UC San Diego
UC San Diego Electronic Theses and Dissertations

Title
Towards Structurally Closing the Academia-Industry Gap in Undergraduate CS

Permalink
https://escholarship.org/uc/item/08m0f1gc

Author
Valstar, Andries

Publication Date
2021

Peer reviewed|Thesis/dissertation

eScholarship.org Powered by the California Digital Library
University of California

UNIVERSITY OF CALIFORNIA SAN DIEGO

Towards Structurally Closing the Academia-Industry Gap in Undergraduate CS

A dissertation submitted in partial satisfaction of the
requirements for the degree
Doctor of Philosophy

in

Computer Science

by

Andries Valstar

Committee in charge:

Professor William G. Griswold, Co-Chair
Professor Leo Porter, Co-Chair
Professor Philip Guo
Professor Gail Heyman
Professor Scott Klemmer
Professor Joe Politz

2021

Copyright

Andries Valstar, 2021

All rights reserved.

The dissertation of Andries Valstar is approved, and it is acceptable in quality and form for publication on microfilm and electronically.

University of California San Diego

2021

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I want to thank my advisors Leo Porter and Bill Griswold for sharing with me their vast knowledge of- and enthusiasm for Computing Education Research as well as for believing in me and sharpening my approach to research, teaching and life.

I want to thank my wife Guan-Lin for being supportive of my ambitions as well as for her own ambitions that ultimately motivated me to come to California. Without her I would not have embarked on this journey, but more importantly this PhD would have been infinitely more difficult to bring to completion without having my best friend and partner by my side.

Of course, I also want to thank my parents Ria and Wim as well the rest of the family for supporting me throughout this adventure and for providing frequent updates and encouragements from back home.

I want to thank my co-authors and lab members Sophia, Adrian, Alex, Caroline and Sooh, it has been a pleasure working with all of you! And finally, I would like to thank anyone I've interacted with during my time in San Diego, professors, TAs, tutors, CSE3150 folks, classmates, students, friends and everyone else. You helped make this a wonderful experience!

Chapter 3, in full, is a reprint of the material as it appears in the Proceedings of the 51st ACM Technical Symposium on Computer Science Education (SIGCSE 2020). Sander Valstar, Sophia Krause-Levy, Alexandra Macedo, William G. Griswold, Leo Porter. "Faculty Views on the Goals of an Undergraduate CS Education and the Academia-Industry Gap". The dissertation author was the first author of this paper.

Chapter 4, in full, is a reprint of the material as it appears in the Proceedings of the 2020 ACM Conference on International Computing Education Research (ICER 2020). Sander Valstar, Caroline Sih, Sophia Krause-Levy, Leo Porter, William G. Griswold. "A Quantitative Study of Faculty Views on the Goals of an Undergraduate CS Program and Preparing Students for Industry". The dissertation author was the first author of this paper.

Chapter 5, in full, is being prepared for publication and may appear in ACM Transactions on Computing Education in 2021. Sander Valstar, Adrian Salguero, Sophia Krause-Levy, William G. Griswold, Leo Porter. “Analyzing the Adoptability of Educational Innovations: A Case Study of the Academia-Industry Gap in CS”. The dissertation author was the first author of this paper.

Chapter 6, in full, is a reprint of the material as it appears in the Proceedings of the 50th ACM Technical Symposium on Computer Science Education (SIGCSE 2019). Sander Valstar, William G. Griswold, Leo Porter. “The Relationship between Prerequisite Proficiency and Student Performance in an Upper-Division Computing Course”. The dissertation author was the first author of this paper.

Chapter 7, in full, is a reprint of the material as it appears in the Proceedings of the 2021 ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE 2021). Sander Valstar, Sophia Krause-Levy, Adrian Salguero, Leo Porter and William G. Griswold. “Proficiency in Basic Data Structures among Various Subpopulations of Students at Different Stages in a CS Program”. The dissertation author was the first author of this paper.

VITA

2017-2021	Ph.D. Computer Science, University of California San Diego, U.S.A.
2015-2017	Software Engineer, bol.com, Utrecht, the Netherlands
2013-2015	M.Sc. Computer Science, National Taiwan University, Taipei, Taiwan
2009-2013	B.Sc. Computer Science, Eindhoven University of Technology, Eindhoven, the Netherlands

FIELD OF STUDY

Computing Education Research

PUBLICATIONS

Sander Valstar, Sophia Krause-Levy, Adrian Salguero, Leo Porter and William G. Griswold. "Proficiency in Basic Data Structures among Various Subpopulations of Students at Different Stages in a CS Program", ITiCSE 2021

Sander Valstar, Caroline Sih, Sophia Krause-Levy, Leo Porter, William G. Griswold, "A Quantitative Study of Faculty Views on the Goals of an Undergraduate CS Program and Preparing Students for Industry", ICER 2020.

Sander Valstar, William G. Griswold, Leo Porter. "Using DevContainers to Standardize Student Development Environments: An Experience Report", ITiCSE 2020.

Sander Valstar, Sophia Krause-Levy, Alexandra Macedo, William G. Griswold, Leo Porter. "Faculty Views on the Goals of an Undergraduate CS Education and the Academia-Industry Gap", SIGCSE 2020.

Sander Valstar. "Closing the Academia-Industry Gap in Undergraduate CS", ICER 2019 Doctoral Symposium.

Sander Valstar, William G. Griswold, Leo Porter. "The Relationship between Prerequisite Proficiency and Student Performance in an Upper-Division Computing Course", SIGCSE 2019.

ABSTRACT OF THE DISSERTATION

Towards Structurally Closing the Academia-Industry Gap in Undergraduate CS

by

Andries Valstar

Doctor of Philosophy in Computer Science

University of California San Diego, 2021

Professor William G. Griswold, Co-Chair
Professor Leo Porter, Co-Chair

The academia-industry gap in undergraduate CS education has been a well recognized problem over the past two decades. Many papers have identified this gap or proposed novel solutions to parts of it. Yet, recent studies found the gap persists. This dissertation explores various possible reasons that could lie behind the persistence of this gap. We uncover that the majority of CS faculty support including more industry-relevant content in their courses. Faculty do however experience a skills, resources and awareness gap which forms a barrier to making their courses more relevant to industry practice. Moreover, we find that these barriers are experienced by CS faculty at top universities world-wide. Furthermore, many of the efforts to close the gap do not adequately address known barriers to adoption and barriers encountered by faculty. Additionally, one concern raised by some faculty is that students may not be learning the skills that we are teaching. These faculty are partially correct as we find evidence indicating that the average student proficiency with prerequisite materials may not be fully aligned with instructor expectations. Moreover, a different study indicates there are performance disparities between several subpopulations of students. However, we also find evidence that indicates that students on average continue to improve their understanding of CS fundamentals as they progress through their program, indicating that students on average do appear to learn the materials that we teach.

Chapter 1

Introduction

This chapter will introduce my motivation for pursuing research on the topic of the academia-industry gap in Computer Science education as well as the early discoveries of my first literature reviews on the topic, the current state of the art, my thesis statement and my contributions to the field.

1.1 Motivation

When I started my first full-time job as a software engineer I realized quickly that there were many skills that I needed to have to be able to perform my job responsibilities that I was not necessarily taught in my Computer Science studies. Now this is of course not all that surprising in and of itself, every job will be unique in some ways and require some form of on-boarding process. However, many things I was unfamiliar with were certainly not unique to the company I worked at. Take testing for example. From my education I was only familiar with unit testing. However, my team was using numerous additional kinds of testing that I had never heard of before: integration testing, UI testing, performance testing, load testing, acceptance testing, A/B testing and even manual testing using formal test scripts. Moreover, I was also unfamiliar with commonly used design techniques that lead to testable code and was unfamiliar with the concept of mocking.

Note that testing is merely one example used to illustrate the point that I experienced a disconnect between my studies and my first job, I struggled with various other things as well. For example, I was overwhelmed by the size of the project code bases which were several orders of magnitude larger than any project I'd ever worked on before. The engineers employed numerous design strategies to keep the code bases manageable, most of which were completely new to me. These approaches and skills were not specific to the company I worked at and will likely be applicable in most professional software engineering contexts.

Having spent about 5-6 years in CS education at two different world-renowned universities, attaining both a Bachelor and a Master's degree in Computer Science, I would have expected to arrive better prepared for the realities of my first job. From this experience, I concluded that there should be things that can be improved in academic CS programs that will narrow the gap that recent CS graduates currently have to bridge by themselves. It is what ultimately motivated me to pursue research in Computing Education aimed at closing the academia-industry gap.

1.2 Early Discoveries

When I first arrived at the University of California San Diego I was under the impression that academia was generally unaware of the gap I had experienced. However, while performing the literature review for my research exam, I was pleasantly surprised by the amount of studies that had already been published on the topic. Studies published as early as 1998, have concluded that there exists a gap between CS education and the realities of the software industry [67]. All throughout the past two decades studies reaching similar conclusions have continued to be published. Table 1.1 presents an overview of these studies.

After discovering this abundance of proof for the existence of the academia-industry gap I thought my main research contribution would be proposing educational innovations that could close the gap. However, when I continued my literature reviews to inventorize the existing

Table 1.1: Publications identifying the academia-industry gap in Computer Science.
Study subject: I=industry professionals, O=opinion article, S=students/recent graduates, L=literature review

Year	Author(s)	Subj.	Title
1998	Lethbridge	I	The relevance of software education: A survey and some recommendations [67]
2002	Haddad	I	Post-graduate Assessment of CS Students: Experience and Position Paper [51]
2003	Brechner	O	Things they would not teach me of in college: what Microsoft developers learn later [25]
2004	Hagan	I	Employer Satisfaction with ICT Graduates [52]
2007	Taft	O	Programming Grads Meet a Skills Gap in the Real World [116]
2008	Begel & Simon	S	Struggles of New College Graduates in Their First Software Development Job [22]
2008	Begel & Simon	S	Novice Software Developers, All over Again [21]
2010	Stroustrup	O	Viewpoint: What should we teach new software developers? Why? [113]
2013	Radermacher & Walia	L	Gaps Between Industry Expectations and the Abilities of Graduates [93]
2014	Radermacher et al.	I	Investigating the Skill Gap Between Graduating Students and Industry Expectations [94]
2018	Craig et al.	S	Listening to Early Career Software Developers [32]
2019	Kapoor & Gardner-McCune	S	Understanding CS Undergraduate Students' Professional Development Through the Lens of Internship Experiences [62]

relevant educational innovations I discovered yet again a large number of studies (see Chapter 5 and Appendix A.1). The presence of this substantial body of educational innovations related to the academia-industry gap somewhat surprised me. After all, if the community is aware that the gap exists and there are already existing solutions to remedy at least parts of it, then why does the gap persist? Since these innovations could in theory be taken straight off the shelf and implemented in classrooms across the globe as solutions to the academia-industry gap, I had to shift my ideal research contribution from a solution focus to an understanding focus. The reason being that, given the abundance of relevant educational innovations, providing insight into possible reasons why the academia-industry gap has persisted in the face of all this relevant research would be a more substantial contribution to the field than adding yet another educational innovation to the literature.

Disregarding the opinion articles, one striking similarity between the studies in Table 1.1 is that they either study students/recent graduates or experienced industry professionals. Thus, my literature study uncovered that there is a lack of studies conducted on Computer Science faculty. Studying faculty then, would be the first avenue for my research to contribute to the understanding within the academic community of the persistence of this problem. The findings in my studies on Computer Science faculty would prompt new studies in several directions. One such direction of study was to investigate existing relevant educational innovations for the presence of barriers to adoption. Another direction of study directly following from faculty comments was to investigate if the academia-industry gap could be caused by students simply not learning as much of what we teach as we might expect.

1.3 The Current "State of the Art"

While there is a wide variety of educational innovations available that claim to address one or more parts of the academia-industry gap (See Chapter 5 as well as a recent effort by MIT that gained some popularity online [80]). There is not one recognized "state of the art" approach to closing the academia-industry gap.

One likely reason for why there is no recognized current best approach is that the academia-industry gap consists of a wide variety of different topics. Thanks to Radermacher and Walia, we do know the dimensions of the academia-industry gap. In their study they identified 15 areas of student deficiency that occur frequently in papers on the gap [93]. Additionally, while the academia-industry gap is a recognized issue in the field there is less recognition on what should be areas of prime focus, and against what baseline innovations should be assessed. The resulting situation is a large body of studies that may claim to be beneficial in their respective settings, but a lack of a wider push to systematically move towards closing the academia-industry gap.

For some areas in Radermacher and Walia's list of gaps it is easier to establish what is the state of the art than for others. For example, in the area of Computer Ethics I am not aware of a state of the art approach even after my literature reviews for Chapter 5. Moreover, since I myself am not a computer ethics expert it would be difficult for me to establish exactly what the state of the art is unless the literature is very clear on this matter. In Chapter 5 I dealt with this problem by using citation frequency as a proxy for state of the art status among available innovations.

An example of an area for which there is a clearer state of the art is the area of Software Testing. In this area several different approaches for the automated assessment of the quality of student written tests have been compared. Shams found that mutation testing and all-pairs testing were better indicators of student test quality than code coverage analysis, which is the current industry standard for analyzing test quality in professional software projects [101]. However, while according to this research it would be better to apply some combination of mutation testing and all-pairs testing instead of code coverage analysis for automatically grading student test quality, applying any approach to grade student test quality would likely already be substantially better than the status quo in which Software Testing is often under emphasized and not graded at all. Again, it was this realization, that adequate approaches exist but appear to simply not be used, that prompted me to focus on investigating why the gap has persisted and what keeps faculty from adopting these approaches instead of focusing on creating new approaches myself.

1.4 Thesis Statement

There is a long-standing academia-industry gap in undergraduate CS education that has been able to persist in part due to a lack of understanding of how faculty view their role in preparing students for industry. It is important to understand how faculty view the academia-industry gap as they would ultimately need to implement at least some of the changes necessary to close it.

Encouragingly, we find that a majority of faculty is supportive of closing the academia-industry gap in CS programs. However, despite substantial study and support, the gap persists. This lack of success in addressing the gap is in part due to barriers experienced by faculty and barriers to adoption present in existing solutions. While there are also some concerns regarding student learning, we find students on average do continue to improve their understanding of CS fundamentals throughout the program.

More precisely, this thesis substantiates the following claims:

1. There is a long-standing and unresolved academia-industry gap in CS, despite substantial study and efforts (Chapters 2 and 5, and Table 1.1).

2. We claim that a majority of faculty are willing to help close the gap, but that many experience barriers that keep them from closing the gap in their own courses (Chapters 3 and 4).

3. Moreover, we claim that many of the efforts to close the gap do not adequately address known barriers to adoption and barriers encountered by faculty (Chapter 5).

4. A concern raised by some faculty is that students may not be learning the skills that we are teaching. Our claims on this front are the following:

    (a) We claim that students on average appear to not be as proficient with prerequisite fundamental CS materials as instructors in later courses might expect (Chapter 6).

    (b) We claim that students on average seem to continue to improve their knowledge of CS fundamentals in later courses (Chapters 6 and 7).

    (c) We claim that there appear to be performance disparities between several subpopulations of students (Chapter 7).

1.5 Contributions

The contributions to the field outlined in this thesis are the following:

1. Insight into the existing faculty views on this issue, the relative support for opposing views and barriers experienced by faculty (Chapters 3 and 4).

2. Insight into which barriers to adoption are frequently present in currently available innovations (Chapter 5).

3. Insight into student learning. More specifically, uncovering the importance of prerequisite knowledge as well as how students continue to improve fundamental CS knowledge throughout their CS program (Chapters 6 and 7).

4. Insight into performance disparities between student subpopulations (Chapter 7).

Chapter 2

Background and Related Work

This chapter summarizes the related work on the academia-industry gap. More in-depth related work sections can be found inside the relevant chapters in this dissertation.

2.1 Studies Uncovering the Gap—What About Faculty?

Many papers published over the past two decades have uncovered the existence of a persisting academia-industry gap in CS. These works can broadly be categorized as follows: studying students/recent graduates, studying industry professionals, literature reviews and opinion articles. As can be seen in Table 1.1, the vast majority of these studies are studying students/recent graduates or industry professionals. Unfortunately there was a striking lack of studies conducted on CS faculty regarding this issue. It is important to understand how faculty view the academia-industry gap and their role in this issue as they are ultimately the ones who will have to bring about the change required to close the gap. The lack of studies on faculty is what prompted me to conduct the research outlined in Chapters 3 and 4.

2.1.1 The Academia-Industry Gap in Other Fields

The presence of an academia-industry gap is not unique to the field of Computer Science. Some examples of other fields in which researchers have noted the presence of a similar gap are:

Hospitality [18], Law [37], Business [78], Medicine [23], and Accounting [104].

Also noteworthy is that a number of papers have identified the presence of an academia-industry gap for Software Engineering (SE) programs and specializations [13, 20, 43, 83, 126]. This indicates that simply offering a separate SE program or specialization for students seeking to become software engineers does not necessarily solve the academia-industry gap in CS.

2.2 Studies Proposing Solutions to (parts of) the Gap

The academia-industry gap is a diverse problem. Radermacher and Walia identified 15 different areas of student deficiency [93]. Some deficiencies are very technical whereas others are more social in nature. For each of these 15 areas, educational innovations have been proposed in academic literature (see Chapter 5 and Appendix A.1). Moreover, there are several different styles of approaches to closing parts of the gap with industry that can take substantially different forms:

Internships (e.g. students spend their summer break working at a company)

Co-op programs (e.g. students spend several terms working at a company, generally more structured than internship programs)

Technological changes (e.g. use a different IDE in a programming course)

Curriculum-wide changes (e.g. grade software testing in *all* programming courses)

Course changes (e.g. teach software architecture using popular open source projects)

New courses (e.g. add a new course to teach communication skills)

Each different type of approach comes with its own benefits and drawbacks. For example, internships may be relatively hands-off solutions for universities. However, since the number of available internship positions is limited, not every student will be able to benefit from such programs. Moreover, the benefit to student learning will likely heavily depend on how serious the companies take their role of creating learning opportunities for the students over simply using the programs as recruitment opportunities. Furthermore, a recent study has found that about half of CS students returning from an internship feel that their CS studies had not prepared them well for their internship [62].

The diversity of the problem combined with the vastly different existing approaches makes it difficult to compare among approaches directly and pick a "best" solution to the academia-industry gap. Instead of trying to pick one "best" solution, it is likely better to implement a combination of multiple different approaches. The most important insight I gained from my reviews of related work on solutions to the academia-industry gap is that there is already an abundance of educational innovations in academic literature that claim to be effective at closing parts of the gap (see Chapter 5 and Appendix A.1). The question remains, why does the gap persist today despite the presence of these innovations?

2.3 Studies on Adoption of Educational Innovations

A possible explanation as to why the gap persists despite the presence of educational innovations can be found in studies on adoption of educational innovations. Multiple studies have been conducted on this topic, in CS as well as in other fields. Notable is an organization called "Increase the Impact" that aims to help innovators with achieving sustained adoption of their innovations [123]. For the field of CS, a working group was started in 2018 to help promote the adoption of educational innovations [119]. Their work led to a paper published at ITiCSE 2018 [118].

Several studies have identified barriers to adoption commonly cited by faculty, such as unawareness of the innovation and a lack of time to implement it [60, 118]. Chapters 3 and 4 investigate how faculty experience these barriers. Chapter 5 investigates which barriers are frequently present in existing solutions to parts of the academia-industry gap.

2.4 Studies on Student Learning

Another possible explanation for the persistence of the academia-industry gap according to some faculty in Chapters 3 and 4 is that students who feel underprepared for industry might simply not have been learning what faculty were trying to teach them. The study described in Chapter 7 was conducted in part as a result of the finding that some faculty hold this opinion.

In the field of Computing Education Research, naturally many studies have been conducted on student learning. A famous example that shows students are not learning as much in courses as instructors might expect is the "rainfall problem" described in Soloway's 1986 paper [108]. Moreover, related work has found that knowledge loss among students can be substantial over time. Tennyson and Beck found that students had a 15.1% knowledge loss over the summer break whereas knowledge loss was only 4% over the much shorter winter break [120]. It is also known that there are performance differences between students with prior experience with CS and those who do not, that persist throughout the entire undergraduate program [16]. Chapter 7 investigates several other subpopulations of students for the presence of similar performance differences.

Chapter 3

Exploring Faculty Views on the Academia-Industry Gap

A Qualitative Study of Faculty Views on the Goals of an Undergraduate CS Education and the Academia-Industry Gap

3.1 Introduction

Students tend to take a relatively pragmatic approach in deciding to attend university. They often expect their degree to prepare them for the job market and increase their employability [44, 84]. This job-focused mentality is even more prevalent in first-generation students [66] and does not change significantly, among psychology students, as they progress through their degree [84]. The expected benefits are most heavily centered along extrinsic motivations, including obtaining a better career or fulfilling family expectations. The most important factor influencing students to study CS, in addition to general interest and prior experience in the field, are career prospects [15, 55].

In contrast to their expectations, many CS graduates feel unprepared for the challenges of their first job [22, 32, 62]. This disconnect between academia and industry in CS was identified at least two decades ago [67], yet studies conducted in the past decade show that not much progress has been made to close this gap [22, 32, 62].

This study aims to uncover some of the reasons why the academia- industry gap persists. Although it is generally well-understood that students desire more career-related training [32, 44], faculty views on the issue remain relatively unstudied in CS. Faculty are important to study, as they design and implement the curriculum (e.g., choosing to what extent to realize the ACM Curricula Recommendations [61]). Faculty views were recently studied focusing on the alignment between CS education and industry needs [30], assuming faculty see industry preparation as a primary goal. Our work explores faculty views more broadly, canvassing their views on the goals of an undergraduate education and a CS major to ultimately answer the question: "Are faculty views on undergraduate CS a potential cause for the persistence of the academia-industry gap?".

Previous work has found that enhancing employability is not always a major focus of curriculum design [44] and that learning outcomes of a degree are generally developed by faculty without referring to students' motivations for attending university [84]. Thus, we expected to find that a sizeable group of faculty oppose the idea that preparing students for industry is a main goal of a CS program. As such, our questions concern (a) whether faculty believe preparation for industry is a primary goal of a CS degree, (b) why they hold that view, and (c) whether there are other obstacles to closing the gap.

To identify the range of faculty views on the goals of a CS education, we conducted semi-structured interviews. We recruited a diverse group of 14 CS faculty across 3 institutions, including a large research-intensive university, a small liberal arts college, and a large primarily undergraduate institution. We employed Phenomenography to identify the full spectrum of experiences and beliefs of the participants.

We found, first, that faculty feel that, beyond the CS fundamentals, the goals of a CS degree should be defined by the student's objectives. Moreover, with the understanding that real-world impact is likely, students should be prepared for ethical leadership. Second, a range of opinions regarding industry preparation were expressed, including faculty either opposing or supporting industry preparation playing a significant role in CS education. Third, faculty admitted struggling with properly preparing students for industry, citing increasing class sizes, lack of background in industry practices, and the difficulties of designing realistic projects without overtaxing both the students and instructors.

