learn. These are not clear "knowledge" items as "writing programs" is the end result of assembling all the lower pieces of knowledge to solve a problem. Testing and debugging also involves several methods and strategies. Selecting the appropriate option for a particular situation is more a matter of proper application of knowledge. The 3rd item, problem solving, is a professional skill-based item. This is one of the two in the list with the other being teamwork and communication. The final three on this list are more classical programming knowledge areas.

Engineering accrediting entities provide another source of data. The ABET curriculum requirements for engineering are too general to be of much help here [43]. They have four broad requirements, but no specific curriculum content requirements. However, in the ABET criteria for computer science [44], we find a little more help. In their 40 semester credit hours of computer science, they list several knowledge categories. However, most of these are still very high-level and not internally defined. Some items—algorithms, computer architecture and origination—match content from some of the other lists, while other items—like networking and communication or operating systems—are advanced topics.

ENAEE, the European Network for Accreditation of Engineering Education, in their EUR-ACE guidelines [45], takes a similar approach by specifying eight learning areas.

Table 2.2 EUR-ACE eight learning areas

Knowledge and understanding	Engineering Practice
Engineering Analysis	Making Judgements
Engineering Design	Communication and Team-working
Investigations	Lifelong Learning

As we will see in the next section, most of these fit more with our professional or graduate skill concept areas than traditional technical knowledge areas. Most of the Becker items would be included in "knowledge and understanding".

The IEA, International Engineering Alliance, seeks common definitions of graduate attributes and professional competencies to facilitate engineering talent to be predictable across international borders [46]. Their assessment areas are also relatively broad.

Table 2.3 IEA Graduate Attributes and Professional Competencies [46] p15-18

	Graduate Attributes
1	Engineering knowledge
2	Problem analysis
3	Design/development of solutions
4	Investigation
5	Tool usage
6	The Engineer and the world
7	Ethics
8	Individual and Collaborative teamwork
9	Communication
10	Project management and finance
11	Lifelong learning

Even with the title, these items are clearly better slated for our next section. Knowledge is generally lumped into item #1.

In the 200-page Computing Curricular Series Report of 2020 [15], we explicitly see what has been inferred in many of these curriculum examples.

> This CC2020 report encompasses most of the themes contained in its predecessor. However, the changing dynamics of computing, computing education research, and changes in the workplace have resulted in many new "add-ons" and features that did not appear in the earlier report. Some of these additions include the following:
>
> • Focusing on competency
>
> • Transitioning from knowledge-based learning to competency-based learning
>
> [15] p12

This emphasis on "competency" includes a focus on professional skills that will be integral in our next section. Knowledge content remains foundational, but without the skill and disposition to apply this to a task, it does not rise to useful engineering.

As accreditation organizations are beginning to move past knowledge towards knowledge and professional skills, or competencies, finding a summary rubric to group knowledge categories has not been a focus in recent years. In 1997, Mcgill & Volet proposed a framework for analyzing students' knowledge [47]. As the knowledge base for computer programming was still developing, they suggested that it could be valuable combining programming knowledge areas (syntactic, conceptual, and strategic) with areas from cognitive psychology literature (declarative, procedural, and conditional). Their final table grouping can be roughly represented by the following graphic:

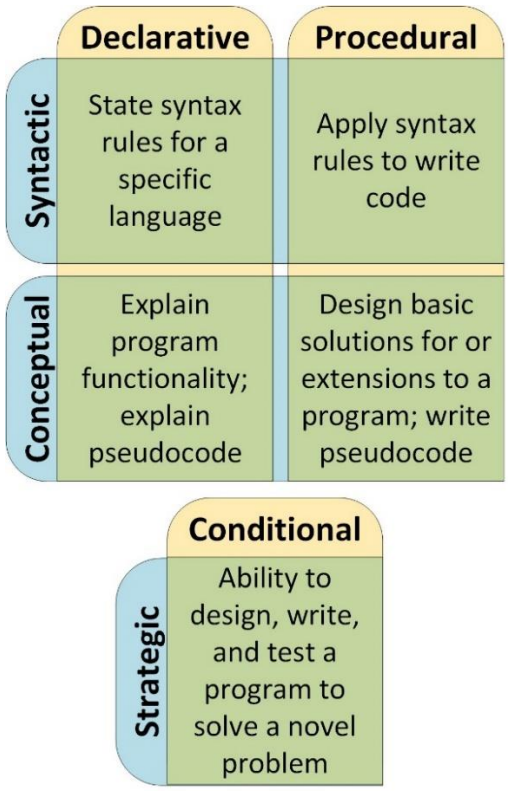


Figure 2.1 Framework for programming knowledge based on [47].

At a conceptual level, this generalization makes a lot of sense. It allows us to group things at a very high level. As we will see in our methods chapter, this grouping may be at too high of a level to be practical for characterizing and grouping our experts' survey results. It still serves as a good indication of how programming knowledge areas can be grouped. Syntactic items, while necessary for doing a program in a specific language, will be secondary to the Conceptual and Strategic items for general programming knowledge.

In our methods, we assemble several of the papers mentioned here to build a knowledge category list to use in our classification framework. This final list will be details in APPENDIX B.

2.3 Research on skills

As we have already noticed in the prior section, accreditation organizations are working to move beyond a simple knowledge-based curriculum to include professional skills in the form of competencies. Returning to the Computing Curricular 2020 report [15], we highlight their compelling case for considering "competency" as a practical educational goal. This committee sees competency as the proper application of skills and knowledge within a task. They also acknowledge that dispositions cannot be removed from how knowledge and skills are applied. Here is an instructive figure from that report.

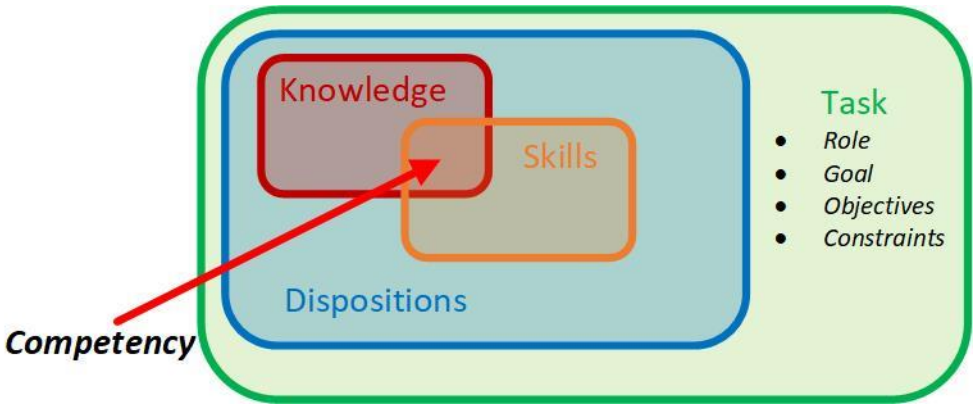


Figure 2.2 Conceptual Structure of the CC2020 Competency Model (based on original figure in [15])

In their table 4.2 titled "Elements of Foundational and Professional Knowledge", there is a list that calls out many general technical skills.

Table 2.4 Elements of Foundational and Professional Knowledge

Analytical and critical thinking	Project and Task Organization and Planning
Collaboration and teamwork	Quality Assurance / Control
Ethical and intercultural perspectives	Relationship Management
Mathematics and statistics	Research and Self-Starter/Learner
Multi-Task Prioritization and Management	Time Management
Oral Communication and Presentation	Written Communication
Problem Solving and Trouble Shooting	

In addition to these, their table 4.4 titled “Prospective Elements of Dispositions” provides a broad selection of non-technical or interpersonal skills:

Table 2.5 Prospective Elements of Dispositions

Adaptable	Professional
Collaborative	Purpose-driven
Inventive	Responsible
Meticulous	Responsive
Passionate	Self-directed
Proactive	

These are characteristics that help make up who an individual is and how they work. From the report: “while it may be difficult to teach disposition, faculty members should instill these concepts in their students through assessment design, exercises, sustained practice, readings, case studies, and their own example.” Every working engineer and teacher, as well as every student, will have a different mixture of these characteristics. They need to leverage what they have, improve upon their strengths, and understand their weaknesses. Becoming well-rounded in this way will help them be successful in both the classroom and future jobs.

One research avenue that has delved directly into this concept of professional skills is the study of “graduateness” [48]. Graduate attributes are a collection of skills that a college graduate may be expected to have mastered. This is very similar to the IEA term of Graduate Attributes.

Marianne Bester, in her dissertation on the subject, noted a "discrepancy between higher education and the needs arising in the world of work"[49]. She agreed with several authors who "suggest that a pertinent focus on developing graduate attributes could possibility [sic] address this mismatch." Her work focused on the South African Higher Education system. Her work is not unique. In the foreword to a 2012 book about South African graduateness [50], Prof Ngambi attributes the need for this study to a combination of "the fast-changing environment and recent economic meltdown". She believes that graduates needed to be **RARE** (**R**esponsible, **A**ccountable, **R**elevant and **E**thical) to meet the challenges of the day and the future[51]. Prof Chetty, in her Chapter 1 introduction of this same book, lays out more details around the background, need, and definition of graduateness [50]. While the concept of graduateness is complex, it includes skills of the "hard" and "soft" variety. Following our definitions, hard skills equate to knowledge. Teaching "soft" skills cannot be done in the same way as "hard" skills. While there is a clear focus in all this work, South Africa is not the only region that has started considering these soft skills.

While it is recognized that professional skills are critical to hiring and career development as well as academic education, there is a recognition of a gap between what is being taught and what employers need. In a survey of nearly 300 computing professionals, the authors study "practitioners' perspectives about how effective computing programs are at preparing graduates for the most important aspects of their job" [52]. From their research, they highlight the top and bottom knowledge areas and skills based on their industry surveys.

Table 2.6 Ten most and least important skills reproduced from [52]

Ten Most Important Skills		Ten Least Important Skills
Problem solving		Waterfall development model
Ability to teach myself		Legal aspects
Critical thinking		Calculus
Concepts across Languages		Code generation tools
Verbal communications		Extreme programming
Logic		Spiral development model
Team problem solving		Soft sciences
Written communications		Assembly
Communicate with other specialties		Physics
Perform different roles		Hard sciences

Following the general pattern from the South African university studies and the educational boards, the majority of the "most important" items are non-technical skills that continue to show up as important to the general job of programming. As this study was focused on non-technical professional skills, the area of general programming knowledge is once again excluded from consideration. While this helps focus on skills, it still falls short of helping us generate a combined knowledge and skills ranked list. In their conclusion, the most important yet least taught skills were critical thinking, problem solving, and lifelong learning. This is no surprise. These have already been noted in many of the works we have reviewed. As computer technology and jobs are changing rapidly, specific knowledge like syntax of a single language may be useless in a few years. Companies, and subsequently employees, will be constantly challenged with new opportunities and problems that require mastering new languages, tools, and skills. Our research goal remains: build a ranked list of knowledge and skills. Just focusing on skills helps us greatly in understanding how important these skills are to programming and engineering jobs. It is not enough to formulate a clear proposal on curriculum changes that must balance knowledge and skills.

For another summary of graduateness and its ascendance into educational boards, we return to a South African professor, Marthie Schoeman. While this paper's main focus is to describe several promising practical methods "to infuse certain aspects of graduateness into an introductory programming module in an ODeL [Open Distance e-Learning] environment" [48], she includes this description of graduateness, summarizing the approach of several other educational boards:

> "The Assessment & Teaching of 21st Century Skills (ATC21S) project classified ten skills in four groups: thinking tactics (creativity and innovation, critical thinking, problem-solving, decision-making, learning to learn/metacognition – understanding own thinking processes); working tactics (communication, cooperation or teamwork); working tools (information literacy, information and communication literacy) and behavior in the world (local and global citizenship, life and career, personal and social responsibility) (Binkley et al., 2014). The World Economic Forum (WEF) in turn identifies sixteen 21st century skills grouped in three clusters: basic literacies (applying fundamental skills in daily life: literacy, numeracy, scientific literacy, ICT literacy, cultural and civic literacy); capabilities (approaching complicated tasks: critical thinking/problem-solving, creativity, communication, collaboration) and personality traits (managing transforming environs: curiosity, initiative, persistence/grit, adaptability, leadership, social and cultural awareness)." [48]

The same skills we have seen show up once again. In addition, the author calls out several other items such as the groups "behavior in the world" and "personal traits". This does show that the actual number of skills we could focus on is large. Schoeman also recognized both the difficulty of classifying which skills could/should be taught in which classes as well as asking the question

of how they could be taught. This is a significant issue, especially in the area of skills that she lists under personality traits. While there are ways for people to develop "grit" or "leadership", how an individual thinks, feels, and believes work in complex ways which make every person unique. No person will be strong in every personal attribute just as no person could master all knowledge. In addition, every programmer will not have identical capabilities in personality traits. People blend their traits with knowledge and skills to find their own way to be a valuable member of any team. If we hope to be successful in improving a CS1 student outcome, we must evaluate any skills which arise in an industry/instructor skills gap while appreciating the complexity and diversity of ways that these can interact in any individual. Skills must be approached dynamically enough to allow for many different individual methods of integrating them into their approach to engineering. Skills like communication or teamwork, which have appeared in almost all these studies, are likely to show up in our results. Introducing these into a curriculum would need care to help teach the high-level skill while allowing for a diversity of individual approaches.

All the studies noted so far reference graduateness as skills learned through a general university education. Our interest is specific to engineering programming inside an engineering degree. We also want to understand the degree as leading to an engineering career. In their paper from 2015, Li et al interviewed 59 software engineers at Microsoft striving to gain more clarity on what we call skills [53]. Using hour-long individual interviews, the authors' "analysis identified a diverse set of 53 attributes of great software engineers" [53]. They organized these skills into four groups: personal characteristics, decision making, teammates, and software product. In each of these areas we see some of the same skills we have found throughout these various analyses. Under personal characteristics we see skills such as "improving", "passionate",

and “curious”. For decision making we find people who are “knowledgeable about people” and “knowledgeable about their technical domain”. In the teammates group, we have people who “create shared context and success”, “mentoring”, and “manages expectations”. Finally in their software product category we have terms like “elegant”, “creative”, and “anticipating needs”. They generate a thought-provoking set of items. In their discussions, they have a section targeted at educators:

> “Our findings also raise significant questions about curriculum choices, teaching methods, and learning objectives in formal computer science and software engineering education. Educators may consider adding courses on topics not found in their current curricula.” [53]

As we have shown, accreditation organizations are all starting to evolve towards knowledge being strongly coupled to these graduate attributes. They also highlight that “our results provide little insight into the relative importance of the attributes” [53].

As we strive towards a competency-based model, where knowledge and skills are blended to solve tasks, we contend that knowing which knowledge and professional skill areas are most impactful is critical. Our work strives to find those critical areas and seeks to map them into engineering, programming, and even CS1 course curricula. Our survey will strive to build this ranked list to inform any curriculum change proposals we might suggest. As we focus on knowledge and professional skill areas needed for a programming job, we hope to extract specific professional skills from our industry and academic gap analysis which could be applied specifically to a engineering programming courses, as well as specific courses like CS1.

Does this application from engineering, to engineering programming, to a CS1 course make sense? Reviewing the dissertation from Marianne Bester indicates that it may be required

[49]. Bester's work began "it is reasonable to argue that a focus on mere academic disciplinary knowledge is not sufficient to meet employers' and students' expectations about higher education studies." She gathered data for analysis by interviewing undergraduate teachers at a South African University of Technology. Her conclusions argued that graduate skills must be fully integrated into the class content to maximize the positive impact on students. In her opportunities for future research she says, "It will also be helpful to focus on issues related to the dynamic interaction between the conceptions of students, employers and academics in terms of graduate attributes" [49].

We are not interested in every piece of knowledge or every professional skill that might be useful to a programmer or an engineer. We are searching for the highest ranked knowledge areas and skills identified by both industry programmers and academic instructors. While not a simple task, we believe it is achievable. We turn to the Delphi Study as our means to gather a consensus ranked list from our expert groups.

2.4 History of Delphi Technique

To understand why the Delphi Study is the right tool for this research, we need to look at the history, purpose, and method of this technique. In the 1950s and 60s, the ever-increasing pace of technology, along with a massive increase in data that could be generated and evaluated by computers, highlighted a need to look into the future and reasonably predict what would likely happen. One of the solutions proposed was the Delphi Method. Olaf Helmer in his foundational article [54] posited on page 2, "The future is no longer viewed as unique, unforeseeable, and inevitable; there are, instead, a multitude of possible futures, with associated probabilities that can be estimated and, to some extent, manipulated." To this end, he championed a method that could help anticipate coming change in the light of insufficient or

overly massive data. He was looking to a revolution in soft sciences to help decision makers of both private and public sectors address these challenges. He called out one of the new methods at the time "that has become known as the Delphi Technique, which attempts to make effective use of informed intuitive judgement" [54]. At its simplest, this is assembling a group of experts and helping guide them to a consensus prediction. Fish and Busby said, "The Delphi method rests on the idea that it is possible and often quite valuable to reach consensus through a collective human intelligence process" [55].

Since the '60s, there have been many reviews of the method [56], [57] as well as books which give more of the history and method variations [55], [58]. While there have been many variations such as "policy Delphi" [59] and "real-time Delphi" [55], [60], the fundamentals have remained like the original:

1. Select a group of experts.
2. Ask careful, open-ended questions to generate a list of key items.
3. Conduct additional rounds of ranking sessions to rank items.

The power of the Delphi Method is in its flexibility to address many situations. Most often, the methodology is focused to drive consensus. While this can be useful for getting experts to arrive at a best combined estimate, it can also help to bring working teams into agreement. It combines a level of anonymity that allows all voices to be heard, with a reconciliation process that allows for give and take in the process of ranking items.

Delphi Surveys have long been used in Medical and Nursing fields [57], [60], [61]. They also have been conducted in fields such as tourism [62], [63], education [64]–[69], food safety [70], psychology [71], business[72], and even curriculum questions during COVID-19 [73]. Applications can also be found in engineering fields such as Construction Management [74],

[75], Architecture and Construction [76], Software [77], [78], Information Systems [79], and Industrial Engineering [80]. More general articles [64], [81] focus on practical implementation with details on the high-level process and recommendations to achieve the best results. While this is not an exhaustive list, it is clear that this methodology can help generate key consensus points from a group of experts. We believe it will help us build a consensus list from our dual groups of industry and academic experts. Further details of how we will conduct our Delphi survey are found in our Methods chapter.

2.5 Pedagogical methods

We hope to achieve ranked lists of knowledge and professional skill areas from our two expert groups. From there, we will recommend possible changes to what we teach in engineering programming courses. If this is done at the professional skill level, the desire would be to have this be agnostic to teaching methods. However, providing a brief review of some of the primary pedagogical methods which are common among teaching of CS1 courses will allow us to cross verify our final recommendations.

While there are many variations, we will list some of those that can be found in research. At the heart of these methodologies, "There are three dominant theories of learning: (a) behaviourism (studying and analysing human behaviours), (b) cognitivism (knowledge constructed by mental cognition), and (c) constructivism (learners construct the knowledge during the learning process)." [82] The concept of cognitive load is woven through all these theories of learning. Cognitive load can be divided into three main loads [83]. Intrinsic load pertains to how many pieces of information must be processed together to understand the target concept. Extraneous cognitive load is when the activities required of the learner are too great. Germane cognitive load is the optimal amount of load that fosters the learning process. For all

pedagogical methods, teachers are always striving to present material at the germane cognitive load. While cognition and cognitive load are much larger topics, we will stop at this high level as we review many of the current instructional methodologies.

2.5.1 Traditional

For traditional learning, the teacher is the dominant source of knowledge in the class [82]. While there are many flavors, this is the default lecture model which remains dominant in many universities. This is mainly influenced by behaviorism. While this is an efficient way to communicate knowledge, it has been criticized for not helping students learn how to learn.

2.5.2 Competency-Based

While traditional learning focuses on knowledge, "the competence is the ability to apply knowledge to effective decision-making both in a specific subject area and in extreme conditions" [84]. From another reference "Competency Based Education (CBE) aims at getting a clear sense of students' capacities in order to optimize their learning and to certify more precisely their acquired knowledge" [85]. We have already seen that competency is becoming a focus of many instructional organizations. However, defining the right competencies as well as assessing them can be challenging [85].

2.5.3 Active learning

At its core, active learning seeks to minimize traditional classroom lecture time while increasing the engagement of the student in activities like interactive learning, more small assignments, or other tasks focused on learning through doing. One study showed that this had a positive effect on CS1 instruction [86]. Like many of the alternative methods, active learning strives to be student centered by requiring "learners to do meaningful learning activities,

combined with reflection on what they are learning and doing" [87]. Studies also indicate that active learning can reduce student failure rate as well as produce general increases in course grades [88], [89].

Hartikainen et al. conducted a survey of literature for active learning and listed a large range of methods which could all be loosely called active learning [90]. It is a strength and a difficulty that active learning can take so many flavors. Since there is no single model, we consider active learning more of a principle than a methodology. Many of the following paradigms could fall under the umbrella of active learning.

2.5.4 Flipped classroom

While traditional learning has lectures to teach the content information and homework to practice the skills, the flipped model asks students to study the knowledge information outside of class individually and attend class together to complete the practice assignments and get help with any concept issues they have. Bergman and Sams, some of the first teachers to champion this model, described the initial question that motivated the method. "What if we prerecorded all of our lectures, students viewed the video as 'homework,' and then we used the entire class period to help students with the concepts they don't understand?" [91]. In their survey of literature, Berssanette and de Francisco found 60% of the reviewed studies showed positive feedback from students [87]. They also echoed that the downside of time, cost, and staffing were the main drawbacks to this method. In addition, students must be motivated to prepare before class [92].

2.5.5 Inquiry-based

Inquiry-based learning combines both learning and practice [82]. Drawing from the constructivist model, students practice as part of acquiring knowledge. Developing problem-solving skills is considered critical to the process. Criticism here is that the method requires both highly motivated student who have a starting base of knowledge and teachers that can serve as guides to the process. This is a much more complicated balance to maintain.

2.5.5.1 Problem-based, Research-based, Design-based

While there are many competing names, the core of this methodology focuses on inspiring students to solve real-life challenges [93]. Following the constructivist model, this is similar to inquiry-based learning but focuses on progressively harder problems as the students' progress to build their knowledge. "In terms of cognitive architecture, two processes are considered crucial to PBL: Activation of prior knowledge and elaboration." [83].

While there is much to be said for these student-centric methodologies, critics have argued that unguided or minimally guided approaches require students to have a sufficiently high prior knowledge to be effective [94].

2.5.5.2 Simulation-based

A variation on Problem-based, simulations are useful when real-life opportunities could be problematic (medicine, for example) [95]. It still relies on practice to help construct a knowledge framework. Depending on the simulated task, skills such as communication or collaboration and teamwork [95] could be the focus instead of pure knowledge.

2.5.5.3 Team-based

A variation on the problem-based model, team-based learning divides a larger group into smaller teams that all work on the same problem. This is more formalized so one tutor could supervise many teams [96]. From the literature, other group-based pedagogies, such as collaborative or cooperative learning have many similarities [97]. While designed to build up teamwork skills as well as improve content retention through the group problem solving process, it is dependent on careful team selection and equal participation from all team members[97]. Like general active learning methods, it may reduce the dropping rate in first term programming classes [98].

2.5.6 Assessments

Testing is generally considered necessary for tracking the progress of students. While this always has a host of challenges, active learning models introduce additional struggles.

2.5.6.1 Challenge-based assessment

This is more of an assessment model where a challenging problem is set for the students to evaluate what they have learned [99]. For programming, this could be a substantial coding assignment. It is closely coupled to the research-based, design-based methodology.

2.5.6.2 Competency-based assessment

In this model, questions and short assignments are set to evaluate specific competencies for the material presented [99]. While this can be applied to traditional learning models, it can also be utilized instead of challenge-based assessment for problem-based models.

2.5.6.3 Peer assessment

For all team-based models, there needs to be some thought to peers evaluating their team and each other. Done well, these assessments aim “to hold individuals accountable to their teams and to lessen the likelihood of social loafing” [96], [100].

2.5.7 Pedagogical methods summary

While this brief overview is not meant to be exhaustive, it shows the broad range of methods that are used in a classroom to both transfer knowledge and encourage students to engage the material and make it their own. Our work, seeking professional skills which could be more emphasized in engineering programming courses, should be easily integrated into any if these methodologies.

CHAPTER III

METHODS: DELPHI SURVEY WITH EXPERT CLASSIFICATION

To gather actionable results, we must establish a classification framework to help parse our open-ended survey results into knowledge and professional skill areas results. Once this has been established, we will detail the specifics of our Delphi survey.

3.1 Classification framework

From our review of literature section, we have seen that knowledge items are starting to be coupled with professional skill items. In the context of competency, we are searching for the overlap of knowledge and skill. While the complete definition would include the individual's dispositions and anchor all of this in the completion of a task, the first step that our research addresses is focused specifically on the knowledge and professional skills which are taught in engineering programming classes and valued in engineering jobs.

3.1.1 Knowledge areas

For classifying knowledge items, the most general groupings follow Mcgill & Volet's mapping [47]. This would have the classifications of:

Table 3.1 Mcgill & Volet category summaries

Syntactic-Declarative	Know syntax rules
Syntactic-Procedural	Use syntax to write code
Conceptual-Declarative	Know program functionality
Conceptual-Procedural	Use functionality to write code
Strategic-Conditional	Code to solve a problem

We will use this structure to potentially group and combine specific items from our survey responses. The “declarative” versus “procedural” differentiation may blur back together when reviewing our survey answers as “knowing” and “doing” will likely focus on “doing”. This would make all the knowledge items lean toward the original computer categories instead of the cognitive psychology additions. We expect that most summary items will also fall closer to the strategic spectrum and move away from the syntactic items as single language syntax has limited use for general training in programming.

While grouping at this high of a level may help us see which of these levels is most important, it seems overly broad for application. We need a finer grained model. Becker’s analysis of syllabi provides 52 knowledge and 2 professional skill ranked categories [14]. In Appendix B, I have used this full list as a super-set to cross-reference several of the other lists from multiple articles [40], [42], [44]. Apart from some advanced topics called out in the ABET-CS guidelines, the Becker list proves to be a good starting point. If we apply the Mcgill and Volet groupings, we find that we need several others to fully cover our complete list.

Table 3.2 Additional grouping summary items beyond Mcgill and Volet

Background	Math, history, CS theory
Tools	Useful tools for coding (IDE, etc)
Debugging	Debugging methodology
Advanced Topics	Topics like parallel processing
Professional Skills	Professional skills

These items are relatively self-explanatory. Background represents information that would be expected to be brought into a programming class (math, logic) or information that supports the learning of the subject (history, theory). Tools would be any specific items like an integrated development environment (IDE) that assist in writing programs. Debugging includes

both learning how to utilize the tools for debugging as well as effective strategies. Advanced topics are things like parallel processing that would be building upon the programming fundamentals. Professional skills, as we have discussed at length, are those skills like teamwork and communication that are critical to most areas of engineering pursuits. I believe that the 52 Becker "knowledge" items will also be a superset of the result items called out by our experts. Our final list can be found in Appendix B.1

3.1.2 Professional skill areas

From the skills side, we have no superset list like Becker. Without this, we followed the model Becker used for ranking items from his syllabi survey. We combined lists from eight of our references and ranked them based on how many of these sources referenced that professional skill. In Appendix B.2, we have the superset list following the knowledge area list. As we saw from the ABET guidelines, "student outcomes describe what students are expected to know and be able to do by the time of graduation. These relate to the knowledge, skills, and behaviors that students acquire as they progress through the program." [43]. From our superset list, the following table shows all professional skills that are acknowledged by two or more of our eight references.

Table 3.3 Professional skills by ranked list.

Skill	Count (out of 8 references)
Communication	8
Teamwork and collaboration	8
Lifelong learning	6
Problem solving	5
Ethical responsibilities	5
Consideration of public factors	4
Experimentation and judgement	4
Knowledge Items	4
Adaptability	2

As the knowledge list had a few skill items mixed in, the skills list also has four references that intertwine and include knowledge items. It will come as no surprise that communication and teamwork were referenced in all eight surveyed references.

3.1.3 Classification framework

Utilizing the two ranked lists in Appendix B for knowledge and professional skill areas, we will parse the Delphi results into the highest ranked item that is reasonable. The flow can be seen in the Figure 3.1.

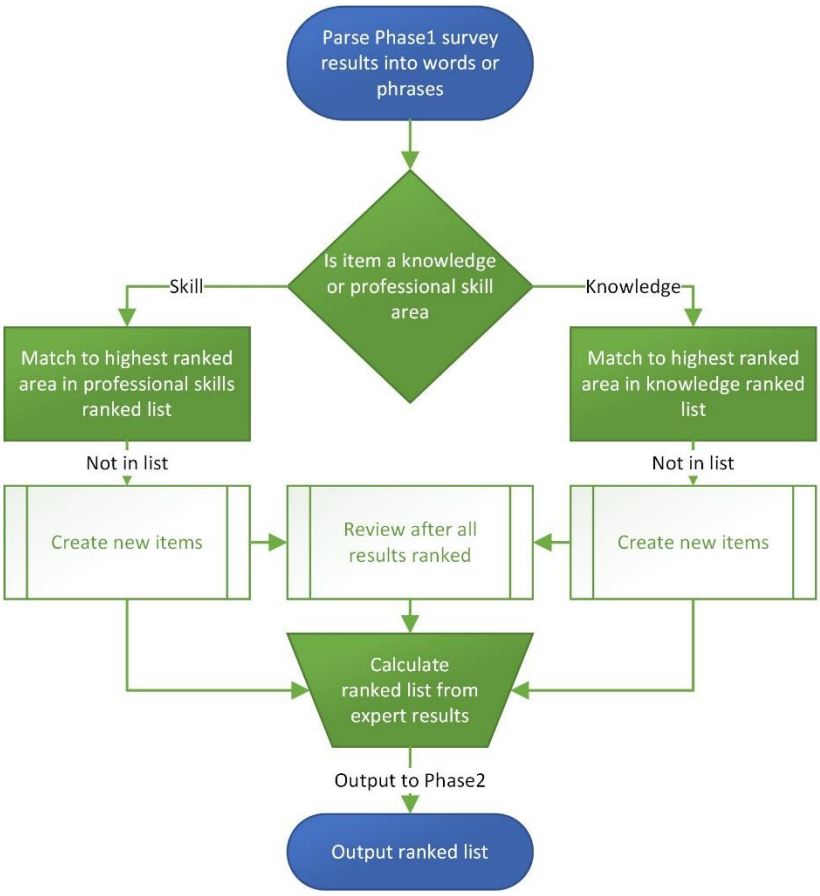


Figure 3.1 Classification framework decision flow

It is possible we will have some items that fall “not in list”. If we have enough of those to rate in the top items, we may need to re-address our framework to understand why our references were not inclusive enough. Our final list of added items can be found in Appendix B.3.

3.2 Delphi Survey

The Delphi Survey is a straight-forward concept, but there are many challenges that can lead to failure. Turoff and Linstone highlighted several in their discussion of techniques and applications for the Delphi [58]. Disagreements in ranking may indicate many different problems with understanding of the questions or ideas. It may indicate fundamental differences in the

thinking of different groups among the experts. Further challenges include how to classify "experts" and find a representative group. Some other technical concerns about how the statistics are managed can also be found in the literature. In this chapter, we will present the specifics of our planned Delphi Survey to build our industry and academic consensus ranked lists. From these lists, we can extract any gap that will become the highlight of future curriculum change recommendations. We address some of the difficulties and how we aim to avoid them. Using the Olsen paper as a high-level guide [64], we walk through main decision points and provide details of this Delphi Survey.

3.2.1 Developing Delphi research questions

To discern if a knowledge and skills gap exists between academic and industry experts, we must first find those items most valued by each group. As we have shown, we expect to see a mix of general knowledge areas and non-technical skills show up in both groups. At a high level, we are trying to have our experts answer what knowledge and skills are most important for both academic and industry success. For our open-ended questions, we want to be focused enough to have the experts thinking about a common outcome, like being hired by a company, while preventing words or phrases that would limit the possible responses of the individuals. From our prior chapters, our desire is to encourage our experts to think about general knowledge and skills in their answer. While not specifically excluding preliminary knowledge or specific knowledge, our expectation is that general skills will rise to the top when we go through our ranking process. For skills, we expect many of the same items presented in our review of literature to appear in these lists.

What we are looking for in our survey would be the most important items from the combination of knowledge, skills, graduate attributes, dispositions, and characteristics. While it

would be somewhat surprising if our Delphi survey replicated any of these referenced lists exactly, I expect to see several of these items score highly both among the industry experts as well as the academic experts.

Working through all these points, we decided on the following two key survey questions.

- Q1: What knowledge, skills, or characteristics should new hires in programming positions possess?

As we have outlined, knowledge, skills, and characteristics include all the different aspects of the lists detailed above. In addition, putting knowledge first, we expect to get any key hard knowledge requirements, general, preliminary, or specific, deemed critical to the expert. "Characteristics" is chosen to elicit thoughts that might lead into items like the list of dispositions. It was thought that "characteristics" might be a more accessible word than "dispositions". As we are targeting industry-applicable skills, "new hires" should enable both our industry experts and our teachers to imagine what they look for in someone just out of college. Some hiring managers may consider experienced programmers in the group of "new hires", but we believe that even these managers will gravitate towards both general knowledge and important skills that will apply at all job levels. The final word "possess" could have been simplified to "have", but "possess" implies a level of ownership. What skills should new hires have already made their own? Placing this focus on new hire skill ownership is intended to significantly reduce the set of skills that the employers would expect engineers to learn on the job.

- Q2: What experiences are helpful to develop into a good programmer?

While this is still looking for skills and dispositions, it is intended to be focused more on skills instead of academic knowledge. "Experiences" is intentionally broad, but still leans away from knowledge that could be classified as book learning. Experiences are situations that

students should go through to learn about important principles such as the value of hard work, or the difficulties and rewards of working in teams. The word “helpful” is also targeted to allow disposition and characteristics to be included in the skill set. We hope this will bring out our earlier idea that every individual must find a way to adapt their personalities to these skills in a diverse way. While some experts may focus on being adaptable, others may focus on being meticulous. Some may revel in the value of being collaborative and team engaged, while others may call out personal creativity and inventiveness. When we see the highest ranked skills, we expect that every individual engineer will develop some way to imbody it. Adding “to develop” also highlights that these skills may not be mastered fully when entering the job market. While the first question is looking for “possessed” skills, the second question is looking for “begun” aptitudes.

3.2.2 Defining panelists and panel size

As our goal is to evaluate the potential alignment of industry versus academic experts, we are designing a two-group survey, consisting of Fortune 500 company programmers and professors of CS1 classes. One of the potential pitfalls of poor Delphi surveys is receiving a small number of responses. From overall literature guidance, our plan is to target 30-35 finished surveys in each expert group. This number is large enough to have group statistical significance yet is small enough to allow for individual contact and follow-up which guidance says is crucial to encouraging participants to stay through all the rounds of the survey.

3.2.2.1 Fortune 500 company programmers

As the author currently works for a Fortune 500 company, he has a large network of coders of differing expertise that will generate a good cross-section for the Delphi survey. By

design, this group of programmers will have varied levels of programming expertise and varied involvement in hiring. Another possible weakness in Delphi surveys is selecting a group of experts that do not have much diversity across the group. While it is impossible to have a completely random group of experts, we will reach into the US, China, Israel, and Finland to get a relatively broad cross-section of geographies and cultures.

3.2.2.2 Professors of introductory programming classes

Our second group represents academics. We will reach out to professors who design programming curriculum and teach programming classes. Our goal would be to have 30-35 finished surveys like the first group. Recruiting for this group will take a little more outreach. Several avenues will be used to find suitable and interested experts. First, the authors will reach out specifically to land grant schools in each state as most of these have some level of engineering program. Second, we will search for lists calling out the top engineering schools in the country and reach out to those schools. Finally, we will search the ASEE report which lists schools that are graduating the most computing majors. With all these lists, we will use school websites to locate the best potential teachers of beginning programming classes. If necessary, we will reach out to department deans to locate CS1 professors. In addition, if we remain short of participants, we will use snowball recruitment from our existing contacts. At this time, we will not specifically search for colleges outside the US. This will reduce the diversity of this expert group but correlating different countries' requirements and expectations could prevent the group from being able to reach consensus. This might be something that could be considered for future research.

3.2.3 Self-classification questionnaire

Another issue that the Delphi survey must contend with is the nebulous definition of "expert". There is not an objective standard that can be easily applied to our group of programming engineers or professors. If we have a large variation in the skill level of our group, we might see different levels of experience emphasize different skills. In order to mitigate this, we will be pulling from programmers of different expertise levels and programming areas. We have classification questions which may help resolve differences within a group. For example, coders that are working on low-level firmware may have different expectations than coders working on application code or test code. If we have any specific differences in skills and/or ranking, we will strive to resolve these based on the secondary classification questions. While there are known concerns with self-classification, this seems to be the best way to have some method of looking for variations within the separate groups of experts. We will ask the same classification questions to our academic experts. By design, teachers of CS1 and engineering programming classes will be more homogenous, but our classification questions should help show some diversity in how long they have been involved in teaching.

The full classification questionnaire can be seen in Appendix A.

3.2.4 Delphi Rounds

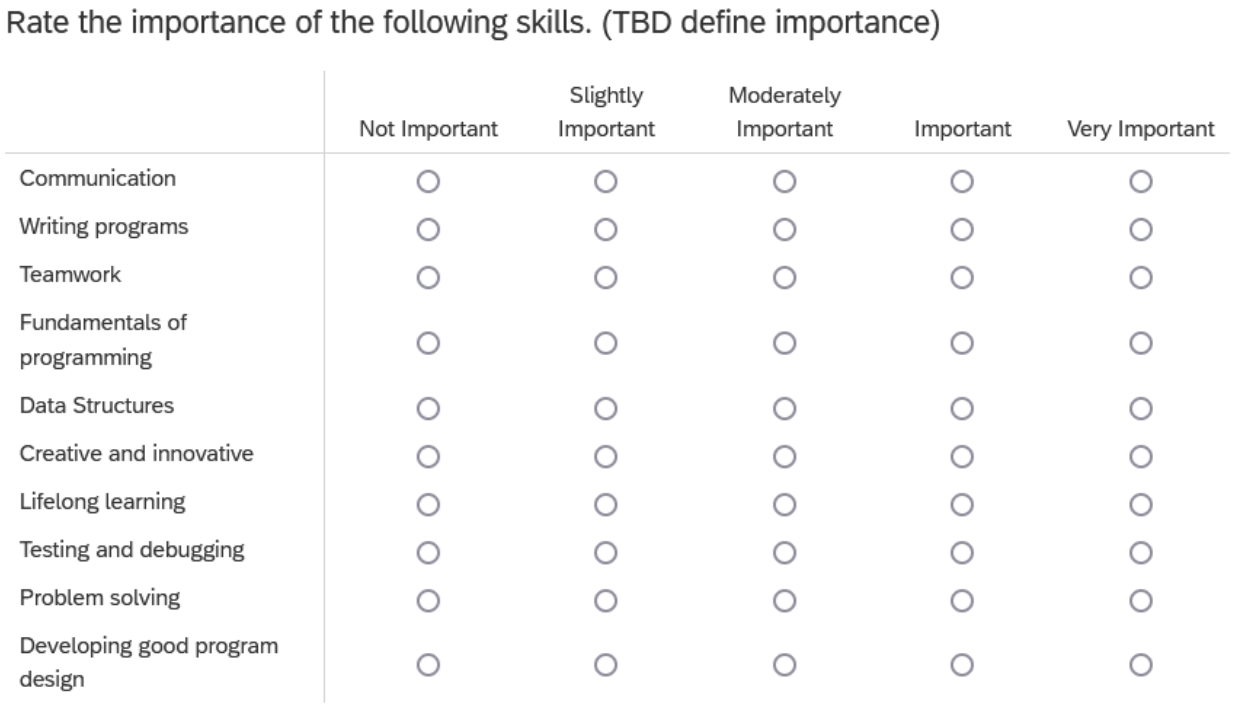
For this survey, we will have three rounds. The first will be our two open-ended questions. The second would be individual ranking of the resulting skills list. The third would be group-versus-individual ranking to see if we can converge on the skills which have the most support.

3.2.4.1 Delphi Round 1

This round simply states the research questions as the main Delphi Survey input. We will also include the classification questions in this round. Once these results are entered, the classification framework will be used to distill the open-ended question answers to a list of knowledge and professional skill areas. This completed list will become the ranking list for the next round of the survey.

3.2.4.2 Delphi Round 2 – Individual ranking

Once we have a list of areas from Round 1, we will send these skills back to the experts and ask them to use a Likert scale to rank them. Following Norman [101], we will utilize a five-point importance/scale questionnaire as shown in the next figure.


Rate the importance of the following skills. (TBD define importance)

	Not Important	Slightly Important	Moderately Important	Important	Very Important
Communication	○	○	○	○	○
Writing programs	○	○	○	○	○
Teamwork	○	○	○	○	○
Fundamentals of programming	○	○	○	○	○
Data Structures	○	○	○	○	○
Creative and innovative	○	○	○	○	○
Lifelong learning	○	○	○	○	○
Testing and debugging	○	○	○	○	○
Problem solving	○	○	○	○	○
Developing good program design	○	○	○	○	○

Figure 3.2 Example Likert 5-point importance scale questionnaire

This scale will equate “1” with Not Important and “5 with Extremely Important”. Once the ranking is complete, we enter basic statistical analysis for these results. Likert scales have the issue of a discrete number of ranking selections for every question. In addition, reviewers may self-select different metrics such as not using the number 1 or 5 as a matter of course. This means that several skills could end up ranked at the same number for many experts. If one expert ranked three skills as “Very Important”, there is no information on how the expert would rate these skills against each other. This does give us a ranked list that clusters skills into importance groups.

Utilizing SPSS, we will do simple mean analysis to rate all of the items from our experts. This will then be provided as part of our Round 3 analysis.

3.2.4.3 Delphi Round 3 – Group classification

After the second round, we compile all the rankings into a single group ranking. Iqbal and Pipon-Young [81] recommend using percentages to show the group rankings, but with our Likert scale the mean number seemed more accessible. Experts would see a mean of 3.89 and realize that this was between 3 “moderately important” and 4” important, with the scale tipped most of the way toward “important”. We allow the subjects to have one more chance to re-rank their items considering the group and their own ranking. According to the Delphi process, seeing the larger group context will help them determine if they agree with the group, or still believe their ranking is better than the group ranking. The results are re-tallied to generate our final ranking.

3.2.5 Industry/Academic gap analysis

Once we have the two expert rankings, we will analyze the lists against each other. We will run T-Test and ANOVA tests on the data from each list. While Likert data is ordinal, which might indicate the Spearman rho assessment or the Mann-Whitney U test should be used for analysis, several researchers have shown that normal parametric tests are generally robust enough to provide good results [101]–[103]. It is possible that we will have some categories in each group that end up excluded from the other groups lists. Again, we will use SPSS to analyze the data. If this analysis is not instructive, we will fall back to simple rank and mean comparison to describe the deltas between the two groups.

From this analysis, we should be able to compare where the two groups agree and where they diverge. This analysis will inform our discussion as well as any recommendations we might make for targeting specific areas for increased emphasis in a CS1 curriculum.

CHAPTER IV

RESULTS: FORTUNE 500 COMPANY INDUSTRY EXPERTS

4.1 Industry experts

As we discussed in chapter three, our industry expert group is entirely comprised of engineers from the same Fortune 500 company. Geographically, participants hail from the United States and Israel, with some additional representation from Finland and China. This selection across different continents is intended to give us some representation from different cultures and experiences. While we expect our results to be applicable to all industry engineers, future research may be needed to confirm there are no single-company biases hidden in these results. In addition, all surveys were done in English, which also can limit the findings from having global reach without additional studies crossing different languages.

4.2 Expert classification breakdown

In our classification questions, we are looking for diversity across several metrics: coding area, primary programming language, involvement in hiring, and self-ranked skill level. The graphs below are based upon the thirty-one completed Round 1 surveys. We have two experts who did not complete the classification questions, so most of the results are based on twenty-nine samples.

4.2.1 Coding area

Our first classification question is:

- Where do you spend most of your coding time (check all that apply)?

For reference, we pulled a chart that provides one cross section of different programming areas [15].

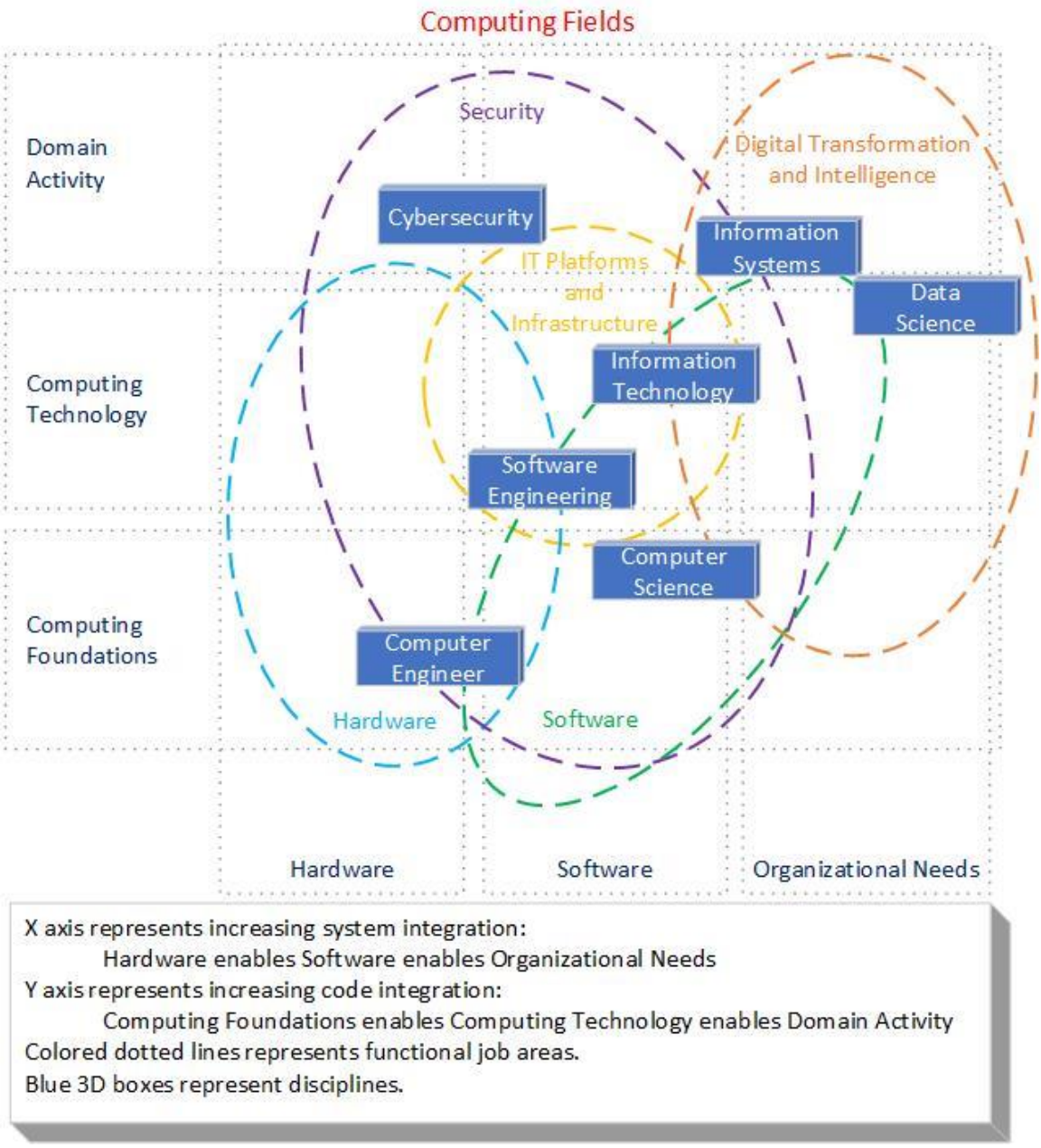


Figure 4.1 Contemporary view of the landscape of computing education (based on original figure in [15])

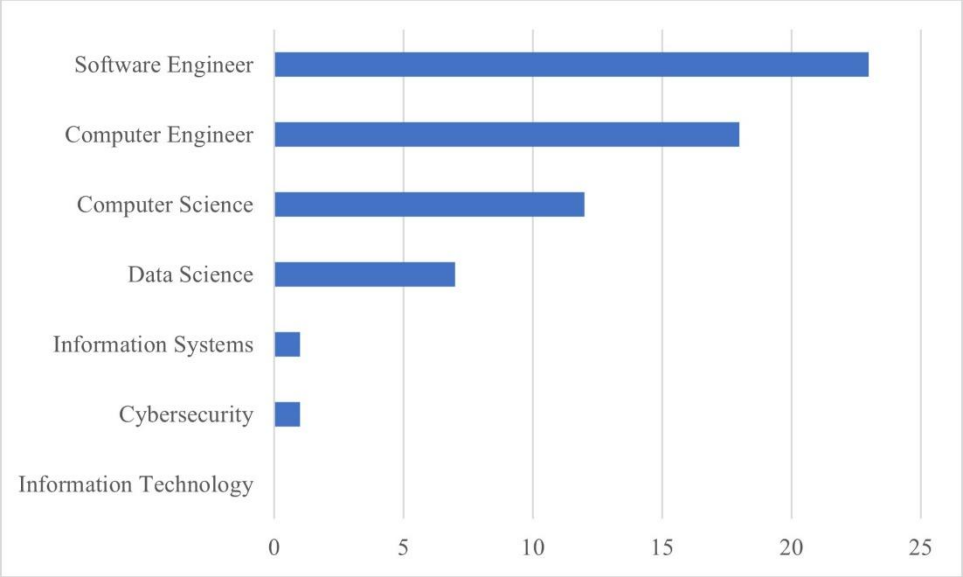


Figure 4.2 Result: Where do you spend most of your coding time?

While we did not have any experts with Information Technology background, most of the other areas are reasonably represented. Not surprisingly for this company, computer engineers and computer science were selected by a significant number of the respondents. Many experts selected more than one area, so the total result is much higher than the 29.

4.2.2 Hiring involvement

Our second classification question is:

- How involved are you with hiring?

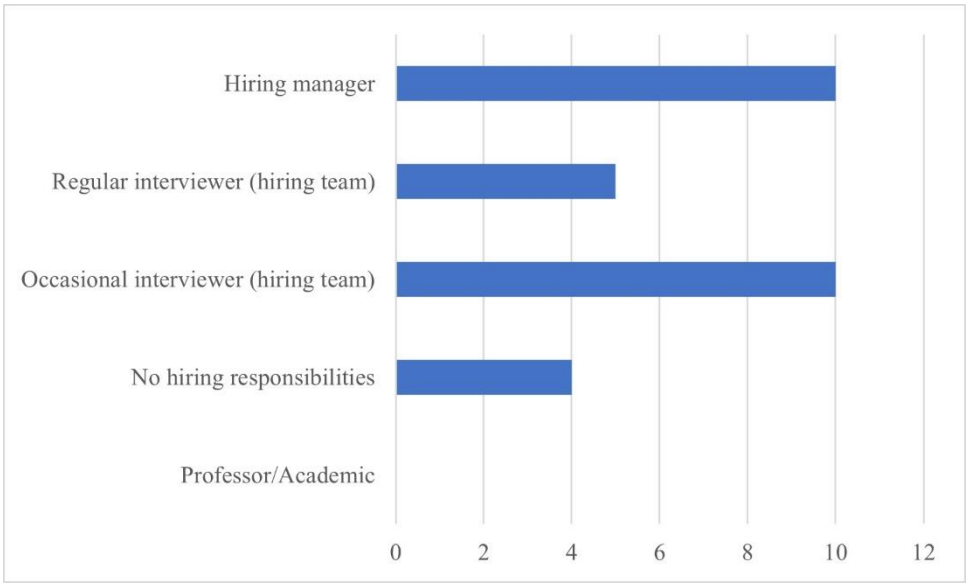


Figure 4.3 Result: How involved are you with hiring?

We have a broad range of hiring involvement. There were only four individuals that said they had no hiring responsibilities. This indicates that our experts reflect the opinions of people who have a cross section of experience with hiring new engineers.

4.2.3 Number of interviews

Our next classification question is:

- How many interviews have you been involved in in the last year?

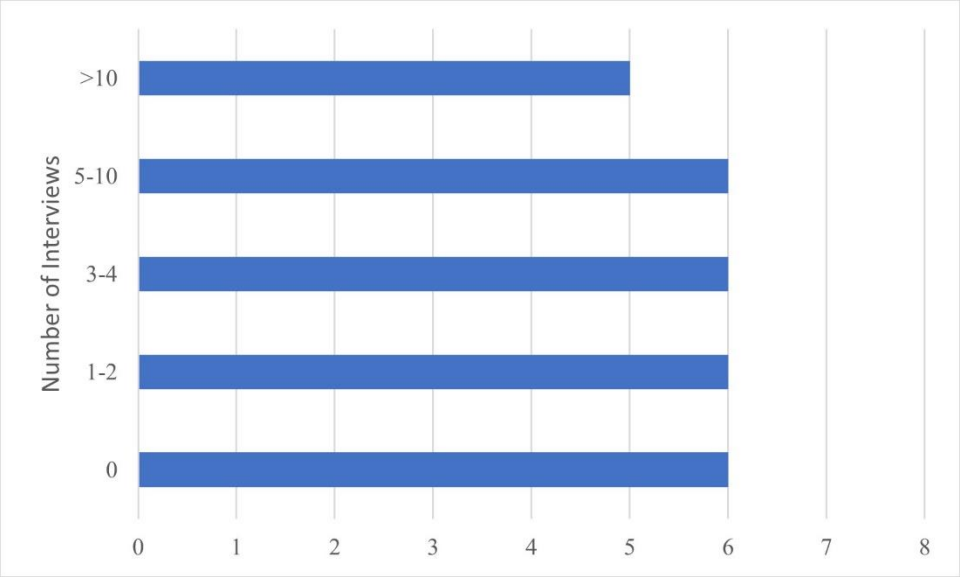


Figure 4.4 Result: How many interviews have you been involved in in the last year?

This question helps qualify the prior question. We have six experts that have not been in an interview in the past year. As we are still coming out of the Covid-19 economy, hiring was not at the same level as in the years before Covid. However, we also have almost a flat spread across all other selections. This supports our prior conclusion that this industry expert group has broad hiring experience with most experts completing multiple interviews in the last year.

4.2.4 Training involvement

Our next classification question is:

- How involved are you with training/mentoring new hires?

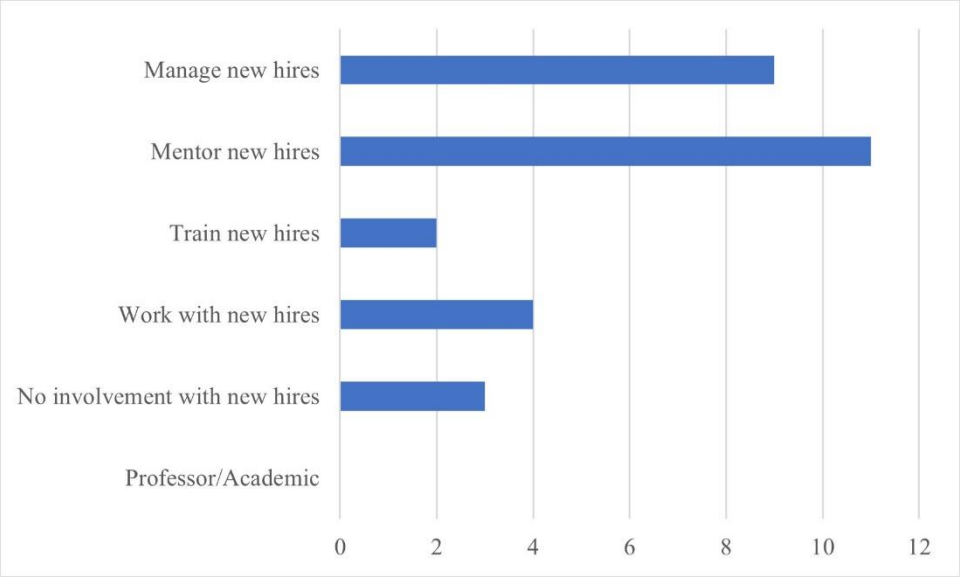


Figure 4.5 Result: How involved are you with training/mentoring new hires?