3.2 Background

3.2.1 Motivations for Attending University

Wilson et al. suggest that there should be a close alignment between student and instructor expectations to improve students' learning and performance. Hence, faculty designing curricula should understand student motives and program expectations [136]. Student motives among geography majors were also studied, finding that undergraduate and graduate students desire that their curricula focus more on career guidance and vocational training [44]. Similarly, in computer science, students value career-relevant training during their undergraduate degree [32] and, in some cases, may not always understand the relevance of many CS courses for their careers [89].

3.2.2 Academia and Industry Preparation Gap

Industry practitioners report that there is a gap between university graduates' abilities and industry expectations, including both technical and non-technical skills [90]. Recent computer science graduates have been found to undervalue software testing [90, 93], struggle with configuration management systems and other software tools [93], lack relevant project experience [94], and lack experience fixing bugs in, or adding features to, large or poorly documented legacy code bases [22]. For a literature review, see [93].

A number of studies have also shown that students lack social and "soft" skills relevant for success in industry. A study that analyzed the alignment of course syllabi and industry needs noticed a lack of mentioning of soft skills in the syllabi [72]. CS graduates have been found to be underprepared to effectively communicate with co-workers and customers [94] and they fail to reach out for help from colleagues or senior engineers when needed [22]. Moreover, they have a number of fundamental misconceptions about how software engineering work is performed in industry, including the importance of team interactions, although internship experience reduced the frequency of these misconceptions to some extent [115].

Craig et al. summarized prior work on this subject and conducted additional student interviews. They found that CS students were underprepared for work on customer-centered projects of vague and evolving scope, work on projects that would span multiple years, collaboration with larger teams to design complex systems, and use of professional tools and formal practices. Notably, half of CS students in a recent study reported that their CS program did not prepare them adequately for their professional experiences [62].

Caskurlu et al. conducted a study on faculty views on the alignment between CS programs and industry needs. They found many faculty believe CS programs prepare students well for industry jobs, citing the fact that the majority of their graduates have no problems finding high paying jobs. However, faculty also mentioned that soft skills are under-taught and that some topics such as software maintenance are hard, if not impossible, to teach [30].

3.2.3 Attitudes and Beliefs about CS

A study by Lewis surveyed 13 faculty and 160 undergraduates on their beliefs and attitudes about a variety of CS-related topics, and then compared how students and faculty differed in their views. Topics included things like the importance of group work and the role of aptitude for success. They found that although students and faculty disagree on many topics in general. In some cases, students later in their CS degree began to better align with faculty views than those earlier in their career, but in other cases the disconnect persisted [68]. One such statement rejected by faculty, but more accepted by the students, was “In the real world, computer scientists spend a lot of time working alone.” This finding is relevant to our study, as it shows how faculty and students may disagree on basic assumptions about applications of CS, even after interacting with these faculty during their undergraduate education. These findings have been replicated and further explored, focusing on areas where faculty and students disagree [69, 88].

3.3 Research Questions

While the motivation for our study is primarily the gap between academia and industry, we framed our interviews more broadly. In particular, we included interview questions concerning the goals of an undergraduate CS education in general and the preparation of students for careers in academia. There are two reasons for this choice. The first is that we wanted our interviews to allow us to identify other important aspects of a CS education besides preparation for industry. The second is that we wanted to avoid potentially offending faculty by focusing solely on industry-specific interview questions that could cause them to take a defensive position, compromising our data collection process. Our research questions are:

RQ1	What are faculty views on the goals of an undergraduate CS education?
RQ2	What are faculty views on the role of an undergraduate CS education in preparing students for industry?
RQ3	What are faculty views on better preparing students for careers in industry?

Whereas RQ1 remains more broadly focused, RQs 2 and 3 focus solely on preparation for industry. For reasons of space we omit results related to preparation for careers in academia, only listing some of the most important findings in Sections 3.5.3 and 3.6.

3.4 Methods

In order to understand the range of faculty views regarding the goals of a CS degree, we found a qualitative interview method to be the best fit. A quantitative study using, say, a multiple choice questionnaire would have limited faculty responses to our predetermined questions and answers, likely biasing the results towards our imagination of what faculty views might be. Taking a qualitative approach allowed the participating faculty more freedom to directly and fully express their views in their own words.

Specifically, to both ensure that our research questions were addressed and allow faculty to express their full views, we decided to employ a semi-structured interview format, which prescribes a short list of questions that can be followed up by additional questions that are prompted by the participant's answers. For data analysis, we chose a phenomenographic approach, which seeks to identify the full spectrum of experiences and beliefs of participants, as opposed to finding a single objective truth [75].

3.4.1 Data Collection

The semi-structured interviews were audio-recorded and then transcribed for data analysis. We first devised an initial set of questions followed by a pilot study on four senior PhD students. During this pilot we revised the interview questions until they rendered responses that adequately answered our research questions. Some examples of the resulting questions are: "What does it mean to you for someone to be a Computer Scientist?" and "..., what do you think is the role of an undergraduate CS education in preparing students for industry?". The full set of interview questions is publicly available online [3].

To achieve a high response rate we applied convenience sampling by sending personalized emails to personal connections of my advisors Leo Porter and Bill Griswold. Unfortunately, recruitment from institutions other than our own proved challenging. We ultimately conducted interviews at one research-intensive university (ours), one large primarily undergraduate institution, and one small private liberal arts college. The distribution of participants across these schools and the size of their CS programs can be seen in Table 3.1. I personally conducted all the interviews and either Sophia Krause-Levy or Alex Macedo was taking notes during the first 10 interviews.

We deliberately invited faculty from a wide variety of research areas. Resulting in

Table 3.1: Participant distribution and size of CS programs.

School Type	Participants	Degrees awarded in 2016 [9]
Research	11	total: 6,477, CS: 470
Undergraduate	1	total: 7,058, CS: 101
Liberal Arts	2	total: 1,463, CS: 9

participants with the following backgrounds: Theory of Computation, Algorithms, Real-Time Scheduling, Cryptography, Cloud Computing, PL, ML, Cyber-Physical Systems, Bioinformatics, Parallel Programming, CER, and Architecture. The seniority of the participants varied greatly, from faculty who were recently hired to faculty with several decades of experience. Of the 14 participants, 5 have some experience working in industry (only 2 as software engineer). Additionally, 1 participant owns a startup and 2 participants collaborated with startups.

3.4.2 Analysis

Phenomenography is a method used in qualitative research as an alternative to Grounded Theory. Where Grounded Theory allows the researcher to develop a theory grounded in the data, Phenomenography focuses on identifying the full range of understandings or beliefs exhibited by the subjects. An excellent explanation of the phenomenographic process can be found in Simon et al.'s work [107, §1.2]. A more in-depth explanation can be found in Åkerlind's work [11]. An overview of how Grounded Theory and Phenomenography are applied in computing education research can be found in Kinnunen and Simon [64].

In short, Phenomenography allows the researcher to identify all the understandings of and beliefs about a phenomenon and structure them in a hierarchical manner, where the lowest level in the hierarchy maps to the simplest understandings of the phenomenon and the highest level maps to the most complete understandings. The process we applied was derived from the process described in Carbone et al.'s work [28, §2.3] and follows the following steps. For brevity, researchers/paper authors are labeled A1-A5 (respectively: myself, Sophia, Alex, Bill and Leo).

Step 1: Individual Labeling

A1 applied labels to all interview transcriptions. A2 and A3 both labeled half of the transcriptions. Each person created their own label codes.

Step 2: Extracting Dimensions of Variation

Next, a list of summarized beliefs per research question was extracted by collectively analyzing all labeled excerpts from step 1. A "summarized belief" refers to a collection of labeled excerpts that can be viewed as all exhibiting the same belief. Each summarized belief was then mapped to each faculty who exhibited it. From this collection of summarized beliefs, we determined the dimensions of variation in our data.

Step 3: Constructing Categories of Description

The summarized beliefs and dimensions of variation from step 2 served as the basis for extracting the hierarchical categories of description. This was done collaboratively with the use of a whiteboard. This process also resulted in subcategories that make it more explicit which belief belongs where.

Step 4: Validation of Categories of Description

We validated the output of step 3 by mapping each summarized belief from step 2 to its corresponding subcategory. Whenever there was a disagreement about where a belief should be placed, the names and descriptions of the (sub)categories were clarified. After creating this hierarchy, A2 and A3 labeled a single participant interview and refined the subcategories based on their disagreements. Step 4 resulted in Figures 3.1 and 3.2. The categories of description are depicted as the dotted boxes, with the blue boxes being their subcategories.

Step 5: Extracting Quotes

The faculty quotes for this paper were extracted by back-tracing from the figures created in step 4, to the summarized beliefs from step 3, which gave us the faculty that exhibited these beliefs. The labels created in step 1 were then used to find the relevant sections in each interview.

3.5 Results

In this section we will discuss the outcomes of our phenomenographic analysis, as well as some other interesting findings that do not directly relate to our research questions. Wherever a participant is quoted, an anonymous identifier is provided (P1-P14).

3.5.1 RQ1: Goals of Undergraduate CS Education

The (sub)categories of description we identified during our analysis of faculty beliefs about the goals of an undergraduate CS education are displayed in Figure 3.1. The hierarchical relationship between the categories is that of dependency or prerequisite, i.e. it is not possible to be a competent problem solver without the required depth of knowledge and hard skills. However, while each subcategory depends on some subcategories in a lower category, it does not necessarily depend on all.

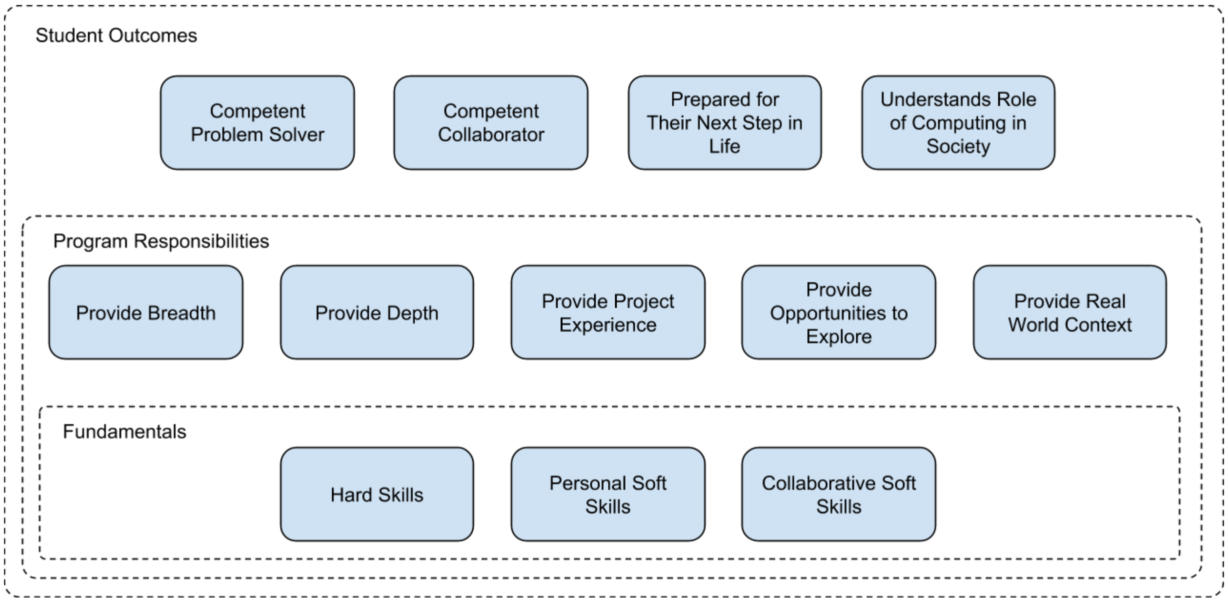

Student Outcomes
Competent Problem Solver
Competent Collaborator
Prepared for Their Next Step in Life
Understands Role of Computing in Society
Program Responsibilities
Provide Breadth
Provide Depth
Provide Project Experience
Provide Opportunities to Explore
Provide Real World Context
Fundamentals
Hard Skills
Personal Soft Skills
Collaborative Soft Skills

Figure 3.1: The Goals of Undergraduate CS Education: Categories of Description and Their Respective Subcategories.

Fundamentals

The simplest understanding of the goals of a CS education are the specific skills a student should attain. In this category, we differentiate between Hard Skills, Personal Soft Skills, and Collaborative Soft Skills.

Hard Skills. Examples include being a good programmer, having good software engineering skills, having a strong understanding of how a computer works, and being able to work with data.

Personal Soft Skills. Examples include having good time management skills or perseverance and dedication.

"... we need to get our students to do things they don't like ... to see through even when it's difficult ... perseverance and dedication." - P5

Collaborative Soft Skills. Examples include socializing and translating between human language, formal language and code.

"... be able to clearly communicate and translate between formal, precise language, and human oriented language." - P5

Program Responsibilities

A more sophisticated understanding of the goals of CS are the responsibilities of the CS program.

Provide Breadth. Faculty believe the program should allow students to develop a broad knowledge base as this will help students approach problems in a more sophisticated way.

Provide Depth. Furthermore, the program should allow students to develop some area of specialization within CS.

Provide Project Experience. Faculty believe providing project experience is essential for students to learn to collaborate.

Provide Opportunities to Explore. Faculty mentioned how every student is a unique individual with their own strengths, weaknesses, and interests. The program should offer opportunities that will allow for each of these unique individuals to excel in their own way and find out what interests them.

"People discover their potential at various points in life ... you can give them multiple paths ... so taking advanced courses, graduate courses, doing a project, doing independent study." - P1

Provide Real World Context. It is important to study computing in the context of the world. The program should teach students what types of problems are good to approach computationally, how computing has changed the world in the past, and how computing itself has evolved over time.

"... what does it mean to be a computational problem? What kinds of characteristics do these problems have in common?" - P14

Student Outcomes

We identified student outcomes as the most complete understanding of the goals of CS. This category applies when high level outcomes are mentioned, which will of course depend on the program, as well as the fundamentals.

Competent Problem Solver. Faculty mentioned that becoming a good problem solver was a major student outcome. Some even considered this to be a generic non-CS specific skill, that becoming a good problem solver is the goal of any undergraduate education.

Competent Collaborator. Often noted by faculty was the fact that outside of the classroom, people tend to solve problems in teams. So a major outcome of CS programs should be that students are able to contribute to such a collaborative environment.

Prepared for Their Next Step in Life. University education is seen by faculty as preparation for a person's next step in life. Hence, one major outcome of a CS education is that a student is reasonably sure what their next step will be, is well prepared for it, and able to gain the position they desire.

"... to prepare themselves for that next step, be it going into industry, starting their own company, going off to grad school ... the goal is to help students be successful in whatever that pursuit is." -

P9

Understands Role of Computing in Society. Some faculty were adamant about the need for students to be able to relate their work to society. Specifically, students should understand the impact their work might have on society, as well as its ethical implications.

3.5.2 RQ2: Preparing Students for Industry

The categories of description we identified during our analysis of faculty beliefs about the role of an undergraduate CS education in preparing students for industry are displayed in Figure 3.2. The quotes in this section are all responses to questions that asked specifically about students headed for industry.

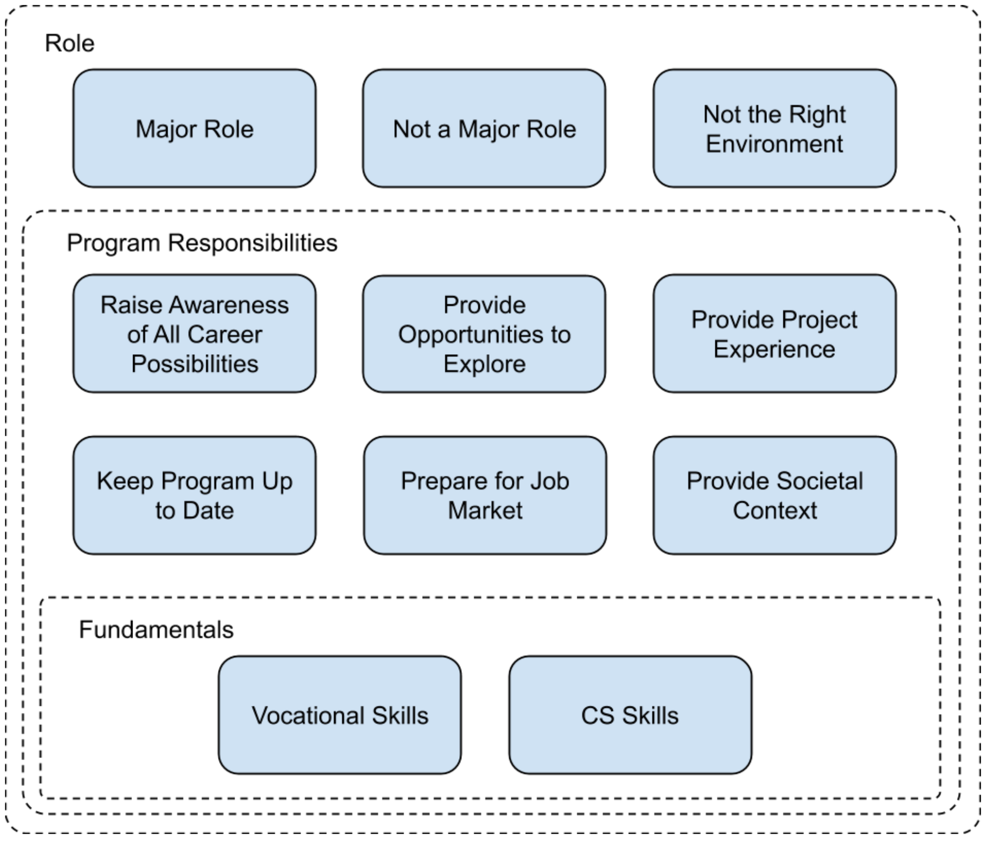


Figure 3.2: The Role of an Undergraduate CS Education in Preparing Students for Industry: Categories of Description.

Fundamentals

Being successful in industry requires a minimal set of fundamental skills. In this category, faculty mentioned some vocational skills as well as fundamental CS skills.

Vocational Skills. Vocational skills mentioned by faculty include being fluent in software engineering principles, familiar with the tools, and able to collaborate and communicate well.

CS Skills. An example of a CS skill that faculty believe is important for industry is the ability to learn a new technology or algorithm quickly by leveraging a strong fundamental understanding of CS.

Program Responsibilities

A CS program has specific responsibilities in preparing students for industry.

Raise Awareness of All Career Possibilities. Faculty believe that a CS program should raise awareness on industry career options and what knowledge is necessary for those careers.

Provide Opportunities to Explore. Along with raising awareness, there should be possibilities available to try things out. This can be an internship, co-op program, or working with a professor.

Provide Project Experience. There should be at least one significant project in the curriculum. Students should gain experience with large code bases that are being worked on by multiple people.

Keep Program Up to Date. Faculty say they have a responsibility to keep the program up to date and relevant. This includes listening to feedback from students returning from internships and alumni.

Prepare for Job Market. Faculty also state we should keep in mind that our students will have to enter the job market and interview with companies. Part of our program's responsibility is to ensure that our students will be able to pass interviews and find a job.

Provide Societal Context. CS is not an isolated field, it has a major role in society. Our programs should make students aware of the role they play in society and how their work can influence the lives of other people in both positive and negative ways.

"I think people should be aware when they take a job in industry, why that company is focusing on what they're focusing on, and that it's aligned to a problem that is a real problem, and that doesn't have sort of negative consequences for somebody, or if it does, that student goes in well aware of that. So I think helping students understand that it's not just about writing code ... but it's about evaluating the overall big picture as well." - P14

Role

The highest level of understanding on the role of CS education in preparing students for industry is how that role should manifest itself.

Major Role. Some faculty believe industry preparation should be a major role of a CS program, perhaps even the most important.

Not a Major Role. Other faculty believe industry preparation is a responsibility of the students themselves or even of the companies they will work for in the future.

Not the Right Environment. Another group of faculty believe it is simply unrealistic to expect universities to provide industry preparation, because they are not well equipped for it.

"... faculty are not suited for preparing them for the workplace. They never worked, and very few worked outside the campus ..." - P1

3.5.3 RQ3: Better Preparing Students

We found that when faculty mentioned what they believed to be goals of a CS degree or the role of a CS degree in preparing students for academia or industry, those same things were usually brought up when asked what CS degrees could do better. Below we list our most notable findings on what faculty believe CS programs can do to improve in the preparation for academia and industry.

Reduce Class Sizes. One of the most widely mentioned problems was that class sizes have become too big. Issues resulting from large classes mentioned were: not being able to go as deep into the material because too many students wouldn't be able to keep up, grade inflation, and the quality of group projects suffering. To increase the quality of undergraduate CS programs, faculty recommended reducing class sizes or providing more resources.

"I would love to [better prepare for ind.], but I don't know how because it would require way more resources, and smaller classes" - P14

Increase the Number of Projects. Another widely mentioned recommendation was to increase the number of project courses and make them more authentic, e.g. working with a real customer or solving real-world problems. Interestingly, some faculty advocated for more vaguely defined problems, whereas other faculty claimed that having more project courses with well-defined problems would provide a more authentic preparation for industry.

"In industry projects there are often times where the requirements are not well defined ..." - P12

"Less open-ended projects. My experience with industry is that projects are usually pretty well-defined, especially for junior folks." - P7

Some faculty also mentioned that it is difficult to create an authentic experience, even in a project class.

"Maybe what they're most lacking is things like real experience on large project teams, and I don't think team projects in undergraduate classes really give you a sense of what it's like to work on a larger ongoing project. Maybe there's no way they could." - P3

The User Perspective. In industry, software is always shipped to a user. However, most classes don't teach how that works.

"There's not a sense across the curriculum that we're always talking how to ... put something out there for people to use ... [e.g.] a desktop app that runs on different operating systems or ... a web app." - P2

More Research Opportunities. There is a call from faculty to scale up research opportunities for undergraduates by creating more programs that make it easy for faculty to work with them.

Software Engineering. Another concern faculty have is that students may not be well prepared for the level of rigor required of software engineers in industry.

"They need to have more software engineering skills 'cause the industry is a lot more rigorous about that stuff than academia is." - P7

Furthermore, students should be given more opportunities to familiarize themselves with important industry standard tools.

"... more experience with the tools that you would use in industry that are kind of standard. Like version control software, or different build systems." - P12

While faculty may be willing to address these deficiencies, it is difficult to implement these changes without increasing the workload on the students or faculty or cutting back on course materials.