Many of our experts have served as mentors and managers. This is not surprising with ten hiring managers in our expert pool. We had a small number of trainers. It appears that mentoring is much more common than training. This is a case where deeper study of the concepts of training and mentoring might explain why we see this result. On the other side of the spectrum, we only had three experts who were not involved with new hires.

4.2.5 Main coding language

Our next classification question is:

- What languages do you spend the most time in (select all that apply)?

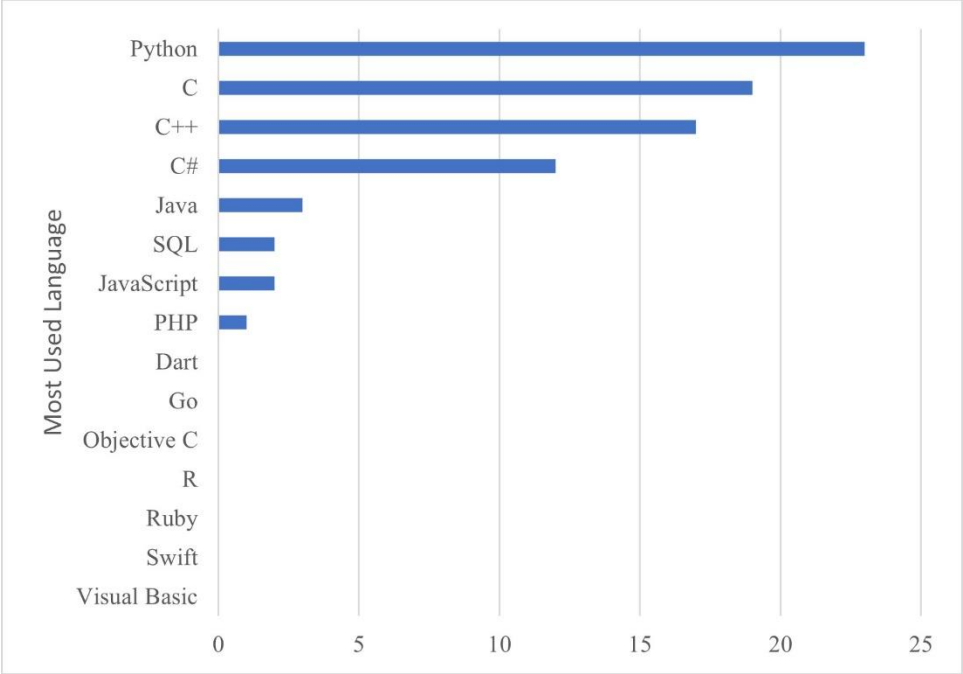


Figure 4.6 Result: What languages do you spend the most time in?

While C, C#, C++ all had several ticks, Python had the largest number of users. It was a little surprising to me how many of the languages from our target list were not used at all at this company. It is not surprising that C and its derivatives were common. Python is a language that has been increasing in popularity, and certainly is used by most of our experts. As we also allowed multiple selections, the total languages mentioned exceed the 29 respondents. Several of my experts called out C, C#, C++, and Python, so a majority of engineers regularly worked in several different languages. Java, SQL, JavaScript, and PHP were always accompanied by one of the top four languages, so no-one called out these coding languages as their only environment.

4.2.5.1 Other languages

Our next classification question was looking to see if our selection of languages was sufficient. While most surveys had this blank, we did have a few additional mentions.

Table 4.1 Additional languages mentioned

Language	Mentions
Powershell	3
Matlab	2
Rust	2
System Verilog	1
Haskell	1
Lisp/scheme	1

None of these rises to a “major” language. Powershell is a scripting language, so this is often a companion to other languages. System Verilog is specific to chip designers. Matlab is very useful for some of our data scientists to do high-end calculations. While we could include these on future lists, this does not impact our primary finding that Python, C, C++, and C# are the most used languages in this expert group.

4.2.6 Percentage of time coding

Our next classification question is:

- How much of your current job involves coding?

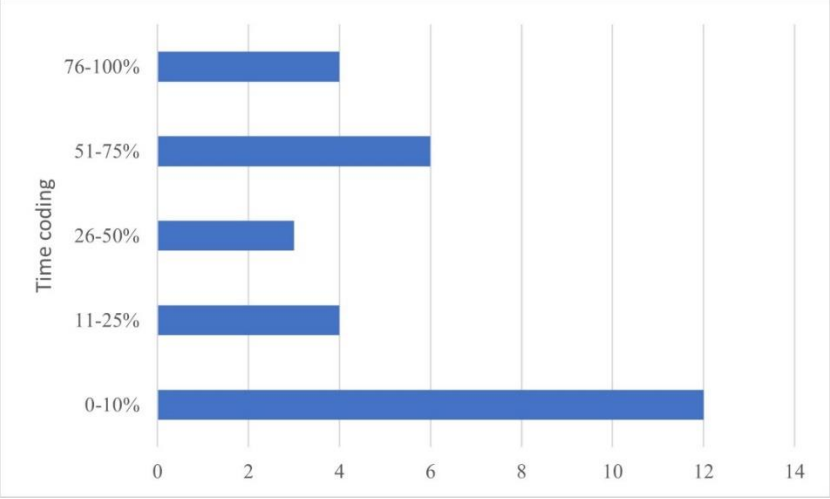


Figure 4.7 Result: How much of your current job involves coding?

Twelve of our respondents do very little coding in their day job. Seven of these were actually hiring managers, which seems to make sense. From a diversity perspective, we once again have representation across all possible categories for this questions.

4.2.7 Self-ranking skill level

Our next classification question is:

- How would you rank your skill level?

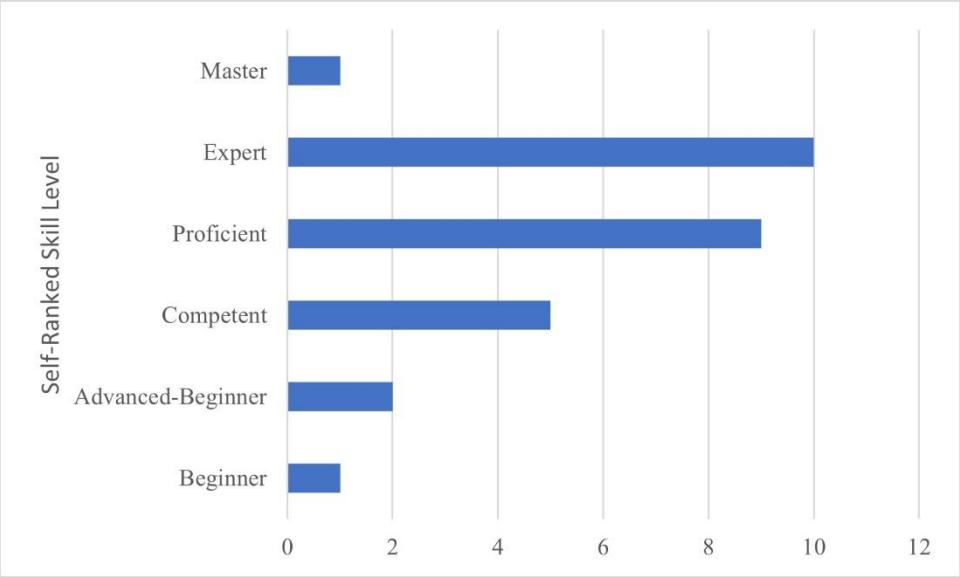


Figure 4.8 Result: How would your rank your skill level?

While we had one “Beginner” and two “Advanced-Beginners” in our group, most of our respondents were in the “Expert” and “Proficient” categories. It is also notable that we had only one self-ranked “Master”. As we did not define these terms, it might be an interesting sub-study to understand what our group of coders considered the difference between an “Expert” coder and a “Master” coder. While every selection has representation, this group is shaded toward the more experienced end of skill.

4.2.7.2 Explain your ranking

Our next classification question is:

- In 1-2 sentences, explain why you chose this ranking.

Our “Beginner” did give more background when answering this question. “I am a PM [Project Manager], I don't get to code very often. However, I do do code reviews.” Even our one “Master” coder added “While there is always room to improve, mastery means that there is really no programming challenge in my domain that I am not equipped to successfully deliver code for.” There were many short personal stories contained in this data that would be fun to mention/investigate further.

We have a broad range of coding expertise in our expert group.

4.2.8 Years at skill level

Our final classification question is:

- How many years have you been this skill level?

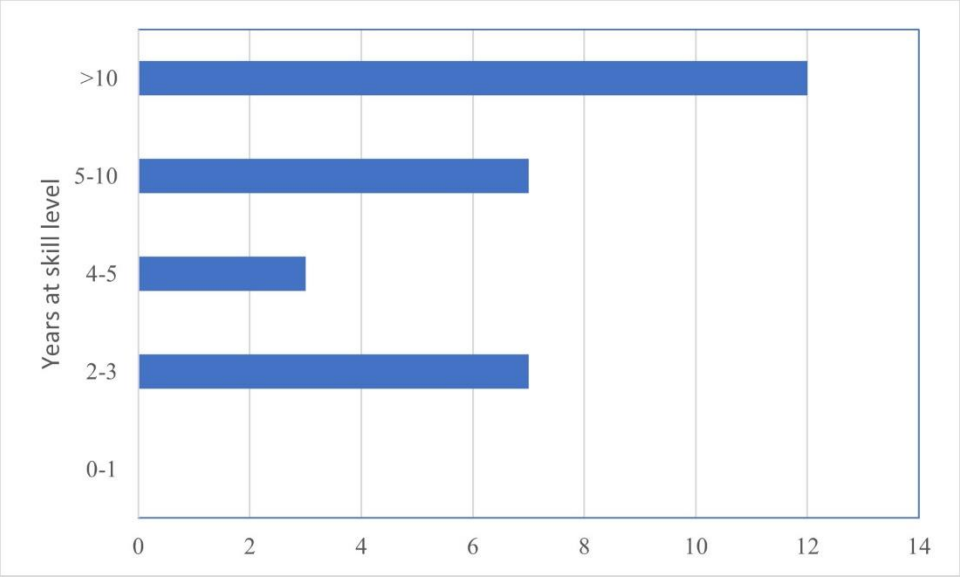


Figure 4.9 Result: How many years have you been this skill level?

For this group, the majority have been at their same skill level for years. This gives some indication that many of these experts are very stable in their current positions. We clearly had no actual “new hires” in this group as everyone had been at their level for two years or more.

4.2.9 Classification summary and discussion

These questions assessed the diversity of our expert group. While we were expecting to have some areas where our individuals would cluster together, the actual findings showed individuals in almost every category for every question.

As we mentioned in several areas, future work to better define many of these terms and details would help provide clear results. Discovering the difference between mentoring and training comes to mind as a clear item which would be interesting to unravel. Extensive work could be done around the concept of which programming language is most useful for particular classes of problems. One of our experts added this comment when discussing programming languages:

> Experience, even if light, with the three main programming paradigms (imperative, object oriented, functional) is definitely very welcome as it helps us approach problems through different lenses. Of course, this implies having experience with different languages (in most cases) as it's hard for a single language to properly support all three paradigms. To that end, my top three choices would be C (imperative), Ruby (object oriented), and Haskell (functional).

There may be many other competing takes on this subject.

For our classification purposes, we believe these results show that our particular group of experts should be an excellent cross-section that represent the industry segment well.

4.3 Round 1 results

The most delicate and time-consuming work of a Delphi survey is taking the open-ended question essay answers and distilling them into a set of rankable items. Our chapter three framework was applied to make this classification process as objective as possible.

4.3.1 Classification examples

In order to show our classification framework in action, we present two examples from our Round 1 surveys. In each of these, we show the initial response from an expert and then decompose their essay response into knowledge and professional skill areas to feed into our Round 2 survey.

To recap, these are the two open-ended questions in the survey:

- Q1: What knowledge, skills, or characteristics should new hires in programming positions possess?
- Q2: What experiences are helpful to develop into a good programmer?

We are trying to get at the same list of knowledge and professional skill areas by presenting two different ways to think about the problem. For our analysis, the questions are parsed into the same single list of category areas.

4.3.1.1 Example one – concise response

Here is the text from one of our experts for our Delphi Q1 and Q2 questions.

Q1 answer

1. *Knowledge of programming language syntax and good programming practices.*
2. *Creativity to solve a problem with their own perspective.*
3. *Thorough. Document, test, and verify their solution.*

Q2 Answer

Learn from mistakes. Don't take it as personal attack but opportunity for growth. Bug failure analysis. Investigate and discover. Peer Code reviews

This engineer was organized and concise. Many of our responses were similar to this mode of communicating. Following our framework, we strive to break out words or phrases that can map into one of the entries in our knowledge list or our professional skills list. If we find no good mapping, we add a line to our list tracking added categories.

Take the first item in their Q1 list "Knowledge of programming language syntax". This is clearly a knowledge area, so we start with our knowledge list. The highest ranked item on this list is "Writing programs". This appears to be a reasonable match. Reviewing the rest of the list, nothing stands out as simpler or clearer. The next table shows how we map these ideas for the rest of the answers.

Table 4.2 Framework Mapping of example expert 1

Expert Text	Area Mapping
Knowledge of programming language syntax	Writing programs (knowledge)
Good programming practices	Developing good program design (knowledge)
Creativity to solve a problem with their own perspective	Creativity and innovation (skill)
	Problem solving (skill)
Thorough. Document, test, and verify their solution.	Generating clear documentation (knowledge)
	Testing and debugging (knowledge)
Learn from mistakes. Don't take it as personal attack but opportunity for growth.	Lifelong learning (skill)
Bug failure analysis. Investigate and discover.	Failure analysis (new)
Peer Code reviews	Code reviews (knowledge)

Many of these mappings are very straightforward, but some lose some nuance when mapped. For example, "thorough" has been grouped into our "generating clear documentation". It is possible that "thorough" may have been intended to be a stand-alone item representing

something more like “attention to detail”. While we stand by our mapping as best representing the intent of the expert based on surrounding context, there are definitely some chance that not every thought of the respondent is fully appreciated in our final mapping list.

As a second example, the phrase “learn from mistakes” was classified as “lifelong learning”. While this is what we think the best mapping is, it is possible the expert thought of this as more of an attitude. They might have really intended to say that individuals should have an attitude of translating mistakes or negative feedback into long-term personal growth. When mistakes happen, is your attitude one of desiring to grow or do you see this feedback around mistakes as a personal attack? Without further feedback from the expert, we continue to feel that “lifelong learning” is more likely to represent what is being expressed here.

While the author has reviewed these groupings for all the entries and believes they are accurate, we are humble enough to realize this process that cannot be completely objective. One of the benefits of the Delphi survey is our Round 2 and Round 3 ranking passes help elevate the most significant items while allowing for feedback to see if our experts feel any critical areas were missed in the mapping. Our top-rated items, even if the mapping was not perfect, should still represent the consensus from the group on what is truly important.

4.3.1.2 Example two – descriptive with additional details

Here is the essay text from our second example. Note that the item numbering was entered by the expert and is reproduced as they wrote it.

Q1 answer

1. *Programming language fluency. To a great degree, programming languages are for human consumption rather than machine consumption (that's why we don't program in machine code). Consequently, language fluency is critical. Like reading other human languages, it's not just basic syntax understanding - developing an intuition for what the code author was trying to accomplish and a sense of the overall way the code hangs together, the nuances that a design conveys about the requirements. In the same way, language fluency enables authoring code that is maximally understandable by other humans and minimizes the cognitive burden required to understand it.*

2. *Trained in the scientific method as applied to programming. Being able to systematically analyze code, synthesize testable predictions about its behavior, and formulate tests to prove or disprove those predictions is a key skill for debug and program understanding.*

3. *Oriented towards knowledge acquisition instead of knowledge retention. Knows how to discover answers to questions quickly rather than relying on memorized domain knowledge.*

4. *Recognizes the importance of mastering the tools of the trade. Characterized by having developed a stable of good tooling that contributes to rapid digestion and exploration of complicated code bases. Most great programmers I know are true masters of the code search tools and editors that they use and are constantly improving the toolchest that they use through little scripts, editor extensions, source control tricks, etc.*

5. *A humility about the correctness and performance of one's own code. My code is always guilty until proven innocent by testing and verification of its expected operation. Knows how to write tests that prove code is correct.*

Q2 Answer

1. *Opportunities to explore the boundaries of one's competence without being thrust into completely alien territory. As experience grows, the frontiers of competence expand - programmers that don't continue to chase that frontier become stagnant with respect to expertise. Conversely, those who are thrown into the deep end without support fail without learning. Staying in the goldilocks zone at the edge of competence is key.*

2. *Exposure and engagement with experts. Even once a programmer has become a highly-experienced "good programmer," continuing to interact with others at high-levels of competence is important for refining and growing one's technique.*

3. *Successfully delivering a product. Creating something that is out there in the world that people actually use is a key motivator for further success. If you only work on things that no one sees or appreciates, or if all your projects are cancelled before release, it is hard to be motivated to be any better.*

4. *Non-programming (e.g. people skills) development. Programming as a field can often attract brilliant people who are somewhat difficult to interact with. Developing non-technical skills such as clear communication, ability to give grace to difficult people, knowing how to set good professional boundaries and build professional relationships can actually be really key to unlocking access to people who have a lot to contribute back to your own technical development.*

On a first pass, we can see this respondent wrote many more words and tried to describe his thought in much more detail. Where our 1st expert used around 50 words, our 2nd expert used around 500. Without going through every phrase, our mapping ended up looking like this:

Table 4.3 Framework Mapping of example expert 2

Expert Text	Area Mapping
Q1 #1	Coding as language (new)
	Developing good program design (knowledge)
Q1 #2	Problem solving (knowledge)
	Testing and debugging (knowledge)
Q1 #3	Lifelong learning (skill)
Q1 #4	Tools (new)
Q1 #5	Humble (skill)
	Unit test (knowledge)
Q2 #1	Writing programs (knowledge)
	Lifelong learning (skill - duplicate)
Q2 #2	Teamwork and collaboration (skill)
Q2 #3	Experimentation and judgement (skill)
	Staying motivated (skill)
Q2 #4	Communication (skill)
	Teamwork and collaboration (skill)

After mapping, we see that our areas were not nearly as divergent as the number of words to describe them. Expert one had 9 areas while expert two has 14. We still have a chance, even with the extra description, of missing some of the nuance when we do the mapping. For example, this expert talks in Q1 #4 of "mastering the tools of the trade" with examples of "utilizing tools to become efficient at their jobs through their 'toolchest'". This is mapped into "tools". Clearly, some of the scope and intent the expert is detailing is lost by the single mapping. Still, the Delphi process should call out whether the consensus is that tools are "very important", "not important" or somewhere in between. It will also allow for comments if the concept of "fully utilizing tools to develop job efficiency" is an item they feel is missing from the overall list.

4.3.2 Round 1 area "hit-list"

For Round 1 results, we have no rankings as key items are mapped from the essay responses. However, counting how many experts "hit" on the same area gives us an initial feel

for how likely experts in our ranking sessions will highly rank the items they mentioned. It also gives us a very rough ability to consider how our Round 2 results compare to this “hit-list”.

Table 4.4 Industry Framework Mapping “Hit-List”

Category	Hits	Area	New
Problem solving	16	Professional skill	
Communication	14	skill	
Lifelong learning	14	skill	
Teamwork and collaboration	13	skill	
Curious	11	skill	
Testing and debugging	11	knowledge	
Writing programs	9	knowledge	
Designing algorithms	7	knowledge	
Creative and innovative	7	skill	
Fundamentals of programming	7	knowledge	
Single language	7	knowledge	yes
Tools	6	knowledge	yes
How computers work	6	knowledge	
Data structures	6	knowledge	
Developing good program design	6	knowledge	
Multiple languages	6	knowledge	yes
Object oriented programming	6	knowledge	
Operating systems	6	knowledge	
Unit test	5	knowledge	yes
Program management	5	knowledge	
Scripting language	4	knowledge	yes
Generating clear documentation	4	knowledge	
Knowledge and understanding	4	knowledge	
Attention to detail	3	skill	yes
Humble	3	skill	yes
Ethical	3	skill	
Accountable	3	skill	yes
Helpful	3	skill	yes

Table 4.4 (Continued)

Category	Hits	Area	New
Computer hardware	3	knowledge	yes
Passionate	3	skill	yes
Code reviews	2	knowledge	yes
Disciplined	2	skill	
Big picture	2	skill	yes
Multithreaded programming	2	knowledge	yes
Empathy	2	skill	
Abstraction	2	knowledge	
Specific language	2	knowledge	yes
Curiosity	1	skill	yes
Scientific method	1	knowledge	yes
Broad experience	1	knowledge	yes
Pointers	1	knowledge	
Asks for help	1	skill	yes
Writing games	1	knowledge	
Networking and communication	1	knowledge	
Asks questions	1	skill	yes
Failure analysis	1	knowledge	yes
Machine learning	1	knowledge	
Imperative Programming	1	knowledge	yes
Memory allocation	1	knowledge	

New = area not found in the current mapping lists derived from literature.

4.3.3 Discussion

While we cannot base any conclusions on the first round of a Delphi survey, there are several interesting observations from this data.

First, this hit list shows that the top five items were professional skills. In addition, all of these were present in the framework skills list. This certainly indicates that our industry experts value skills highly. This supports the first half of our hypothesis H1 where we posit that industry experts will have professional skills highly rated.

Second, all these highest hit-list skills were on the initial framework list. In addition, we can see that many of the knowledge categories from our list also were called out by many experts. This indicates that our framework appears to be doing a reasonable job predicting many of our content areas.

Third, we do see several categories that are new from this breakdown. While we could make a case for the area of "single language" as being possible to combine under one of our knowledge list categories like "writing programs", the language in the essays was direct enough that we want to take this category into our Round 2 ranking. This does support that our initial framework lists from research may not be comprehensive enough to cover all the areas our experts find as having importance.

4.4 Delphi Round 2 results

The results from the hit-list table become the inputs into our Round 2 industry expert survey. However, doing a ranking list on 49 items felt burdensome and could discourage some experts from completing the Round 2 entry. To reduce this number, we removed items that were only mentioned by one expert. While there is some risk one of these items might be seen by experts and rated very high, this seems to be a reasonable trade-off to build a manageable ranking list. Eliminating these entries, we end up with 37 items which seems much more reasonable. We considered further dropping those categories that had only two hits from our experts but dropping an additional seven items did not seem to change the overall number enough. We also further risk an area recognized by two experts being seen as very important when ranked alongside the other items. Further details about our survey structure can be seen in Appendix A.

Below, the results are presented both as statistical data in tabular form and a box-plot form.

Table 4.5 Round 2 statistical data of all industry results (28/31 samples)

Category	Rank	Mean	Std Dev
Communication	1	4.43	0.69
Problem solving	2	4.39	0.79
Teamwork and collaboration	2	4.39	0.69
Accountable	2	4.39	0.63
Fundamentals of programming	5	4.36	0.83
Passionate	5	4.36	0.78
Ethical	7	4.21	0.83
Attention to detail	8	4.18	0.86
Testing and debugging	9	4.14	0.89
Knowledge and understanding	9	4.14	0.76
Lifelong learning	11	4.11	0.74
Big picture	11	4.11	0.74
Curious	13	4.04	0.84
Developing good program design	14	3.93	0.94
Creative and innovative	15	3.89	0.88
Helpful	16	3.86	0.85
Disciplined	17	3.71	0.76
Code reviews	18	3.68	0.94
Unit test	19	3.61	1.13
Generating clear documentation	20	3.57	1.14
Writing programs	21	3.50	1.07
Data structures	21	3.50	1.07
Empathetic	23	3.46	1.00
Abstraction	24	3.39	0.99
Humble	25	3.32	0.98
Tools	26	3.25	0.97
Multithreaded programming	27	3.14	0.80
Operating systems	28	3.11	0.99
How computers work	29	3.07	1.05
Designing algorithms	30	3.04	1.07
Computer hardware	30	3.04	1.10
Object oriented programming	32	3.00	0.98
Scripting language	33	2.93	0.90

Table 4.5 (Continued)

Category	Rank	Mean	Std Dev
Program management	34	2.62	1.21
Multiple languages	35	2.39	1.07
Single language	36	2.36	1.19
Specific language	36	2.36	1.19

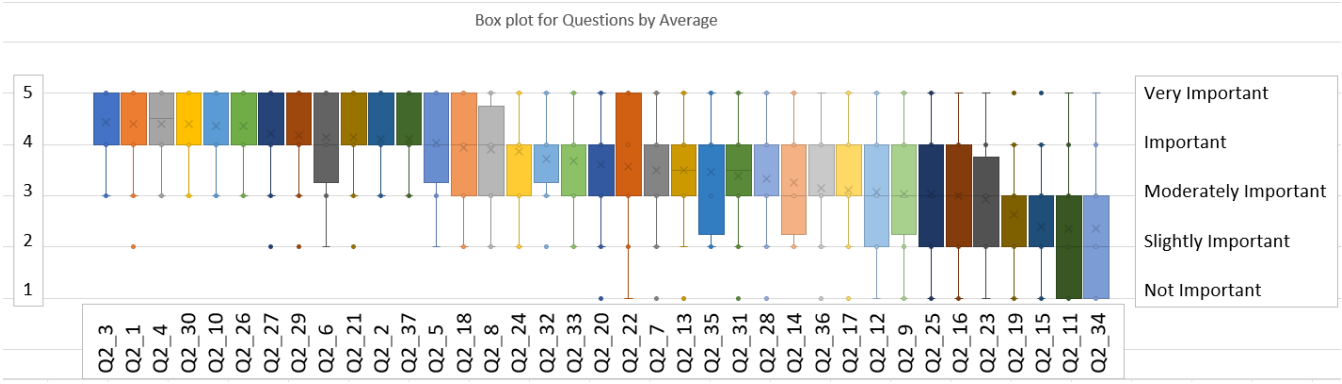


Figure 4.10 Round 2 boxplot of all industry result question statistics sorted by mean.

These results, statistically, show that we do have a reasonable separation in our data. The highest ranked item had a mean of 4.43 while the lowest item was only 2.36. Nothing came in under a mean of 2.00, which makes sense as everything on the initial Round 2 list had at least two people in the initial results call out that item. The highest ranked items are all professional skills: “communication”, “problem solving”, “teamwork and collaboration”, and “accountable”. The lowest ranked items are knowledge items: “multiple languages”, “specific languages”, “single languages”. This continues to support our hypothesis that industry experts will highly value professional skills while ranking some specific programming knowledge categories much lower.

On the knowledge areas, the highest ranked are: “fundamentals of programming”, “testing and debugging”, and “knowledge and understanding”. These appear to be more general topics than specific.

4.4.1 Comparison between Round 2 and Round 1 results

In order to evaluate how our expert rankings changed between these two survey passes, we calculated a delta change between where items were ranked in the Round 1 hit-list, and where they landed in the Round 2 results. The following table shows these deltas.

Table 4.6 Round 2 ranking versus Round 1 hit list ranking

Category	Rank	Hit List Rank	Delta
Teamwork and collaboration	1	4	3
Accountable	2	27	24
Communication	2	2	0
Problem solving	4	1	-3
Fundamentals of programming	5	10	4
Passionate	5	30	23
Testing and debugging	5	6	1
Ethical	8	26	17
Knowledge and understanding	8	23	15
Attention to detail	10	24	13
Big picture	10	33	21
Curious	10	5	-5
Lifelong learning	13	3	-10
Developing good program design	14	15	1
Helpful	15	28	13
Creative and innovative	16	9	-7
Code reviews	17	31	14
Unit test	18	19	1
Disciplined	19	32	13
Data structures	20	14	-6
Generating clear documentation	21	22	1
Tools	22	12	-11

Table 4.6 (continued)

Category	Rank	Hit List Rank	Delta
Writing programs	22	7	-15
Abstraction	24	36	12
Empathetic	25	35	10
Humble	26	25	-1
Operating systems	27	18	-9
Multithreaded programming	28	34	6
Object oriented programming	29	17	-12
Computer hardware	30	29	-3
Designing algorithms	30	8	-22
Scripting language	30	21	-10
How computers work	33	13	-20
Program management	34	20	-14
Multiple languages	35	16	-19
Specific language	36	37	1
Single language	37	11	-26

- Ties are ranked are the same level, and subsequent ranks are skipped. (If we have two items tied at #2 the next available rank is #4 and we have no #3.
- Delta is positive if Round 2 ranked was higher, negative if Round 2 rank was lower, and zero if the same.