"And then in terms of like code quality ... I can't find the time in my classes to really teach it or to really get them to do it because I'm putting so much technical challenge on them, like, "You're going to solve this really hard problem, and then solve this other really hard problem." And then I don't have the time to check up on whether they've really thought about all the different edge cases, or whether their code looks good ... So I think we'd need to cut way, way back on the amount of content we're throwing at them, and have them focus on some of these other either more open-ended issues, or communication issues, or impact issues, or quality issues." - P14

Side Projects. Faculty mentioned the value of students having side projects in being better prepared for industry.

"That's often a distinction between strong and weak students, if they really love programming and like doing it in their spare time." - P3

Communication. Many faculty mentioned communication and other soft skills are not well addressed in the curriculum.

Inspire Students. Faculty are concerned we may focus too much on teaching content and forget to make students excited about CS.

Societal Context. Similarly, we often forget to put CS in the context of society and think critically about our work.

Curriculum is Already Adequate. While most faculty see many ways to improve our degrees, some faculty mostly mentioned things that were going well such as students securing good jobs.

"My feeling is that we do a great job of preparation and if they're feeling unprepared, then maybe they didn't do all their work as an undergrad or didn't achieve very well or something of that sort." - P9

Everything Needs to Be Done Differently. On the other end of the spectrum, a few faculty believed the way we teach our CS programs is fundamentally flawed.

"in many classes, attendance is below 50%. Which means, is that our undergraduates are paying for the product, but are not taking it, which is very unusual when you are a customer and paying ... Which means that our education is inferior and has serious, serious problems ... My response would be to change educational system. To ... ban traditional classroom lectures." - P10

3.6 Discussion and Take-Aways

This section summarizes our key findings.

Industry preparation is important Contrary to our initial hypothesis that industry preparation might be a controversial programmatic goal, we found general agreement among the faculty in support of industry career preparation. Although Phenomenography does not offer quantitative conclusions, we note that the vast majority of our participants recognize that industry is the next step for most students and view preparing them for their industry careers as a key goal. Part of their motivation is simply to see their students succeed. However, as we discuss below, they also understand that CS plays a critical role in the making of the modern world and much of the impact students have on society comes from what they do after being hired by companies.

Two Curriculum Drivers: The Role of CS in the World and Student Choice Beyond the belief that learning CS fundamentals is an essential goal of CS education, we've identified two additional organizing principles.

The first, as stated above, is that faculty understand that CS plays a critical role in the making of the modern world. This perceived role of computing in society drives several beliefs. Faculty believe that they should prepare their students to be leaders who can make decisions with a regard for the social and ethical implications of their work. Thus, faculty believe their courses should include ways for students to practice with real-world implications. In turn, faculty value project courses, authentic assignments, and devoting more time to code quality, testing, and industry standard tools. Additionally, they understand that real-world problems tend to be solved by teams of people. As such, they understand that "soft" skills are important, even while acknowledging that they are under-supported in their curriculum.

The second organizing principle that faculty expressed is that every student is unique. Hence, beyond having a common base of CS fundamentals, students should be allowed to create their own path through the curriculum by means of opportunities, including conducting research with a professor, side projects, internships, electives, graduate courses, etc. Furthermore, faculty advocate for creating more of these types of opportunities for students.

The Academia-Industry Gap May be a Resource Gap The interviews reveal some possible explanations for why the academia-industry gap has persisted over the years in spite of the perceived importance of industry preparation and a significant body of research on the topic.

For one, faculty expressed concern over how the recent meteoric rise in enrollments has impacted the quality of education they are able to offer, specifically citing difficulties in achieving depth, grade inflation, and the difficulty of managing group projects. While previous enrollment booms have waned, the present one seems unlikely to subside. We would note that the gap has persisted even during periods of low enrollment, but the challenges related to enrollments may now be an additional barrier for change.

Second, a few faculty confided that they do not believe preparation for industry is a principal role of an undergraduate CS education. While this group is seemingly small, they no doubt have some influence on curriculum design.

Third, and perhaps most impactful, is the observation that, despite the fact that faculty cited interest in helping students prepare for careers in industry, many may not be sufficiently equipped to do so. Most of the faculty we interviewed never worked in industry and some told us to discount their views due to lack of experience: *"I've never worked in industry, so take that all with a grain of salt."* - P14. However, alumni boards and teaching assistants with industry experience were identified as possible resources for faculty to gain an industry perspective.

Finally, faculty cited struggling with finding ways to provide authentic experiences. One reason for this struggle is class size, but another is that it remains unclear for faculty how to make experiences such as projects and programming assignments more authentic without unacceptably increasing the workload of the students or the instructional team.

3.7 Limitations and Threats to Validity

This study is subject to the following threats and limitations. Participants were recruited through the personal networks of my advisors Leo Porter and Bill Griswold; this may have biased some participants' answers due to their relationships with my advisors. We believe this threat to be limited by the fact that Leo and Bill were never present during the interviews. On the other hand, we attempted to limit bias regarding our research questions by framing them about all goals, not just, say, industry preparation. Additionally, the participants were drawn from three institutions, however most participants were from the same research-intensive institution. Moreover, all three institutions are located in the same (US) metropolitan area. Both potentially limit the generalizability of the results and warrant replication to other institutions and regions. Finally, while Phenomenography identifies the range of understandings of a phenomenon, it, combined with the study's scale, did not clearly determine their prevalence. Since we identified conflicting beliefs—e.g., both opposition and support of industry as a major role of CS education—it will be valuable for future work to establish the relative support for such views.

3.8 Summary

We performed a phenomenographic analysis to identify the range of views faculty have on the role of undergraduate CS education, focused on student preparation for industry. We categorized these findings in this work while identifying multiple key themes relevant to remedying the gap between academia and industry: faculty generally want to prepare their students well for industry and want the program to address their career needs, however lack of instructional resources and, in some cases, industry experience are barriers to facilitating this preparation.

3.9 Acknowledgements

We thank the participating faculty for making time for us in their busy schedules. This work was supported in part by NSF award DUE-1712508.

Chapter 3, in full, is a reprint of the material as it appears in the Proceedings of the 51st ACM Technical Symposium on Computer Science Education (SIGCSE 2020). Sander Valstar, Sophia Krause-Levy, Alexandra Macedo, William G. Griswold, Leo Porter. “Faculty Views on the Goals of an Undergraduate CS Education and the Academia-Industry Gap”. The dissertation author was the first author of this paper.

Chapter 4

Investigating Relative Support for Differing Faculty Views

A Quantitative Study of Faculty Views on the Goals of an Undergraduate CS Program and Preparing Students for Industry

4.1 Introduction

Computer Science students and their ultimate employers have long lamented that their undergraduate education did not adequately prepare them for their career. Many stakeholders in the so-called *academia-industry gap* have been studied over the years, including students [29, 62], recent graduates [22, 32], employers and practitioners [51, 94]. Given that faculty are essential to closing any such gap, it is surprising that the faculty view on this topic had remained relatively unstudied.

In Chapter 3, we described an in-depth interview study of 14 computer science faculty to learn the range of faculty views on the goals of a CS program, where they felt their programs were falling short, and what was hindering improvement. One might think that faculty would claim "we are not a vocational school" in regards to questions about closing the academia-industry gap. Such views were indeed voiced by a small number of the interviewed faculty, but the other faculty desired to teach a curriculum relevant to industry preparation. These faculty cited a range of barriers to better preparing students for industry, including lack of knowledge of industry practices, lack of resources, and large class sizes.

Although this prior qualitative study uncovered a wide range of faculty views on these matters, including on the academia-industry gap, the relative support for those views in the broader community of CS faculty is unknown. Is the attitude of "we are not a vocational school" a major contributor to the academia-industry gap, or is the larger contributor these barriers to improvement cited by faculty? To learn the answers to these questions, we sought to quantify faculty views on the goals of a CS education, how best to prepare students for their academic or industry career, and what barriers they perceive inhibiting achievement of those goals.

Building on the results of our prior qualitative study described in Chapter 3, we conducted a broad survey study to establish which views are predominant among faculty, and how those views vary by institutional type and other factors. Specifically, we translated the range of faculty views captured in our prior qualitative work into multiple choice questions, and then sent them out as a survey to thousands of CS faculty across the globe, using mailing lists, contact information from Google Scholar, and scraping e-mail addresses from computer science department web sites (described further in Section 4.3). In analyzing the survey responses, we found the following:

- In terms of the goals of a CS education, we found the highest support for traditional CS goals such as students learning problem-solving, programming, and how to acquire new computing knowledge on their own.
- We found consistently high support for imparting skills and experiences related to industry

preparation. Right behind the above-cited traditional CS goals were understanding the real-world context of computing as well as soft skills, including time management and working in teams, ethics and the role of computing in society, and software engineering skills. Yet, we found split support for goals related to personal development, such as helping students explore new interests or discover their path in life. When asked about the relative importance of supporting industry preparation and academic (graduate school) preparation, faculty favored industry preparation. Faculty also felt that much of the responsibility for industry (and academia) preparation should fall on the shoulders of their institution.

- The majority of faculty expressed support for improving their courses by integrating material and experiences related to industry preparation into their courses, such as software engineering best practices and soft skills. This varied widely by the primary course taught by faculty, with faculty who teach courses related to algorithms being more split about the appropriateness of preparing students for industry in their course, whereas faculty who teach courses involving the construction of software rate industry preparation more highly.

- Faculty were split about how many and which barriers they encounter to offering students better industry preparation, with roughly half of participants perceiving only a few barriers and a quarter of participants perceiving substantial barriers.

With generally strong faculty support for industry preparation, and the split views on the difficulty of providing that preparation, there remains a puzzle about why the gap remains.

In the next section we review the literature on the academia-industry gap. We then describe our methods and the survey results, and finally close with a discussion of the results and a conclusion.

4.2 Background

4.2.1 The Academia-Industry Gap

The gap between recent CS graduates' knowledge and industry expectations has been a well-known problem for decades, yet it still exists. Industry practitioners have stated that many recent graduates lack relevant technical and non-technical skills, leading to failed interviews and requiring graduates to go through large amounts of additional training upon being hired. Recent graduates are known to lack the ability to test software in real world settings [90], lack relevant project experience [93, 94], have poor communication and team building skills [22, 93, 94] and lack the ability to configure and use software tools [93].

A study by Radermacher and Walia, interviewed 23 industry professionals to understand areas where recent graduates were underprepared. They found the most common issue to be struggling with software tools, followed by misunderstanding the expectations of the job. Another study interviewed recent graduates and found that there were six main areas where the graduates fell short, these included: "communication, team work, working on large, long-lasting projects that are open-scoped and using complex software systems" [32].

To gain a deeper understanding of new graduates' experiences in industry, Begel and Simon followed eight new graduates during their first six months in industry. They found that the graduates had many misconceptions about their new roles such as "I must do everything myself so that I look good to my manager." and "I must be the one to fix any bug I see – and I should fix it the 'right' way, even if I do not have time for it." [22].

In addition to technical skills and general misconceptions about industry, many students lack non-technical, "soft" skills such as oral and written communication, being able to work in a large group and understanding the ethics of their work [32, 94]. A study by Mardis et al. compared universities' syllabi to industry certifications, job postings and internship postings and found that many of the syllabi did not list any non-technical skills.

The motivation for remedying the gap is grounded in satisfying student career goals but also in our theoretical understanding of how students learn. Under Situated Learning, students learn better while working with others and they desire to be part of a community of professional practice [65]. Also relevant is Cognitive Apprenticeship where experts model behaviors that are meaningful to learners in a real-world context [31]. Learners then attempt to imitate those skills with coaching by the experts. Under Cognitive Apprenticeships, students want to learn from experts in their future discipline in a real-life context and this approach may enable students to achieve a high degree of learning [129].

4.2.2 Relationship Between Faculty and Undergraduate Beliefs

Some of the academia-industry gap—real or perceived—could be due to divergent views on CS education. Indeed, faculty and undergraduate attitudes and beliefs about computer science and computer science education do not always align. A study by Lewis et al. surveyed 13 faculty and 160 undergraduates on their beliefs and attitudes about a variety of CS-related topics, and then compared how student and faculty views differed. The survey questions were clustered into two topic areas: "Computer Science as Accomplishment" and "Computer Science as an Intellectual Discipline" [68, 69]. The survey results showed that while student and faculty views aligned in some cases, there were many areas of misalignment, including for students in their final year at the university. Several statements that were rejected by faculty and more accepted by students included "A significant problem in learning computer science is being able to memorize all the information I need to know", "Doing things the 'right' way is not as important as just pushing through to a solution", and "In the real world, computer scientists spend a lot of time working alone." These findings show that faculty and undergraduate students may not agree on important aspects of being a computer scientist, even when students are close to completing their degree. Such misconceptions could lead to related misconceptions as to what a computer science education should contain. On the other hand, such misconceptions could be influenced by how

CS courses are conducted (e.g., testing for memorization and having students work alone). As will be shown below, if courses are run this way, it could be due to challenges in instruction rather than faculty beliefs about the goals of a CS education. Lewis et al.'s findings have been replicated and further explored, focusing on areas where faculty and students disagree [88].

It has been suggested that faculty designing curricula should understand student motives and program expectations in order to improve students' learning and performance [136]. A study aimed at identifying motives among geography majors found that students desire their curricula to focus more on career guidance and vocational training [44]. This is similarly true in computer science [32]. In some cases, students may not understand the relevance of many CS courses for their careers [89]. A recent study reported that half of CS students believed their CS program did not sufficiently prepare them for their professional experiences [62]. On the other hand, a study interviewing faculty found that many faculty believe CS programs do sufficiently prepare students for jobs in industry, stating that the majority of their graduates are able to find high paying jobs. However, faculty also cited topics like software maintenance as being difficult to teach and "soft" skills as under-taught [30].

In Chapter 3, we described a study focused on faculty views on the goals of an undergraduate computer science education. We conducted semi-structured interviews with 14 faculty members from three different universities, including a large research-focused university, a large undergraduate-focused university, and a small liberal arts university. Study participants came from a variety of technical backgrounds such as Theory, Machine Learning, and Bioinformatics. Coding the interviews via phenomenography [11], a wide-range of views were uncovered on the goals of a CS education, their institution's role in achieving those goals, and the barriers encountered in pursuing those goals. Separately, we also observed that our 14 faculty interviewees generally, although not universally, viewed industry preparation as an important part of an undergraduate degree in CS, yet many felt that they did not have the resources to make their courses successful in that regard. Our present study attempts to put these observations and related questions on a quantitative footing, asking similar questions in an online survey sent to faculty worldwide.

4.2.3 Barriers to Change

An element uncovered in our interview study in Chapter 3 was that faculty face barriers to narrowing the academia-industry gap. There have been a number of studies investigating the challenges to changing faculty practices [56] and how best to address them (please see Henderson et al. for a summary of the literature [58]). Guidelines for fostering faculty change require identifying the barriers to adoption that faculty may face before attempting to create potential solutions. This allows solutions to account for those adoption challenges [63]. As such, this study seeks to identify which barriers are most commonly faced by faculty in order to inform future solutions.

4.3 Study Design

4.3.1 Research Questions

Based on the results of our prior qualitative study in Chapter 3, we formulated the following research questions.

RQ1 What are the views of the broad community of CS faculty on the goals of an undergraduate CS education?

RQ2 What are the views of the broad community of CS faculty on the role of an undergraduate CS education in preparing students for industry?

RQ3 What are the views of the broad community of CS faculty on better preparing students for careers in industry?

RQ4 What barriers to providing more industry-relevant course content are the broad community of CS faculty encountering?

4.3.2 Methods

To gain insight on the above research questions, we selected relevant questions and findings from our prior qualitative study in Chapter 3 and formulated them as survey questions. In order to assess whether the survey questions were being interpreted properly, we tested our survey on three graduate students in think-aloud sessions.

The final survey consisted of the following questions (a full version of the survey is posted online [2]):

1. What course/area do you primarily teach or identify with?
2. Do you teach at a public or private institution?
3. Which of the following terms best defines your institution?
4. In what country is your institution located?
5. How would you rate the size of your institution in terms of estimated total number of undergraduate students?
6. How would you rate the size of your CS program in terms of estimated number of undergraduate CS degrees awarded per year?
7. What type of term system does your institution use?
8. For each of the following items please list how important you believe this goal should be in your institution's undergraduate CS program.[1]
9. Please indicate your agreement with the following statement: Preparing students for the academic challenges of a Masters or PhD program should be a principal goal of an undergraduate CS education.

10. Please indicate your agreement with the following statement: Preparing students for careers in industry should be a principal goal of an undergraduate CS education.

11. Out of a total of 100%, what do you believe should be the balance of responsibilities for preparing students for the academic challenges of their Masters or PhD programs?

12. Out of a total of 100%, what do you believe should be the balance of responsibilities for preparing students for careers in industry?

13. Please indicate your agreement with the following statement: It would be beneficial for the quality of my institution's CS program if we investigated how we could better prepare our students for careers in industry.

14. Please list your agreement with the following statements. In order to better prepare students for their future careers than we are doing today at my institution, I believe we should...[1]

15. Please indicate your agreement with the following statements. The strongest barriers to me providing more directly industry-relevant content in my primary course are...[1]

16. What is your gender?

17. What is your age group?

18. Did one (or more) of your parents/caregivers receive a college or university diploma?

19. Do you identify with a group that is underrepresented in computer science programs in your country? (e.g. ethnicity)

[1]The full item list can be found online [2]

4.3.3 Dissemination of Survey

We employed a range of methods to maximize the reach of our survey. First, we posted our survey invitation to a few established CS community mailing lists, although we were unable to send to many mailing list as most do not allow the posting of surveys. Second, using automated methods, we scraped information from Google Scholar by searching for profiles with tags related to computer science. Finally, we similarly scraped the faculty listings of a wide selection of computer science department web sites. We obtained these department websites by searching on Google for the institution names as they appeared in the Times Higher Education Ranking [5] together with the words “computer science department”. In all, we emailed approximately 7250 faculty at 140 institutions in 35 countries.

An example of the email template used for participant recruitment can be found online [2].

4.4 Results

We received 325 responses in total, 249 of which are complete responses (77% completion rate). Unless stated otherwise, the results we report are based on the 249 complete surveys.

4.4.1 Demographics

Of the 249 respondents, 200 were men, 37 were women, 2 were non-binary, and 10 chose prefer not to answer. The age distribution of the respondents is shown in Figure 4.1. A total of 66 respondents were first-generation students and 40 respondents identified with an underrepresented group in CS, including gender, sexual orientation, ethnicity, disability, socioeconomic status and age.

As illustrated by Figure 4.2, the great majority of our respondents are from doctoral granting institutions. We also found that 70% of respondents are from public institutions and 30% work at private institutions. The great majority of respondents work with a semester system (74%), followed by the quarter system (13%), and the trimester system (10%). The remaining

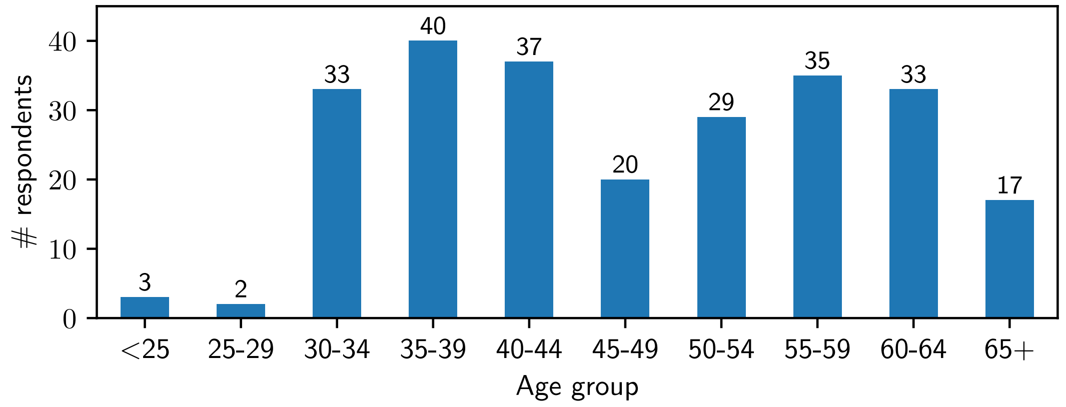


Figure 4.1: Q17: Age distribution of respondents.

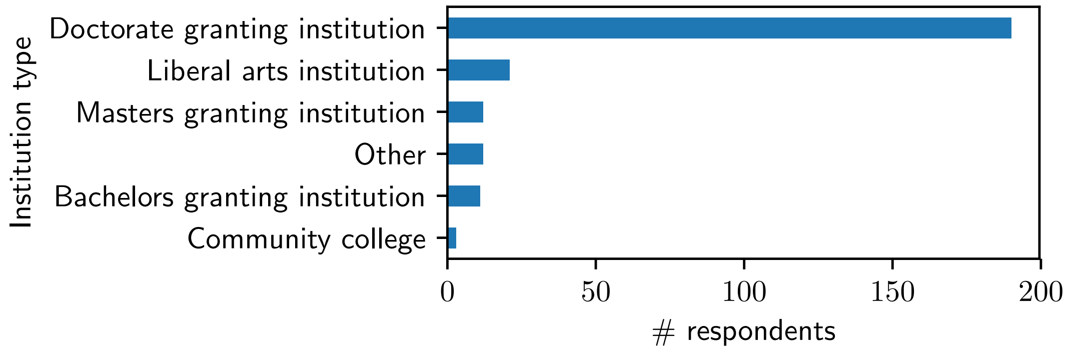


Figure 4.2: Q3: Institution type of respondents.

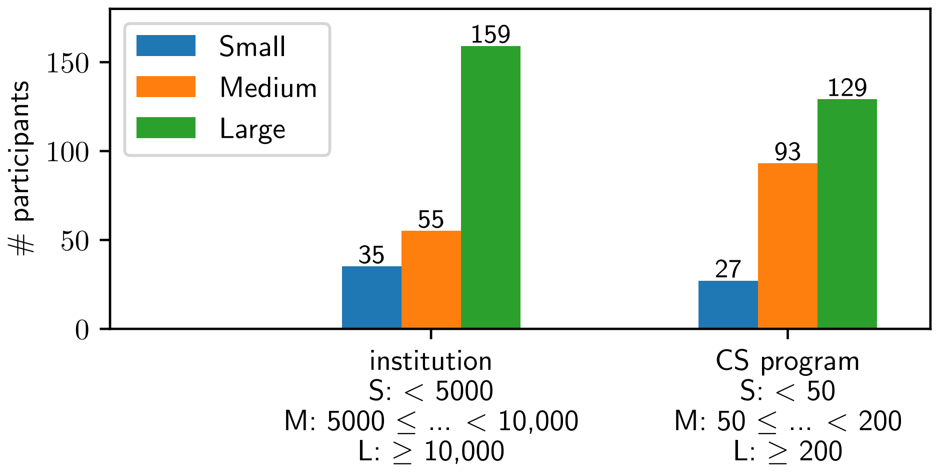
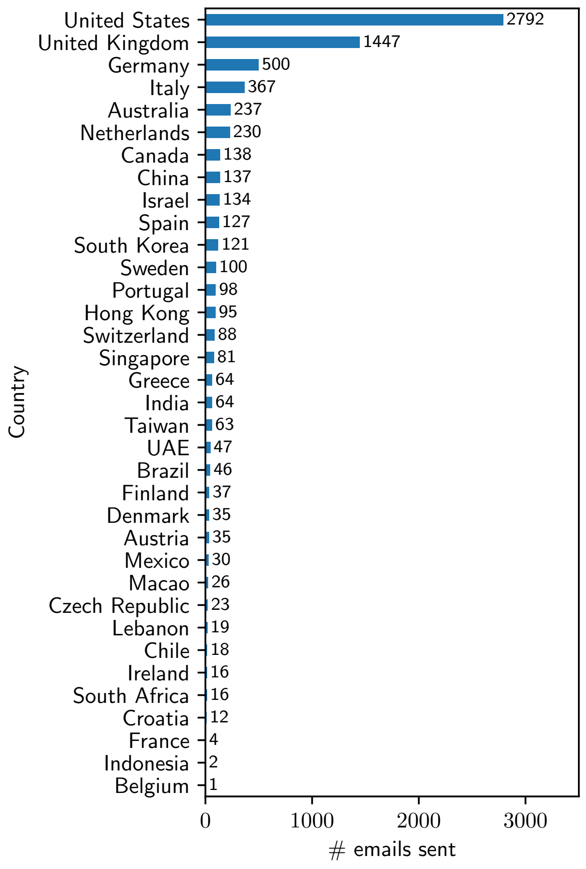
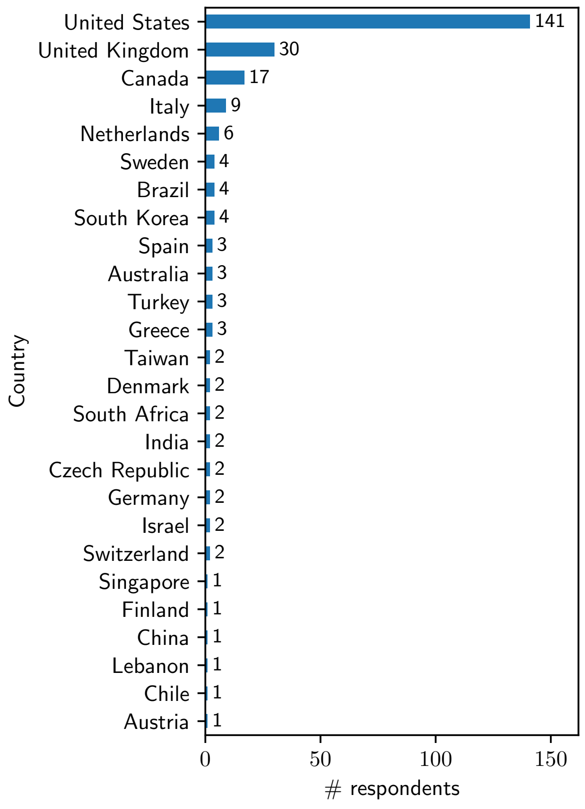


Figure 4.3: Q5-6: Institution and CS program sizes by number of undergraduates.



(a) The number of emails sent per country.



(b) Q4: Country distribution of respondents.

Figure 4.4: Emails sent and number of complete responses collected per country.

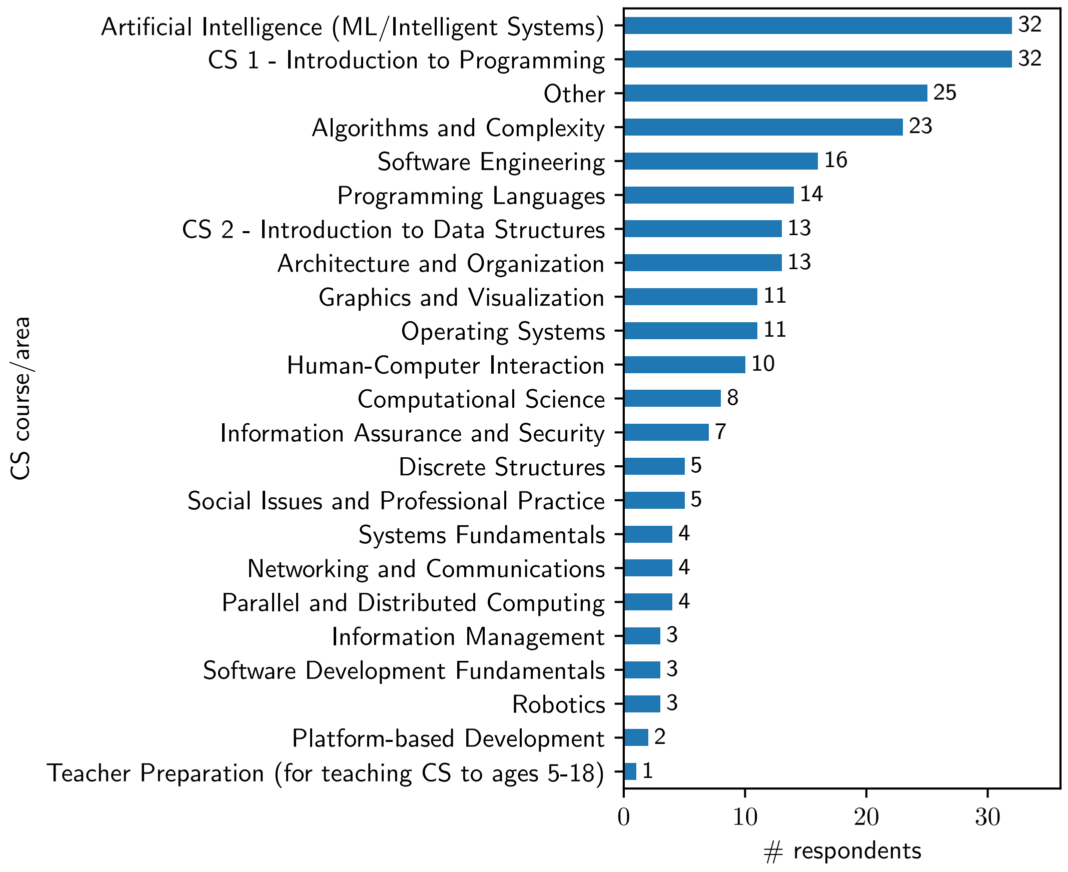


Figure 4.5: Q1: CS course/area with which faculty respondents primarily identify themselves. Courses and areas taken from the ACM curricula guidelines [61].

3% use other term systems. Figure 4.3 shows that most participants are from large schools with medium to large CS programs. Responses were collected from 26 countries in all.

As seen in Figure 4.4b, the majority of our respondents are from the United States (57%). Figure 4.4a shows that while did send out a larger number of emails to the United States than to any other country, it was not the majority of our emails (39%). However, the response rate for emails sent to the United States at 5% was higher than the average response rate of 3%.

Around 13% of our participants identified CS1 as their primary course. The full distribution of faculty respondents' primary courses/ areas can be found in Figure 4.5.

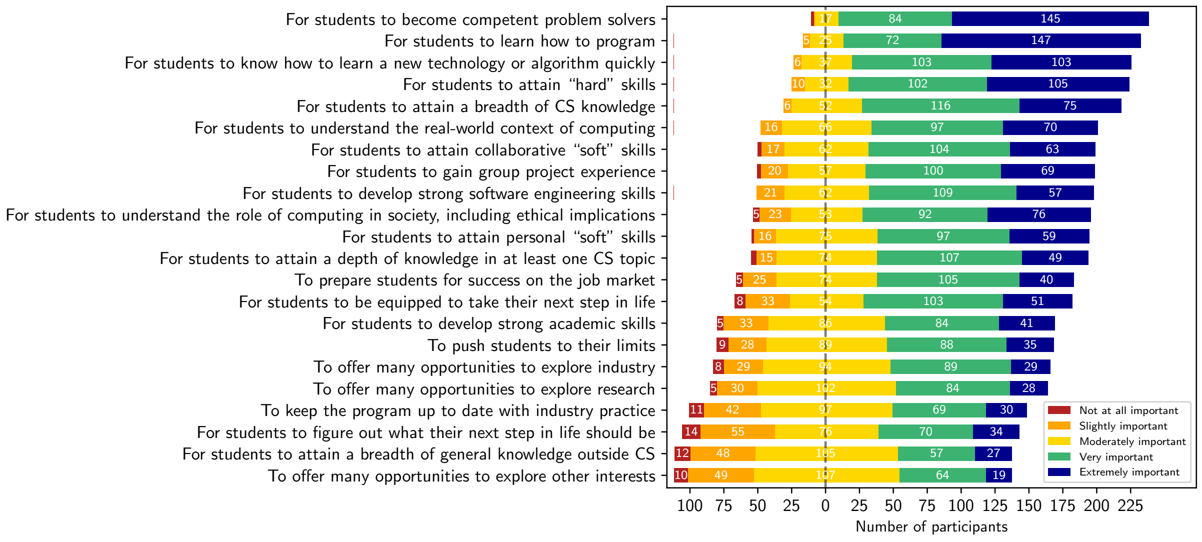


Figure 4.6: Q8: Relative support for goals of a CS program, "For each of the following items please list how important you believe this goal should be in your institution's undergraduate CS program."

4.4.2 Faculty Views on the Goals of CS

With the exception of just two people who voted "Not at all important" and one person who voted "Slightly important", faculty overwhelmingly support "For students to become competent problem solvers" as a goal of a CS program. Other strongly supported goals include "For students to learn how to program", "For students to know how to learn a new technology or algorithm quickly", "For students to attain 'hard' skills" and "For students to attain a breadth of CS knowledge". The vast majority of the listed goals received only a very small number of "Not at all important" votes, with "For students to figure out what their next step in life should be" as the item receiving the most dissidence. This high level of agreement is partially to be expected because the stated goals were taken from those expressed by faculty in our previous qualitative study in Chapter 3.

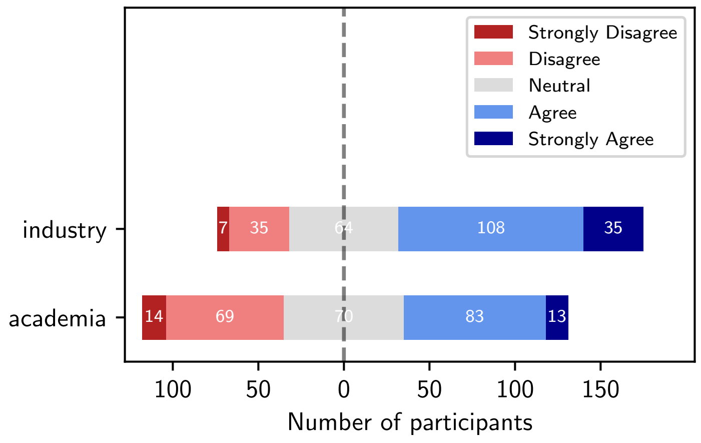


Figure 4.7: Q9-10: Relative support for industry preparation and academic preparation as a principal goal of a CS program. “Preparing students for [careers in industry/the academic challenges of a Masters or PhD program] should be a principal goal of an undergraduate CS education”

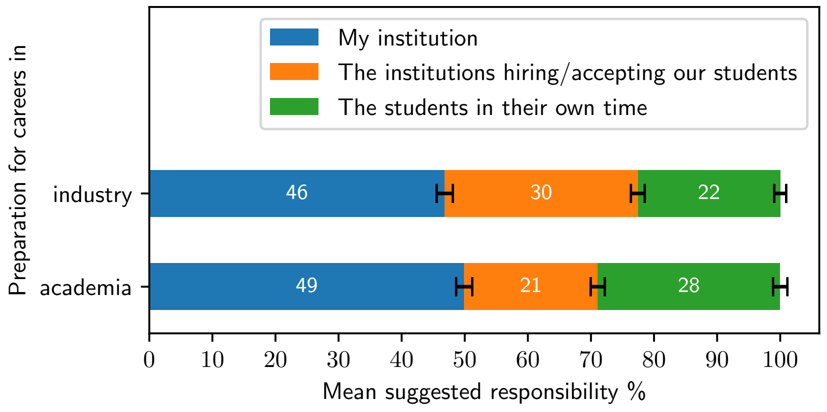


Figure 4.8: Q11-12: Balance of responsibilities in preparing students for careers in academia/industry according to faculty. “Out of a total of 100%, what do you believe should be the balance of responsibilities for preparing students for [the academic challenges of their Masters or PhD programs/careers in industry]?”

4.4.3 Views on Industry-Specific Program Goals

Figure 4.6 shows that, while there is a minority who disagree, most faculty support industry specific goals such as developing strong software engineering skills and preparing students for the job market. In an interesting contrast, faculty seem to be split on the importance of keeping the program up to date with industry practice. This could be due to faculty perceptions that software industry trends are not always driven by fundamental changes to practice, but rather by factors like cost, tool compatibility, or levels of scalability not relevant to the course context.

Figure 4.7 shows that faculty seem to have a stronger agreement that industry preparation is a principal goal of a CS program. The medians were “Agree” for industry preparation and “Neutral” for academic preparation. We ran a one-sided Mann-Whitney U test to determine whether or not the agreement was significantly greater for industry preparation. We found significance for $\alpha = 0.05$ with a Mann-Whitney U statistic of 38792.0 and a p-value of 1.935e-07.

More importantly, Figure 4.8 shows that for the preparation of students for careers in both industry and academia, faculty view their own institutions as the primary responsible party. This is encouraging as finding other results here would likely imply that faculty are not willing to close the academia-industry gap. Faculty do, however, place a significant portion (around 25%) of the responsibility on both the students themselves and the party hiring the students after graduation. For industry preparation, faculty expect the companies hiring the students to take on a larger part of the remaining responsibility whereas for preparation for academia, faculty expect the students to take on a larger part of the remaining responsibility.

We also explored the role that age has on faculty perceptions of the value of industry preparation. Our hypothesis was that older faculty may have entered CS when it was part of mathematics departments and hence more theory-heavy, perhaps making older faculty less open to teaching content relevant for industry preparation. As such, we grouped faculty by age, binning those under 50 years old as younger. Comparing medians led to the interesting finding that,

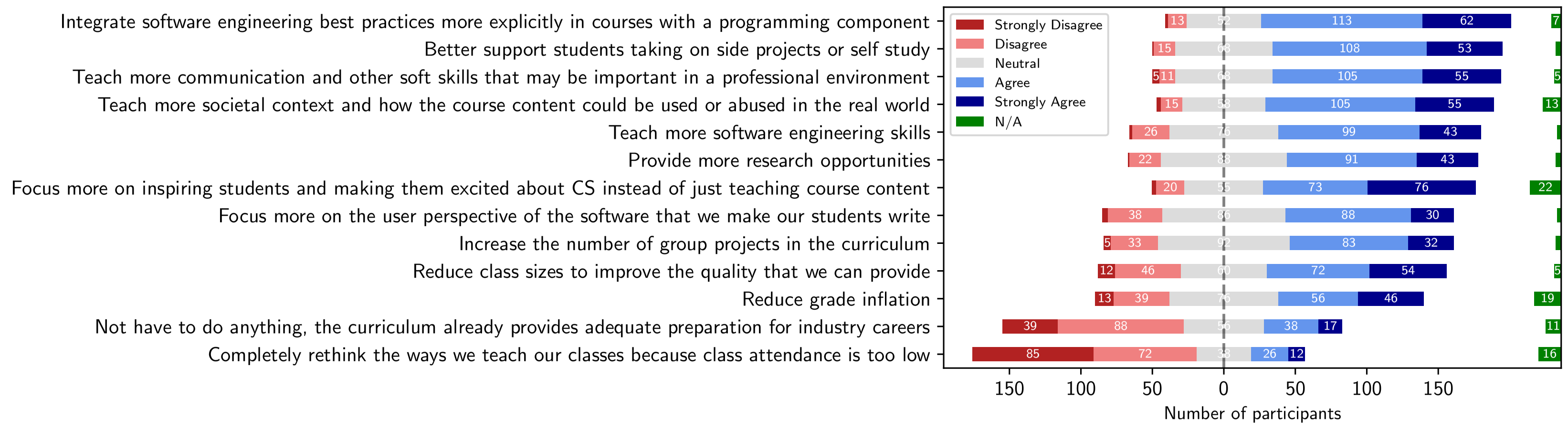


Figure 4.9: Q14: Faculty views on better preparing students for industry. "Please list your agreement with the following statements. In order to better prepare students for their future careers than we are doing today at my institution, I believe we should..."

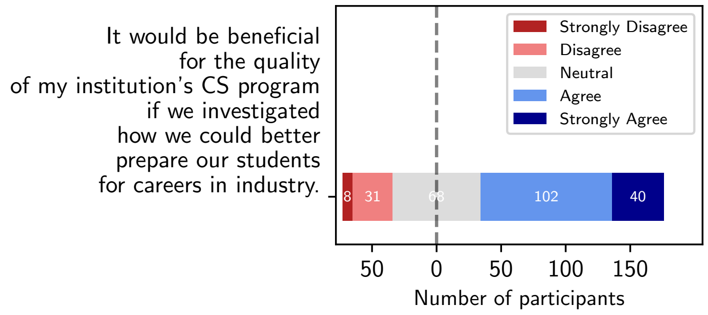


Figure 4.10: Q13: Relative support for investigating how to better prepare students for industry. "It would be beneficial for the quality of my institution's CS program if we investigated how we could better prepare our students for careers in industry."

contrary to our expectation, older faculty had a higher median rating for industry preparation as a principal goal ("Agree", vs. "Neutral" for early-career faculty). And indeed when running a one-sided Mann-Whitney U test to establish significance, we find that older faculty reported a significantly greater agreement with industry preparation as a principal goal for $\alpha = 0.05$ with a statistic of 6406.5 and a p-value of 0.008.

For preparation for academia as a principal goal, we find no significant differences between younger and older faculty. Both groups report a median of "Neutral". A two-sided (to identify any difference in either direction) Mann-Whitney U test led to a statistic of 7475.5 and a p-value of 0.343, indicating no significant differences for $\alpha = 0.05$.

4.4.4 What Should Be Improved?

The majority of faculty believe their curriculum should be improved with respect to preparing students for industry careers, as can be seen in Figure 4.9. This finding contradicts popular claims that "ivory tower" faculty attitudes would be responsible for perpetuating the academia-industry gap. The same graph also shows that most of the needs for improvements that surfaced from our prior interviews in Chapter 3 can count on broad support among faculty. The strongest support can be found for integrating software engineering best practices, such as software testing, more explicitly in courses with a programming component. Faculty appear to be split on the topics of reducing grade inflation and reducing class sizes. Overall, we see broad support for many ideas for improvement, but for each of them there is also a number of faculty who oppose this idea. In Figure 4.10, we also see broad support from faculty for investigating how their institution could better prepare students for industry careers.

4.4.5 Barriers to Improvement

Figure 4.11 shows that faculty appear to be quite split in terms of which barriers limit their ability to provide more industry-relevant content in their courses. Although there are perceived barriers, we see a continued trend in desiring to better prepare students for industry in the fact that the majority of faculty do not oppose including industry-relevant content in their courses. However, there is a sizeable group of faculty who agree or strongly agree that such content does not belong in their course. This difference in the number of faculty supporting better industry preparation from the number who support doing so in their own courses may be a factor in perpetuating the academic-industry gap. As such, we wanted to determine if faculty from certain courses or areas of CS are more likely than others to find that industry content does or does not belong in their course. Figure 4.12 explores participant agreement with this statement based on the course they most commonly teach. From this figure, it appears that instructors who teach CS1, Software Engineering or Human-Computer Interaction are the most supportive of teaching industry-relevant content in their course. In contrast, instructors who teach Algorithms and Complexity represent the largest group opposing industry-relevant content in their courses. Somewhat surprisingly though, we also find a sizeable portion of Programming Languages faculty believe such content does not belong in their course.

Since faculty appear to be split on whether large class size is a barrier and on the need to reduce class sizes to improve the quality of education, we decided to compare the results on these two survey items for faculty from small and large institutions.

We found that the median for “The class size is too large” as a barrier to improvement to

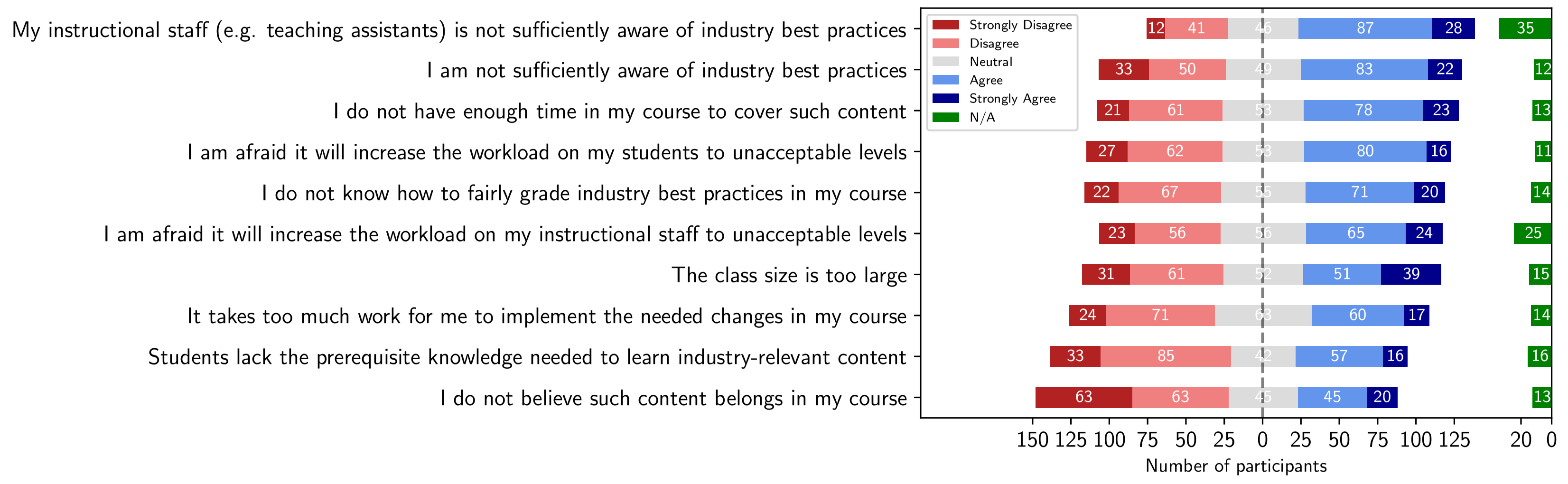


Figure 4.11: Q15: Barriers experienced by faculty. “Please indicate your agreement with the following statements. The strongest barriers to me providing more directly industry-relevant content in my primary course are”

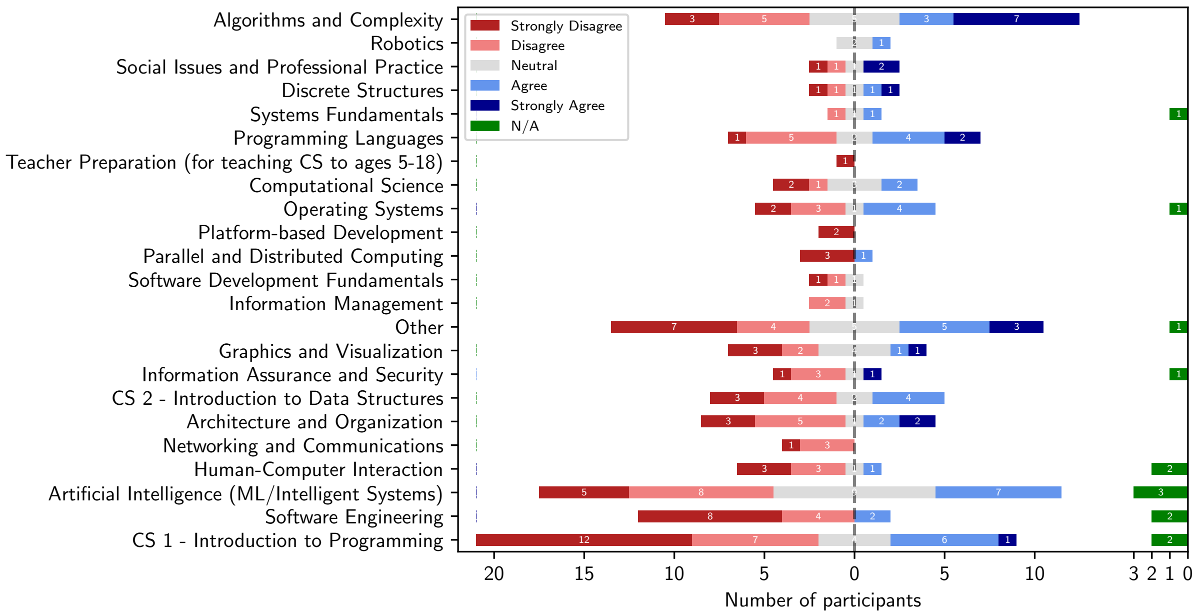


Figure 4.12: Q15: Participant level of agreement with "I do *not* believe such content belongs in my course" as a barrier to providing directly industry-relevant content in their course, plotted against the main CS course/area they identify with. Courses and areas taken from the ACM curricula guidelines [61].

be "Disagree" for faculty from small CS departments, whereas faculty from large CS departments responded with a median of "Neutral". For "Reduce class sizes to improve the quality that we can provide" as an improvement, the median was "Neutral" for faculty from small CS departments and "Agree" for faculty from large CS departments.