Four of our top six are very similar. In particular, “communication” is #2 in both, “problem solving” fell from #1 in the hit list to #4 in the Round 2, and “teamwork and collaboration” is ranked #4 in the hit list and rose to #1 in the Round 2. From our review of literature, “teamwork and collaboration” and “communication” are the only skills included in our initial survey of syllabi. We can see that this data appears to backup that those may be the most important items in this list.

We have several significant ranking moves. The delta column shows how much the category rose or fell from the Round 1 hit list to the Round 2 ranking. The top five largest moves up are: “accountable”, “passionate”, “big picture”, “ethical”, and “knowledge and

understanding". Several of the ten largest moves up in our Round 2 ranking list were professional skills. The three knowledge items that have a move of greater than ten points were "code reviews", "knowledge and understanding", and "abstraction".

There appear to be two primary reasons that could motivate these moves in our Round 2 rankings. The first seems to be items that were assumed in the Round 1 answers. For example, "knowledge and understanding" was ranked 23rd in our Round 1 hit list with only 4 experts mentioning it, but its Round 2 rank was #8. This clearly indicates that most experts thought this was important, but few of them thought of this distinctly when filling out their Round 1 essay answer responses. Professional skills seemed to show this type of move clearly. While most experts did not mention many professional skills in their initial open-ended questions, many of them were ranked as important in the Round 2 results. Several of these professional skills fall into the skill subcategory that CC 2020 calls "attributes" [15]. Topping this list is "accountable". While this was 27th in our Round 1 hit list, with only three experts calling this out, it jumped 24 spots to tie for rank #2. This means that while only three of our expert group thought to call out this item when describing what knowledge and professional skills new engineers need to have, almost all of our experts acknowledged how important this attribute is. A corollary of this finding is that several knowledge items were ranked much lower in this survey step. Areas that fell ten or more slots were all knowledge items, with four falling more than twenty spots: "object oriented programming", "designing algorithms", "multiple languages", and "single language". "Single language", for example, was #11 in our initial list, but was #37 in the Round 2 list. While more work would be needed to expose the underlying reasons, we suggest that specific language knowledge is clearly not as important as most other items.

On the other end of the spectrum, we had professional skills that dropped in the Round 2 assessment. “Lifelong learning”, which was ranked #3 in our hit list, fell to #13. While this professional skill is still considered important with a mean of 4.11, it was not ranked as “very important” by many experts. We have foreshadowed that we expected knowledge areas which are very specific, language knowledge in particular, would rank lower for our industry experts. The data bears this out. While I expected the top knowledge category from our review of literature, “writing programs”, would have remained reasonably high in the Round 2 ranking, it fell 15 ranks from #7 to #22. This seemed to be overshadowed by “knowledge and understanding” which rose from #23 to #8. If engineers have base level programming knowledge, it may be that it is assumed they have the ability to write programs.

From only this result, we already see a clear signal that industry experts generally place more emphasis on many professional skills over particular knowledge categories. Of the top thirteen ranked Round 2 items, 10 are skills. Of the bottom ten ranked items, all are knowledge-based items. It is clear that a focus on professional skills in the engineering degree, and programming specifically, must be undertaken to produce graduates that are fully capable of stepping into first-time jobs.

4.4.2 Discussion

While this data clearly supports our main hypothesis, there seems to be some industry assumptions that every candidate will meet some minimum level of programming knowledge to be considered for a job. If a job applicant could not program in any language with some proficiency, they would be a non-starter for a programming job. Future work to discover what this minimum standard is would be useful as this needs to be fully covered in the knowledge requirements from college degrees and programming course syllabi.

4.5 Delphi Round 3 results

Round 3 shows both the expert's group responses as well as their initial response on each ranked area. They are then asked to consider if they want to move their rating based on the group result. The survey was presented to our experts like this:

Figure 4.11 Example of Round 3 survey

This has three notable items which are added to what was in the Round 2 survey. First, the default answers in this survey were populated from the individual results during the Round 2 round. Second, the ranking in the list and the mean data are presented in the areas. Third, a small text window was provided for feedback. Filling in the Round 3 survey, experts can keep or change their response. In the instructions, the experts were asked to make a short note in the text field if they either a) made a significant change to their prior answer, or b) kept their prior answer in spite of a significant delta from the group results. This is intended to provide some deeper ability to understand what an expert was thinking if they had a divergent response versus the group. Thirteen of my twenty-nine industry respondents utilized these text fields to some degree.

Here are the final Round 3 results.

Table 4.7 Statistical data of all industry Round 3 results (24/31 samples)

Category	Area	Rank	Mean	Std Dev
Problem solving	Skill	1	4.58	0.50
Accountable	Skill	2	4.46	0.51
Fundamentals of programming	Knowledge	2	4.46	0.66
Communication	Skill	4	4.38	0.71
Testing and debugging	Knowledge	4	4.38	0.65
Attention to detail	Skill	6	4.29	0.86
Teamwork and collaboration	Skill	7	4.21	0.66
Ethical	Skill	8	4.17	0.92
Lifelong learning	Skill	8	4.17	0.70
Passionate	Skill	8	4.17	0.82
Curious	Skill	11	4.08	0.72
Developing good program design	Knowledge	11	4.08	0.83
Knowledge and understanding	Knowledge	11	4.08	0.72
Big picture	Skill	14	4.04	0.75
Creative and innovative	Skill	15	3.96	0.75
Code reviews	Knowledge	16	3.92	0.83
Unit test	Knowledge	17	3.88	0.90
Disciplined	Skill	18	3.83	0.70
Data structures	Knowledge	19	3.71	0.81
Generating clear documentation	Knowledge	19	3.71	1.12
Helpful	Skill	21	3.67	0.76
Abstraction	Knowledge	22	3.58	0.83
Empathetic	Skill	23	3.50	0.93
Writing programs	Knowledge	23	3.50	1.10
Designing algorithms	Knowledge	25	3.42	0.78
Humble	Skill	25	3.42	0.78
Tools	Knowledge	27	3.38	0.92
Multithreaded programming	Knowledge	28	3.29	0.86
How computers work	Knowledge	29	3.21	0.88
Operating systems	Knowledge	29	3.21	0.83
Object oriented programming	Knowledge	31	3.17	0.92
Scripting language	Knowledge	32	3.04	0.95
Computer hardware	Knowledge	33	2.88	0.74
Multiple languages	Knowledge	34	2.54	1.02

Table 4.7 (continued)

Category	Area	Rank	Mean	Std Dev
Program management	Knowledge	35	2.42	0.88
Single language	Knowledge	35	2.42	1.02
Specific language	Knowledge	37	2.29	0.95

If we compare these results to our Round 2 data, we see that we have some changes, but things remained relatively stable. At most items moved no more than 5 rank items up or down. Here is the delta list.

Table 4.8 Round 3 ranking and mean versus Round 2

Category	Round3 Rank	Round2 Rank	Rank Delta	Mean Delta
Problem solving	1	2	1	0.19
Accountable	2	2	0	0.07
Fundamentals of programming	2	5	3	0.10
Communication	4	1	-3	-0.06
Testing and debugging	4	9	5	0.24
Attention to detail	6	8	2	0.11
Teamwork and collaboration	7	2	-5	-0.18
Ethical	8	7	-1	-0.04
Lifelong learning	8	11	3	0.06
Passionate	8	5	-3	-0.19
Curious	11	13	2	0.04
Developing good program design	11	14	3	0.15
Knowledge and understanding	11	9	-2	-0.06
Big picture	14	11	-3	-0.07
Creative and innovative	15	15	0	0.07
Code reviews	16	18	2	0.24
Unit test	17	19	2	0.27
Disciplined	18	17	-1	0.12
Data structures	19	21	2	0.21
Generating clear documentation	19	20	1	0.14
Helpful	21	16	-5	-0.19
Abstraction	22	24	2	0.19

Table 4.8 (continued)

Category	Round3 Rank	Round2 Rank	Rank Delta	Mean Delta
Empathetic	23	23	0	0.04
Writing programs	23	21	-2	0.00
Designing algorithms	25	30	5	0.38
Humble	25	25	0	0.10
Tools	27	26	-1	0.13
Multithreaded programming	28	27	-1	0.15
How computers work	29	29	0	0.14
Operating systems	29	28	-1	0.10
Object oriented programming	31	32	1	0.17
Scripting language	32	33	1	0.11
Computer hardware	33	30	-3	-0.17
Multiple languages	34	35	1	0.15
Program management	35	34	-1	-0.20
Single language	35	36	1	0.06
Specific language	37	36	-1	-0.07

The mean delta was also a max drop of -0.20 points and a max gain of 0.38 points. This is around one-third of a rating point.

4.5.1 Discussion

The collective data did have subtle but significant effects. “Problem solving” is a good example to review. From a rank perspective, the Round 2 survey had this tied for #2. In the Round 3 results, we saw this overtake the #1 position. A histogram of the results shows why this happened.

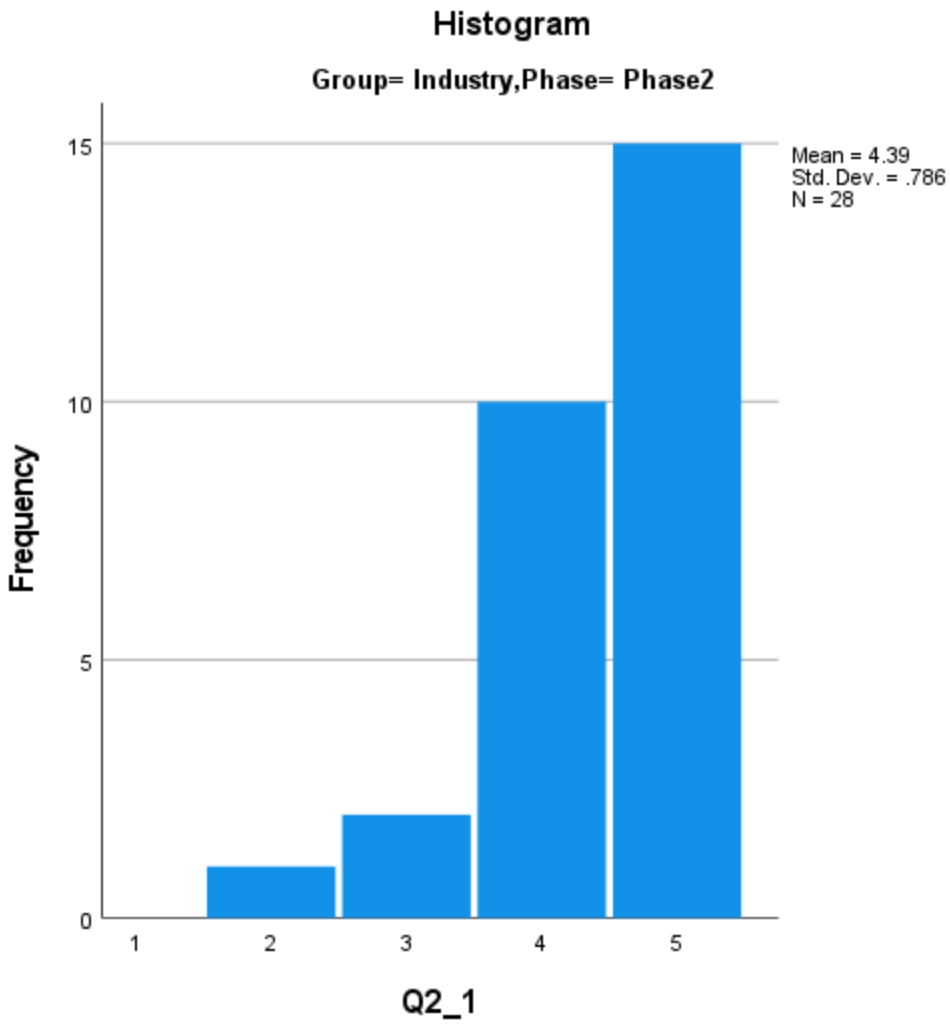


Figure 4.12 Round 2 histogram for “Problem solving”

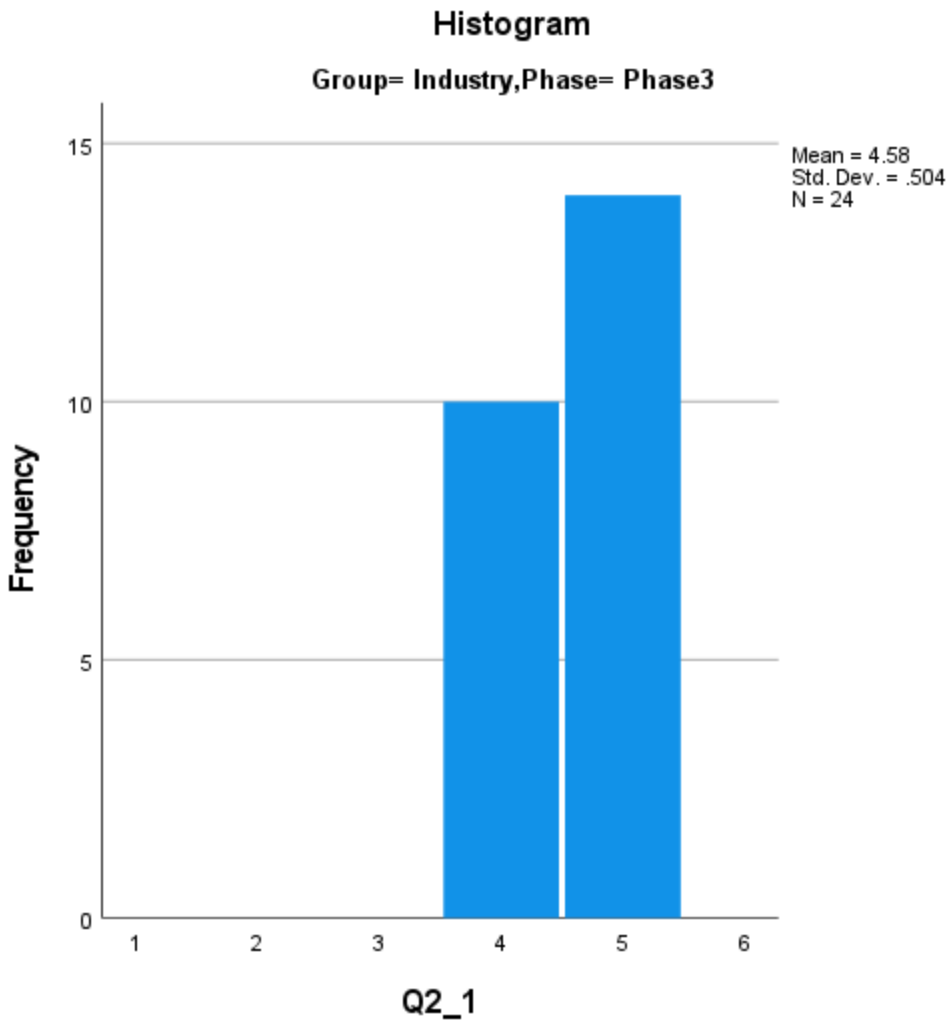


Figure 4.13 Round 3 histogram for “Problem solving”

We can see that Round 2 had one expert who rated this “Slightly important (2)”, and two experts who rated this “Moderately important (3)”, in the round three results we had no-one who placed this below “Important (4)”. In particular, the expert who had coded this at a “2” in the Round 2 ranking changed this to a “4” in the Round 3 and added this comment: “Moved it to important; original thinking was that not all of the work requires solving problems in its classic meaning.” This is a great example of the extra round of survey with the group results giving

experts an opportunity to compare their initial ranking with the overall feel of the group. In this case, this increased the mean of this category because we moved an outlier up significantly.

The largest move up the ranking was “Testing and debugging” which went from #9 to #4. While there were a few single point moves up in this group, we also had the expert which rated this “2” in Round 2 not respond to Round 3. This takes that low number out of the calculations. It is difficult to know whether that expert would have increased his ranking based on the group statistics or not. While there were not many comments on this category, one of the experts, who rated this a “Very important (5)”, did add this note: “Most of the work is testing and debugging. Probably the most crucial skill.” While there are some questions about the corners of the data, we believe this is still the best consensus result from the team with the data we have.

The category of “teamwork and collaboration” was the largest drop from #2 to #7. This was a little surprising as this seemed to be at the top of most of our lists going into Round 3. Looking at the detail this seems to be the opposite of what we saw in “testing and debugging”. Several experts who ranked this “5” did not participate in the Round 3 survey. This totaled six experts. Looking at the other respondents, there were almost no changes in ranking. This led to a reduction in the overall mean because we lost more high rankings based upon who did not complete the Round 3 survey. We are under our desired limit of thirty responses which means we can get some larger swings from this type of phenomenon. In a similar way, “helpful” also has several “5” responses in the Round 2 results which were also missing from Round three causing a drop of five ranks. It makes sense that experts who rated “teamwork and collaboration” as very important, would rate “helpful” equally high. One of the other experts, who rated this a “3” added this clarifying comment. “Being helpful is nice, but the majority of the time in coding

is spent alone." This is an example of where our expert group was not uniformly agreed on this particular category which resulted in a lower ranking.

While there are many additional levels of detail that could be gleaned from this data, this gives enough examples to show that our final consensus list does represent the overall importance of these knowledge and skill areas relative to each other.

4.6 Implications

The Round 3 of the Delphi survey did not show significant moves away from the Round 2 results, but it did give us a consensus rating which allows us to believe the top and bottom categories have general agreement from our industry experts. Recapping our top ten:

Table 4.9 Top ten Round 3 categories

Category	Area
Problem solving	Skill
Accountable	Skill
Fundamentals of programming	Knowledge
Communication	Skill
Testing and debugging	Knowledge
Attention to detail	Skill
Teamwork and collaboration	Skill
Ethical	Skill
Lifelong learning	Skill
Passionate	Skill

We see, like the Round 2 results, that only two knowledge categories were rated higher than professional skills. As the mean for this group went from 4.17 to 4.58, we have the consensus rating at between "Important (4)" and "Very important (5)".

Looking at the bottom ten, we see the reverse trend.

Table 4.10 Bottom ten Round 3 categories

Category	Area
Multithreaded programming	Knowledge
How computers work	Knowledge
Operating systems	Knowledge
Object oriented programming	Knowledge
Scripting language	Knowledge
Computer hardware	Knowledge
Multiple languages	Knowledge
Program management	Knowledge
Single language	Knowledge
Specific language	Knowledge

Only one of these categories is a professional skill. In addition, all the focus on languages (scripting, multiple, single, specific) rated much lower than other areas.

It appears to be the consensus places much more importance on professional skills then on detailed programming knowledge. This is not a completely conclusive statement. Even with the bottom two categories here, “single language” and “specific language”, our means were still 2.29 and 2.42. This means they rated between “slightly important (2)” and “moderately important (3)”. Where some of the experts did classify these two as “not important (1)”, others had rankings as high as “very important (5)”. There may still be some assumption that programmers will have some basic mastery of a single, or even of multiple, programming languages. It would take some more detailed surveys to understand this nuance. However, the overall finding is clear. For a new hire to excel as an engineer and a programmer, they must have some mastery of these top-rated professional skills.

CHAPTER V

RESULTS: ACADEMIC EXPERTS

5.1 Academic experts

Academic experts are the second group critical to understanding knowledge and professional skills rankings. Surveying academics who teach CS1 as well as more advanced programming engineering courses will help understand what those closest to the classroom feel are important. While there is expected to be some level of overlap, we expect academic experts to have more highly ranked knowledge categories compared to their ranked professional skill areas.

In order to build our list, we pulled a list of over 1300 ABET accredited engineering program, randomized them, and started searching websites for EE or ECE department chairs. We sent a blind request for assistance email to 156 chairs asking if they might have professors with some experience with teaching introductory programming courses to engineering students. While we only had 40 chairs respond, we were able to assembly a list of 71 potential professors who were good candidates. Out of this list, we had 33 which agreed to join our Delphi survey.

5.2 Expert classification breakdown

As with our Industry experts, we asked our Academic experts several classification questions to see if we had a broad representation of time teaching and experience in programming areas.

5.2.1 Coding area

This question is identical to our Industry question.

- Where do you spend most of your coding time (check all that apply)?

For reference, we pulled a chart that provides one cross section of different programming areas [15].

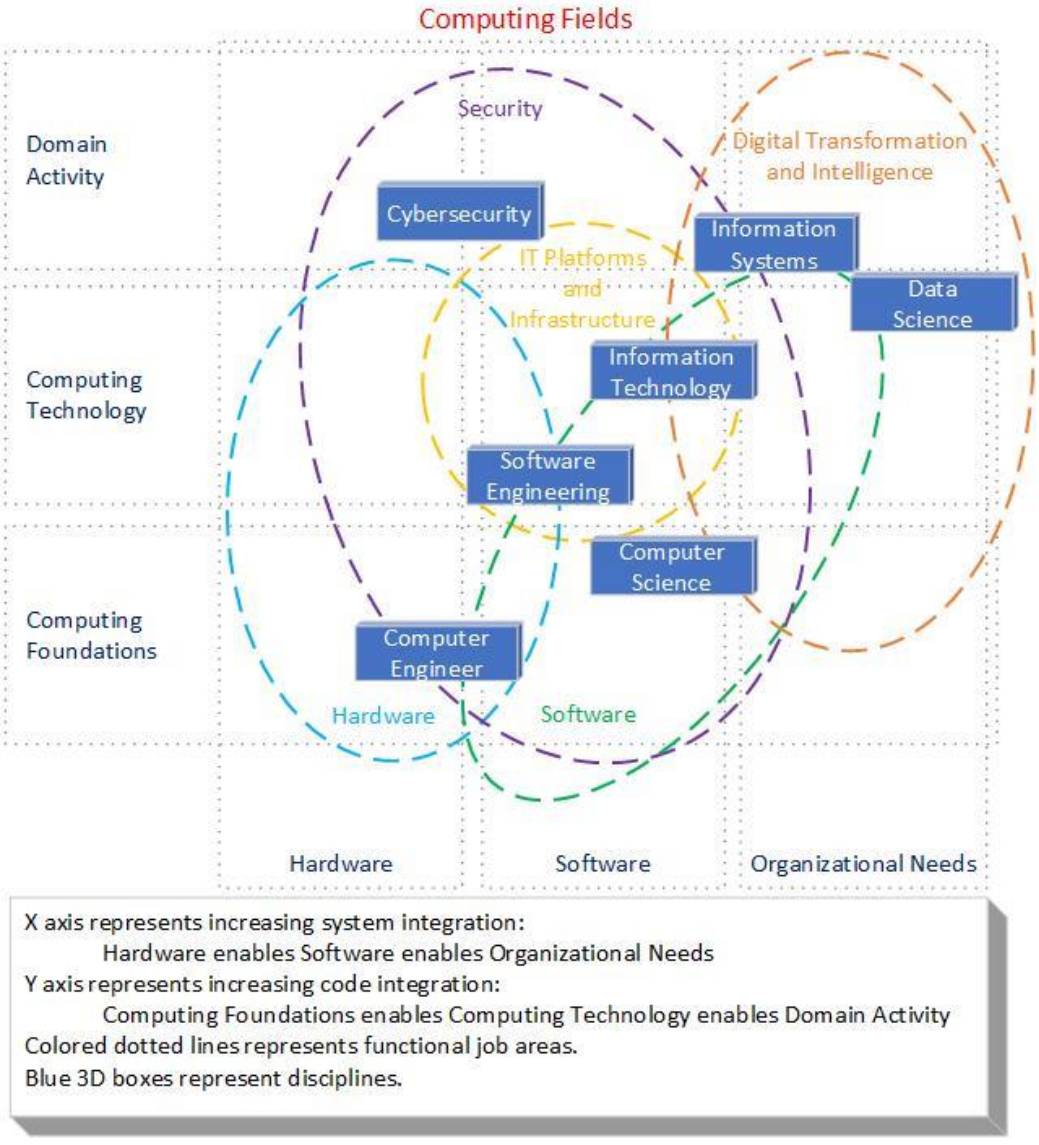


Figure 5.1 Contemporary view of the landscape of computing education (based on original figure in [15])

From our expert sample we have coverage across several disciplines, but Computer Science and Software Engineer were clearly the most common self-identified areas.

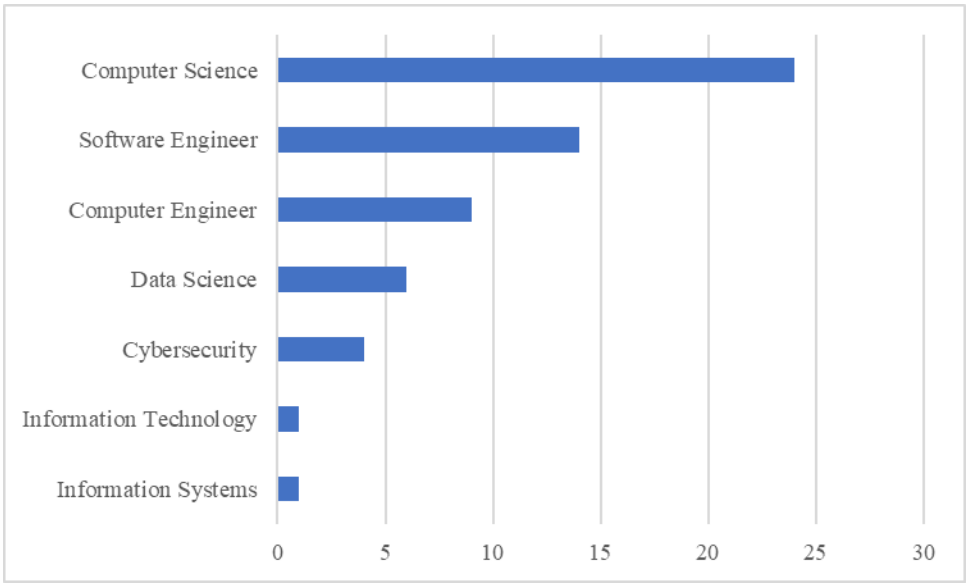


Figure 5.2 Result: Where do you spend most of your coding time

While "computer engineer" was our #1 category in our industry group, "computer science" and "software engineering" rank above "computer engineer" for our academics. This is not surprising as many of our academic experts were actually in the computer science field.

5.2.2 Courses taught

Our second question focuses on what level of courses our expert teach.

- What level of engineering/computer science programming courses do you teach (check all that apply)?

Beginning programming is ranked the highest, which matches our methodology to find experts in this area, but we have some representation through all the other categories as well.

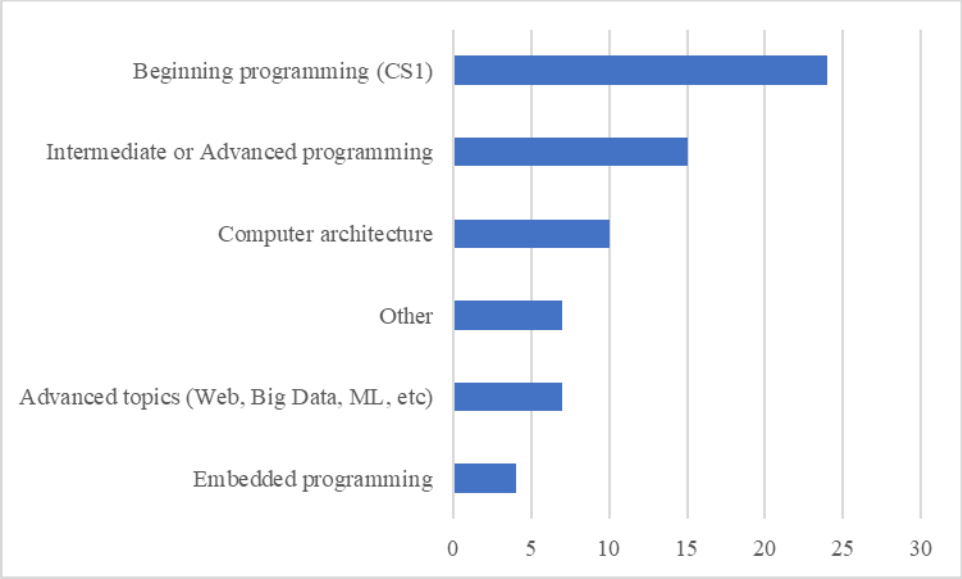


Figure 5.3 Result: What level of engineering/computer science programming courses do you teach

This shows we should be getting a reasonable cross section of experts who teach across the spectrum of engineering programming courses included Embedded programming.

5.2.3 Years teaching

Our third question is:

- How long have you been teaching?

Almost half of our experts had been teaching for longer than 10 years. While there may be some differences between new teachers and experienced teachers, we expect all of these professors are keeping up with current expectations and standards.

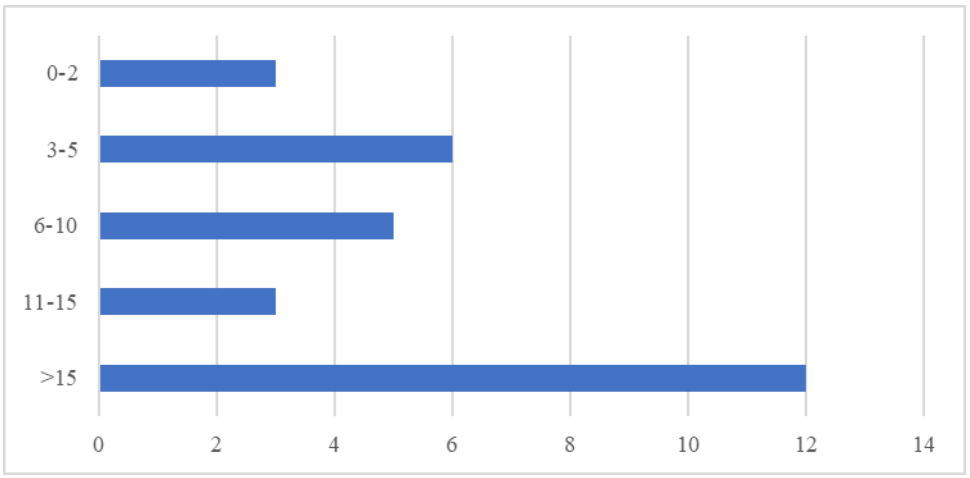


Figure 5.4 Result: How long have you been teaching

5.2.4 Conducts research

In order to discover how many professors are doing research in programming, we asked this question.

- Do you conduct research in programming or programming educations?

We would expect professors doing research in this area may be more involved with various pedagogical techniques.

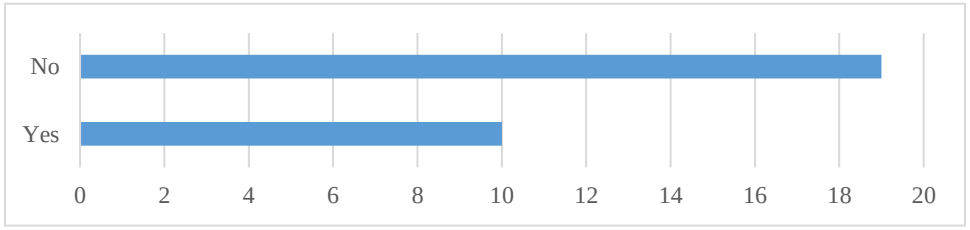


Figure 5.5 Result: Do you conduct research in programming or programming educations

Only a third of our expert group are also researching in this area. This could lead to a more “traditional” bent among these teachers, however, that assertion would take more direct questions. We do not add questions at this level of detail.

5.2.5 Main coding language

This question also mirrors the same question we ask our Industry group.

- What languages do you spend the most time in or teach (select all that apply)?

We can see that Python is also a slight winner among our academics as it was with our industry experts.

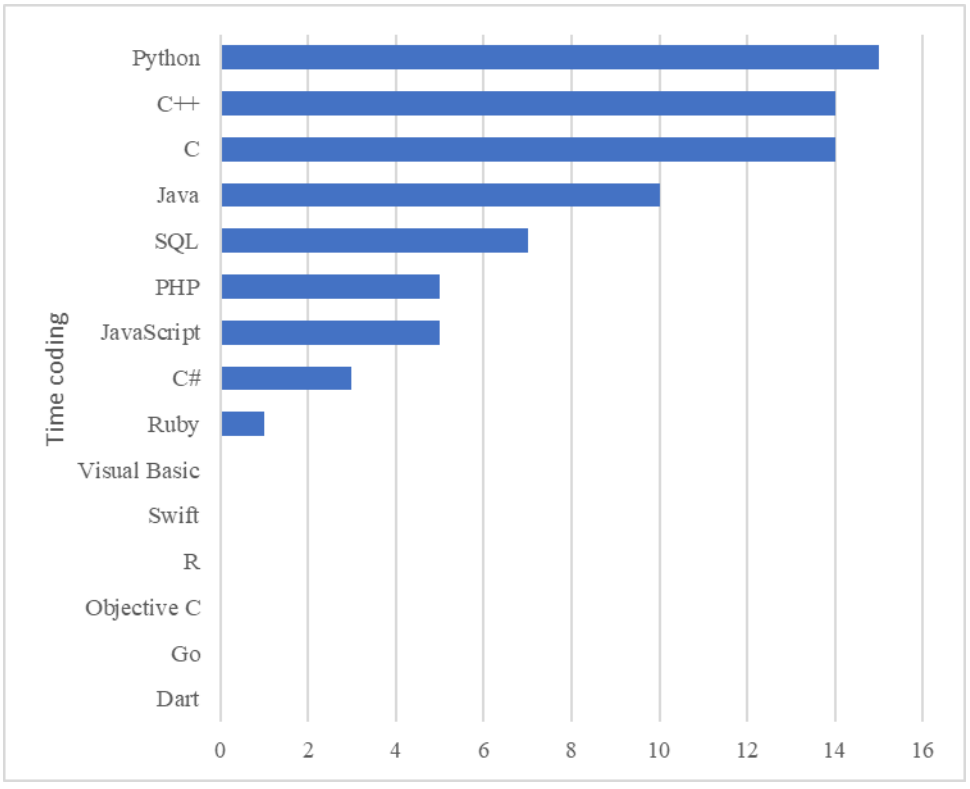


Figure 5.6 Result: What languages do you spend the most time in or teach

The other languages (C++, C, Java, etc) are very similar to our Industry group. C# had a much higher number of users in our industry group, which may be particular to the company we surveyed. In generally, this shows high agreement on which languages are most valuable to teach and to know in industry. While a small sub-finding, we believe this is a good area to have industry and academic agreement. We did not ask specifically what languages they taught, but we would expect these align closely with what they program in.

5.2.6 Industry experience

To learn whether our Academic group had interactions with industry programmers, we ask this question.

- Do you have industry experience?

While this is slightly vague and could mean worked in industry or could mean worked with industry, the results show that two-thirds of our academics self-identified as having industry experience.

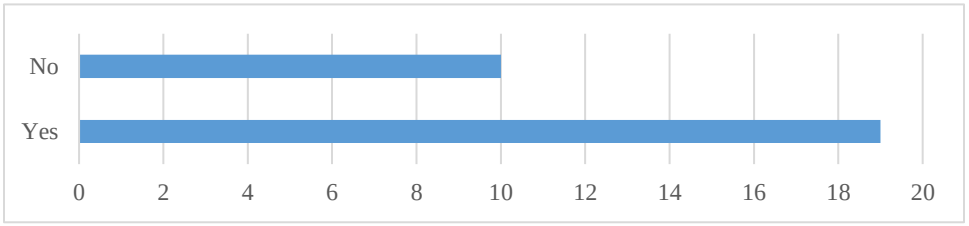


Figure 5.7 Result: Do you have industry experience

This would lead us to assume that there should be a fair amount of overlap between these two groups when we have our final ranking.

5.2.6.1 Years of industry experience

If our experts answered “yes” to the prior question, we followed up with a question on duration.

- How many years of industry experience?

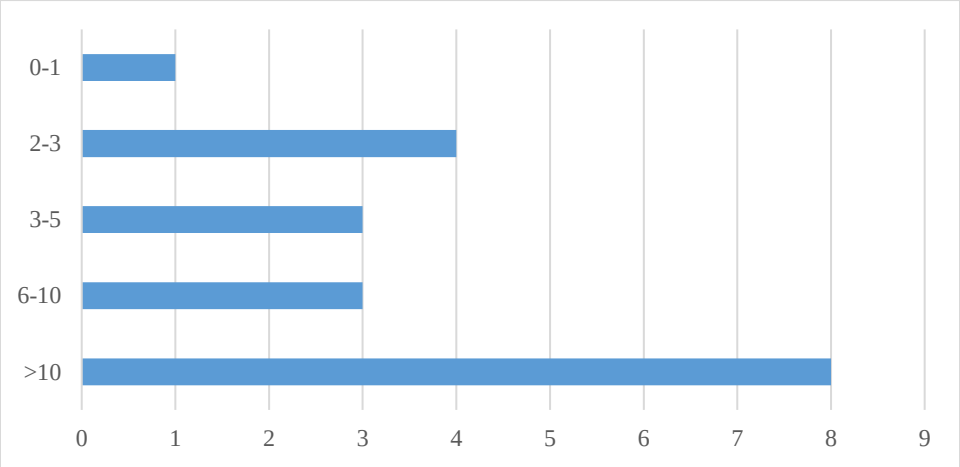


Figure 5.8 Academic years of industry experience

While we had some answers in every category, eight of our nineteen experts claimed over 10 years of experience. This leads us to believe that these experts are likely to be cooperating with industry concurrent with their teaching. Understanding the opinions of second career teachers as well as teachers who also work with industry might be an interesting topic for further study.

5.3 Classification summary

These questions assess the diversity of our expert group. Like our industry group, we show reasonable diversity across all of these metrics. However, it is clear our expert group is weighted to professors who generally have ten or more years teaching.

5.4 Round 1 results

The process for classification matches follows the methodology laid out in Chapter 3 with examples of how this is done in Chapter 4. This matched what was done with our industry experts.

5.4.1 Round 1 area "hit list"

For Round 1 results, we have no ranking as all the key items are extracted from the essay responses. However, counting how many experts "hit" on the same area gives us some initial assessment around how likely experts in our ranking sessions will highly rank the items they mentioned. Here are the classification mapping results.

Table 5.1 Academic framework mapping "hit-list"

Category	Rank	Hits	Area	New
Writing programs	1	18	knowledge	
Problem solving	2	16	skill	
Teamwork and collaboration	3	15	skill	
Testing and debugging	4	13	knowledge	
Designing algorithms	5	12	knowledge	
Lifelong learning	6	11	skill	
Single language	6	11	knowledge	yes
Developing good program design	8	10	knowledge	
Communication	9	9	skill	
Data structures	9	9	knowledge	
Version control	9	9	knowledge	yes
Abstraction	12	8	knowledge	
Fundamentals of programming	12	8	knowledge	
Generating clear documentation	14	7	knowledge	
Multiple languages	14	7	knowledge	
Tools	14	7	knowledge	
Receives feedback well	17	6	skill	
Comprehending programs	18	5	knowledge	
Evaluating time/space complexity	18	5	knowledge	
Internships	20	4	knowledge	yes

Table 5.1 (continued)

Category	Rank	Hits	Area	New
Networking and communication	20	4	knowledge	
Object oriented programming	20	4	knowledge	yes
Persistence	20	4	skill	yes
Code reviews	24	3	knowledge	
Control structures and logic	24	3	knowledge	yes
File handling and I/O	24	3	knowledge	
How computers work	24	3	knowledge	
IDE	24	3	knowledge	
Operating systems	24	3	knowledge	
Pseudocode	24	3	knowledge	
Refactoring code	24	3	knowledge	
Specifications	24	3	knowledge	
Unit test	24	3	knowledge	
Web development	24	3	knowledge	
Arrays dictionaries lists vectors	35	2	knowledge	
Asks for help	35	2	skill	
Assembly language	35	2	knowledge	
Attention to detail	35	2	skill	
Coding to API	35	2	knowledge	
Command prompt for compilation and execution	35	2	knowledge	
Databases	35	2	knowledge	
Ethics	35	2	skill	
Life Cycle	35	2	knowledge	
Memory allocation	35	2	knowledge	
Threading and concurrency	35	2	knowledge	yes
Accountability	46	1	skill	
Advanced data structures	46	1	knowledge	
Designing a user interface	46	1	knowledge	
Experimentation and judgement	46	1	skill	yes
Gathering client requirements	46	1	skill	
Imperative programming	46	1	knowledge	
Inheriting and extending others' code	46	1	knowledge	
Meets deadlines	46	1	skill	
Pattern recognition	46	1	knowledge	yes
Pointers	46	1	knowledge	
Program comprehension	46	1	knowledge	
Program management	46	1	knowledge	
Project management	46	1	knowledge	

Table 5.1 (continued)

Category	Rank	Hits	Area	New
Regression testing	46	1	knowledge	
Repetition and loops	46	1	knowledge	
Scope of code	46	1	knowledge	
Scripting language	46	1	knowledge	
Searching algorithms	46	1	knowledge	
Security	46	1	knowledge	
Skill in stay motivated	46	1	skill	
Sorting algorithms	46	1	knowledge	
Specific language	46	1	knowledge	
Teachable	46	1	skill	
Tracing execution of program	46	1	knowledge	
UML	46	1	knowledge	yes
Variables assignments	46	1	knowledge	
Writing large program	46	1	knowledge	

5.4.2 Discussion

From a volume perspective, the Academic group called out 72 separate categories where the industry experts only enumerated 50. For our academic group, we have only 15 professional skills out of their 74 total items, and only 4 of their top ten items were skills. If we look at how many professional skill items between the groups, we see our industry experts had 18 out of 50, but 6 of the top 10 ranked items were skills. At an aggregate level, it does seem that our initial hypothesis is correct. Academic experts have a higher importance on knowledge areas versus professional skills.

If we look at the two professional skills from both groups, we see some overlap.

Table 5.2 Professional Skills comparison of Industry and Academic Round 1 results

Ind Skill	Ind Rank	Aca Skill	Aca Rank
Problem solving	1	Problem solving	2
Communication	2	Teamwork and collaboration	3
Lifelong learning	2	Lifelong learning	6
Teamwork and collaboration	4	Communication	11
Curious	5		
Creative and innovative	8		

The top four are identical, if in a slightly different order. This may be significant as agreement on these professional skills may provide a platform for where increased focus could be applied in programming classes. "Curious" and "Creative and innovative" are interesting. While they ranked high in the industry list, they were not even mentioned in the Round 1 academic survey.

We remove everything with only one or two expert callouts to reduce the list to 33 items which is more manageable for our Round 2 ranking. In addition, we add "curious" and "creative and innovative" which were not mentioned in the Academic Round 1. We hope to determine whether these professional skills were excluded as an oversite or if these items are really not valued by our Academic experts. Including Round 2 areas that did not come from the Round 1 results is a deviation from our Delphi survey method, but we believe this is warranted allow a broader analysis of these high ranked industry professional skills in our final gap analysis. Having 35 items to rank is similar to the 37 items we had on the industry Round 2 survey.

5.5 Academic Round 2 analysis

Our survey structure is identical to the industry survey asking for an importance selection across a randomized list of the items. That statistical tabular results and the box-plot format below show the result of our Round 2 data.

Table 5.3 Round 2 statistical data of all academic results (23/33)

Category	Rank	Mean	Std Dev	Variance	Kurtosis	Skewness
Fundamentals of programming	1	4.64	0.12	0.34	1.20	-1.39
Problem solving	2	4.61	0.14	0.43	1.20	-1.50
Testing and debugging	3	4.52	0.14	0.44	0.19	-1.10
Writing programs	3	4.52	0.12	0.35	-0.22	-0.81
Control structures and logic	5	4.50	0.17	0.64	-0.20	-1.22
Comprehending programs	6	4.32	0.15	0.51	-0.76	-0.57
Developing-and coding to-specifications	6	4.32	0.18	0.70	1.31	-1.23
Developing good program design	8	4.30	0.17	0.68	1.33	-1.18
Persistence	9	4.26	0.16	0.57	-1.00	-0.49
Communication	10	4.13	0.16	0.57	1.61	-0.92
Data structures	10	4.13	0.16	0.57	-1.14	-0.23
Teamwork and collaboration	12	4.09	0.12	0.36	0.16	-0.01
Abstraction	13	4.00	0.20	0.91	-0.28	-0.69
Generating clear documentation	13	4.00	0.18	0.73	-0.29	-0.48
Lifelong learning	13	4.00	0.25	1.33	-0.77	-0.82
Receives feedback well	16	3.91	0.20	0.90	-0.51	-0.51
Object oriented programming	17	3.87	0.17	0.66	-0.23	-0.30
Curious	18	3.77	0.25	1.42	-0.37	-0.63
Creative and innovative	19	3.70	0.16	0.58	-0.15	-0.07
File handling and I/O	20	3.65	0.20	0.96	1.14	-0.78
Multiple languages	21	3.57	0.19	0.80	-0.51	-0.21
Networking and communication	22	3.48	0.18	0.72	-0.34	0.32
Refactoring code	22	3.48	0.21	0.99	-0.92	0.07
Internships	24	3.45	0.21	0.93	-0.82	-0.03
Evaluating time/space complexity	25	3.43	0.14	0.44	0.14	0.26
Designing algorithms	26	3.39	0.21	0.98	-0.94	0.02
Code reviews	27	3.35	0.18	0.78	-0.21	0.51

Table 5.3 (continued)

Category	Rank	Mean	Std Dev	Variance	Kurtosis	Skewness
How computers work	27	3.35	0.23	1.24	0.11	-0.56
Version control	29	3.30	0.23	1.22	-0.68	-0.23
Single language	30	3.27	0.24	1.26	-0.51	0.07
Operating systems	31	3.13	0.17	0.66	-0.23	0.30
Tools	31	3.13	0.20	0.94	-0.18	0.71
IDEs	33	2.95	0.22	1.09	-0.98	0.10
Pseudocode	34	2.91	0.24	1.23	-0.58	0.89
Web development	35	2.78	0.22	1.09	-0.16	-0.05

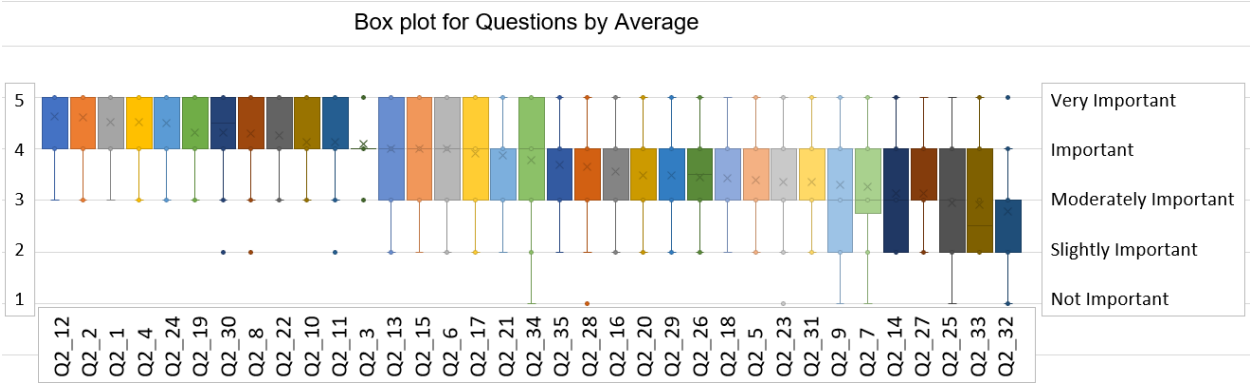


Figure 5.9 Round 2 boxplot of all academic result question statistics sorted by mean rank.

Like our industry data, there is a clear mean separation between the highest and lowest ranked items. In addition, we see most of the low ranked items have some “very important” and some “not important” classifications, while the highest ranked items have nothing lower than “moderately important” which supports the high mean. Also matching our Industry group, the lowest mean was 2.78 which is still just slightly under “moderately important”. All of the items listed to be ranked were generally considered important to some degree. Our added “curious” and “creative and innovative” ranked in the middle of the pack with a 3.77 and a 3.70 mean

respectively. These are still close to the “important” group to our Academic group. While these were not called out in the Round 1 by any expert, they did value these above many of their other explicitly called out items. They did rank slightly lower than then did with our Industry group (4.04 and 3.89).

Looking at the table, the top twelve items are dominated by knowledge areas. There are only 4 professional skills: “problem solving” at #2, “persistence” at #9, “communication” at #11, and “teamwork and collaboration” at #12. It is not surprising that “fundamentals of programming” ended up at #1 while “writing programs” and “testing and debugging” came in at #2 and tied for #3. In addition, three items in the top 10 were items that did not show up in the industry list at all: “control structure and logic”, “comprehending programs”, and “developing- and coding to-specifications”.

The bottom three items were all knowledge categories, but they still had a mean close to “moderately important”. These items were “IDEs”, “pseudocode”, and “web development”. While these are in the bottom of the pack, we see nothing special that separate these from the other items slightly higher in mean.

5.5.1 Academic hit-list to Round 2 results deltas

The following table shows how our Round 2 results varied from our initial hit-list.

Table 5.4 Round 2 ranking versus Round 1 hit-list ranking

Category	Rank	Hit List Rank	Delta
Fundamentals of programming	1	12	11
Problem solving	2	2	0
Testing and debugging	3	4	1
Writing programs	3	1	-2

Table 5.4 (continued)

Category	Rank	Hit List Rank	Delta
Control structures and logic	5	18	13
Comprehending programs	6	9	3
Developing-and coding to-specifications	6	24	18
Developing good program design	8	8	0
Persistence	9	20	11
Communication	10	35	25
Data structures	10	9	-1
Teamwork and collaboration	12	3	-9
Abstraction	13	12	-1
Generating clear documentation	13	14	1
Lifelong learning	13	6	-7
Receives feedback well	16	17	1
Object oriented programming	17	20	3
Curious	18	73	55
Creative and innovative	19	73	54
File handling and I/O	20	24	4
Multiple languages	21	14	-7
Networking and communication	22	20	-2
Refactoring code	22	24	2
Internships	24	20	-4
Evaluating time/space complexity	25	18	-7
Designing algorithms	26	5	-21
Code reviews	27	24	-3
How computers work	27	24	-3
Version control	29	9	-20
Single language	30	6	-24
Operating systems	31	24	-7
Tools	31	14	-17
IDEs	33	24	-9
Pseudocode	34	24	-10
Web development	35	24	-11

Similar to our Industry results, we do see some big moves from our Round 1 to our Round 2 results. Several items ended up high in each list: “problem solving” was #2 in each list,

“testing and debugging” went from #4 to #2, and “writing programs” went from #1 to #3. Our Round 2 #1 result “fundamentals of programming” moved up significantly from its #12 Round 1 position.

For our largest moves upwards, “curious” and “creative and innovative”, were not in the Round 1 list. This means they were ranked #73 (beyond the 72 total items) so the delta move of 54 and 55 is a little deceptive. Still, they ended up #18 and #19 for the Round 2 list, as discussed in the last section. The professional skill “communication” also moved from #35 to #10 while “teamwork and collaboration” dropped from #3 to #12. This does not completely match the thought that both would finish high. We don’t have any particular data to understand why “teamwork and collaboration” dropped so much. We will see if this corrects at all in the Round 3 data. “Persistence” also saw an 11 position move from #20 to #9. On the knowledge area side, “developing-and coding to-specifications” increased from #24 to #6, and “control structures and logic” jumped from #18 to #5.

The downward moves were equally interesting. “Single language” lost 24 ranks moving from #6 to #30. “Version control”, “tools”, “web development”, and “pseudocode” all moved down 10-20 ranks to end up in the high twenties and thirties. The most significant drops were in knowledge areas apart from two interesting professional skills. “Teamwork and collaboration”, as we have already mentioned, dropped 9 spots to fall out of the top 10. “Lifelong learning” also fell out of the top 10 dropping 7 ranks.

5.5.2 Discussion

At this point, we once again see that our hypothesis concerning academics rating knowledge areas higher than skill areas continues to appear true. Seven of our top ten items are knowledge based where only two knowledge areas broke the top ten in the industry group.

Overall, our Round 2 items only started with 8 out of 35 items being skills, and two of those were added from our industry list.

At the same time, the bottom 16 items in the Round 2 list were all knowledge areas and had mean values from 2.78 to 3.65. While this is still in the "moderately important" range, it does show that there are several knowledge areas that rated lower than all of the 8 professional skills in the overall ranking. In the comment section of my survey, one of the professors made what we feel is a telling comment, "The problem is that it's always easy to find lots of things that are important, but there is only so much time."

"Teamwork and collaboration" and "lifelong learning" both dropping out of the top ten are interesting data points. This seems to indicate that these are not as highly valued as many other areas. Our industry group Round 3 data had "lifelong learning" at #11, so that group also dropped this out of the top ten. "Teamwork and collaboration" ended up at #2, so there seems to be some disconnect here that needs to be investigated further.

We will see how much our Round 3 data moves versus the Round 2 results. If the industry results hold true, we will only see small moves at the next level.

5.6 Academic Round 3 analysis

Our academic Round 3 results are shown below.

Table 5.5 Academic Round 3 statistical results with ranking

Category	Area	Rank	Mean	Std Dev
Fundamentals of programming	Knowledge	1	4.76	0.10
Control structures and logic	Knowledge	2	4.67	0.14
Problem solving	Skill	3	4.62	0.15
Writing programs	Knowledge	3	4.62	0.13
Testing and debugging	Knowledge	5	4.57	0.15
Comprehending programs	Knowledge	6	4.52	0.13
Developing-and coding to-specifications	Knowledge	7	4.38	0.18
Persistence	Skill	7	4.38	0.16
Developing good program design	Knowledge	9	4.29	0.18
Communication	Skill	10	4.10	0.15
Data structures	Knowledge	10	4.10	0.18
Lifelong learning	Skill	12	4.05	0.26
Teamwork and collaboration	Skill	13	3.95	0.19
Generating clear documentation	Knowledge	14	3.90	0.15
Abstraction	Knowledge	15	3.86	0.23
Receives feedback well	Skill	15	3.86	0.23
Curious	Skill	17	3.81	0.25
File handling and I/O	Knowledge	18	3.71	0.24
Creative and innovative	Skill	19	3.57	0.16
Object oriented programming	Knowledge	19	3.57	0.22
Evaluating time/space complexity	Knowledge	21	3.38	0.18
Internships	Skill	21	3.38	0.23
Multiple languages	Knowledge	21	3.38	0.22
Single language	Knowledge	21	3.38	0.24
Designing algorithms	Knowledge	25	3.33	0.20
How computers work	Knowledge	25	3.33	0.21
Refactoring code	Knowledge	25	3.33	0.25
Networking and communication	Knowledge	28	3.29	0.18
Version control	Knowledge	29	3.14	0.20
Code reviews	Knowledge	30	3.05	0.16
Operating systems	Knowledge	31	2.95	0.18
Pseudocode	Knowledge	31	2.95	0.26
Tools	Knowledge	31	2.95	0.18
IDEs	Knowledge	34	2.90	0.23
Web development	Knowledge	35	2.57	0.16

If we look at the delta from the Round 2 to the Round 3, we see very little changes in the top ten items.

Table 5.6 Academic Round 2 to Round 3 top 10 deltas

Category	Round3 Rank	Round2 Rank	Delta Rank	Mean Delta
Fundamentals of programming	1	1	0	0.12
Control structures and logic	2	5	3	0.17
Problem solving	3	2	-1	0.01
Writing programs	3	3	0	0.10
Testing and debugging	5	3	-2	0.05
Comprehending programs	6	6	0	0.20
Developing-and coding to-specifications	7	6	-1	0.06
Persistence	7	9	2	0.12
Developing good program design	9	8	-1	-0.01
Communication	10	10	0	-0.03
Data structures	10	10	0	-0.03

If we look at the largest deltas, we do see some larger changes (for example “single language” rose 9 ranks from phase 2 to phase three), but all other moves were small.