We followed up by running a one-sided Mann-Whitney U test to determine whether the responses for faculty from larger CS departments were statistically greater than responses from faculty from smaller CS departments for these two survey items. We found "The class size is too large" had a Mann-Whitney U statistic of 2369.0 and a p-value of 0.001, indicating that the response for faculty from large CS departments was significantly higher for $\alpha = 0.05$. For "Reduce class sizes to improve the quality that we can provide", we found a Mann-Whitney U statistic of 2349.0 and a p-value of 0.002, also indicating a significantly greater agreement for faculty from larger CS departments for $\alpha = 0.05$.

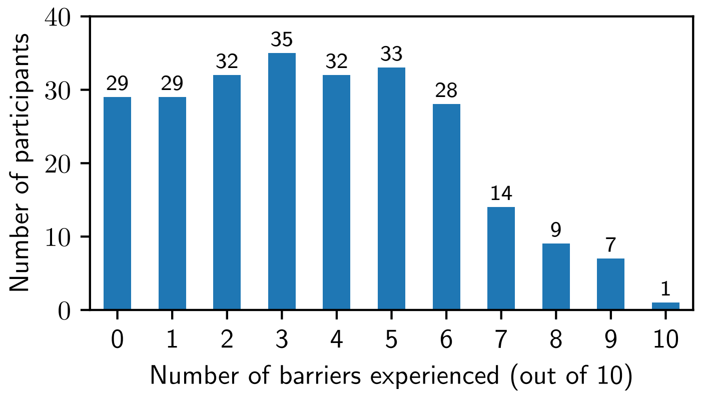


Figure 4.13: Q15: Number of barriers experienced per participant (counting number of "Agree" and "Strongly Agree" ratings)

Another interesting point to note is the large numbers of "N/A" responses for the questions regarding instructional staff in Figure 4.11. Specifically, for the item "My instructional staff (e.g. teaching assistants) is not sufficiently aware of industry best practices", 41% of the faculty from small CS departments responded with "N/A". We suspect the reason for this is that due to smaller class sizes they may not have instructional staff available to them.

Figure 4.13 shows that there is not a two group split between one group of faculty who experience all the barriers and another group of faculty who experience no barriers. While there is a sizable group of faculty who experience no barriers at all, the vast majority of faculty experience at least some barriers to providing more directly industry-relevant content in their courses.

4.5 Discussion

Concerning RQ1, we find that faculty demonstrate strong support for the following goals of an undergraduate CS education:

1. For students to become competent problem solvers
2. For students to learn how to program
3. For students to know how to learn a new technology or algorithm quickly

4. For students to attain "hard" skills

5. For students to attain a breadth of CS knowledge

Whereas faculty are in disagreement about the importance of the following goals:

1. To keep the program up to date with industry practice

2. For students to figure out what their next step in life should be

3. For students to attain a breadth of general knowledge outside CS

4. To offer many opportunities to explore other interests

Regarding RQ2, we find that 57% of faculty agree that industry preparation is a principal goal of an undergraduate CS education whereas only 17% of faculty disagree. The faculty opinion is more split on whether or not preparing students for academia is a principal goal (39% agree, 33% disagree). Moreover, faculty view their own institution as the primary responsible party for preparing students for both industry and academia.

For RQ3, we find that a majority of faculty believes it would be beneficial for the quality of their institution's CS program if they would investigate how they could better prepare students for careers in industry (57% agree, 16% disagree). We also find that many faculty expressed support for improving their courses by integrating materials and experiences related to industry preparation into their courses. Examples of such materials are software engineering best practices and soft skills. However support for such course changes varied widely by course area.

With respect to RQ4, we find that faculty were split regarding the number of barriers encountered as well as which barriers they encountered. Unfortunately this finding implies that the academia-industry gap is a multi-faceted problem that most likely cannot be solved by addressing a single barrier.

The remainder of this section explores these research questions in greater depth.

4.5.1 Dissenting Views

Although a majority of our respondents agree that preparing students for industry should be a primary goal of undergraduate CS education and that the quality of their institution's CS program would be improved by better preparing students for industry, there is a substantial group of faculty who disagree with these statements. These faculty may believe, as the quote attributed to a famous CS professor in Stroustrup [114]: "We don't teach programming; we teach computer science", that software engineering skills important for industry are different than the skills of a computer scientist. These beliefs are apparent in the distinction between software engineering [17] and computer science [97] degree programs. What matters most, perhaps, is that these disagreements about the goals of CS education exist. If institutions are working to improve student preparation for industry, these dissenting voices could slow progress and represent a new barrier not currently addressed in our survey.

4.5.2 Entrenched Beliefs?

One might wonder if those faculty who have been in the present system longer would be more resistant to change, in particular change towards aligning CS curricula with better industry preparation. This could potentially explain the persistence of the gap, i.e., younger faculty are eager to make changes, while entrenched interests resist that change, causing the younger faculty to lose momentum.

However, as discussed in Section 4.4.3 we found that older faculty were be more inclined to support industry preparation as a primary goal. Thus entrenched beliefs, if present, would appear to support the goal of industry preparation. Interestingly, Florian et al. found that employee age is negatively correlated with resistance to change [41]. This result would support the idea that older faculty, having greater support for industry preparation, would be more open to curriculum changes that improve industry preparation.

Why older faculty report higher agreement with industry preparation as an important programmatic goal remains unclear. However, we wonder if it might be that faculty learn more about the merits of an industry background throughout their interactions over the years with graduated students or with people returning to academia from industry. They may also be more likely to have older children who are making career choices; having a concern for their success could cause a broader pragmatic turn.

4.5.3 First-Generation Respondents

In contrast with previous work which finds first-generation students have a stronger job-focused mentality [66], we did not find any statistically significant difference for faculty who were first-generation students and their agreement on the importance of industry preparation as an important goal. We find that for the item "To prepare students for success on the job market" as a goal of CS programs, both faculty who were first-generation students and faculty who were not first-generation students report a median of "Very important". A two-sided Mann-Whitney U test reported a statistic of 5630.0 and a p-value of 0.426, indicating no significant difference for $\alpha = 0.05$.

Why there appears to be a difference in job-focused mentality between first-generation students and faculty who used to be first-generation students remains unclear. We wonder if perhaps faculty having tenured jobs could play a role here or if the subset of students who pursue faculty positions is different from the general population of first-generation students. Future work could investigate these questions.

4.5.4 CS Subdisciplines

We find some evidence that certain areas of CS may be more biased than others against industry preparation. As discussed in Section 4.4.5, we find that faculty identifying with the areas of Algorithms and Complexity as well as Programming Languages are more likely to believe that industry-relevant content does not belong in their course. However, we should keep in mind that a possible explanation for this is that it might be the case that academia is ahead of industry in these specific fields. For example, one participant who identifies with the Programming Languages area reached out to us over email after completing the survey. They stated that it would be counter-productive to cover directly industry-relevant content in their course because it would mean not covering the state of the art in their field. According to this participant, it was not necessarily the case that such content does not belong in their course, but more that they believed it would be counter-productive to inspiring students and preparing them for life-long learning. It is unclear how many of our respondents from these areas may share this perspective.

4.5.5 Barriers Encountered

As demonstrated in Section 4.4.5, faculty face barriers towards providing more directly industry-related content in their courses. Identifying common barriers is important as they inherently impact the likelihood that faculty will adopt possible solutions [58, 63]. We already see some evidence of this because the academia-industry gap persists despite the many papers that have been published on how to effectively teach industry-relevant content. For instance, there is a plethora of studies on how to teach software testing as well as on how to automatically grade it; some examples are [38, 101, 109]. Unfortunately, these practices are not yet commonplace at many institutions (among which our own) which suggests these barriers may be inhibiting adoption. Future work is needed to investigate why such studies have not gained traction and whether or not that may be related to the barriers discussed in Section 4.4.5.

Furthermore, we find in Section 4.4.5 that not all faculty experience the same barriers. Figure 4.13 demonstrates that we find about half of the faculty experience relatively few barriers while about a quarter experience substantial barriers. For example, we unsurprisingly find that faculty from schools with larger CS programs experience class size as a barrier to improvement. Similarly, for smaller CS programs, their faculty report experiencing fewer barriers pertaining their instructional staff as they may not have any. For most barriers, however, it remains unclear why certain faculty are experiencing these barriers while others are not. Future work is needed to uncover what separates these groups of faculty for each barrier such that the barrier can be addressed.

4.5.6 Research and Policy Implications

The results of this study have implications for researchers as well as administrators. For instance, we find substantial support for industry preparation among CS faculty, yet the academia-industry gap in CS remains. Furthermore, faculty cite experiencing substantial and diverse barriers that obstruct the way towards providing more industry-relevant content in their courses. We do not find one or two clear barriers that almost all faculty experience. Instead we find that each of the barriers in Figure 4.11 is experienced by a substantial number of faculty. These findings indicate that the academia-industry gap is a multi-faceted issue and that more work is needed in understanding and overcoming these barriers.

4.6 Limits and Risks to Validity

As a closed-ended survey study, the range of responses to questions was limited to the responses previously elicited in our open-ended interviews in Chapter 3. Although there may have been other responses that respondents may have wished to choose, we do note that our questions achieved a sufficiently wide range of responses to give us confidence that the responses to our survey represented real alternatives and choices for the respondents.

Our survey results are subject to potential selection bias. Participation in our survey was voluntary, without remuneration, so there could be a bias in our results due to the type of person who answers such surveys (e.g., respondents may be more community-minded than the average faculty member). We attempted to counter other selection biases by using multiple methods to target respondents and building department mailing lists from a wide variety of institutions.

Related to the above, although we solicited responses from thousands of faculty, we achieved a response rate of only 3%, a relatively low proportion, although typical for survey studies. For example, Sheehan reports the following factors are strongly correlated to email survey response rates: Survey Length, Respondent Pre-Notification, Follow-Up Contact/Reminders, and Relevance of the Issue to the People Approached [103]. Our survey does not do well on these factors, except for the relevance of the issue to the people we approached. We could of course have sent pre-notifications or reminders, however we did not want to bother people with more than one unsolicited email about our survey. Our completion rate of 77% gives us confidence that the survey design itself was not deterring people from completing it.

We used statistical methods where appropriate in order to test for confidence, and were clear on when we did and did not achieve sufficient confidence.

Furthermore, although we solicited responses from 35 countries, a large portion of our participants are from universities from the US and other English speaking countries. This may limit the generalizability of our results.

As a survey study, our results reflect what respondents consciously chose as answers in the survey, not necessarily what they actually believe or enact in their daily lives. While an observational study (e.g., of department faculty meetings or the conduct of courses) might reveal actual beliefs or tendencies, it would be extraordinarily difficult to achieve the quantitative results required to lend insight on the research questions of this study. On the other hand, because survey responses were anonymous, there was no incentive for our respondents to intentionally mislead.

4.7 Summary

Although the academia-industry gap has been long-studied, little was known about the perspectives of the broad community of CS faculty. In our survey study of 249 faculty, we find that the community is strongly supportive of industry preparation as a goal of their undergraduate CS programs. Moreover, they generally view their own institution as the primary party responsible for industry preparation. Faculty are also strongly supportive of teaching more industry-related content, for example by integrating more software engineering practices into existing courses and by teaching more soft skills that may be important in a professional context. That said, this support varies by the faculty's primary course area, with instructors of software-related courses offering more support and instructors of more mathematical courses offering less support.

The enthusiasm for the goal of industry preparation is tempered by an acknowledgment that there are many barriers to success. Although faculty varied widely in the number of barriers they cited as preventing their inclusion of more industry-related content, over half encountered more than three barriers. There were no discernible patterns in the number of barriers according to institution or program characteristics.

With such strong faculty support for industry preparation, the common barriers uncovered in this work may offer a partial answer to the question of why the gap persists. A clear research priority should then be learning more about the characteristics of these barriers and how we might begin to remedy them.

4.8 Acknowledgements

We thank the participating faculty for making time in their busy schedules to fill out our survey as well as the reviewers and the faculty who responded to our email for their thoughtful suggestions. This work was supported in part by NSF award DUE-1712508. This material is based upon work supported by the National Science Foundation Graduate Research Fellowship Program under Grant No. DGE-1650112. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

Chapter 4, in full, is a reprint of the material as it appears in the Proceedings of the 2020 ACM Conference on International Computing Education Research (ICER 2020). Sander Valstar, Caroline Sih, Sophia Krause-Levy, Leo Porter, William G. Griswold. "A Quantitative Study of Faculty Views on the Goals of an Undergraduate CS Program and Preparing Students for Industry". The dissertation author was the first author of this paper.

Chapter 5

Barriers to Adoption

Analyzing the Adoptability of Educational Innovations: A Case Study of the Academia-Industry Gap in CS

5.1 Introduction

Based on struggles with gaining faculty adoption of evidence-based educational practices [57], extensive work to identify the common barriers to adoption of educational innovations has been conducted within the past decade [63, 112, 118, 125]. As a prime barrier is lack of time for adoption [60], educational innovations must either be extremely lightweight or innovators must provide extensive support.

The group "Increase the Impact" seeks to improve the rate at which faculty adopt pedagogical innovations by providing recommendations that educational innovators can employ to make adoption more likely [123]. These recommendations have been only recently developed (within the past decade) and require additional effort from innovators—particularly by asking them to prioritize meeting the needs of potential adopters from the start.

We believe these previous works can provide insight into why certain innovations have failed to gain traction in our community. To that end, we developed a checklist that helps the user identify potential barriers to adoption present in a CS educational innovation paper.

The use of checklist guidelines is not new to computer science. McGill et al. [77] provided guidelines on what data to report in future papers focusing on experience reports in pre-college computing activities. The need for consistent reported information allows for better replication studies, [54] analysis and comparisons of these initiatives. These frameworks have also been used to evaluate the current literature of the CS education community, as seen in the work of Malmi et al. [71], allowing for better understanding of how research is carried out along with what key areas of the research process need improvement.

We based the contents of this checklist on Increase the Impact's formal framework for analyzing the potential for sustained adoption [111, 123], the work of Taylor et al. on propagating the adoption of CS educational innovations [118], the barriers experienced by faculty as identified in Chapters 3 and 4, and our own experience as educators. Some examples of items on the checklist are: whether or not materials such as a topic schedule, assigned readings and assignments are publicly available and whether or not the innovation requires paid products or services. The full checklist can be found in Appendix A.2.

As part of the checklist development process and to explore its value in a relevant context, we used our checklist to evaluate the adoptability of proposed innovations that seek to solve the academia-industry gap. The academia-industry gap has been well studied over the past two decades, with numerous studies uncovering the divide between what is taught in academia and what is important to CS graduates for their careers [93]. While this academia-industry gap was identified at least two decades ago [67], recent work has shown that it still persists today [22, 32, 62]. Although one might say this is because faculty do not care about remedying the problem, in Chapter 4 we actually identified that a majority of CS faculty view industry preparation as a major goal of CS degrees.

To demonstrate the presence of solutions to this problem, we provide an overview of the dimensions of the academia-industry gap in CS (Section 5.4.1) as well as an overview of the solutions that have been proposed (Section 5.4.2). Were these solutions widely adopted, we believe we would have seen a narrowing of the academia-industry gap. However, because papers on the gap have continued to be published in recent years [32, 62], we suspect these listed solutions have not yet made it into standard practice in CS curricula.

The status quo of the academia-industry gap can thus be summarized as follows:

1. The academia-industry gap has been a recognized issue in the field of CS education for at least 2 decades [67].

2. Many faculty would like to better prepare students for industry, but lack awareness, skills and resources necessary to provide more industry-relevant content in their courses (see Chapter 4).

3. There are many solutions proposed in academic literature waiting to be used in practice (see Section 5.4.2).

4. The gap continues to persist today [22, 32, 62], thus there appears to be little adoption of these solutions.

Hence, for the case study in this work we hypothesize that published solutions to the academia-industry gap contain substantial barriers to adoption. To assess our hypothesis we use our checklist to analyze these innovations. We find the most common areas of improvement are: (1) examining the scalability of the proposed innovation and examining the innovation beyond a single institution, (2) providing concrete teaching materials that can directly be used by a potential adopter, and (3) providing innovations that do not require expertise beyond what can be reasonably expected of someone who holds a CS degree.

We hope that this case study on the importance of minimizing potential barriers to adoption helps both with identifying next steps for innovations that seek to close the academia-industry gap and with underscoring the importance of minimizing these barriers during the development and publication of CS educational innovations.

5.2 Related Work

5.2.1 Faculty Adoption of Pedagogical Practices

Stanford et al. found that most published educational innovations are not developed with a clear plan for propagation and gaining adoption [112].

Al-Zubidy et al. conducted a review of empirical evaluations in papers presented at a large CS Education conference and found that there was a lack of replication which may be to due a number of reasons including “papers do not report methodologies clearly enough to allow for replication”, “faculty are unaware of other similar studies from which they could build their work upon” and “faculty are not able to recruit others to use their methodologies” [12].

In a similar scope, Sanders et al. conducted a review of inferential statistics used in computing education research by reviewing 270 papers from a large CS Education conference and found that many of the papers that do use inferential statistics do not provide enough information to understand the statistical tests and for the purpose of being able to replicate the studies. Surprisingly only 51% of the papers even using inferential statistics and 28% of the papers used no numbers to describe their data. This provides further evidence that mnay papers do not provide a clear path for replication and therefore adoption of the conducted research.

This is worrying as Khatri et al. found that devoting attention to adoption from the start is important for developing a successful innovation [63].

A 2018 literature review assessed what motivates and discourages educators to adopt educational innovations, particularly in CS [118]. Their work lists the following barriers to the adoption of educational innovations:

1. Unawareness of the innovation

2. The time and effort required to implement the innovation
3. The skills required to implement the innovation
4. Incompatibility of the innovation with existing practice
5. The worry that the innovation will take so much time that not all course topics can be covered anymore
6. The worry that the innovation will have a negative impact on student evaluations of the course and its instructor
7. The potentially limited amount of time a research track faculty devotes to teaching and updating their teaching methods
8. Unconscious habits and (potentially irrational) beliefs that influence teaching decisions

In a survey of 821 CS faculty, Hovey et al. identified "not having time to try it out" as the most important barrier to adoption of educational innovations. With other strong barriers being satisfaction with the current practice, lacking the necessary resources, and being unfamiliar with the innovation [60]. These findings exemplify the importance of minimizing the time and resources required to adopt an innovation.

Turpen et al. found that educators who have chosen not to adopt an innovation perceive different barriers to adoption than educators who claim to have adopted the innovation [125]. Unfortunately, self-reporting has been found to be an unreliable way of measuring the adoption rate of an educational innovation as many faculty will claim to have adopted an innovation, but will in fact not adhere to all of its implementation details [110]. This indicates that innovations should come with clear implementation guidelines in terms of what is "required" and what is "okay to modify" [118]. To address this issue, Walter et al. designed a validated survey which minimizes this self-reporting bias when measuring faculty use of an innovation [131]. Other work has found that the fewer implementation components an innovation has, the more likely faculty are to adopt it properly [24].

5.2.2 The Academia-Industry Gap

In this work, we use our checklist to run a case study on a problem that has been well studied by our community over the last two decades: the academia-industry gap. Notably, Radermacher and Walia analyzed 30 papers on this gap between industry expectations and the abilities of CS graduates [93]. In a follow up study, they interviewed 23 managers and hiring personnel from a variety of companies and found that the most frequent problem companies experience with recent graduates is lacking proficiency in- or awareness of different software tools [94]. The managers mentioned that it takes about six months for recent graduates to reach the same degree of proficiency with the tooling as other employees. One example mentioned is that recent graduates are unfamiliar with version control beyond its basic usage, i.e. they know how to commit and push code, but have no idea how to deal with things like merging and branching. Also, most have never been exposed to core topics, such as continuous integration, that are essential to quality assurance in a professional setting.

Craig et al. interviewed recent CS graduates and performed an analysis of five studies on the academia-industry gap [32]. They extracted a number of differences between academia and industry that lie at the basis of this gap. For example, the well-defined fixed scopes of student projects versus the large continually evolving scopes of industry projects and the difference in scale of codebases for student projects compared to industry projects.

The academia-industry gap is not unique to CS education, some examples from other fields include: Hospitality [18], Law [37], Business [78], Medicine [23], and Accounting [104]. Also noteworthy is that there are a number of papers identifying an academia-industry gap in Software Engineering (SE) programs and specializations [13, 20, 43, 83, 126], indicating that simply offering a separate SE program or specialization for students seeking to become software engineers does not necessarily solve the academia-industry gap in CS. Fortunately, many innovative solutions have been proposed to address parts of the academia-industry gap in CS. The case study we run in this work analyzes these innovations for the presence of potential barriers to adoption.

5.3 Methods: Developing the Checklist

For educational innovations to gain "Sustained Adoption", they need to address three challenges:

1. Be Effective. Any published innovation should show evidence, either qualitative or quantitative, that it is effective in achieving its goals. Based on the fact that these papers were peer reviewed and published we expect these papers address this challenge. We assume that not showing sufficient evidence would provide a reviewer with strong reasons to reject a paper submission.

2. Gain Faculty Awareness. In order for faculty to begin considering adoption of any educational innovation, the innovator first needs to make sure faculty are aware of the problem. However, assessing faculty awareness is challenging to do in our case study because much of the dissemination and outreach efforts for a proposed innovation occurs outside of what one might find in a publication. As such, our study does not examine this element of "Sustained Adoption", but we revisit this topic in Section 7.5.

3. Be Easy to Adopt. As the most commonly cited reason faculty choose to not adopt educational innovations is lack of time [60], it is critical that solutions are well described, well tested, and are easy to obtain and deploy (preferably at scale). How well solutions address this challenge can be answered by analyzing the potential barriers to adoption that arise from information provided in the papers. As such, the remainder of this section details how we developed an instrument to formally analyze the potential barriers to adoption that may be present in a CS educational innovation.

5.3.1 Developing a Checklist to Identify Barriers to Adoption

A group called “Increase the Impact” [123] has already published a formal framework for assessing the “Sustained Adoption” of an educational innovation: “Design for Sustained Adoption Assessment Instrument (DSAAI)” [111, 122]. However, as mentioned above in point 3, ease of adoption (or “Adoptability”) is only one aspect of “Sustained Adoption”. Where Adoptability only concerns whether it is easy to adopt a solution, “Sustained Adoption” also concerns developing a strategy to achieve high adoption of an innovation and sustaining that adoption into the future. This means the DSAAI includes items on propagation and increasing faculty awareness, however these items are difficult to assess without intimate knowledge of the work (beyond what is available in a publication). Hence, our checklist does not address this issue and instead focuses on identifying the potential barriers to adoption of a solution in order to provide insight in the Adoptability of the solution.

Our final checklist is based on Increase the Impact’s DSAAI instrument [122], the barriers to adoption of innovations identified by Taylor et al. [118], the barriers faculty experience as identified in Chapters 3 and 4, and our own experience as educators. The full checklist is appended to this article in Appendix A.2. Moreover, Table 5.1 lists abbreviations of all items on the checklist. Please note that for some items on the checklist, not checking the item may indicate a potential barrier (e.g. items under 3 and 9).

Items on the checklist were heavily based on the Sustained Adoption Assessment with some additional changes [111, 122]. First, items that we could not realistically and accurately rate on from simply reading a paper were removed. This included items under the ”Dissemination through:” category in Section 3 of the instrument, such as methods and venues the dissemination of an innovation could be done through. Considering we only looked at the papers in isolation, we could not determine if seminars, conference booths or other methods of dissemination occurred for an intervention. Next, items were added to the checklist based upon prior work done on analyzing the effectiveness of innovation propagation. An example of an addition made is the specification of scalability issues and specific sample details and setting (i.e. size of sample, institutions, country) the innovation is reported on. This was done to gain a more detailed understanding of what aspect of the innovation may be difficult to scale up, such as high amounts of manual grading, class size or physical working space for students. Finally, in order to increase the inter-rater reliability, Likert scales were changed to check boxes and some items were rephrased, split or combined.

5.3.2 Checklist Reliability Validation

We desired to make an instrument that would be easy to use by authors of innovations, potential adopters, and paper reviewers, as well as provide consistent results. We tried multiple approaches, such as 5 point and 3 point Likert scales, but ultimately arrived a checklist similar to the Teaching Practices Inventory as that proved to be more reliable than using scales [135]. The first three authors iteratively created the checklist using the following process:

1. The first author picked two new papers to rate.
2. Each of the first three authors then independently analyzed these two papers using the latest version of the checklist while using a spreadsheet to keep track of the items checked as well as any corresponding notes.
3. The three authors then discussed the differences in their analyses in a meeting where they updated the checklist.

This process was repeated until an average pairwise agreement of 87% and an average pairwise Cohen's Kappa of 0.72 was achieved across the three raters for both papers in the seventh iteration, indicating substantial inter-rater reliability for the final checklist.

5.4 Methods: The Case Study

We first determined the different dimensions of the academia-industry gap. Next, we determined whether solutions exist for those dimensions. And finally, we selected a subset of all identified solutions and analyzed those with the checklist.

5.4.1 Determining the Dimensions of the Academia-Industry Gap

To classify the various dimensions of the academia-industry gap, we use a literature review of papers describing the academia-industry gap by Radermacher and Walia. In their work, they analyzed 30 papers and found 31 knowledge deficiencies [93]. Of these 31 deficiencies, they report the 15 most frequent.

We amended this list of deficiencies with several clarifications and additions where we felt the deficiencies on the list were not complete or well defined enough based on our own literature review. They are differentiated below by italic text and added citations.

Written Communication: Technical writing and documentation.

Oral Communication: Communicating with customers, listening skills, communicating about struggles and the need for assistance.

Project Management: General project management skills.

Software Tools: Tools for version control, code development (IDEs), project management, issue tracking, code analysis, database management, testing, debugging, and continuous integration.

Testing: Writing high quality test cases, making use of test quality metrics such as code coverage. *Understanding the importance of testing* [51]. *Not giving up when writing tests is difficult* [22].

Teamwork: Working in large teams, working with experts from other disciplines, *working in the same codebase with multiple teams* [25].

Problem Solving & Critical Thinking: Analyzing problems. Being able to come up with alternative solutions to a problem. *Critiquing and justifying a solution* [51].

Programming: General programming ability, multi-threaded programming, language independence.

Requirements *Engineering*: Elicitation and specification.

Personal Skills: Motivation, attitude, disposition, face and learn from errors, accept feedback, *time management* [32, 116].

Ethics: Being able to distinguish between ethical and unethical behavior. Acting ethically.

Leadership: General leadership skills.

***Software* Design:** Describing and producing software designs.

Databases: Query languages. Database design.

Development & Improvement Processes: *Familiarity with development processes such as code reviews and shipping cycles* [116]. Awareness of process standards such as the Capability Maturity Model (CMM) and process improvement frameworks such as Capability Maturity Model Integration (CMMI).

5.4.2 Process Used to Find Solutions

Given the dimensions of the problem from Section 5.4.1, we next sought out solutions in academic articles for the identified gaps. We formalized our search for solutions with the following process. We used the ACM digital library and Google Scholar to find published studies. The search range used was 2000–2019 as we suspect teaching methods published before 2000 may be less relevant to the way we teach CS today. Our queries took the form of: “Teaching ¡gap¿ in CS” (e.g., “Teaching ethics in CS”). Because the ACM digital library contains papers focused on Computer Science, we found that querying it led to more directly relevant results than Google Scholar. So we used it as our primary query platform, only moving to Google Scholar whenever we did not find relevant results. However, we found that searching the ACM digital library through Google was faster, so we used the following strategy to search on Google: *“site:dl.acm.org Teaching $\ll$ gap $\gg$ in CS”*.

If the high level gap name did not render any results, we searched for more specific descriptions of the gap. For instance, *"Teaching Software Tools in CS"* did not render any relevant results, so we instead searched for: *'site:dl.acm.org teaching "version control" OR "IDE" OR "project management tools" OR "issue tracking" OR "debugging" OR "continuous integration" in CS'*

We found a total of 78 relevant papers [4] proposing solutions to the 15 gaps listed in Section 5.4.1. We do not claim that this is an exhaustive lists of all solutions, but rather proof of existence of solutions to each of the academia-industry gap's dimensions.

5.4.3 Selection and Analysis of Solutions

For each of the 15 gaps we selected the 3 papers with the highest number of citations per year as papers to analyze in our case study. The following formula was used to calculate the number of citations per year: $\#citations/max(1,\ 2019-year)$. We used the number of citations reported by Google as the numbers were consistently higher than those in the ACM library. We used this metric as a proxy for papers with the highest impact and possibly greatest likelihood for adoption. Moreover, by selecting exactly three papers from each gap dimension we avoid potentially biasing our results towards dimensions with relatively many proposed solutions. Next, the 3 papers per gap were divided up among the first 3 authors of the paper such that each author analyzed exactly 1 paper for each of the 15 areas of the academia-industry gap. The final list of the 45 solution papers analyzed for barriers to adoption can be found in Appendix A.1. The complete list of all 78 identified solution papers can be found online [4].

5.5 Results

Table 5.1 shows the results of the analysis of these 45 papers. For each item on the checklist, one can see for how many of the 45 solutions this item was checked. Please note that for some items a potential barrier may be implicated when the item is *not* checked (e.g. items under 3 and 9).

Table 5.1: The results of applying the checklist to analyze 45 papers proposing innovative solutions to aspects of the academia-industry gap. The count indicates the number of papers for which a checklist item applied, the percentage indicates the count relative to the total of 45 papers.

Checklist Item	Count	%
1. Project type		
Curricular Change	9	20%
Course Change	18	40%
New Course	14	31%
Technology	4	9%
Other	1	2%
2. Components mentioned in the paper		
Course topic schedule	19	42%
Lectures	22	49%
Readings	18	40%
In-class activities	29	64%
Videos	4	9%
Individual assignments	36	80%
Group projects	21	47%
Labs	10	22%
Required software	20	44%
Other: ________	2	4%
3. Materials/examples publicly available		
Course topic schedule	11	24%
Lecture materials	0	0%
Reading materials	11	24%
In-class activity materials	9	20%
Video materials	1	2%
Assignment materials	9	20%
Project materials	2	4%
Lab materials	5	11%
Grading strategies	6	13%
All required software	19	42%
Software manuals/documentation	19	42%
Other: ________	2	4%
4. Expertise requirements		
Author/collaborator has expertise in __	21	47%
Author provides info to help learn	12	27%
5. Monetary requirements		
Products/services author or students paid	11	24%
Other: ________	0	0%
6. Hardware requirements		
Raspberry Pi, Arduino or similarly priced	0	0%
Generic web servers	0	0%
Self-designed or self-assembled hardware	0	0%
Other hardware: __________	3	7%
7. Teaching assistant requirements		
Paper mentions __ TAs per __ students	5	11%
Author's TAs have expertise in: ________	1	2%
8. Author collaborated with		
a prof in their own department	14	31%
a prof from the _______ department	3	7%
________ at their university	2	4%
another university	4	9%
industry	9	20%
the open source community	2	4%
other: ________	3	7%
9. Paper reports innovation was used		
On a max class size of _____ students	22	49%
On a min class size of _____ students	22	49%
In multiple offerings of the same course	22	49%
In multiple different courses	9	20%
By multiple instructors	11	24%
At multiple institutions	5	11%
In multiple countries	1	2%
10. Paper reports statistical evidence		
That shows innovation is beneficial	20	44%
11. Potential scalability issues		
Class size NOT mentioned in paper	20	44%
Paper does NOT report any usage on a large class	43	96%
Requires manual grading	30	67%
Requires significant reading for grading	24	53%
Requires presentations	12	27%
Requires in-class student interaction	25	56%
Requires all students to work at the same time	20	44%
Requires students to have employment	1	2%
Requires interaction with customer/product owner	6	13%
Not all students can participate (limited spots)	4	9%

We find that 14 out of the 45 solutions are of the “New Course” type. Proposing new courses can be problematic, because the CS curriculum is already full with courses. Thus the adopter will have to figure out what course to remove. Consequently, proposing a new course may not be particularly helpful unless it is clear which course it should replace. Course changes and curricular changes may not require the removal of an entire course from the curriculum, however there is likely still some content that should be considered for removal in favor of the change.

For many course components we find that, even though the components are explicitly mentioned in the paper, actual materials or detailed examples are missing, leaving any potential adopter with the burden to recreate these materials from scratch. When papers described, or linked, course materials these generally included topics schedules, readings, labs, and required software. The course components for which often no materials were provided were lectures, in-class activities, group-projects, individual assignments and grading strategies.

For close to half of the papers, the author of the paper or a collaborator had special expertise that a potential adopter (who just has a CS degree) could not be expected to have. An example of this comes in the work of Fioravanti et al. [39]. The innovation presented in this paper requires the adopter to have experience with industry project management. Although industry is a popular choice for many computer scientists, it is possible (and in our own experience quite common) for instructors to have a pure educational background with minimal to no industry experience. Another example is that of the innovation presented by Burton et al. [26]. This innovation requires the adopter to have knowledge on science fiction literature and computer ethics. Although computer ethics is an important aspect of the field, it is typically not a requirement of computer science degree programs. For about half of these papers, the author seems to realize this and provides resources to help a potential adopter learn the topic. For the other half, potential adopters are seemingly expected to either already have this expertise or to be able to attain it independently. Moreover, papers often did not mention how many teaching assistants were used and if these teaching assistants had any special expertise. This makes it hard for potential adopters to judge whether or not the innovation would for instance require more teaching assistants than is supported by their budget. About a third of papers require some form of collaboration with other professors, industry professionals, or other outside entities. This may form a barrier for potential adopters from smaller institutions or for adopters who are not as well connected as the author.

The 22 papers that reported maximum class sizes had an average class size of 70.45 students (median 42.5 and standard deviation 64.03). In total only 2 papers were evaluated on a class size larger than 200, and 5 papers on a class size between 100 and 200. The 22 reported minimum class sizes had an average of 62.32 (median 31.5 and standard deviation 69.02). Some papers mentioned aggregate class sizes across multiple course instances, hence the class size related items under 9 and 11 on the checklist do not add up to 45.

About half of the papers reported the solution had been used in multiple offerings of the same course, giving confidence that these innovations gained some traction at the author's university. However, few papers report usage in multiple courses, by multiple instructors, at multiple institutions, or in multiple countries.

Perhaps somewhat surprising is that for 25 out of 45 solutions, the published paper does not provide any statistical evidence that the proposed innovation is beneficial. This contradicts our original expectation that papers would certainly provide such information, as we believed that either the lack of or minimal evidence would be clear reason for a paper rejection. This is worrying as it may mean that the innovation does not have the desired effects. Moreover, it may make it harder for potential adopters to convince their colleagues that it is worthwhile to adopt an innovation.

One of the things that stands out in Table 5.1 are the scalability issues that apply to many of the solutions. For example, 43 papers did not report that the solution had been evaluated on a large class. Moreover, 20 papers did not mention class size at all. This is particularly concerning because if a solution has not been used on a large class, it is likely that it may hide some scalability issues that an adopter has to uncover and correct (likely making them less willing to adopt it). Moreover, as class sizes continue to rise in computer science, scalable solutions become a necessity [27]. Another point of worry is grading at scale—30 solutions require manual grading and 24 solutions require a significant amount of reading for grading (e.g. when grading essays or project reports), which is difficult to do in large classes. Only 9 solutions provide explicit grading strategies.

There were other issues related to scalability as well. We found many solutions require some form of in-class student interaction. Depending on the type of in-class activity, this may not work well in large classes, especially if student participation in the activity is graded. Just under a quarter of solutions include some form of student presentations. This is also a potential scalability issue as student presentations can consume considerable class time. That said, one innovation had an interesting approach: the author proposed having students present lecture content to practice presenting, that way there is no need to scrap course content in order to make time for student presentations [45]. However, there may be quality concerns if core course content is presented by students. Requiring all students to work at the same time of day may also pose a scalability issue, especially if students are expected to work in a computer lab, because there may not be enough workstations to fit a large class. Only few solutions required employment, interaction with actual customers, or access to a student program with limited spots available. This is encouraging as innovations depending on such things are inherently difficult to scale.

5.6 Discussion

Our checklist can be applied to CS educational innovation papers by authors, reviewers and adopters alike in order to uncover any potential barriers to adoption and inform the decision making process on how to move forward with a paper or project. For example, before submitting a paper for publication, the author(s) of an innovation might analyze their work using our checklist and find that adding a link to their publicly posted lecture slides would be beneficial for potential adopters.

5.6.1 Gaining Adoption

As discussed before, minimizing barriers to adoption of an educational innovation is by no means the only aspect that is important for successful sustained adoption. For instance, one of the main barriers to adoption as listed in Section 5.2.1 is "Unawareness of the innovation" which is tied to the key challenge of gaining faculty awareness of the solution. In addition, Chapters 3 and 4 have shown that faculty may not always have knowledge of current industry practices, so proposed solutions should help to close that divide. It is also important to note that simply because barriers to adoption are lowered does not mean an innovation will be universally adopted. As discussed by Hovey et al., faculty are also motivated by additional factors such as how the innovation may benefit their students' learning and how compatible it is with their current teaching practices [60].

Thus, before an author of an innovation arrives at the point where they might use our checklist to analyze their work for potential barriers to adoption, they should ideally already have followed some of the best practices for sustained adoption. As only reducing barriers to adoption may for instance still lead to an innovation that other people are not interested in.

5.6.2 Many Papers Contain Substantial Barriers

Judging by our findings in Section 5.5, it is clear that many educational innovations proposed in our community, specifically on topics related to the academia-industry gap, pose substantial barriers to adoption. For example, many papers do not provide concrete teaching materials, did not evaluate the innovation on large class sizes and may contain several potential scalability issues. With the prime reason for not adopting an innovation being a lack of time [60], this is worrying information as regenerating course materials or making innovations suitable for larger classes could place a heavy time burden on the adopter.

5.6.3 Developments in the CSE Community

With our work, we hope to see more attention paid to adoptability, particularly for research papers and experience reports submitted to SIGCSE conferences. Fortunately, we believe our community has already began to recognize the importance of adoptability in educational innovations as new criteria relevant to adoption were added just two years ago to the SIGCSE symposium reviewer guidelines. For example, according to the reviewer guidelines for the SIGCSE 2021 Technical Symposium [105], papers in the "Experience Reports and Tools Paper Track" should "provide enough detail so that others could adopt the new innovation". This is a welcome addition and we laud the changes to the reviewer guidelines. We hope this paper will lead to further investigations into how these guidelines may be improved with respect to helping reviewers identify potential barriers to adoption that may be present in submitted papers.

5.7 Limitations and Threats

This study has several limitations and threats to validity. For example, we created an instrument that can be used by authors, reviewers and adopters to analyze CS educational innovation papers and we have provided evidence that the instrument performs reliably between different raters. However, a minor threat is that we ourselves were not an author, reviewer or adopter of the innovations we analyzed. Moreover, it is unknown how well the instrument performs for people who were not involved with this work.

Since each paper is somewhat unique in its own way it is near impossible to eliminate all chance for ambiguity from our checklist, especially in cases where the paper itself is ambiguous. For example, we found that differences between raters during our validation iterations were often caused by a rater misinterpreting parts of a paper. In addition, as our knowledge of the solutions in our case study is limited to what is presented in the papers, we may not be aware of all efforts made by authors to achieve sustained adoption.

We also wish to note that research on designing for sustained adoption [122, 123] is relatively new to the CSE community, thus designers of the solutions we analyzed were likely unaware of its recommendations. However, many items on our checklist do not require a thorough focus on sustained adoption. For example, virtually any innovation should be able to make relevant course materials publicly available.

Despite these limitations we believe our checklist is beneficial to authors, reviewers and adopters of CS educational innovations. Moreover, our case study clearly uncovers a lack of focus on minimizing barriers to adoption, which may detract from the closing of the academia-industry gap.

5.8 Summary

The goal of this study is to uncover the barriers to adoption that may be present in educational innovations. Our checklist may assist authors, reviewers and adopters alike in analyzing papers for the presence of potential barriers to adoption. This in turn can help identify missing information to improve the potential adoptability of a promising educational innovation. The checklist then, can help guide discussion whether to submit, accept or adopt an innovation or if more focus on reducing barriers is desired.

Further more, we used our checklist to run a case study on a well studied problem in our community: the academia-industry gap. We examined whether the presence of barriers to adoption in CS educational innovations addressing this gap could be an explanation for its persistence. The results of this case study clearly show that many potential barriers to adoption are often present in these papers. Although we do not claim this to be the key reason why this issue has persisted, these barriers may be increasing the difficulty when trying to implement these innovations at another institution. This leads us to conclude that the presence of barriers to adoption in the proposed solutions is indeed one reason the academia-industry gap has been able to persist for two decades despite the many publications on the topic.

Hence we encourage the CSE community to reiterate on these solutions by addressing the potential barriers to adoption that arise from applying our checklist. Table 5.1 provides an overview of which potential barriers to adoption are commonly present. Key areas we believe are relatively easy to improve include providing ready-to-use course materials and making sure the solution does not contain aspects that will hinder its scalability in larger classes. On the other hand, checklist items such as evaluation at multiple institutions may be too costly to address in initial publications.

Finally, we want to recognize the valuable recent addition of adoptability-related questions to the review criteria for SIGCSE conferences and encourage the CSE community to continue these efforts by devoting more attention to uncovering potential barriers to adoption in the review process going forward. Future community conversations should determine how we can raise the bar regarding adoptability of submissions to SIGCSE conferences.

5.9 Acknowledgements

Chapter 5, in full, is being prepared for publication and may appear in ACM Transactions on Computing Education in 2021. Sander Valstar, Adrian Salguero, Sophia Krause-Levy, William G. Griswold, Leo Porter. “Analyzing the Adoptability of Educational Innovations: A Case Study of the Academia-Industry Gap in CS”. The dissertation author was the first author of this paper.

Chapter 6

Prerequisite Proficiency and Student Performance

A Study on The Relationship between Prerequisite Proficiency and Student Performance in an Upper-Division Computing Course

6.1 Introduction

Although prerequisites have been shown to be valuable for student outcomes outside computing [74, 128], their value has not been empirically studied in computing. Studying the importance of prerequisite success takes on greater meaning in light of institutional pressure to reduce failure rates, as well as recent discussions about reducing prerequisites to speed time to degree [130].

Many instructors have a story about the student who somehow enrolled in their course without a required prerequisite, suffering dire consequences. But what happens when a student has taken the prerequisite and just did poorly in that class—does low prerequisite knowledge relate to poor performance, and if so, to what extent? What percentage of students enter courses with insufficient proficiency of required prerequisites? Do students improve on their deficiencies in prerequisite knowledge during the course?

To answer these questions, we measured the prerequisite knowledge of 208 students in an upper-division data structures class through a pre- and post- survey. From these results, we examined changes in proficiency over the term, correlated pre-term and post-term proficiency with the students' final exam scores, and analyzed which prerequisite content was most salient for student success.