Table 5.7 Academic Round 2 to Round 3 deltas by largest moves

Category	Round2 Rank	Round3 Rank	Delta Rank	Mean Delta
Single language	30	21	9	0.11
Evaluating time/space complexity	25	21	4	-0.05
Control structures and logic	5	2	3	0.17
Internships	24	21	3	-0.07
Pseudocode	34	31	3	0.04
File handling and I/O	20	18	2	0.06
How computers work	27	25	2	-0.02
Persistence	9	7	2	0.12
Abstraction	13	15	-2	-0.14
Object oriented programming	17	19	-2	-0.30
Testing and debugging	3	5	-2	0.05
Code reviews	27	30	-3	-0.30
Refactoring code	22	25	-3	-0.15
Networking and communication	22	28	-6	-0.19

Note: Delta Rank = Round2 – Round3. Positive means increasing rank in the Round 3 survey.

Like our industry Round 3 results, most of the moves were minor and many items did not change at all. In the mean difference column, we can see that the importance average delta was very small. “Single language” bumped from #30 to #21. “Evaluating time/space complexity” increased in rank even though the average importance dropped by a few tenths of a point. “Networking and communication” had the largest drop in the Round 3 results moving from #22 to #28. Like our industry results, we did not have identical participation in the Round 3 survey, so having a high or low value respondent in Round 2 not fill out Round 3 can cause some of these small mean moves.

If we focus on our Round 2 top ten, we only have three moves to note in this list: “testing and debugging” dropped from #3 to #5, “control structure” increased from #5 to #2, and “persistence” stepped from #9 to #7.

5.6.1 Discussion

Having the Round #3 results show mostly minor changes, like our industry results, shows confidence that our final list does reach some level of consensus among our academic experts. In our top twelve items, we now only have three professional skills, but both #13 and #14 are professional skills that just missed the cut. It still holds true that our industry group appears to rate professional skills as more important than academic experts.

5.7 Industry/academic gap analysis

While our method called out running t-test and ANOVA statistics to compare the results, these small data sets of ordinal data did not produce any interesting results. The t-test results basically highlighted every mean difference in our comparison. Problem solving, for example, has mean of 4.48 for the industry group and 4.61 for the academic group. This gave us a t-test significance of 0.62 and a Cohen's d of -0.200 which rejects the hypothesis that the variances are the same. The ANOVA test also generated high significance. As this is not helpful to evaluate our gap data, we simplified our analysis to look at the top ten ranked items and the delta rank between groups.

5.7.1 Where industry and academic experts agree

If we review the top 10 list from each expert groups, we can find three areas where we have strong agreement.

Ind P3 Rank	Category	Aca Match	Aca P3 Rank	Category	Ind Match
1	Problem solving	3	1	Fundamentals of programming	2
2	Accountable	na	2	Control structures and logic	na
2	Fundamentals of programming	1	3	Problem solving	1
4	Communication	10	3	Writing programs	23
5	Testing and debugging	5	5	Testing and debugging	5
6	Attention to detail	na	6	Comprehending programs	na
7	Teamwork and collaboration	13	7	Developing-and coding to-specifications	na
8	Ethical	na	7	Persistence	na
8	Lifelong learning	12	9	Developing good program design	11
8	Passionate	na	10	Communication	4
			10	Data structures	19

Figure 5.10 Industry and academic Round 3 top-ten comparison

We have several items which seem to be important for both groups.

- Problem solving (Industry #1, Academic #3)
- Fundamentals of programming (Industry #2, Academic #1)
- Testing and debugging (Industry #5, Academic #5)

These items provide some common ground for these two groups to consider whether or not we should invest more work in reaching higher levels of agreement. Without much deep thought, the three that have the most agreement do not seem very surprising. “Problem solving” is one of the fundamental skills for all engineers. “Fundamentals of programing”, which we would need some future work to define this crisply, seems to talk to the minimum required knowledge to understand how to program. “Testing and debugging” tied at #5 in both groups, which makes sense as this is the only way to know you have a program that is doing what you planned. In many ways, every programming assignment in a CS1 course should have some level

of all three of these required to be successful. This will form the core of our first recommendation.

5.7.2 Second tier results, key professional skills

We also have a few items that are important to one group and not as important to the other group.

- Communication (Industry #4, Academic #10)
- Teamwork and collaboration (Industry #7, Academic #13)
- Lifelong learning (Industry #8, Academic #13)
- Developing good program design (Industry #11, Academic #9)
- Data structures (Industry #19, Academic #10)
- Writing programs (Industry #23, Academic #3)

From our review of literature, we expected "teamwork and collaboration" as well as "communication" to be in this list. As we called out in proclaiming our hypothesis #1 true, these were top-ten for our industry group but just out of our top-ten for our academic group. Our abet professional skill of "lifelong learning" also fared better amount our industry experts than our academic experts. While the mean from both groups would place these areas as "important", we must understand, and consider improving, why we have this disconnect between our groups. Our hypothesis is further solidified by noting that three of the Academic areas which were not in the academic list were all more detailed knowledge areas.

On the flip side, we had three items which were in the top-ten on our academic list that were out of the top-ten for our industry group. "Developing a good program design" was close enough to be considered an agreement. "Data structures" were clearly not as important to our industry group, and "writing programs" was much lower. We would consider that "writing

programs" and "testing and debugging" are almost foils of each other. Without the 1st, there is not much you can do on the 2nd. This is a disconnect on the industry side that needs to be better understood. If you had a candidate who showed up for an industry and could show he excelled at "problem solving", "fundamentals of programming", and "testing and debugging", but had no experience "writing programs", we do not think they would be offered a job. It might be assumed that our industry experts folded the "writing programs' into "fundamentals if programming" at some level, but without future work to clarify this we cannot base any of our recommendations on that result.

We have several areas which were completely absent from the other expert groups list. There was also one academic professional skill, "persistence", which also did not appear in the industry results. Was this an oversite by the industry group? Similarly, the industry items of "accountable", "attention to detail", "ethical", and "passionate" were not mentioned in the academic group. Future work could be designed to target some of these important differences.

If we focus on professional skills, we do see that academics do rate these areas lower than their industry counterparts. If we adjust for that, we see five skills considered important by both groups. If we add the four skills which were in the industry's top ten which were not listed by our academic experts, we have nine items which could be considered important.

Table 5.8 Highest rated professional skills

Item	Industry Rank	Academic Rank
Problem solving	1	3
Accountable	2	none
Communication	4	10
Attention to detail	6	none
Teamwork and collaboration	7	13
Ethical	8	none
Lifelong learning	8	12
Passionate	8	none
Persistence	none	7

This list will form the foundation for our second recommendation.

5.7.3 Comparison of low ranked items between expert groups

With how our survey was structured, our phase one questions focused on items that were important. This means that the final ranked-ordered list showed those areas on the bottom which were simply not as important as the items on the top. As we mentioned in our review of the importance mean, none of these items were “not important”. This means the value of looking at the bottom end of our ranked list is not as valuable as the top ranked items. However, there are some learnings that are suggested by these results. Here is the list of the bottom eleven items.

Ind P3 Rank	Category	Aca Rank		Aca P3 Rank	Category	Ind Rank
27	Tools	31		25	Designing algorithms	25
28	Multithreaded programming	na		25	How computers work	29
29	How computers work	25		25	Refactoring code	na
29	Operating systems	31		28	Networking and communication	na
31	Object oriented programming	19		29	Version control	na
32	Scripting language	na		30	Code reviews	16
33	Computer hardware	na		31	Operating systems	29
34	Multiple languages	21		31	Pseudocode	na
35	Program management	na		31	Tools	27
35	Single language	21		34	IDEs	na
37	Specific language	na		35	Web development	na

Figure 5.11 Industry and academic Round 3 bottom-eleven comparison

Only three of these matched, and the differences in rank were minor: “tools”, “how computers work”, and “operating systems”. Academics had two items that were ranked slightly higher than on the industry list: “designing algorithms”, and “code reviews”. Industry had three items that were ranked higher by the academics: “object-oriented programming”, “multiple languages”, and “single language”. But even the “code reviews” ranking by the academics only reached rank #16.

The biggest take away was how many items were in each list that were not even in the survey for the other group. There were five items that were on the industry list that did not even make the cut for the academic list, and six items on the academic side that did not make the industry list. Two possible areas of future work could be undertaken to build stronger consensus. First, a Delphi survey which included equal numbers of industry and academic experts would be valuable to see how these different experts would rank items together. Second, a survey could be constructed with a research question asking, “what items are taught in engineering programming

courses which are least useful for new hires to master". The focus of this question would produce a better list of items that might be seen in the classroom which were considered not-important, especially to our industry experts.

While this result is not as strong as our other results, our third recommendation is drawn from this information.

CHAPTER VI

CONCLUSION

We began this work outlining the challenges seen from introductory programming (CS1) as one of the first courses in the engineering degree program. While research on CS1 difficulties as well as pedagogical improvements show some promise, we proposed that aligning CS1 teaching outcomes with increased academic focus on professional skills may help improve understanding of engineering as a career. This has the promise of improving morale and performance in courses like CS1. However, there is no current research which ranks traditional knowledge areas along with professional skill areas. This dissertation has been a first step to begin this investigation and analysis.

To investigate this link between knowledge and professional skill areas, we sought to gather understand of experts from industry and academia. To assess these groups, we conducted two three round Delphi survey to build consensus ranked lists. We then were able to compare the results from each expert group.

Our first hypothesis predicted a gap between these two groups when discussing programming skills for new hire engineers.

- H1: Academic experts and industry experts will have one or more gaps regarding critical knowledge and professional skill areas required for programming in an industry engineering position.

To investigate this hypothesis, we had two primary research questions that formed the basis of our Round 1 Delphi survey:

- RQ1: According to industry experts, what are the most important knowledge and professional skills to consider for an industry programmer?
- RQ2: According to academic experts, what are the most important knowledge and professional skills to consider for an industry programmer?

Based on the results presented in previous chapters, our hypothesis was shown to be true. While there are areas of agreement between where our industry and academic expert groups, our academic experts mentioned more knowledge areas and less professional skills in their survey. When our academic and industry group had a matching professional skill, the academics generally rated that skill lower in their listings. Specific examples will be provided in our key findings section below.

With our two consensus-based lists of important knowledge and professional skill areas, we analyzed these lists side by side to address our third research question:

- RQ3: What is the gap between industry and academic experts in their answers to these questions?

This question combines initial research on knowledge from syllabi work of Becker and Fitzpatrick [14] and extends through the industry professional skills research of Groeneveld [20]. With our gap analysis, we have a foundation to addressing our final research question:

- RQ4: Is there knowledge or a set of skills which should be emphasized or deemphasized in a CS1 curriculum which could give students a better ability to know whether engineering is a degree they want to pursue?

While our research does not fully answer this final question, our recommendations section below will discuss several importing items supported by our analysis.

6.1 Review of our industry and academic individual results

From our survey results, we discovered how our industry experts rated many professional skills as important, which academic experts rated more knowledge areas at the top of their ranked list.

6.1.1 Industry results

Our industry results highly rated professional skills as shown by the listing in Table 6.1. Eight of the "top ten" items were professional skills. While several of these were predicted by prior literature, "accountable", "attention to detail", and "passionate" were new categories called out by our experts. Overall, 41% of the 37 areas ranked by the industry experts were professional skills.

Table 6.1 Top industry results (from Table 4.7)

Category	Area	Rank	Mean
Problem solving	Skill	1	4.58
Accountable	Skill	2	4.46
Fundamentals of programming	Knowledge	2	4.46
Communication	Skill	4	4.38
Testing and debugging	Knowledge	4	4.38
Attention to detail	Skill	6	4.29
Teamwork and collaboration	Skill	7	4.21
Ethical	Skill	8	4.17
Lifelong learning	Skill	8	4.17
Passionate	Skill	8	4.17

6.1.2 Academic results

Our academic experts placed higher importance on knowledge areas as shown by the listing in Table 6.2. Only three professional skills made the top 11, "problem solving",

"persistence", and "communication". The academic experts highest rated professional skill, "persistence", was unique to these experts, not showing up in either literature or our industry list. These top knowledge items strongly confirm the findings of Becker and Fitzpratrick [14]. The top four Becker results ("writing programs", "testing and debugging", "control structures and logic", and "problem solving") made the top five on our academic expert's list. Only 26% of the 35 academic areas were professional skills.

Table 6.2 Top academic results (from Table 5.5)

Category	Area	Rank	Mean
Fundamentals of programming	Knowledge	1	4.76
Control structures and logic	Knowledge	2	4.67
Problem solving	Skill	3	4.62
Writing programs	Knowledge	3	4.62
Testing and debugging	Knowledge	5	4.57
Comprehending programs	Knowledge	6	4.52
Developing-and coding to-specifications	Knowledge	7	4.38
Persistence	Skill	7	4.38
Developing good program design	Knowledge	9	4.29
Communication	Skill	10	4.10
Data structures	Knowledge	10	4.10

6.2 Key findings between our industry and academic experts

From the results described in sections 6.1.1. and 6.1.2, we notice many differences rankings. While professional skills made up 41% of our industry expert's list, our academic experts only identified professional skills in 26% of their items. To make recommendations, our comparative analysis of all industry and academic panel results identified several areas overlap between the two groups. These are critical as these overlaps will provide the basis of our following recommendations.

In three areas, the final rank in both groups was near the top and closely matched:

- Problem solving (Industry #1, Academic #3)
- Fundamentals of programming (Industry #2, Academic #1)
- Testing and debugging (Industry #4, Academic #5)

These items are common ground. As almost all programming, at its simplest, is writing code to solve a particular problem, we were not surprised that "problem solving" was near the top of both lists. To be an effective programmer, you must have mastery of the "fundamentals of programming". "Testing and debugging", which also appeared often in our literature review, plays a significant role in programming. Any good coder needs to be able to test and debug code. While "problem solving" is a professional skill, the other items are knowledge-based areas. This common ground between our two expert groups will be foundational to our recommendation section which follows.

We had five other areas which had some common ground.

- Communication (Industry #4, Academic #10)
- Teamwork and collaboration (Industry #7, Academic #13)
- Lifelong learning (Industry #8, Academic #13)
- Developing good program design (Industry #11, Academic #9)
- Data structures (Industry #19, Academic #10)

While the agreement is not as strong in these areas, there is enough support that these should be included in our recommendations.

Finally, in addition to the eight areas where we have some common ground on importance, we also identified eight knowledge areas that rated near the bottom of both of our expert's lists.

- Tools (Industry #27, Academic #31)
- How computers work (Industry #29, Academic #NA)
- Operating systems (Industry #29, Academic #31)
- Designing algorithms (Industry #25, Academic #25)
- Object oriented programming (Industry #31, Academic #19)
- Multiple languages (Industry #34, Academic #21)
- Single language (Industry #35, Academic #21)
- Code reviews (Industry #16, Academic #30)

These items are all knowledge areas. There are some differences in ranking between the two groups, but neither group ranked any of these higher than #16.

With this collection of areas which have both high and low rank, we can present our recommendations.

6.3 Recommendations for engineering-based computer programming courses

As we began in our introduction, we believe increasing the focus on professional skills in programming courses will have positive impacts on motivation and retention. From our results, we present three recommendations that work into this overall goal. First, we highlight what is working and should remain foundational moving forward. Second, we detail the highest rated professional skills which should be considered for integration into existing curriculums. Third, we suggest lower rated knowledge areas which could be deemphasized to make room for new content.

6.3.1 Recommendation #1: Continue to emphasize the importance of problem solving, fundamentals of programming, and testing and debugging in all engineering programming courses.

These three areas, which include one professional skill and two knowledge areas, showed strong support with both of our expert groups. As these were also in the top five items of Becker's research, this is not a new recommendation. It is encouraging to note that our results clearly reconfirm that these items, which are already a focus of many of the syllabi that Becker reviewed, exist today. Indeed, if there was a way to deepen or strengthen the focus of these three areas it should be considered. Industry managers could include specific questions seeking a candidate's mastery of these three items. Academics could use these three items to focus and reenforce the teaching objectives through the course material.

6.3.2 Recommendation #2: Find new ways to instruct, highlight, and assess important professional skills.

From our results, our industry experts highly rated professional skills. This seems to align with the increased focus in this area from accreditation boards. These professional skills, as a critical part of industry jobs, represent a significant side of what is needed to be successful in engineering jobs. While the importance of knowledge cannot be discounted, as emphasized by Recommendation #1, student understanding and practice of professional skills would give them a more complete view of what engineering feels like in practice.

To find which professional skills are the most like candidates for consideration, we focus on the professional skills from the top ten of both industry and academic lists. "Problem solving" would be #1 on this list, but it is covered in Recommendation #1. The additional eight areas are shown in the next table.

Table 6.3 Professional skills to emphasize in degree, programming, and individual classes (from Table 5.8)

Item	Industry Rank	Academic Rank
Accountable	2	none
Communication	4	10
Attention to detail	6	none
Teamwork and collaboration	7	13
Ethical	8	none
Lifelong learning	8	12
Passionate	8	none
Persistence	none	7

We will address these in three different groups.

First, we have a group of three professional skills that have support from literature as well as general support among both of our expert groups: “communication”, “teamwork and collaboration”, and “lifelong learning”. While academic experts rated these lower than several other knowledge areas, there is enough overlap to consider codifying these as key elements of engineering and programming courses.

Second, the area “accountable”, “ethical”, “passionate”, and “persistent” present an extremely interesting group of professional skills. While “ethical” is found in current research such as the ABET criterion guidelines, the others were unique to this study. Industry came up with “accountable” and “passionate” and ranked them highly during their Delphi survey.

Third, academics added an area outside of our professional skills research and ranked it at #7 in their final list. “Persistent” does sound like a skill that would be useful for students as well as career engineers. As this represents new items which have not been well researched, we believe this presents us with an additional professional skill area which merits further research.

Distilling these eight items from our Delphi ranked lists of industry and academic expert groups, along with our gap analysis, present the most significant finding from this research. We hope this may provide some needed focus to drive future work investigating how these could be included in future teaching objectives and analyzed to discover if these professional skills could have a notable impact on student performance and retention.

6.3.3 Recommendation #3: Deemphasize less important knowledge areas to make room for additional focus on professional skills.

Any addition to current curriculum would require having some candidate items which could be deemphasized or dropped from the existing course load. While our open-ended Delphi questions clearly focused on needed knowledge and professional skills, the areas which bubbled to the bottom of the list would be considered candidates for deemphasizing or removing to make space for our Recommendation #2. The table below is assembled from the lowest ranked items in both groups of our Delphi surveys.

Table 6.4 Knowledge areas to deemphasize in degree, programming, and individual classes

Item	Industry Rank	Academic Rank
Language (specific, single, multiple, scripting)	37,35,34,32	none
Operating systems	29	31
How computers work	29	25
Tools, IDEs	27	31,34
Web development	none	35
Pseudocode	none	31
Designing algorithms	25	25

In many ways, all of these could be seen as supporting knowledge to programming. Learning a “language” is required to do programming, but I believe this data suggests that teaching objectives should not focus on the details of language (syntax, details, etc.). In the same

way, having some understanding of “operating systems”, “how computers work”, “Tools, IDEs”, “web development”, “pseudocode”, and “designing algorithms” may be needed to complete a simple program, but it should not be the focus. A discussion of how this item could be better studied is presented in future research.

These three recommendations, supported by our results, present some clear direction for future research.

6.4 Contributions to the field of Computing Education

As we look at some of the results as well as some of the research and development of methods, we have contributions that impact educators as well as future researchers in the field.

6.4.1 What does an educator know now?

Teaching is a challenging vocation. Not only must instructors keep up with start of the art in their field, but they must also be checking and upgrading their teaching methods. As we are focusing on engineering courses that teach programming, including CS1 courses, we have two key findings and one critical question that will continue to be relevant as they review and evaluate their material from year to year.

First, our Recommendation #1 clearly shows that the foundation of engineering courses is solid. Instructors can be confident that both industry and academic experts agree the core of programming courses should continue to be problem solving, fundamentals of programming, and testing and debugging the heart of a programming course. As they consider making future changes and improvements, these areas should be maintained and strengthened.

Second, or Recommendation #2 gives eight professional skills which are strong candidates to consider integrating into their courses. While these should all be included through

the course of an engineering education, selecting key areas which align with their current material would likely be a benefit to their students. We believe these professional skills, particularly in beginning engineering courses like CS1, will help students gain a better understand of what engineering careers require.

Finally, we believe the simple question "what mix of knowledge and professional skills will be most valuable to students" will allow an active focus at a broad program level, a course syllabus level, and even at a week-by-week teaching outcome plan level. Our Recommendation #3 has some thoughts about what lower-level knowledge items might be deemphasized to make sure the focus remains on the most important areas, but every educator would need to find the right balance in their courses and with their individual classes.

These three items, supported by our research, should help provide some clarity for educators as they look across their courses and their programs.

6.4.2 What does a researcher know now?

As research into the efficacy and application of professional skills continues, we show XXX areas from our research which should make future studies easier.

First, we have shown with our results that starting with a broader, holistic approach can provide better ability to compare and analyze results. This applies to expert team selection as well as looking at knowledge and professional skill areas together. In our expert selection, we have shown that including both industry and academic experts allows a broader range of opinions and experiences to be sampled. As both viewpoints are necessary for educational outcomes, research would be wise to include both areas in future work.

In a similar way, combining knowledge and professional skill areas into one ranked list allowed us to calculate relative ranks which could not be seen when studying each area

separately. As we believe future studies are needed to better define and rank professional skills, continuing to refine the related knowledge areas is best done concurrently.

Secondly, we have shown two methods which should be extensible to other areas of research: classification framework, and gap analysis. Our classification framework not only supported our work, but the framework itself provides a template for future work in many fields. Building a list of terms from research, following a classification flow, and including a process to add terms when the original list is not sufficient, show a general method that others could leverage for different research areas.

When comparing our two survey results, our gap analysis provides a simple yet instructive comparison between the two different expert groups. While an argument could be made for research which includes both industry and academic experts in the same group, when this is not reasonable analysis that follows our model can be useful.

Third, our results produced individual ranked list for each of our groups. These results, as seen in our final recommendations, can be used to verify and extend our findings. Having a starting list, like the initial lists we used from sources like Becker and the ABET recommendations, is useful to launch into similar or extended future research. Our three final recommendations can directly be reenforced or challenged in future research. At a high level, the idea that professional skills should be integrated into engineering and programming courses is clear from our results. We also have called out specific professional skills which might be candidates for individual attention to discover of they would show statistically significant improvement in student performance and retention. In our future research section, we will suggest some specific areas we can see that would be useful extensions of our results.

It is our hope that these three contributions will encourage expansion of our methods and additional work to help further codify how inclusion of professional skills in engineering programming education could improve both student performance and retention.

6.5 Limitations

While we showed several clear results from our work, we note four limitations which could impact the reach of this work.

6.5.1 Defining and building a hierarchy of terms

With our category mapping framework and ranking methodology, we left the definition of our categories open to interpretation to our experts. A good example of where this presents a problem is around the area “writing programs”. As mentioned earlier, this was ranked #3 by our academics while ranked #23 by our industry experts. We suggested that the industry experts may have assumed this would be above a minimum bar or may have lumped the minimum requirements into “fundamentals of programming”. Without an agreed upon definition, we cannot understand what this really implies. Several of our categories suffer from this definition problem to some degree. Future research might take a few key items and work with experts to build clear definitions.

In the same way, our areas have no hierarchy built into our analysis. For example, can areas like “data structures” or “file handling” be grouped under “control structure and logic”? Could “helpful”, “empathetic”, and “humble” be grouped under “teamwork and collaboration”? Building our terms into a clear hierarchy list may help explain differences where broader terms in the hierarchy may be rated well above or below more detailed sub-terms.

Future research that sought to clarify term definitions might allow for results like ours to be developed with a little more hierarchy in the terminology as well as uniformity between different groups.

6.5.2 Diversity across our industry and academic groups.

While we believe we assembled a diverse industry and academic group, there were limitations which may impact the generally applicability of our results. In our industry group, all of our experts were from the same fortune 500 company. While we showed a diversity of experience, hiring involvement, as well as found representatives from four continents, there might be some general training/hiring/development within one company that might skew these results away from a general population. Our 33 academic experts were pulled from 27 different colleges of engineering, but all within the US. In addition, with both our expert groups our surveys were conducted in English.

While we believe our results are representative of a much broader population, future research with different groups would be needed to corroborate our results.

6.5.3 Attrition through the survey process

Because if attrition, we ended up with less than 30 completed surveys after Round three. This can limit the statistical findings. The following table shows these numbers.

Table 6.5 Expert survey completion rate

Round	Industry Counts	Industry Percent	Academic Counts	Academic Percentage
Round 1 Start	36		33	
Round 1 Completion	31	86%	29	88%
Round 2 Start	31		33	
Round 2 Completion	29	94%	24	73%
Round 3 Start	31		33	
Round 3 Completion	25	81%	21	64%

For our Round 2 and Round 3 completion numbers, we were short of our 30 completions. With over 29% attrition amount our industry group and 34% attrition amount our academic group, we should have begun with at least 38 industry experts and 47 academic experts. This can have statistical impact to findings; however, we believe the overall impact was minor as detailed in our results discussion sections.

6.5.4 Clear identification of knowledge areas which could be deemphasized

In our Recommendation #3, we generated a list of items which rated at the bottom of both our Delphi surveys. While we believe this is generally accurate, our initial list was searching for items which could be important to new hires in engineering jobs. This means that every item on our list was considered a possible positive area. The ranking clearly showed these bottom items were up to two importance points lower in the final ranking. However, if we had asked an open-ended question like “what are the most taught and least important knowledge areas in a beginning programming class”, or possibly “what are the most taught programming knowledge areas in a college engineering degree that are not useful in an industry job”, we may have gotten a more tightly targeted list. Future work could be done to strengthen the best knowledge areas to deemphasize according to our Recommendation #3.

6.6 Future research

With all research, we build on those that have gone before us and look forward to the work that will come after. Here are three areas of future research that we hope will flow out of our three key recommendations.

6.6.1 Professional skills in the classroom

While our results produced some clear recommendations, the obvious next step would be to design classroom experiments to evaluate integration of each of these professional skills and measure the impact. Pedagogical and methodological work needs to be done that integrates one or several of our targeted professional skills into a particular course. Then controlled work would need to be done to analyze if the integrating of that material has a notable impact on performance and retention. Some work has been done in this area (assessing problem solving [104]–[106], professional skills in engineering [107]–[110], etc.), but more is needed.

For our eight recommended professional skills, we need to consider how they could be best taught and assessed. From our initial review of literature, we outlined several pedagogical methods from literature. Do methods help teach particular professional skills? Would problem-based learning be a good way to introduce the skill problem solving? Does team-based learning do a good job developing teamwork? Most of these pedagogical models still focus on the goal of teaching knowledge areas. Simply having the professional skill name in the model does not mean it will work well. For example, team-based learning suffers greatly when there is not equal participation from all team members. This situation places individuals at odds instead of fostering teamwork. If teams work well together, it might be a way to help the students simulate, on a small scale, some of the realities engineers might face when working in industry.

We may consider different pedagogical techniques, or possibly just a difference in focus, around the teaching of professional skills. Could we use interactive class teamwork exercises to emphasize the importance of teamwork without making this the teaching focus of the class? Could the professor simply talk about professional skills as they apply to the knowledge items being learned? A focus on methods for teaching professional skills within the existing pedagogical methods or with completely new methods could both be pursued.

We look forward to seeing this type of work attempted in the future. In particular, we believe CS1 is a prime place where critical professional skills could serve to increase student motivation and allow them to get a better understanding of what an engineering career would entail.

6.6.2 Fleshing out "fundamentals of programming" and "testing and debugging"

While industry experts clearly rated professional skills as very important, "fundamentals of programming" was ranked #2 and "testing and debugging" was ranked #4 overall in the industry list. Among the academic experts, the corresponding ranks were #1 and #5. This shows a high level of agreement between both expert groups. However, further understanding of these particular knowledge areas is needed to uncover the underlaying items which make up these knowledge areas.