We expected that proficiency with prerequisite knowledge would be generally high as the prerequisites for this course are reported to be rigorous and demanding. Yet we also had concerns that the very low rates of D's and F's ($< 7\%$ *combined*) in the prerequisites could mean that some instructors were giving passing grades to students who are not truly prepared for the next step. Other possible factors include fading knowledge for those not taking these courses back-to-back [19] and cheating in prerequisite courses [82, 106].

The results were both sobering and encouraging. Prerequisite proficiency on entry to the course was far lower than anticipated. We classified student prerequisite proficiency as low (under half of the questions correct), medium (50-70%), or high (over 70%). Nearly a third of students (29%) demonstrated low proficiency and only 27% demonstrated high proficiency. On the other hand, students improved their proficiency over the term, by 8 percentage points on average, with a number of low-proficiency students improving to at least medium proficiency by the time of the final.

Not entirely surprising is our finding that final exam performance is correlated with prerequisite proficiency, 0.366 (Pearson correlation coefficient) for the pre-term test, 0.384 for the post-term test, and 0.434 for the two taken together. Improvement in prerequisite proficiency during the term mattered: for students demonstrating low proficiency on the pre-term prerequisite test, their improvement on the post-term test was correlated (0.371) to their performance on the final. All of these correlations were statistically significant. Not all prerequisite knowledge was found to be equally important, however, nor was proficiency uniform over the subject matter. For example, a student's ability to perform a basic runtime analysis was the strongest predictor of performance on the final. We discuss which topics were of most importance in detail.

Taken together, these results show the importance of proficiency with prerequisite knowledge for future performance in computing, and highlight the possible benefits of putting more attention on the reinforcement of prerequisite knowledge.

6.2 Background

Course prerequisites are commonly used in curricula to ensure students have the necessary prerequisite knowledge to succeed in subsequent courses. In computing, course prerequisites are described in the ACM curricula recommendations CS 2013 [85]. Prerequisites can play a variety of roles in the curriculum, including ensuring students have the proper background knowledge or sufficient maturity, that they display a certain degree of commitment to a program, or to convey program requirements [130]. In particular, the role of mathematics and introductory computing prerequisites have been well discussed [130]. There is pressure, however, to reduce the depth of prerequisite chains to aid with students' time to degree [130]. In view of these discussions, the role prerequisite proficiency plays in student success in computing courses deserves further study.

Outside computing, Martin found that grades in prerequisite courses, specifically statistics and mathematics, correlated with student performance in an agriculture price analysis course [74]. Expanding on the work of Martin in agriculture, Vitale et al. found that cumulative GPA and grades in prerequisite courses predict student grades in a Farm and Agribusiness Management course [128].

In computing, direct connections between prerequisite course grades or knowledge from prerequisite classes has not been similarly established. However, there has been substantial work examining a myriad of factors that determine students success in introductory programming courses. These factors include prior knowledge, gender, motivation, self-efficacy, and achievement goals [70, 95, 137, 138]. Although each of these factors have shown promise, Wilson et al. recently cautioned that these relationships may not always replicate [132], which has led to large replication efforts [139]. Our work expands on our understanding in computing by establishing a direct relationship between prerequisite course knowledge and success in upper-division computing courses.

The study most closely related to ours is that of Muralidharan et al. [81]. Muralidharan et al. examined student performance in Hindi and mathematics with 8th grade students in India, finding that a surprising majority of students are lacking prerequisite knowledge from prior grade levels. Their study goes on to show that an intervention program targeting prerequisite knowledge can significantly improve outcomes. In our study, we similarly examine student prerequisite knowledge entering an Advanced Data Structures course, finding that incoming students demonstrate a wide range of proficiency and that improvement in prerequisite proficiency during the course is correlated with improved final scores.

6.3 Research Questions

Our study has five primary research questions:

RQ1. Are most students arriving with sufficient prerequisite knowledge to start subsequent courses?

RQ2. Do students, particularly those with low preparedness, improve their proficiency in prerequisite content as they progress through subsequent courses?

RQ3. Does incoming prerequisite proficiency impact final exam scores?

RQ4. If there is a relationship between incoming prerequisite proficiency and final exam scores, which prerequisites are most meaningful?

RQ5. Does improvement in prerequisite proficiency translate to improved final exam scores?

6.4 Study Design

To understand the influence of prerequisite proficiency on final exam performance we chose to study an instance of the course "Advanced Data Structures" (ADS), an early upper-division course in our CS program. This is an excellent candidate for our study because it has a wide range of prerequisites including software, theory, and hardware. As we were interested in actual proficiency at the time of taking the course, not just performance in the prerequisite courses, we tested proficiency at the beginning and the end of the term. Our study has institutional Human Subjects approval.

6.4.1 Course Context

The ADS course we studied was taught at a large North American research university on the quarter system. This 10-week course is designed for CS majors and minors, and is one of the gateway classes to other advanced courses in the major. The goal of the class is for students to learn how to program and analyze high-performance data structures. The data structures studied include trees, hash tables, Huffmann coding, and graphs. Related topics include memory management, pointers, and recursion. Assignments are programmed in C++. The course has three principal prerequisites, all lower-division courses: introductory data structures (CS2), advanced discrete math, and computer organization.

6.4.2 Data Collection

In order to be able to study the relative levels of prerequisite proficiency we needed a tool to broadly examine the prerequisite knowledge of the students in this class. We chose to test the prerequisite knowledge of students by means of two surveys. In a pre-study for the present study, we ran the surveys online. However, we found a significant mismatch between student correctness in the online survey versus questions which were subsequently given in class. This suggested that students collaborated or used outside aids while completing the survey online. So, for this study, we performed the data collection with an in-class survey, on paper. The first survey was taken during the first day of class and the second survey was taken during the last week of class, before the final exam. Both surveys consisted of 15 multiple-choice questions and students were given 20 minutes for each survey. This format was chosen to balance the needs of data capture with the time taken away from the class lecture time.

6.4.3 Survey Content

The following courses are prerequisites for majors for the upper-division data structures class at our institution:

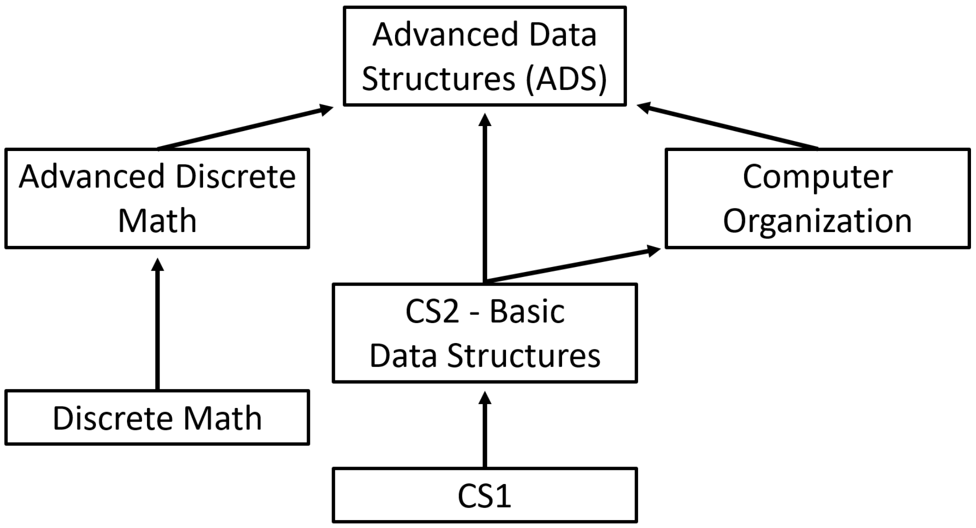


Figure 6.1: Prerequisite chains for ADS.

Table 6.1: Survey questions listed by question topic and related prerequisite courses.

Q	Topic	Related courses
1	Pass by value/reference	CS1
2	Valid Heap not BST	CS2 - Basic Data Structures
3	Valid Heap not BST	CS2 - Basic Data Structures
4	Valid BST not Heap	CS2 - Basic Data Structures
5	Neither Heap nor BST	CS2 - Basic Data Structures
6	Best List implementation	CS2 - Basic Data Structures
7	Runtime analysis	CS2 - Basic Data Structures and Adv. Discrete Math
8	Invariants	Discrete Mathematics and Adv. Discrete Math
9	Tracing recursion	CS1 and Adv. Discrete Math
10	Assembly	Computer Organization
11	Bitwise arithmetic	Computer Organization
12	Little-endian addressing	Computer Organization
13	Pointers	Computer Organization
14	Probability	Discrete Mathematics
15	Tracing recursion	CS1 and Adv. Discrete Math

Lacking validated assessments for direct prerequisite courses, we selected questions for our survey from those used in prior research studies on these topics [34, 47, 53, 102]. In cases where such a question was not available, we asked the instructor of the corresponding prerequisite course to supply us with two easy and two medium difficulty multiple-choice questions for us to use.

The two surveys were designed to be essentially equivalent, matched question-for-question. To limit the effects of memorization between the two surveys, small changes were made to the questions in the second survey, making sure that the changes did not make a question harder or easier. For example, on a tracing question, it was important that the number of statements that needed to be traced stayed the same. Table 6.1 lists the question topic and associated prerequisite courses for each survey question. Publicly hosted PDF versions of both surveys can be found online [8]. Because questions 2-5 are topically related, we reversed the order of these questions between the first and second survey. In the table (and the rest of this paper), the order of questions 2-5 for Survey 2 has thus been reversed so that they align with the equivalent questions in Survey

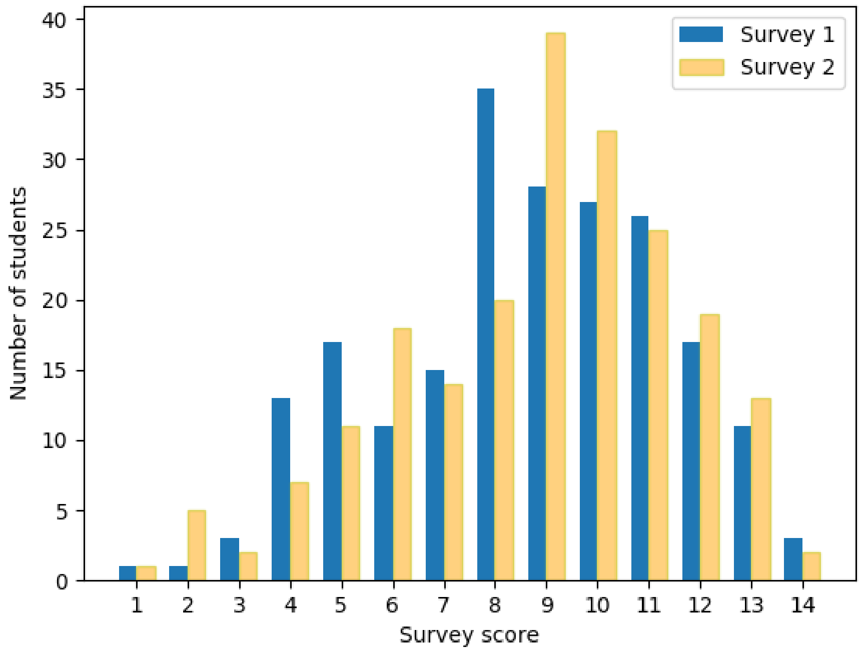


Figure 6.2: Distribution of Scores on Surveys 1 and 2. Respectively: avg=8.6 (57%), 9.8 (65%); median=9, 10; std dev=2.7, 2.7.

1. For example, question 5 on Survey 2 is treated as question 2 in the table.

6.4.4 Population

The first survey was taken by 365 students, the second by 236 students, and the final by 363 students. A total of 209 students took both surveys and 208 of those students took both surveys and the final exam. Unless stated otherwise, the statistics in this paper are based on these 208 students.

6.5 Results

6.5.1 RQ1: Early-Term Prerequisite Proficiency

For Survey 1 students scored an average of 8.6 out of 15 points with a median of 9 and a standard deviation of 2.7. Figure 6.2 shows a bar chart of the distribution.

Table 6.2 shows the amount of low-, medium-, and high-performing students for Survey 1.

Table 6.2: Low, medium, and high groups on Survey 1.

Group	Threshold	# students	% students
lowS1	$0 \leq score\ S1 < 8$	61	29.3%
mediumS1	$8 \leq score\ S1 \leq 10$	90	43.3%
highS1	$10 < score\ S1 \leq 15$	57	27.4%

Table 6.3: Low, medium, and high groups on Survey 2.

Group	Threshold	# students	% students
lowS2	$0 \leq score\ S2 < 8$	44	21.2%
mediumS2	$8 \leq score\ S2 \leq 10$	73	35.1%
highS2	$10 < score\ S2 \leq 15$	91	43.8%

We define low performance as below 50% (below 8) and high performance as above 70% (above 10). As listed in Table 6.2, just over 29% of students answered fewer than 50% of the questions on prerequisite topics correctly.

In sum, the students entering a subsequent course like ADS exhibit a wide distribution of prerequisite proficiency, with about a third having considerably lower proficiency than expected.

6.5.2 RQ2: Change in Proficiency during Term

For Survey 2 students scored an average of 9.8 out of 15 points with a median of 10 and a standard deviation of 2.7. Figure 6.2 shows a bar chart of the score distribution for Survey 2. Table 6.3 shows the number of low-, medium-, and high-performing students for Survey 2 using the same thresholds as Survey 1.

The score improvement between Surveys 1 and 2 was 1.2 (8%) on average, with a median of 1 and a standard deviation of 2.7. The scores on Survey 2 were significantly different compared to Survey 1 with a p-value less than 10^{-4}. As seen in Figure 6.3, most students improved their score by 0, 1, or 2 points out of 15. This resulted in over a quarter of the low performers on Survey 1 moving to medium or high proficiency. Overall, these results demonstrate that for the prerequisites in this study, as students progress in a subsequent course, they improve their prerequisite proficiency.

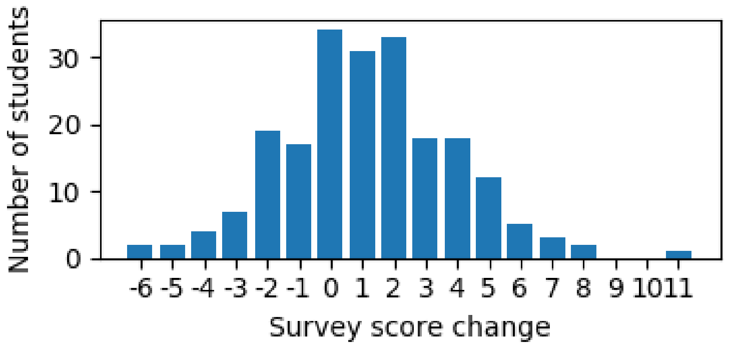


Figure 6.3: Distribution Score Change. The average score change was 1.2 points (8%), with a median of 1 and a standard deviation of 2.7.

Table 6.4: Low, medium, and high groups on the Final Exam.

Group	Threshold	# students	% students
lowFinal	$0 \leq score < 59$	64	30.8%
mediumFinal	$59 \leq score \leq 65.5$	76	36.5%
highFinal	$65.5 < score \leq 75$	68	32.7%

6.5.3 RQ3: Relationship between Prerequisite Proficiency and Final Exam Scores

The average score on the final exam was 61.8 points out of 75 (82%), with a median of 63 (84%) and a standard deviation of 6.9 (9%). For both surveys, the students' scores are significantly correlated to their final exam performance. For Survey 1 the correlation to final exam grades is 0.366, for Survey 2 the correlation is 0.384, and for the sum of the scores for Survey 1 and 2 the correlation is 0.434. The p-values for these Pearson correlations are all below 10^{-7}.

The same correlations are illustrated in Figure 6.4. We binned students based on their final exam performance as low, medium, or high performers. To obtain a balanced grouping we used the two tertile points of the score distribution as thresholds. The reason the group sizes are not perfectly in balance despite using the tertile points as thresholds is because multiple students received the same final score. Notable is that these thresholds are at 79% and 87% of a full score, so on average students did much better on the final exam than on our prerequisite surveys.

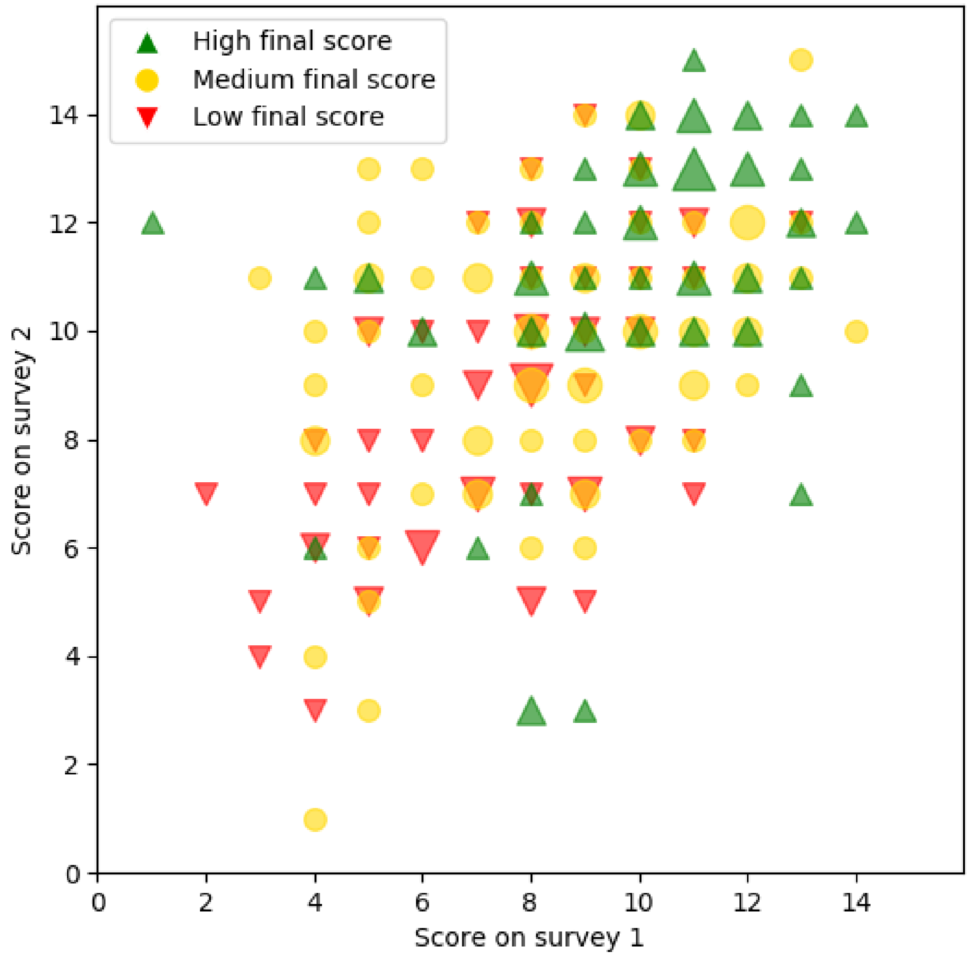


Figure 6.4: Comparison of Survey 1, Survey 2, and final scores. Low, medium, and high scores are defined in Table 6.4. A data point's size conveys the number of students under the point.

Table 6.4 shows the groupings.

In Figure 6.4, each student is plotted in the scatter plot with the *x* axis being their score on Survey 1 and the *y* axis being their score on Survey 2. The color and shape of the data point indicates whether the student performed low, medium, or high on the final. The size of the data point indicates the number of students represented by it. Note that in cases when there were students from multiple performance levels in the same point, multiple data points appear (e.g., a large green triangle and a small yellow circle). Where these multiple points occur, the color may appear as combinations of others (e.g., orange is a combination of red and yellow).

Figure 6.4 hence visualizes the correlation between prerequisite proficiency and final

Table 6.5: Statistics for survey questions: percentage of correct answers and comparison to final exam scores: correlation with p-values, sufficiency, and inverse necessity. Bolded values indicate significant correlation for $alpha = 0.05$.

		Survey 1					Survey 2				
Q	Subject	correct	correl	p-val	suff	nec^{-1}	correct	correl	p-val	suff	nec^{-1}
1	Pass by value/reference	64%	0.050	0.470	1.13	1.25	71%	0.103	0.139	1.24	1.68
2	Valid Heap not BST	44%	0.006	0.937	0.99	1.00	44%	0.065	0.350	1.27	1.20
3	Valid Heap not BST	56%	**0.190**	**0.006**	1.36	1.46	75%	**0.193**	**0.005**	1.21	1.80
4	Valid BST not Heap	62%	0.055	0.434	1.04	1.07	83%	0.126	0.069	1.14	1.90
5	Neither Heap nor BST	76%	0.108	0.120	1.15	1.55	90%	0.063	0.370	1.03	1.37
6	Best List implementation	61%	0.032	0.646	1.13	1.21	65%	0.071	0.307	1.06	1.12
7	Runtime analysis	61%	**0.329**	**0.000**	1.85	2.56	69%	**0.284**	**0.000**	1.34	1.92
8	Invariants	34%	0.009	0.895	0.89	0.94	41%	0.072	0.299	1.05	1.03
9	Tracing recursion	78%	0.119	0.087	1.05	1.17	77%	**0.211**	**0.002**	1.16	1.65
10	Assembly	56%	**0.200**	**0.004**	1.43	1.57	43%	**0.235**	**0.001**	1.59	1.40
11	Bitwise arithmetic	58%	**0.155**	**0.025**	1.25	1.35	67%	**0.217**	**0.002**	1.36	1.86
12	Little-endian addressing	21%	**0.194**	**0.005**	1.91	1.18	27%	**0.252**	**0.000**	2.10	1.30
13	Pointers	69%	**0.244**	**0.000**	1.47	2.35	69%	**0.203**	**0.003**	1.32	1.87
14	Probability	55%	**0.215**	**0.002**	1.34	1.42	72%	**0.265**	**0.000**	1.36	2.19
15	Tracing recursion	70%	**0.198**	**0.004**	1.16	1.40	88%	-0.049	0.479	1.02	1.12

exam performance. Most strikingly, it shows that most of the high performing students on the final were those who performed well on both Survey 1 and Survey 2 (upper right quadrant). Overall, we find a strong correlation between prerequisite proficiency and final exam performance.

6.5.4 RQ4: Which Prerequisites Relate to Final

To gain insight on whether prerequisite knowledge can be unbundled into more- and less-relevant topics, we performed a few per-question statistical analyses. Table 6.5 lists the correlation of the individual survey questions to the final exam scores, providing a hint as to which prerequisites might be important for performance on the final. Significantly correlated questions are listed in bold. Question 7, which concerns runtime complexity analysis, is the highest (significantly) correlated question for both Survey 1 and 2.