6.6.3 Bridging the gap between industry and academics

While our separation of our experts into two groups to build two consensus lists to compare was intentional, the results show several gaps and ranking differences that we have no data to explain. Future Delphi surveys could be completed which strove to build a single group combining industry and academic experts. While there may need to be a better definition of

terms or additional rounds to achieve consensus, this would serve to remove some of the ambiguity that exist in our data when one group had a ranked category that did not even show up on the other groups list.

We do not believe this would change our critical results or our recommendations, but it would help provide a more complete picture of the cross-group differences.

6.7 Closing remarks

In engineering education, how do we help prepare students for their future? At the lowest level, this happens uniquely for each individual student and involves friends, family, teachers, staff, experiences, and opportunities. Every path is different. It is also true that no one can jump forward in time to know for sure that the path they are taking will end up exactly where they wanted to go. We all simply make the best choices we can along the way, and work to adjust if we find ourselves somewhere we did not want to be.

In light of these factors, teaching students is a challenging task. There is never one simple way that works for all teachers or all students. Understanding some of the complexities and limitations of the academic environment is important. Most engineering students will end up in industry positions. This means teachers must also strive to understand the complexities and limitations of the industry positions for which they are preparing their students for.

Our research attempted to bring some understanding to the gap between industry and academic expectations in the specific area of engineering programming education. It is hoped this can help guide deeper understanding on both sides. Our three recommendations provide a base framework we hope is thought provoking to educators. It is hoped that encouraging deeper integration of critical professional skills in our engineering courses, programming courses, and even the

introductory CS1 course can be something that helps students develop a better understanding of engineering careers and how they might integrate into that world.

REFERENCES

[1] Y. Bosse and M. A. Gerosa, “Why is programming so difficult to learn?,” *ACM SIGSOFT Software Engineering Notes*, vol. 41, no. 6, pp. 1–6, Jan. 2017, doi: 10.1145/3011286.3011301.

[2] J. Bennedsen and M. E. Caspersen, “Failure rates in introductory programming,” *ACM SIGCSE Bulletin*, vol. 39, no. 2, p. 32, Jun. 2007, doi: 10.1145/1272848.1272879.

[3] A. Petersen, M. Craig, J. Campbell, and A. Tafliovich, “Revisiting why students drop CS1,” in *Proceedings of the 16th Koli Calling International Conference on Computing Education Research - Koli Calling ’16*, 2016, pp. 71–80. doi: 10.1145/2999541.2999552.

[4] A. Baist, A. P.-V. J. I. P. Teknik, and undefined 2017, “Analysis of Student Difficulties in Computer Programming,” *jurnal.untirta.ac.id*, Accessed: Jun. 12, 2020. [Online]. Available: http://www.jurnal.untirta.ac.id/index.php/VOLT/article/view/2211

[5] L. Porter and B. Simon, “Retaining nearly one-third more majors with a trio of instructional best practices in CS1,” *SIGCSE 2013 - Proceedings of the 44th ACM Technical Symposium on Computer Science Education*, pp. 165–170, 2013, doi: 10.1145/2445196.2445248.

[6] L. Porter, C. Bailey Lee, and B. Simon, “Halving fail rates using peer instruction,” p. 177, 2013, doi: 10.1145/2445196.2445250.

[7] B. B. Morrison and J. A. Preston, “Engagement,” *ACM SIGCSE Bulletin*, vol. 41, no. 1, pp. 342–346, Mar. 2009, doi: 10.1145/1539024.1508990.

[8] S. Russell, “Automated Code Tracing Exercises for CS1,” *Computing Education Practice 2022*, pp. 13–16, Jan. 2022, doi: 10.1145/3498343.3498347.

[9] P. M. Phothilimthana and S. Sridhara, “High-coverage hint generation for massive courses: Do automated hints help CS1 students?,” *Annual Conference on Innovation and Technology in Computer Science Education, ITiCSE*, vol. Part F128680, pp. 182–187, Jun. 2017, doi: 10.1145/3059009.3059058.

[10] K. M. Ala-Mutka, “A Survey of Automated Assessment Approaches for Programming Assignments,” *Computer Science Education*, vol. 15, no. 2, pp. 83–102, Jun. 2005, doi: 10.1080/08993400500150747.

[11] Simon *et al.*, "Automated assessment in CS1," *Conferences in Research and Practice in Information Technology Series*, vol. 52, pp. 189–196, 2006.

[12] A. Pears *et al.*, "A survey of literature on the teaching of introductory programming," *ITiCSE-WGR 2007 - Working Group Reports on ITiCSE on Innovation and Technology in Computer Science Education*, pp. 204–223, Dec. 2007, doi: 10.1145/1345443.1345441.

[13] B. A. Becker and K. Quille, "50 Years of CS1 at SIGCSE: A Re-view of the Evolution of Introductory Programming Education Research," *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, 2019, doi: 10.1145/3287324.

[14] B. A. Becker and T. Fitzpatrick, "What Do CS1 Syllabi Reveal About Our Expectations of Introductory Programming Students," *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, 2019, doi: 10.1145/3287324.

[15] A. Clear, J. Impagliazzo, A. S. Parrish, M. Zhang, J. Impagliazzo, and M. Zhang, "Computing Curricula 2020," *SIGCSE 2019 - Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, pp. 653–654, Feb. 2019, doi: 10.1145/3287324.3287517.

[16] A. Luxton-Reilly, "Learning to program is easy," *Annual Conference on Innovation and Technology in Computer Science Education, ITiCSE*, vol. 11-13-July-2016, pp. 284–289, Jul. 2016, doi: 10.1145/2899415.2899432.

[17] C. G. P. Berdanier, "A hard stop to the term 'soft skills,'" *Journal of Engineering Education*, vol. 111, no. 1. John Wiley and Sons Inc, pp. 14–18, Jan. 01, 2022. doi: 10.1002/jee.20442.

[18] M. Minnes, S. G. Serslev, and O. Padilla, "What Do CS Students Value in Industry Internships?," *ACM Transactions on Computing Education (TOCE)*, vol. 21, no. 1, Mar. 2021, doi: 10.1145/3427595.

[19] Charlotte Ruhl, "Bloom's Taxonomy of Learning Classification System," May 24, 2021. https://teachersupport.info/blooms-taxonomy/ (accessed Apr. 23, 2022).

[20] W. Groeneveld, J. Vennekens, K. Aerts, and K. Leuven, "Identifying Non-Technical Skill Gaps in Software Engineering Education: What Experts Expect But Students Don't Learn," *ACM Transactions on Computing Education (TOCE)*, vol. 22, no. 1, pp. 1–21, Oct. 2021, doi: 10.1145/3464431.

[21] A. Luxton-Reilly and Andrew, "Learning to Program is Easy," in *Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education - ITiCSE '16*, 2016, pp. 284–289. doi: 10.1145/2899415.2899432.

[22] A. Yadin and Aharon, "Reducing the dropout rate in an introductory programming course," *ACM Inroads*, vol. 2, no. 4, p. 71, Dec. 2011, doi: 10.1145/2038876.2038894.

[23] R. Medeiros, … G. R.-I. T. on, and undefined 2018, “A systematic literature review on teaching and learning introductory programming in higher education,” *Ieeexplore.Ieee.Org*, pp. 1–14, 2018.

[24] F. Y. Assiri, “Recommendations to improve programming skills of students of computer science,” in *Proceedings of 2016 SAI Computing Conference, SAI 2016*, Aug. 2016, pp. 886–889. doi: 10.1109/SAI.2016.7556084.

[25] J. Coffey, “Relationship between design and programming skills in an advanced computer programming class,” *Journal of Computing Sciences in Colleges*, vol. 30, no. 5, pp. 39–45, 2015.

[26] J. M. Wing, “Computational thinking,” *Commun ACM*, vol. 49, no. 3, pp. 33–35, 2006, doi: 10.1145/1118178.1118215.

[27] K. Brennan and M. Resnick, “New frameworks for studying and assessing the development of computational thinking.”

[28] D. Barr, J. Harrison, and L. Conery, “Computational Thinking: A Digital Age Skill for Everyone.,” *Learning & Leading with Technology*, vol. 38, no. 6, pp. 20–23, 2011, Accessed: Feb. 05, 2022. [Online]. Available: http://csta.acm.org.

[29] S.-C. Kong and H. Abelson, “Computational Thinking Education,” *Computational Thinking Education*, p. 382, 2019, doi: 10.1007/978-981-13-6528-7.

[30] P. J. Denning, “Remaining Trouble Spots with Computational Thinking,” *Commun ACM*, vol. 60, no. 6, pp. 33–39, Jun. 2017, doi: 10.1145/2998438.

[31] V. J. Shute, C. Sun, and J. Asbell-Clarke, “Demystifying computational thinking,” *Educ Res Rev*, vol. 22, pp. 142–158, Nov. 2017, doi: 10.1016/J.EDUREV.2017.09.003.

[32] M. Tedre and P. J. Denning, “The long quest for computational thinking,” *ACM International Conference Proceeding Series*, pp. 120–129, Nov. 2016, doi: 10.1145/2999541.2999542.

[33] C. Angeli and M. Giannakos, “Computational thinking education: Issues and challenges,” *Comput Human Behav*, vol. 105, p. 106185, Apr. 2020, doi: 10.1016/J.CHB.2019.106185.

[34] Y. Li, “Teaching programming based on Computational Thinking,” *Proceedings - Frontiers in Education Conference, FIE*, vol. 2016-November, Nov. 2016, doi: 10.1109/FIE.2016.7757408.

[35] S. Gross, M. Kim, J. Schlosser, C. Mohtadi, D. Lluch, and D. Schneider, “Fostering computational thinking in engineering education: Challenges, examples, and best practices,” *IEEE Global Engineering Education Conference, EDUCON*, pp. 450–459, 2014, doi: 10.1109/EDUCON.2014.6826132.

[36] N. A. of E. NAE, *Educating the Engineer of 2020*. Washington, D.C.: National Academies Press, 2005. doi: 10.17226/11338.

[37] L. Miller, L.-K. Soh, V. Chiriacescu, E. Ingraham, and D. F. Shell Melissa Patterson Hazley, "Integrating Computational and Creative Thinking to Improve Learning and Performance in CS1," *Proceedings of the 45th ACM technical symposium on Computer science education*, 2014, doi: 10.1145/2538862.

[38] X. Tang, Y. Yin, Q. Lin, R. Hadad, and X. Zhai, "Assessing computational thinking: A systematic review of empirical studies," *Comput Educ*, vol. 148, p. 103798, Apr. 2020, doi: 10.1016/J.COMPEDU.2019.103798.

[39] R. P. Y. Lai, "Beyond Programming: A Computer-Based Assessment of Computational Thinking Competency," *ACM Transactions on Computing Education*, vol. 22, no. 2, pp. 1–27, Jun. 2022, doi: 10.1145/3486598.

[40] Y. Qian and J. Lehman, "Students' misconceptions and other difficulties in introductory programming: A literature review," *ACM Transactions on Computing Education*, vol. 18, no. 1. Association for Computing Machinery, Oct. 01, 2017. doi: 10.1145/3077618.

[41] Y. Qian, S. Hambrusch, A. Yadav, S. Gretter, and Y. Li, "Teachers' Perceptions of Student Misconceptions in Introductory Programming," *Journal of Educational Computing Research*, vol. 58, no. 2, pp. 364–397, Apr. 2020, doi: 10.1177/0735633119845413.

[42] C. Schulte and J. Bennedsen, "What do teachers teach in introductory programming?," *ICER 2006 - Proceedings of the 2nd International Computing Education Research Workshop*, vol. 2006, pp. 17–28, 2006, doi: 10.1145/1151588.1151593.

[43] ABET, *Criteria for Accrediting Engineering Programs*, November 2. Baltimore, MD 21201: ABET, 2020. [Online]. Available: www.abet.org

[44] ABET, "Computing Accreditation Commission CRITERIA FOR ACCREDITING COMPUTING PROGRAMS," 2019.

[45] ENAEE, "EUR-ACE ® Framework Standards and Guidelines (EAFSG)."

[46] I. E. A. IEA, World Federation of Engineering Organizationa, and unesco, "International Engineering Alliance Graduate Attributes and Professional Competencies," 2021. [Online]. Available: http://www.ieagreements.org

[47] T. J. Mcgill and S. E. Volet, "Title: A conceptual framework for analyzing students' knowledge." [Online]. Available: http://eds.a.ebscohost.com/eds/delivery?sid=85d39948-40e7-4195...

[48] M Schoeman (University of South Africa), “Exploring infusing graduateness in an introductory programming module in an ODeL environment,” in *digiTAL 2021 Conference Proceedings*, 2021, pp. 178–188. doi: 10.13140/RG.2.2.32529.56167.

[49] M. (Stellenbosch U. Bester, “Academics’ conceptions and orientations of graduate attributes in applied design programmes at a university of technology,” Stellenbosch : Stellenbosch University, 2014. Accessed: Dec. 26, 2021. [Online]. Available: https://scholar.sun.ac.za:443/handle/10019.1/86447

[50] Y. Chetty, “Graduateness and employability within the higher education environment: A focused review of the literature,” in *Developing student graduateness and employability: Issues, provocations, theory and practical application*, 2012, pp. 5–24.

[51] H. Ngambi, “Cultivating the RARE Graduate,” in *Developing Student Graduateness and Employability: Issues, provocations, theory and practical guidelines*, 2012, pp. xix–xx.

[52] M. Exter, S. Caskurlu, and T. Fernandez, “Comparing Computing Professionals’ Perceptions of Importance of Skills and Knowledge on the Job and Coverage in Undergraduate Experiences,” *ACM Transactions on Computing Education (TOCE)*, vol. 18, no. 4, Nov. 2018, doi: 10.1145/3218430.

[53] P. L. Li, A. J. Ko, and J. Zhu, “What Makes A Great Software Engineer?,” in *Proceedings - International Conference on Software Engineering*, Aug. 2015, vol. 1, pp. 700–710. doi: 10.1109/ICSE.2015.335.

[54] O. Helmer, “Analysis of the Future: The Delphi Method,” *RAND Corporation, Santa Monica, California*, no. March, 1967.

[55] P. Galanis, L. S. Fish, and D. M. Busby, “The Delphi method,” in *Faculty Publications - BYU Scholars Archive*, vol. 35, no. 4, BETA Medical Publishers Ltd, 2005, pp. 238–253. doi: 10.4324/9781315728513-10.

[56] T. J. Gordon, “The Delphi Method,” *Futures research methodology*, vol. 2, no. 3, pp. 1–30, 1994.

[57] F. Hasson, S. Keeney, and H. McKenna, “Research guidelines for the Delphi survey technique,” *J Adv Nurs*, vol. 32, no. 4, pp. 1008–1015, Oct. 2000, doi: 10.1046/j.1365-2648.2000.t01-1-01567.x.

[58] M. Turoff and H. Linstone, *The Delphi method-techniques and applications*. 2002.

[59] M. Turoff and H. A. Linstone, “The Policy Delphi,” *The Delphi Method: Techniques and Applications*, vol. 2, no. 2, pp. 80–96, 2002.

[60] S. Keeney, F. Hasson, and H. McKenna, “Consulting the oracle: Ten lessons from using the Delphi technique in nursing research,” *J Adv Nurs*, vol. 53, no. 2, pp. 205–212, 2006, doi: 10.1111/j.1365-2648.2006.03716.x.

[61] N. J. Shariff, “Utilizing the Delphi Survey Approach: A Review,” *Journal of Nursing & Care*, vol. 04, no. 03, pp. 246–251, 2015, doi: 10.4172/2167-1168.1000246.

[62] R. E. Taylor and L. L. Judd, “Delphi method applied to tourism.,” *Delphi method applied to tourism.*, pp. 95–98, 1989.

[63] B. Garrod, A. F.-T. research methods: Integrating, and undefined 2005, “Revisiting Delphi: the Delphi technique in tourism research,” *books.google.com*.

[64] A. A. Olsen, M. D. Wolcott, S. T. Haines, K. K. Janke, and J. E. McLaughlin, “How to use the Delphi method to aid in decision making and build consensus in pharmacy education,” *Curr Pharm Teach Learn*, vol. 13, no. 10, pp. 1376–1385, Oct. 2021, doi: 10.1016/J.CPTL.2021.07.018.

[65] D. Menke, S. Stuck, and S. Ackerson, “Assessing Advisor Competencies: A Delphi Method Study,” *NACADA Journal*, vol. 38, no. 1, pp. 12–21, 2018, doi: 10.12930/nacada-16-040.

[66] M. J. Scott, “A Concept Inventory for Functional Reasoning in Engineering Design,” 2019.

[67] S. Blair and N. P. Uhl, “Using the Delphi Method to Improve the Curriculum,” *Canadian Journal of Higher Education*, vol. 23, no. 3, pp. 107–128, 1993, doi: 10.47678/cjhe.v23i3.183175.

[68] A. J. Magana, “Modeling and Simulation in Engineering Education: A Learning Progression,” *Journal of Professional Issues in Engineering Education and Practice*, vol. 143, no. 4, 2017, doi: 10.1061/(ASCE)EI.1943-5541.0000338.

[69] R. A. Streveler, B. M. Olds, R. L. Miller, and M. A. Nelson, “Using a delphi study to identity the most difficult concepts for students to master in thermal and transport science,” *ASEE Annual Conference Proceedings*, pp. 4447–4454, 2003, doi: 10.18260/1-2--12592.

[70] K. W. Lamm, N. L. Randall, and F. Diez-Gonzalez, “Critical food safety issues facing the food industry: A delphi analysis,” *J Food Prot*, vol. 84, no. 4, pp. 680–687, Apr. 2021, doi: 10.4315/JFP-20-372/449189/CRITICAL-FOOD-SAFETY-ISSUES-FACING-THE-FOOD.

[71] L. F. Graham and D. L. Milne, “Developing basic training programmes: A case study illustration using the delphi method in clinical psychology,” *Clin Psychol Psychother*, vol. 10, no. 1, pp. 55–63, Jan. 2003, doi: 10.1002/CPP.353.

[72] W. H. Middendorf, “Modified Delphi Method of Solving Business Problems.,” *IEEE Trans Eng Manag*, vol. EM-20, no. 4, pp. 130–133, 1973, doi: 10.1109/TEM.1973.6448448.

[73] V. Rajhans, S. Rege, U. Memon, and A. Shinde, “Adopting a modified Delphi technique for revisiting the curriculum: a useful approach during the COVID-19 pandemic,” *Qualitative Research Journal*, vol. 20, no. 4, pp. 373–382, Oct. 2020, doi: 10.1108/QRJ-05-2020-0043/FULL/HTML.

[74] A. Sourani and M. Sohail, “The Delphi Method: Review and Use in Construction Management Research,” *http://dx.doi.org/10.1080/15578771.2014.917132*, vol. 11, no. 1, pp. 54–76, Jan. 2015, doi: 10.1080/15578771.2014.917132.

[75] N. Mehta, P. Verma, N. Seth, and N. Shrivastava, “Identification of TQM criterions for engineering education using Delphi technique,” *International Journal of Intelligent Enterprise*, vol. 2, no. 4, pp. 325–352, 2014, doi: 10.1504/IJIE.2014.069075.

[76] J. Jones, G. Belcher, and K. Elliott, “Identifying Technical Competencies for Architecture and Construction Education using the Delphi Method,” *Career and Technical Education Research*, vol. 46, no. 1, pp. 3–15, Jul. 2021, doi: 10.5328/CTER46.1.3.

[77] D. Varona and L. F. Capretz, “Using the DELPHI Method for Model for Role Assignment in the Software Industry,” *IECON 2021 – 47th Annual Conference of the IEEE Industrial Electronics Society*, pp. 1–7, Oct. 2021, doi: 10.1109/IECON48115.2021.9589957.

[78] K. K. Lilja, K. Laakso, and J. Palomki, “Using the Delphi method,” *PICMET: Portland International Center for Management of Engineering and Technology, Proceedings*, no. 1999, 2011.

[79] D. Skulmoski, Gegory J. (Zayed University, F. T. (University of C. Hartman, and J. (University of C. Krahn, “The Delphi Method for Graduate Research,” *Journal of Information Technology Education*, vol. 6, pp. 93–105, 2007, doi: 10.1007/3-540-47847-7_10.

[80] F. Lestari, I. Kusumanto, S. Hasri, and Akmaluhadi, “Independent campus on industrial engineering undergraduate program in Indonesia: A delphi method,” *IEEE International Conference on Industrial Engineering and Engineering Management*, vol. 2020-Decem, pp. 1083–1087, Dec. 2020, doi: 10.1109/IEEM45057.2020.9309800.

[81] S. Iqbal and L. Pipon-Young, “The Delphi method: Susanne Iqbal and Laura Pipon-Young with a step-by-step guide,” *Archives of Hellenic Medicine*, vol. 35, no. 4, pp. 564–570, 2018, doi: 10.4324/9781315728513-10.

[82] B. K. Khalaf and Z. B. M. Zin, “Traditional and Inquiry-Based Learning Pedagogy: A Systematic Critical Review,” *International Journal of Instruction*, vol. 11, no. 4, pp. 545–564, 2018, Accessed: Mar. 12, 2022. [Online]. Available: https://eric.ed.gov/?id=EJ1191725

[83] H. G. Schmidt, S. M. M. Loyens, T. van Gog, and F. Paas, “Problem-based learning is compatible with human cognitive architecture: Commentary on Kirschner, Sweller, and Clark (2006),” *Educ Psychol*, vol. 42, no. 2, pp. 91–97, 2007, doi: 10.1080/00461520701263350.

[84] S. Olena, T. Iryna, S. Olena, and S. Serhii, “Implementing Competency-Based Education for the Engineering Specialties’ Students,” *Proceedings of the 25th IEEE International Conference on Problems of Automated Electric Drive. Theory and Practice, PAEP 2020*, Sep. 2020, doi: 10.1109/PAEP49887.2020.9240850.

[85] G. Durand, C. Goutte, N. Belacel, Y. Bouslimani, and S. Léger, “A diagnostic tool for competency-Based program engineering,” *ACM International Conference Proceeding Series*, pp. 315–319, Mar. 2018, doi: 10.1145/3170358.3170402.

[86] M. D. Mckinney and D. Mckinney, “Improving Pass Rates by Switching from a Passive to an Active Learning Textbook in CS0,” in *ASEE’S Virtual Confrence: At Home with Engineering Education*, Jun. 2020, p. 16.

[87] J. H. Berssanette and A. C. de Francisco, “Active learning in the context of the teaching/learning of computer programming: A systematic review,” *Journal of Information Technology Education: Research*, vol. 20, pp. 201–220, 2021, doi: 10.28945/4767.

[88] S. Freeman *et al.*, “Active learning increases student performance in science, engineering, and mathematics,” *Proc Natl Acad Sci U S A*, vol. 111, no. 23, pp. 8410–8415, Jun. 2014, doi: 10.1073/PNAS.1319030111/SUPPL_FILE/PNAS.1319030111.ST04.DOCX.

[89] G. Gonzalez, “A systematic approach to active and cooperative learning in CS1 and its effects on CS2,” *Proceedings of the Thirty-Seventh SIGCSE Technical Symposium on Computer Science Education*, pp. 133–137, 2007, doi: 10.1145/1121341.1121386.

[90] S. Hartikainen, H. Rintala, L. Pylväs, and P. Nokelainen, “The Concept of Active Learning and the Measurement of Learning Outcomes: A Review of Research in Engineering Higher Education,” *Education Sciences 2019, Vol. 9, Page 276*, vol. 9, no. 4, p. 276, Nov. 2019, doi: 10.3390/EDUCSCI9040276.

[91] J. Bergmann and A. Sams, *Flip Your Classroom: Reach Every Student in Every Class Every Day - Jonathan Bergmann, Aaron Sams - Google Books*. 2012. Accessed: Mar. 31, 2022. [Online]. Available: https://books.google.com/books?hl=en&lr=&id=-YOZCgAAQBAJ&oi=fnd&pg=PR7&dq=+flip+your+classroom&ots=AGghQJmokh&sig=JFvlCpecJH-WB6rJEK3eIhkq0ro#v=onepage&q=flip%20your%20classroom&f=false

[92] G. Akçayır and M. Akçayır, “The flipped classroom: A review of its advantages and challenges,” *Comput Educ*, vol. 126, pp. 334–345, Nov. 2018, doi: 10.1016/J.COMPEDU.2018.07.021.

[93] Z. Huang, A. Peng, T. Yang, S. Deng, and Y. He, “A Design-Based Learning Approach for Fostering Sustainability Competency in Engineering Education,” *Sustainability 2020, Vol. 12, Page 2958*, vol. 12, no. 7, p. 2958, Apr. 2020, doi: 10.3390/SU12072958.

[94] P. A. Kirschner, J. Sweller, and R. E. Clark, “Why minimal guidance during instruction does not work: An analysis of the failure of constructivist, discovery, problem-based, experiential, and inquiry-based teaching,” *Educ Psychol*, vol. 41, no. 2, pp. 75–86, Mar. 2006, doi: 10.1207/S15326985EP4102_1.

[95] O. Chernikova, N. Heitzmann, M. Stadler, D. Holzberger, T. Seidel, and F. Fischer, “Simulation-Based Learning in Higher Education: A Meta-Analysis:,” *https://doi.org/10.3102/0034654320933544*, vol. 90, no. 4, pp. 499–541, Jun. 2020, doi: 10.3102/0034654320933544.

[96] V. Najdanovic-Visak, “Team-based learning for first year engineering students,” *Education for Chemical Engineers*, vol. 18, pp. 26–34, Jan. 2017, doi: 10.1016/J.ECE.2016.09.001.

[97] M. Homero and G. Murzi, “Team-Based Learning Theory Applied to Engineering Education: A System-atic Review of Literature Team-Based Learning Theory Applied to Engineering Education: A Systematic Review of Literature,” *ASEE Annual Conference and Exposition, Conference Proceedings*, pp. 24.1175.2-24.1175.12, 2014.

[98] P. Lasserre, “Adaptation of Team-Based Learning on a First Term Programming Class,” *Proceedings of the 14th annual ACM SIGCSE conference on Innovation and technology in computer science education - ITiCSE ’09*, 2009, doi: 10.1145/1562877.

[99] G. Sidhu, S. Srinivasan, and N. Muhammad, “Challenge-Based and Competency-Based Assessments in an Undergraduate Programming Course,” *International Journal of Emerging Technologies in Learning*, vol. 16, no. 13, pp. 17–28, 2021, doi: 10.3991/ijet.v16i13.23147.

[100] R. Anson and J. A. Goodman, “A Peer Assessment System to Improve Student Team Experiences,” *http://dx.doi.org/10.1080/08832323.2012.754735*, vol. 89, no. 1, pp. 27–34, Jan. 2013, doi: 10.1080/08832323.2012.754735.

[101] G. Norman, “Likert scales, levels of measurement and the ‘laws’ of statistics,” *Advances in Health Sciences Education*, vol. 15, no. 5, pp. 625–632, 2010, doi: 10.1007/s10459-010-9222-y.

[102] G. M. Sullivan, A. R. Artino, and Jr, “Analyzing and Interpreting Data From Likert-Type Scales,” *J Grad Med Educ*, vol. 5, no. 4, p. 541, Dec. 2013, doi: 10.4300/JGME-5-4-18.

[103] A. Joshi, S. Kale, S. Chandel, and D. Pal, “Likert Scale: Explored and Explained,” *Br J Appl Sci Technol*, vol. 7, no. 4, pp. 396–403, 2015, doi: 10.9734/bjast/2015/14975.

[104] N. Mourtos, N. DeJong Okamoto, and J. Rhee, “Defining, teaching, and assessing problem solving skills,” *7th UICEE Annual Conference on Engineering Education*, vol. Mumbai, India, no. 9-13 February, pp. 1–5, 2004.

[105] A. K. Veerasamy, D. D’Souza, R. Lindén, and M. J. Laakso, “Relationship between perceived problem-solving skills and academic performance of novice learners in introductory programming courses,” *J Comput Assist Learn*, vol. 35, no. 2, pp. 246–255, Apr. 2019, doi: 10.1111/JCAL.12326.

[106] C. Lertyosbordin, S. Maneewan, and D. Srikaew, “Components and Indicators of Problem-solving Skills in Robot Programming Activities,” *International Journal of Advanced Computer Science and Applications*, vol. 12, no. 9, pp. 132–140, 2021, doi: 10.14569/IJACSA.2021.0120917.

[107] A. Mohan, D. Merle, C. Jackson, J. Lannin, and S. S. Nair, “Professional skills in the engineering curriculum,” *IEEE Transactions on Education*, vol. 53, no. 4, pp. 562–571, Nov. 2010, doi: 10.1109/TE.2009.2033041.

[108] T. J. Siller, A. Rosales, J. Haines, and A. Benally, “Development of Undergraduate Students’ Professional Skills,” *Journal of Professional Issues in Engineering Education and Practice*, vol. 135, no. 3, pp. 102–108, Jun. 2009, doi: 10.1061/(ASCE)1052-3928(2009)135:3(102).

[109] L. J. Shuman, M. Besterfield-Sacre, and J. McGourty, “The ABET ‘professional skills’ - Can they be taught? Can they be assessed?,” *Journal of Engineering Education*, vol. 94, no. 1, pp. 41–55, 2005, doi: 10.1002/J.2168-9830.2005.TB00828.X.

[110] Å. Cajander, M. Daniels, R. Mcdermott, and B. R. von Konsky, “Assessing Professional Skills in Engineering Education,” 2011.

[111] H. J. Passow, “Which ABET competencies do engineering graduates find most important in their work?,” *Journal of Engineering Education*, vol. 101, no. 1, pp. 95–118, 2012, doi: 10.1002/J.2168-9830.2012.TB00043.X.

[112] M. E. Armstrong *et al.*, “Knowledge, Skills, and Abilities for Specialized Curricula in Cyber Defense,” *ACM Transactions on Computing Education (TOCE)*, vol. 20, no. 4, Nov. 2020, doi: 10.1145/3421254.

[113] S. Chidthachack, M. A. Schulte, F. D. Ntow, J.-L. Lin, and T. J. Moore, “Engineering Students Learn ABET Professional Skills: A Comparative Study of Project-Based-Learning (PBL) versus Traditional Students,” Mar. 2021, doi: 10.18260/1-2-1153-36216.

APPENDIX A

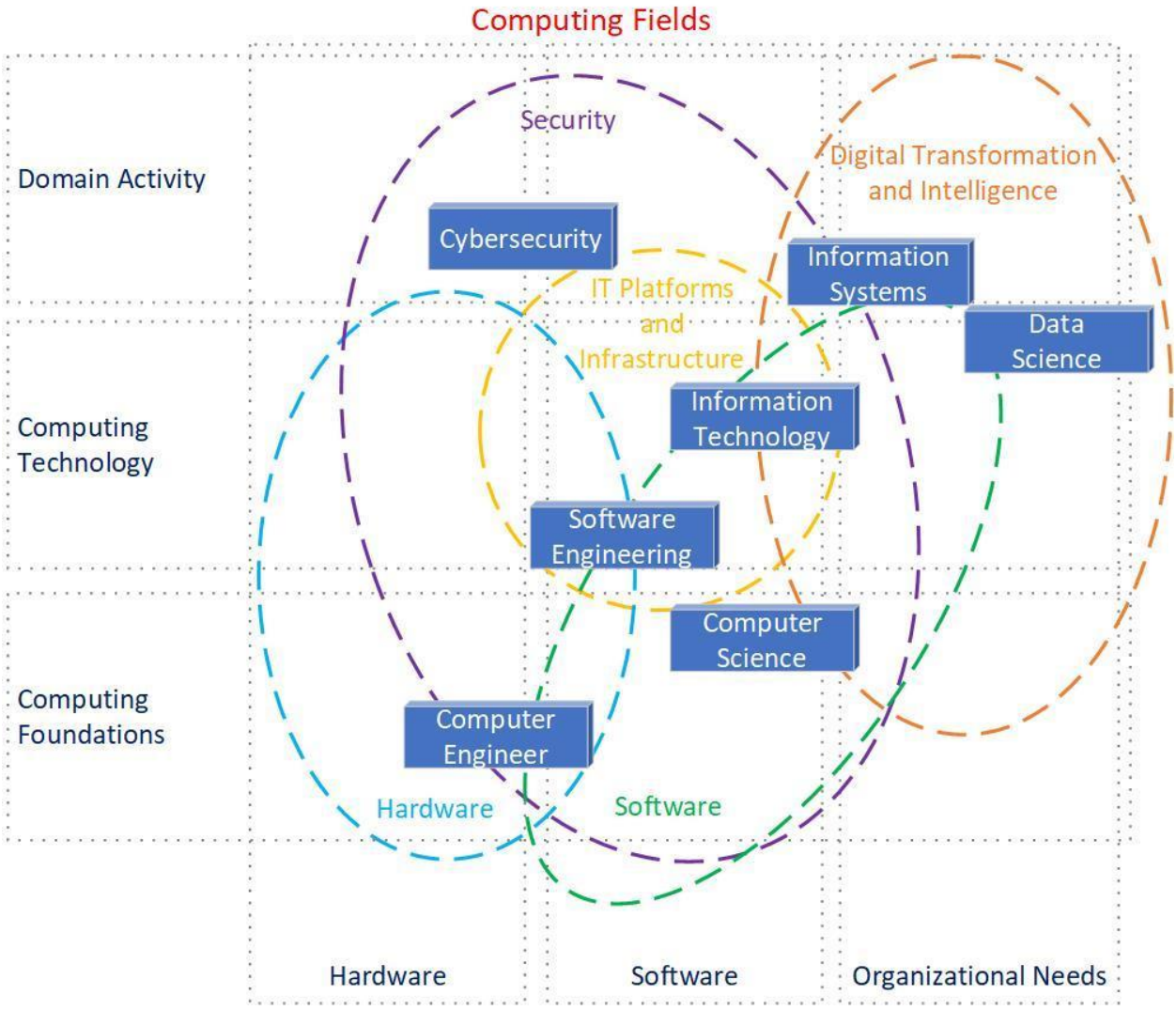
ROUND 1: DELPHI SURVEY OPENENDED AND CLASSIFICATION QUESTIONS

A.1 Delphi Questions

- Q1: What knowledge, skills, or characteristics should new hires in programming positions possess? (please be detailed)
    - (Long answer)
- Q2: What experiences are helpful to develop into a good programmer? (please be detailed)
    - (Long answer)

A.2 Classification Questions – Industry Experts

Computer technologies can be divided in many ways. Here is one diagram and description:



X axis represents increasing system integration:
Hardware enables Software enables Organizational Needs
Y axis represents increasing code integration:
Computing Foundations enable Computing Technology enables Domain Activity
Colored dotted lines represent functional job areas.
Blue boxes represent disciplines.

- Q3: Where do you spend most of your coding time (check all that apply)?

    - (checkboxes)
- Q4: How involved are you with hiring?
    - (pulldown)
- Q5: How many interviews have you been involved in in the last year?
    - (pulldown – 0 through 10+)
- Q6: How involved are you with training/mentoring new hires?
    - (pulldown)
- Q7: What languages do you spend the most time in (select all that apply)?
    - (checkboxs – many)
- Q8: If you spend most of your time in another language(s), list here?
    - (short answer)
- Q9: How much of your current job involves coding?
    - (pulldown – 0 to 100%)
- Q10: How would your rank your skill level?
    - (pulldown)
- Q11: In 1-2 sentences, explain why you chose this ranking.
    - (short answer)
- Q12: How many years have you been this skill level?
    - (pulldown – 0 to >10)

A.3 Classification Questions – Academic Experts

- Q3: (IDENTICAL) Where do you spend/target most of your coding/teaching time (check all that apply)?
    - (checkboxes)
- Q4: What level of engineering/computer science programming courses do you teach (check all that apply)?

    - (Checkboxes – Beginning programming “CS1”, advanced programming, embedded programming, architecture, advanced topics, other)

- Q4a: Briefly describe the “other” type of class you teach?
- Q5: How long have you been teaching?
    - (pulldown – 0 to >15)
- Q6: Do you conduct research in programming or programming education?
    - Yes/no
- Q6a: Briefly describe your research area.
    - (short answer)
- Q7: What languages do you spend the most time in (select all that apply)?
    - (checkboxs – many)
- Q8: If you spend most of your time in another language(s), list here?
    - (short answer)
- Q9: Do you have industry experience?
    - Yes/No
- Q9a: How many years?
- (pulldown – 0 to >10)

APPENDIX B

KNOWLEDGE AND SKILL AREAS – CLASSIFICATION FRAMEWORK LISTS

B.1 Partial Framework - Knowledge

The following table takes several of the lists from literature and strives to group rank them against each other. As the Becker list is the longest, with 54 items in ranked order, the other categories are aligned with these. Note that the ABET list has a few items that do not fall in the Becker list. The final category is an attempt to classify each item against the Mcgill and Volet framework. While this works for most items, we have a few areas that fall out of the straight programing categories. It would be a stretch to place these even in the "Strategic-Conditional" category. These are the additional categories added to this list: tools, debugging, professional skill, background, and advanced topic.

Table B.1 Comparison of knowledge areas from several references [14], [40], [42], [44], [47].

Becker 2019	# of Results	Schulte 2006	Qian 2017	ABET CS	Knowledge Area
Writing programs	112			Substantial coverage of at least one general-purpose programming language	Str-Cond
Testing & Debugging code	110	Debugging			Tools, Debugging
Control Structures & logic (if/else etc)	107	Sel&Iter			Syn-Proc
Problem Solving (also things like computational thinking)	106	ProbSolStrat			Professional skill
Arrays, Lists, dictionaries, vectors, sets	93	AdvDataStr			Syn-Proc
Variables, assignment, arithmetic expressions, declarations, data types	91	VarTypes	Variables		Syn-Proc
(Object oriented programming) Basic OOP	89		OOP		Str-Cond

Repetition & loops (for/while etc)	81		Loops		Con-Proc
Functions, methods, and procedures	78		Functions		Con-Proc
Designing Algorithms	73	AlgDesign		Algorithms and complexity	Con-Proc
File handling & I/O	59				Syn-Proc
Data Structures (general or implied complex like stacks, queues etc.)	54		DataStructures		Con-Proc
Classes & objects	52	Obj&Class			Con-Proc
Recursion	43	Recursion			Con-Proc
Generating clear documentation	41				Skill
How Computers & computational systems work & history of computing	38			Exposure to computer architecture and organization, computer science theory	Background
Abstraction	37			Study of computing-based systems at varying levels of abstraction	Con-Proc
Developing good program Design methodology & styling	34				Str-Cond
Strings	34				Syn-Proc
Searching algorithms	29				Con-Proc
Inheritance	28	Poly&Inheri			Con-Proc
"Fundamentals of Programming"	27			Concepts of programming languages and software development	Background
IDE use	27	IDE			Tools
Sorting Algorithms	26				Con-Proc
Program Comprehension	24				Con-Dec
Evaluating Time/Space Complexity	21				Con-Dec
Simple Graphics & GUIs	21				Tools
Polymorphism	20				Con-Proc
Exception Handling	19				Con-Proc
Pointers	19	Ptr&Refs			Syn-Proc

Encapsulation	18	Encapsulation			Syn-Proc
Teamwork skills & Communication	18				Professional skill
Abstract Classes & Interfaces	13	DesignClasses			Syn-Proc
Scope of code	12	Scope			Con-Proc
Memory allocation	9				Syn-Proc
Information Representation	6				Syn-Proc
Tracing execution of Program	6				Tools
UML Modelling language	6	UMLClassDiag			Advanced Topic
Command Prompt for Compilation and Execution	5				Tools
Detecting logic errors	5				Debuging
Detecting syntax errors	5				Syn-Dec
Pseudocode	5				Con-Proc
Web Development	5				Advanced Topic
Functional Programming	4				Str-Cond
Code Manipulation	3				Con-Proc
Multi Threading & Concurrency	3				Con-Proc
Coupling & Cohesion concepts	2				Con-Proc
Boolean Logic	1				Background
Induction	1				Background
Information Technology & Data Science skills	1			Information management	Advanced Topic
Security	1				Advanced Topic
Version Control	1				Tools
(Networking and communication)				Networking and communication	Advanced Topic
(Parallel and distributed computing)				Parallel and distributed computing	Advanced Topic
(Operating systems)				Operating systems	Advanced Topic

B.2 Partial Framework – Professional Skills

In our review of literature, we did not have a reference that presented a comprehensive list of skills or graduate attributes similar to the Becker survey. We therefore assembled lists from many of these references to build a superset table similar to the one above [17], [44], [45], [48], [52], [109], [111]–[113]. The list was then ranked by number of sources that references the same skill.

Table B.2 Comparison of professional skill areas from eight references [17], [44], [45], [48], [52], [109], [111]–[113].

#	ABET Shortened	EUR-ACE	Schoeman (ATC21S)	Passow 2012	Armstrong
8	Communication	Communication and Team-working	Communication (Working tactics)	Communication	Skill in written communication, communication with clients, communication with management
8	Teamwork and collaboration	Communication and Team-working	Cooperation or teamwork (Working tactics)	Teams	Skill in collaborating with the people you work with
6	Lifelong learning	Lifelong Learning	Learning to learn/metacognition--understanding own thinking processes (Thinking tactics)	Lifelong learning	
5	Problem solving	Engineering Analysis, Engineering Design, Engineering Practice	Critical thinking problem-solving (Thinking tactics)	Problem solving	
5	Ethical responsibilities		Personal and social responsibility (Behavior in the world)	Ethics	
4	Consideration of public factors		Local and global citizenship (Behavior in the world)		

4	Experimentation and judgement	Making Judgements, Investigations	Decision-making (Thinking tactics)	Data analysis	
4		Knowledge and understanding		Math, science & engineering	Knowledge of OS, Logic, etc
2					Ability to be adaptable
1			Creativity and innovation (Thinking tactics)		
1			Information literacy (Working tools)		
1			Information and communication literacy (Working tools)		
1			Life and career (Behavior in the world)		
1				Design	
1				Engineering tools	
1				Contemporary issues	
1				Experiments	
1				Impact	
1					Ability to be curious
1					Skill in stay motivated

(Only five references indicated. The others were completely covered by these existing rows.)

B.3 Added Items

From our industry expert classification framework, we added the following items. In our added items list, we had: 13 industry knowledge items, 13 academic knowledge items, 11 industry professional skills, and 5 academic professional skills. Clearly, the current state of research was not broad enough to cover everything mentioned by our experts in their open-ended questions.

- (knowledge) Single Language (7)
- (knowledge) Tools (7)
- (knowledge) Multiple languages (6)
- (knowledge) Unit test (5)
- (knowledge) Program management (5)
- (knowledge) Scripting language (4)
- (skill) Attention to detail (4)
- (skill) Receives feedback well (4)
- (skill) Humble (3)
- (skill) Accountable (3)
- (skill) Helpful (3)
- (knowledge) Computer hardware (3)
- (skill) Passionate (3)
- (skill-academic) Persistence (3)
- (knowledge-academic) Refactoring code (3)
- (knowledge) Code Reviews (2)
- (skill) Big picture (2)

- (knowledge) Multithreaded programming (2)
- (knowledge) Specific language (2)
- (knowledge) Imperative programming (2)
- (knowledge-academic) Life cycle (2)
- (skill) Asks for help (2)
- (knowledge-academic) Assembly language (2)
- (skill) Curiosity (1)
- (knowledge) Scientific method (1)
- (skill) Broad experience (1)
- (skill) Asks questions (1)
- (knowledge) Failure analysis (1)
- (knowledge-academic) Internship (1)
- (knowledge-academic) Databases (1)
- (skill-academic) Teachable (1)
- (knowledge-academic) Detailed logical thinking (1)
- (knowledge-academic) Inheriting and extending others' code (1)
- (skill-academic) Gathering client requirements (1)
- (knowledge-academic) Design a user interface (1)
- (knowledge-academic) Regression testing (1)
- (knowledge-academic) Coding to API (1)
- (skill-academic) Meets deadlines (1)
- (skill-academic) Self-confidence (1)
- (knowledge-academic) Pattern recognition (1)
- (knowledge-academic) Developing-and coding to-specifications (1)

- (knowledge-academic) Advanced data structures (heaps, B-trees) (1)

(knowledge-academic) Writing large program (multiple files) (1)

APPENDIX C

SURVEY COMPLETION RATES AND BOILERPLATE EMAILS

C.1 Delphi survey emails and interactions

Running the Delphi survey proved to be challenging to find participants as well as encourage them to participate in all three rounds of the survey. Here are some of the key statistics on participation rates.

Table C.1 Round completion rates for both expert groups

Group	Industry	Academic
Request Emails	49	71
P1 Start	36	33
P1 Completion	31	29
P2 Start	31	33
P2 Completion	29	24
P3 Start	31	33
P3 Completion	25	21
Participate	73%	46%
P1 Percentage	86%	88%
P1 of total	63%	41%
P2 Percentage	94%	73%
P2 of total	59%	34%
P3 Percentage	81%	64%
P3 of total	51%	30%

For the industry, the total Round 3 completion rate was 51%. For the academic group, where I did not have personal contact with any of the experts, we were at 30%. This means to reach our target of 30 completed Round 3 surveys we should have lined up 100 initial contacts. However, the start to complete ratios are much better, but still the personal contact with the industry group showed better participation. Of my 36 initial “yes” responses on the industry side, I had 31, 29, and 25 completed surveys for each round. Of my initial 33 “yes” responses from my academic side, I had 29, 24, and 21 respondents.

From the review of literature, individual communication was highly encouraged, so all my initial invite messages and follow-up were also one-on-one. While this took a fair about of time, I do believe my completion rates would have been significantly worse if I had done more group emails.

C.2 Boilerplate email

For each of the rounds, I build templates to be consistent with my participants. For Round 3, all the communication was one-on-one as the surveys were individual.

C.2.1 Personal industry initial invite to join my expert team

Subject: Looking for your help (personal research)

Hi <name> -- I'm hoping you can help me out! I have been working on my distance PhD in EE, and I'm currently doing some research on what skills make for a good programmer. With my professor, we are trying to discover how the skills needed for quality programmer compared to skills taught in the classroom. As someone who has programming and industry experience, you would provide a valuable perspective. Can I add you to my survey distribution list?

The commitment would be a total of about an hour spread out over three surveys. There would be an initial survey (to get what skills and experiences you think are needed to make a good programmer along with a few background questions), followed by two separate ranking requests. The time commitment is small, but your input would strengthen my work and help create consensus on the skills and experiences needed to become a "good" programmer.

Game?

Sincerely, John Hutton

C.2.2 Personal "thank you" of someone agrees

Excellent. I'm still getting some details together. Expect an email from me in the next 1-2 weeks. (The next email will be coming from my university account, which is a @msstate.edu address. I'll ping you from work as this may end up caught in a spam filter.)

Many thanks, John

C.2.3 Personal "survey away" email.
Subject: Round 1 survey sent!

Excellent! Thanks again for helping me with my survey! The email will be something like "John Hutton <noreply@qemailserver.com" from the Qualtrics.com service. It is also possible you may see communication from my jfh232@msstate.edu account. If you do not get the survey email, you may need to check your spam filter. Any issues just let me know!

Yours,

C.2.4 Initial Email with Qualtric link (from university email)
Subject: Delphi Survey Round 1 Invitation

Dear <name> -- In the field of programming, matching skills taught in college programming courses to skills needed to be successful in industry is a difficult task! Even trying to evaluate what mix of technical and interpersonal skills matter can be challenging.

The survey link here below is part of a Delphi survey to attempt to quantify some of these details. This survey has two key questions where I ask you to list skills and life experiences that you feel are needed to make a good programmer. There is no "right" answer here, so please think about it and list as many items as you think matter. These questions are followed by some quick background questions that we will use for classifying responses. Your results will be tabulated with about 40 other people and your individual identity will not be disclosed. This is Round 1.

We will send more information along with future surveys, but here are a few details to give you an idea of what will happen next. After we aggregate all the data, Round 2 will present you with a list of all the skills from all the survey responses. We will ask you to rank skills. Finally, in Round 3, we will show you the aggregated rankings versus your ranking and see if you would like to make any tweaks to your original ranking.

Again, I appreciate your willingness to participate. If possible, I'm hoping to have my 1st round of surveys completed by Tuesday Dec 7th.

Sincerely, John Hutton

C.2.5 Thanks for completing survey

Subject: Thanks for completing the Round 1 survey!

I appreciate you taking the time to complete the survey. This will really help me out. I have another batch of "Round 1" surveys to send out, so you should not expect the "Round 2" for 2-4 more weeks.

Sincerely, John Hutton

C.2.6 Round 2 email

Delphi Survey for John Hutton's PhD Research - Round 2

Dear Friends –

It has been much longer than I initially planned to get this Round 2 out to you! However, the Round 1 results were excellent. Everyone had thoughtful and inciteful things to say and the final list covered quite a lot of items.

This next pass should be much quicker than the 1st. There are 37 areas I'm asking you to rank from "Not Important" to "Very Important". These items range across "traditional"

programming items and into many items about professional skills and even personal attitudes and habits. There are no wrong answers.

Following this message, you will receive another email from Qualtric (email something like noreply@qemailserver.com). Check your spam if you don't see this as I have continued to have spam filters stop these for some people. I would like to have this round completed by 5/2, so try to schedule the 5-10 minutes when you can.

Sincerely, John Hutton

C.2.7 Round 2 Qualtric Message

Friends –

Here is the Round 2 ranking survey. There are 37 areas to rank. Take a quick pass through the list (if possible, difficult if you are doing this on the phone) and then try to rank everything from "Not Important" to "Very Important". Remember that our target would be things that would be important in a new hire you were interviewing, or skills needed by a new hire that would set them on the rode to being an excellent engineer.

There are no wrong answers.

Thanks again for your help throughout this process!

Yours, John Hutton

C.2.8 Round 3 email

Subject: Delphi Survey for John Hutton's PhD Research - Round 3

Dear -

We are I the home stretch! The link below is to my Round 3 survey which is the last. This may take slightly longer than the Round 2, but it should be relatively quick.

Here is your link:

For a Delphi survey, Round 3 is looking to build consensus. This will never be complete, but the methodology works like this. This list of items is ranked based on the aggerate statistics from all the respondents in Round 2. The items go from most highly ranked to least highly ranked. Each item has the statistical "mean" from the data. A mean of 3.00 means that the average of all the ratings was equivalent to "moderately important.

Your Round 2 answers will be entered as the starting default for Round 3. Now that the group data is also present, take a quick look through your responses and pay particular attention to items where your selection was different from the group statistics. Reconsider that particular item and consider whether there may be some positive or negative side you might not have considered. If you decide to move your answer closer to the group answer, that is great. If you decide to stay where you are or even move further away, that is equally valid. On any item where you put in some quick, but serious, though, there is a text field under the item where you can enter comments. If the group had a rating different than yours, you can enter a short section of why you continue to support your selection. If you change an entry, you can enter what thought moved you to a change. You do not have to put comments on every question, but the more you can call out specific details on differences (in particular) the better I will be able to aggerate the final data and possibly see threads among all of the responses.

If you can finish this survey by Friday (5/20) that would be excellent, but my actual deadline for this is next Tuesday (5/24).

Thanks again for your help!

Sincerely, John

C.2.9 Academic ECE and CS Head Request for help
Subject: Assistance with PhD research in Engineering Programming

(Note: Per our method, we initially reached out to department heads to ask for possible professors who might agree to join our academic expert group.)

Hi --

Forgive this blind email, but I could use your help. I'm looking for some professors who teach programming to joining my academic expert team for a Delphi survey I'm conducting for my PhD research. My name is John Hutton from Mississippi State University, and my research is striving to build a consensus list of knowledge and professional skill areas that are critical for engineering and CS programmers. As my academic network is small, I am reaching out to ECE and CS department heads at several Abet accredited ECE/EE schools hoping they could connect me with interested professors (but Abet accreditation is NOT required for participation).

Could you help me? If you can provide names of professors who teach CS1 (or other programming courses), I will reach out to them 1:1 to confirm interest.

Sincerely, John Hutton

C.2.10 Academic Personal
Subject: Re: Assistance with PhD research in Engineering Programming

Hi --

I was given your contact information by <> as someone who might be able to help me. I'm looking for professors who teach programming to joining my "academic expert" team for a Delphi survey I'm conducting for my PhD research. My name is John Hutton from Mississippi State University, and my research is striving to build a consensus list of knowledge and professional skill areas that are critical for engineering and CS programmers. The commitment will be around one hours spread across the traditional three rounds of a Delphi method survey.

Would you be willing to help me?

Sincerely, John Hutton

C.2.11 Academic Round 1 Qualtric Initial Email
Subject: Delphi Survey Invitation

In the field of programming, matching skills taught in college programming courses to skills needed to be successful in industry is a difficult task! Even trying to evaluate what mix of technical knowledge and professional skills to teach can be challenging.

The survey link below is Round 1 of my Delphi survey to attempt to quantify some of these details. This survey has two key questions where I ask you to list knowledge, skills, and life experiences that you feel are helpful to make a good programmer. There is no "right" answer here, so please think about it and list as many items as you think matter. These questions are followed by some quick background questions that we will use for classifying responses. Your results will be tabulated with about 30 other people and your individual identity will not be disclosed.

We will send more information along with future surveys, but here are a few details to give you an idea of what will happen next. After we aggregate all the data, Round 2 will present you with a list of all the skills from all the survey responses. We will ask you to rank them. Finally, in Round 3, we will show you the aggregated rankings versus your ranking and see if you would like to make any tweaks to your original ranking.

Again, I really appreciate your willingness to participate. If possible, I'm hoping to have this round of surveys completed by 4/18.

Sincerely, John Hutton

C.2.12 Personal email if I fear spam capture!
Subject: Round 1 Survey on-the-way

Hello Friends –

Just a quick note from my school email to let you know that your survey has been queued up and should be in your inbox 30-60 minutes after this message arrives. I have already had four people figure out that their school spam filter caught the survey message with the link. If you do not see a 2nd message (will be from an email like noreply@qemailserver.com as this is generated by the Qualtrix service).

Thanks again for being a part of my academic team.

John

C.2.13 Academic Round 2 email
Subject: Delphi Survey for John Hutton's PhD Research - Round 2

Dear Friends –

The Round 1 results are completed! Thanks for all the effort you put into the open-ended questions. We have analyzed these answers and generated a list of 35 categories that you will be ranking in our Round 2 survey. This should be quicker than the Round 1 as most of my prior group were able to complete the ranking in under 5 minutes. I'm asking you to rank from "Not Important" to "Very Important". These items range across "traditional" programming items and into many items about professional skills and even personal attitudes and habits. There are no wrong answers.

Following this message, you will receive another email from Qualtric (email something like noreply@qemailserver.com). Check your spam if you don't see this as I have continued to have spam filters stop these for some people. I would like to have this round completed by 6/4. I know we will be fighting some summer plans and vacations, but please try to complete this as

you are able. I hope to turn around Round 3 in a few days after these results are compiled. It is a ranking check, so also should be in the 5-minute range.

Thanks again for being a part of my research!

Sincerely, John Hutton

C.2.14 Aca Round 2 Qualtric Message

Subject: Delphi Survey for John Hutton's PhD Research - Round 2

Friends –

Here is the Round 2 ranking survey. There are 35 areas to rank. Take a quick pass through the list (if possible, difficult if you are doing this on the phone) and then try to rank everything from “Not Important” to “Very Important”. Remember that our target would be things that would be important for a graduate going into a job interview situation. There are no wrong answers.

Thanks again for your help throughout this process!

Yours, John Hutton

C.2.15 Aca Round 2 email

Subject: Delphi Survey for John Hutton's PhD Research - Round 3

Dear –

Home stretch! Thanks again for your time so far. Our Round 3 should be relatively quick. Your Round 2 answers will be pre-populated into the ranking matrix. However, this time, the matrix is listed in group mean result order. The highest ranked item first and the lowest ranked item last. Consider your response versus the group statistics. If you are significantly higher or lower than the group, consider if this might be more (or less) important than your first ranking.

There is a text field under every item where you can enter a short comment. If you make a change, or if you stay at a delta from the group, you can put a statement about why.

Survey Link:

Once again, thanks for your help! My goal is to have this round done by next Friday (6/24). I know we will have some vacations, etc. that will prevent this from working for everyone, but try to take a few minutes if you can. 😊

Note: If you did not fill out the Round 2 survey, you will not have any default answers. I would still love to have your feedback on these items.

Sincerely, John Hutton