Some questions may be harder than others, reducing correlation at the extremes because there is little to distinguish high and low proficiency. Additional signal might be teased out of such questions by using conditional probabilities. For example, getting the right answer on a really hard survey question might be associated with high performance on the final, yet getting it wrong might say little about doing poorly on the final. Such distinctions are captured by the concepts of Sufficiency and Necessity [36]. A high likelihood of high performance when getting a survey question right is interpreted as meaning that the prerequisite knowledge is *sufficient* for success. A low likelihood of high performance when getting a survey question wrong is interpreted as meaning that the knowledge is *necessary* for success.

We express sufficiency as a likelihood ratio—the ratio that one outcome is more likely than another—specifically the positive likelihood [36]:

$$\textit{Sufficiency} = \frac{P(E|H)}{P(E|\sim H)}$$

where the variable E stands for Evidence (in our case answering a question correctly), H stands for Hypothesis (the student falling in the top 50% of the final exam scores), and $E|H$ means $E\,given H$ (all students that answered the survey question correctly and were in the top 50% of the final exam scores).

Necessity is the negative likelihood, that of finishing in the top half of the final if a student answers a question *incorrectly*, that is, $\sim E|H$. Since we are interested in the association between answering the question incorrectly and being in the *bottom* 50% on the final exam, $\sim E|\sim H$, we use inverse necessity (inverse negative likelihood):

$$\textit{Inverse Necessity} = \frac{P(\sim E|\sim H)}{P(\sim E|H)}$$

The question most strongly associated with high final exam performance when answered correctly (sufficiency) is question 12 for both Survey 1 and 2. Its sufficiency of 2.10 on Survey 2 means that students who answered it correctly had more than twice the probability of ending up in the top half of the class than in the bottom half. Question 12 was deemed difficult by the authors of this paper, and the students performed worst on it.

The question most highly associated with low final exam performance when answered

Table 6.6: Correlation between change in survey score and final exam score, broken down by the groups defined in Table 6.2, Significant correlations are in bold for $alpha = 0.05$ and underlined for $alpha = 0.005$.

Group	avg. impr.	med. impr.	σ	corr.	p-val
lowS1	2.90	3	2.8	**<u>0.371</u>**	**<u>0.003</u>**
mediumS1	1.01	1	2.4	0.114	0.284
highS1	-0.37	0	2.0	**0.306**	**0.020**

incorrectly (inverse necessity) is question 7 on Survey 1 and 14 on Survey 2. Question 7 was deemed by the authors as solvable with just a basic proficiency. Question 7 stands out for being strong for sufficiency, inverse necessity, and correlation. As such, we can safely (and unsurprisingly) conclude that early-term proficiency in runtime analysis is crucial to good performance in an Advanced Data Structures class. The second highest question for inverse necessity was question 13, a basic question on pointers. As the data structures class jointly teaches C++ programming, the importance of having a background in C is unsurprising. The high necessity for question 14 on Survey 2, regarding basic probability, was a surprise, as ADS does not explicitly use probability. Question 10, regarding assembly language, inexplicably exhibits a similar pattern. The necessity of an apparently unrelated topic raises the possibility that there are dependences that remain to be understood or may be indicative of an intervening variable (e.g., study skills, mathematical reasoning, maturity, etc.).

Overall, we find that most prerequisite topics exhibit positive correlations and modest sufficiency and necessity. Some related topics that exhibit the strongest values, like runtime analysis, make sense. However, the high inverse necessities for unrelated topics leave many open questions.

6.5.5 RQ5: Relationship between *Improvement* in Prerequisite Mastery and Final Score

The correlation of score change between Surveys 1 and 2 for the entire class was small and insignificant at 0.02, p-value of 0.77. Examining the course on a whole can mask effects for subpopulations, and we were curious if low performers on Survey 1 saw a significant benefit from improving their prerequisite proficiency. Examining student performance based on the groups defined in Table 6.2, lowS1 and highS1 have significant correlation, as shown in Table 6.6. We do not find significance for mediumS1, however their 1-point improvement is small. The scores of the high performers barely moved, interestingly. The movement and correlation for lowS1 is especially strong, improving a median 3 points, with a correlation of 0.371 and a p-value of 0.003. This effect can also be seen in the left half of Figure 6.4, which contains those students who performed poorly on Survey 1. For those students, if they continue to do poorly on Survey 2 (students in lower-left corner) they tend to have poor final exam scores, whereas students who improved on Survey 2 (students in upper-left corner) they tend to have better final exam scores.

In sum, most of the improvement in proficiency is due to the improvement of those with low entering proficiency, and that improvement had a significant correlation with their final score.

6.6 Discussion

In this section we reflect on our research questions, results, limitations, and threats to validity.

6.6.1 Findings

The relatively low early-term prerequisite proficiency that we identified, combined with the correlation between proficiency and final exam performance, shows that many students will indeed be struggling. This is especially worrying as Muralidharan et al. [81] found that virtually no learning at the 8th grade math level is occurring for students who lack proficiency with the lower grade math materials. They found intervention with a focus on proficiency with lower grade math materials is required in order for students to make progress with their learning at the 8th grade math level.

In our study, we find that students do improve their prerequisite proficiency over the term, however they do so only marginally on average. But we also find that the amount of improvement in prerequisite proficiency is especially correlated to final exam performance for students with low early-term prerequisite proficiency.

One group we did not discuss was the 16 students who dropped the course (and hence are not included among our 208 participants). Their average early-term proficiency was 7 percentage points lower than those who finished the course, with 43% demonstrating low proficiency and only 12% high proficiency. While not statistically significant (likely due to the small sample size), this result suggests that prerequisite proficiency may be even lower at the start of the term than we found here. Moreover, it may indicate that lack of proficiency may be a reason for students dropping the course.

Given these findings, we suggest two possible courses of action: (1) final exams should be carefully designed to ensure that students who pass have achieved proficiency before proceeding to the next course (see discussion of course grades in the next section) and/or (2) teachers of upper-division computer science courses, especially those with many prerequisites, should consider helping students reinforce their prerequisite knowledge at the beginning of the term.

6.6.2 Why Students Lack Prerequisite Knowledge

Grades in Prerequisite Courses. In this study, we found that prerequisite proficiency in an upper-division Computer Science class is much lower than one might expect, given the students achieved passing grades in prerequisite courses. When we analyzed the failure rate of the prerequisite courses for Advanced Data Structures [6] we found that at our institution, on average, only 2.9% and 3.6% of students will receive a D or F (respectively) with the remainder of the students received a C or higher. (A 'D' grade presently allows students to progress at our institution). Our study highlights that not just 3.6% of students but likely 29% of students are failing in their prerequisite proficiency. We suspect there might be some form of grade inflation occurring and the consequence is that we are sending under-prepared students into subsequent classes.

Multiple Prerequisite Paths. At our institution, we offer non-majors options for prerequisite courses to provide flexibility. For example, non-majors can complete either a computer organization course that follows CS2, teaching C and assembly programming, or a non-major's CS1 course in C. Either course should teach students the necessary knowledge in C (e.g., pointers) to begin a course in C++, however one may do so in more detail. Additionally, one interdisciplinary major at our institution (with a large number of required units and few students) requires the Advanced Data Structures course but waives the Computer Organization prerequisite. As such, part of the effect seen in this study may be due to students bypassing prerequisites or due to some prerequisite courses being inferior to others as preparation for this course. Further inquiry is the topic of ongoing future work.

Unrealistic Expectations. Prior work in CS1 has shown that students often underperform instructor expectations. For example, instructors expect students completing CS1 to be able to complete the rainfall problem when, in reality, few students are so capable [108]. It is possible that this mismatch is again at play, where our survey questions on prerequisites were viewed as relatively easy by the authors when in reality, they may be difficult for students. This underscores the importance of using meaningful assessments at the end of courses (e.g., concept inventories [117]) to measure student learning in order to better understand their abilities.

6.6.3 Limitations and Threats to Validity

Limitations. Generalizability from our study is limited as we only studied one course (ADS) and one term. Because correlation is not causation, the correlations found might not imply that increased prerequisite proficiency causes increased performance on the final. However, the fact that increased proficiency over the term was correlated with increased final exam performance for low- and high-performing groups provides compelling complementary evidence.

Underlying Factors. Although our survey focused on prerequisite knowledge, the relationship between the survey and the final exam may not be entirely related to prerequisite content. There may be intervening variables as perhaps prerequisite proficiency is related to students' general study habits or motivation in the field. Moreover, perhaps "better" students took the survey seriously and hence did better on the survey than those who did not. However, evidence that questions on particularly critical prerequisite topics were most strongly correlated with student exam performance suggests that at least some portion of the effect found related to the content itself.

Threats to Validity. The two surveys we used as a measuring tool are not validated and the questions were drawn from many different sources. Some survey questions may have been too hard for all students, making them less useful in our study. Although most students answered all questions, students were given a relatively short amount of time to complete them. As such, students may have hurried through the last questions and hence impacted findings. In addition, those who completed the survey were those who were in class on the two days the survey was given, so the results may be biased for the population of students who frequently attend class.

6.7 Summary

In our study of prerequisite proficiency in an upper-division advanced data structures course, we find that students varied in their proficiency to an unexpected degree, that prerequisite proficiency correlated with higher final exam scores, and that students improve on their understanding of prerequisites in a way that correlates to improved final exam scores. In sum, these findings support more care by instructors in ensuring students have fundamental proficiency with their course content before allowing them to move on to subsequent courses. In turn, instructors who teach courses with multiple prerequisites may wish to provide students with resources to buttress their mastery of prerequisites early in the term. Additional study is needed to understand whether these findings replicate across multiple institutions and across multiple courses, however, these results offer a promising glimpse of how prerequisites relate to student success in upper-division computing courses.

6.8 Acknowledgements

We wish to thank the reviewers for their helpful suggestions and the course instructor, Hau Nguyen, for his assistance. This work was supported in part by NSF award 1712508.

Chapter 6, in full, is a reprint of the material as it appears in the Proceedings of the 50th ACM Technical Symposium on Computer Science Education (SIGCSE 2019). Sander Valstar, William G. Griswold, Leo Porter. “The Relationship between Prerequisite Proficiency and Student Performance in an Upper-Division Computing Course”. The dissertation author was the first author of this paper.

Chapter 7

Student Proficiency in CS Fundamentals

A Study on Proficiency in Basic Data Structures among Various Subpopulations of Students at Different Stages in a CS Program

7.1 Introduction

Inspired by the findings in Chapter 6 that a surprisingly large percentage of students enter later CS classes unable to demonstrate knowledge from earlier classes, we sought to gain better insight into how students' understanding of a core computing topic changes in a progression of courses in our curriculum. To this end, we administered the validated Basic Data Structures Inventory (BDSI) [92] in three classes in a sequence of dependent courses at our institution to see 1) how much student understanding varies and 2) whether their understanding improves in later courses.

Similar to our study on prerequisite knowledge in Chapter 6, we found that student performance varied widely on the BDSI, which might pose challenges for instructors trying to address the pedagogical needs of the students in those classes. Encouragingly, we found that student performance on the BDSI increased for each course in the sequence. This increase suggests that either students are improving in their knowledge of basic data structures over time or students who struggled are leaving. If students are improving on their knowledge from prior courses, this is an interesting finding as a previous study documented notable knowledge loss of CS1 material over the period of a summer [120]. However, unlike that study, we believe basic data structure concepts are likely reinforced in each subsequent course rather than being left to atrophy over the summer.

In addition to looking at student performance for the whole class, we examined performance differences between various subpopulations of the course including: 1) transfer and non-transfer students,[1] 2) students from demographically represented and underrepresented groups in CS in the United States (RG and URG respectively, see Section 7.4.4 for definitions), and 3) by gender.

We found a difference in performance by gender for one of the three courses, but no clear overall trend. We also found that students from URGs on average do slightly worse on the BDSI than students from RGs for all three courses. However, we find no evidence that this performance gap is widening or closing over time. Our most concerning finding is for the transfer students. Although transfer students perform roughly the same as non-transfer in the first course in the series, they do not improve like their non-transfer peers in the two later courses. This causes a growing knowledge gap between transfer and non-transfer students over time.

These findings are consistent with the recent finding from Alvarado et al. that students who took computing courses prior to college (AP-CS) did better than those who did not, through the entire major [16]. This makes the findings no less problematic. Indeed, these findings highlight the need for research on interventions capable of offsetting the societal barriers presented to students from lower socioeconomic backgrounds (common among transfer students at our institution) and to offset the lack of access to early computing courses in school districts in the United States with a majority of Black and Latinx students [73].

[1]transfer students are generally those who join our university after completing two years of school at a community college

7.2 Related Work

It has long been known that students are not learning as much in courses as instructors think they do [49, ch. 2], [108]. Moreover, evidence indicates an issue of knowledge retention among students. After an extensive literature review on knowledge retention, Custers found support for about a third of knowledge being lost in the first year, half being lost after a few years, but leveling off subsequently [33]. Tennyson and Beck recently showed that CS2 students who took CS1 in the Fall semester and CS2 in the Spring semester had a knowledge loss of only 4%, whereas students who took CS1 in the Spring semester and CS2 in the Fall semester showed a knowledge loss of 15.1% [120]. This difference may be explained by the long summer break between the Spring and Fall semesters. Chapter 6 has shown that upper division CS students enter their courses with substantially less prerequisite knowledge than teachers might expect. However, students did show evidence of improving their prerequisite knowledge during the course that was studied. While these studies all provide great insights, the knowledge retention of a core CS topic for students at various stages in a CS program has not yet been studied.

Considerable work has been done in the space of the recruitment and retention of students from groups that are underrepresented in computing. For example, women in computing have been studied extensively over the years [35, 40, 42]. More recently the Computing Education Research (CER) community has begun to recognize and study the underrepresentation of certain minority populations in computing, such as Black and Latinx students in the United States [73].

The gap between the performance of students from represented populations relative to underrepresented populations has been described in STEM education studies, particularly in publications that aim to narrow the achievement gap using active learning techniques [50, 124]. In CS, a recent study showed that adopting Computing in Context [48], Pair Programming [134], and Peer Instruction [76] improved performance for all groups of students with students from URGs benefiting more in absolute numbers, but because the relative rates of improvement were comparable, there was no evidence of the achievement gap narrowing [98].

Transfer students have been studied less extensively, but a study from 2010 presents several recommendations [86]. For example, it is suggested a university should appoint study advisors specifically to advise transfer students in order to provide them with better guidance. But to our knowledge, less is known about the performance of transfer students in computing relative to their non-transfer peers.

There are several assessments available to assess student understanding of CS topics [14, 59, 87, 91, 92, 121, 133]. In this paper we chose to use the recently developed Basic Data Structures Inventory (BDSI) [92] because of its rigorous validation process and because we believe data structures is a central topic in virtually any CS program. As such, other institutions should be able to repeat our methods as part of potential replication efforts. Lastly, because basic data structures also appear on industry coding interviews, this is a topic that should be important to students. At the time of writing, the BDSI is not publicly available, but it can be obtained via the instructions appearing in the article by Porter et al. [92].

7.3 Methods

This study was conducted at the University of California San Diego and is approved for Human Subjects Research. The Basic Data Structures Inventory (BDSI), a Concept Inventory (CI) specifically designed to assess student knowledge on fundamental data structures knowledge, was used as our instrument for measuring student proficiency [92].

7.3.1 Research Questions

We formalized our research questions for the study as follows:

RQ 1: Does general student performance on Basic Data Structures change throughout the various stages of a CS program?

RQ 2: Are there differences in student performance on a Basic Data Structures test for the following subpopulations?

- Transfer students vs non-transfer students
- Men vs women
- Students from URGs vs RGs

RQ 3: Do the performance differences between subpopulations (if any) change across the various stages of a CS program?

7.3.2 Data Collection

Participants were recruited from the following three CS courses at our university towards the end of the Fall 2020 term. We chose to measure in multiple different courses during the same term instead of following a single cohort because if a student were to complete the BDSI more than once, particularly if answers were discussed at the end of the administration, it would bias later results.

1. Introduction to Data Structures (CS2)
2. Introduction to Computer Organization (ORG)
3. Advanced Data Structures (ADS)

These courses form a prerequisite chain in our department:

$$CS2 \rightarrow ORG \rightarrow ADS$$

CS2 is taken after our introductory CS course (CS1). CS2 and ORG are lower division courses primarily taken by first and second year students. ADS is the gateway course into our upper division which is generally taken by students in their second or third year.

We approached the instructors of these courses to ask for permission to run a study on their students. For most of these courses the content addressed by the BDSI is extremely relevant course prerequisite material. This enabled us to frame our BDSI sessions as prerequisite material review sessions for the final exams.

The instructors informed their students about these sessions. Participation was voluntary and, depending on the course, the students were compensated at the instructor's discretion with extra credit on their final exam or on a programming assignment. Since the BDSI is a Concept Inventory, we cannot give performance based rewards to the students as this would heighten the chances for cheating to occur. Thus students were graded based solely on participation. In order to discourage students from submitting random answers to receive the participation credit without making an honest attempt at the BDSI, we told students they would only receive the extra credit if they took at least 30 minutes to complete the test.

For each course, students were given one hour to complete the test. To make it the promised review session (and to provide the students with an additional incentive to take the test seriously), we discussed the answers with the students after they had completed the test and submitted their answers.

We are using the recommended process for BDSI administration as presented in the BDSI paper [92]. However, because of the ongoing COVID-19 pandemic at the time of this study, we were unable to administer the BDSI in person. Instead, we held video conference calls with our participants. The BDSI was provided through a non-downloadable link and responses to the BDSI questions were collected using a Google Form. On the final page of the form we asked students to reflect on the test and provide some personal background information.

7.4 Results

7.4.1 BDSI Performance per Course

Table 7.1 shows the of BDSI score distributions for the three courses. Figure 7.1 shows a visual overview of the distribution of BDSI performance in CS2, ORG and ADS. An upward

Table 7.1: Overall BDSI performance per course. The maximum possible BDSI score is 13.

	CS2	ORG	ADS
n	221	187	325
mean	5.52	6.14	6.93
std	2.67	2.70	2.92
25%	3	4	5
50%	5	6	7
75%	8	8	9

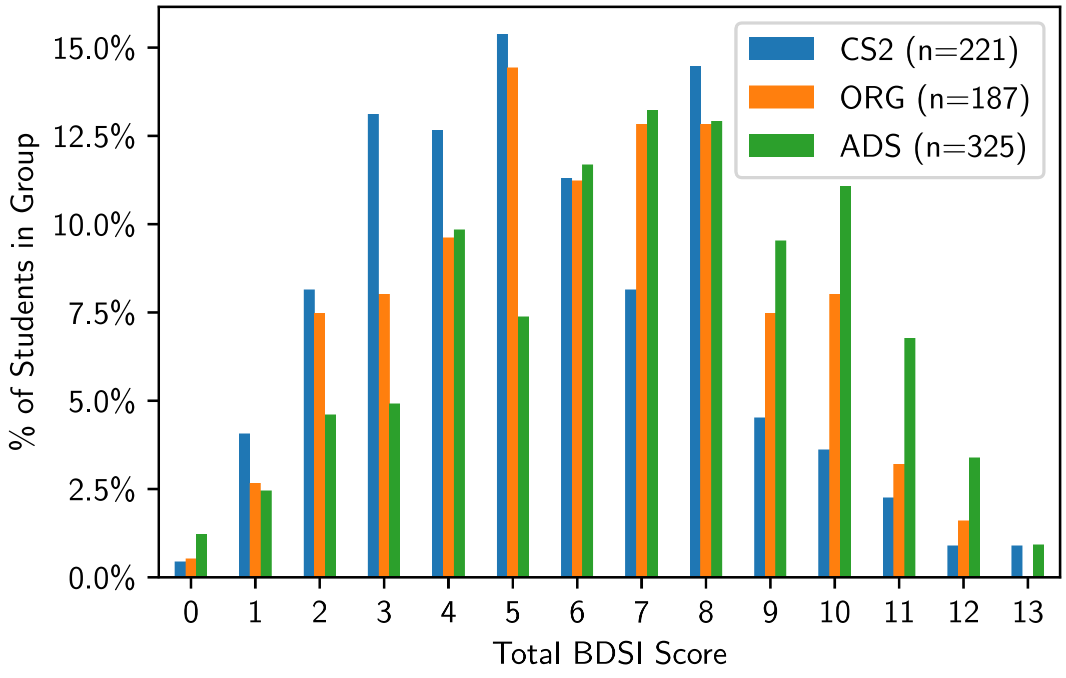


Figure 7.1: Student BDSI performance plotted per course. A clear shift to the right is visible.

trend in later courses is evident, suggesting that students continue to improve their knowledge of basic data structures as they progress through their CS program after completing their CS2 course.

We determined with a normality test that the BDSI performance data per course is not normally distributed, thus we apply nonparametric tests for our analyses. For example, we use Kruskal–Wallis H tests instead of ANOVAs and Mann-Whitney U tests instead of t-tests. As we are running many statistical tests in this study, we adjust our p-values in order to address the

Table 7.2: All Students: Holm-Bonferroni-adjusted p-values resulting from a post-hoc Dunn's test for the general student population in each course. Significant values for $\alpha = 0.05$ are bolded. Results indicate significantly different BDSI scores for ADS compared to CS2 and ORG as well as for ORG compared to CS2.

	CS2	ORG	ADS
CS2	1	**0.02**	**2.15e-08**
ORG	**0.02**	1	**0.01**
ADS	**2.15e-08**	**0.01**	1

increased chances for Type I errors. We chose to apply the Holm-Bonferroni adjustment method as it is allows for more statistical power and reduces the probability for introduction of Type II errors when compared to the simpler Bonferroni adjustment method [10].

A Kruskal–Wallis H test shows there are significant differences among the BDSI results for the three courses (Kruskal Stat: 34.16, p: 3.83e-08). In order to assess all pairwise differences for significance we ran a post-hoc test for nonparametric data (Dunn's test). As can be seen in Table 7.2, this test shows that the scores for ADS differ significantly from the scores in both CS2 and ORG and that the scores for ORG differ significantly from the scores for CS2 as well.

7.4.2 Transfer students

As can be seen in Table 7.4b, transfer students performed significantly worse than non-transfer students for ORG and ADS. Moreover, Figure 7.2b shows a clear upward trend for non-transfer students, whereas there is no clear upward trend visible for transfer students. When we run a Kruskal-Wallis H test on the non-transfer students, we find there is a significant difference among the three courses (stat: 33.97, p: 4.19e-08). Following up with a post-hoc Dunn's test (details in Table 7.3), we find that the scores for non-transfer students significantly improved throughout the entire program. Concerningly, we find no evidence that transfer students improve their performance on the BDSI anywhere from CS2 through ADS (Kruskal-Wallis H stat: 0.52, p: 0.77).

Table 7.3: Non-Transfer Students: Holm-Bonferroni-adjusted p-values resulting from a post-hoc Dunn's test on non-transfer student BDSI total scores for the three courses. Values indicating significance for $\alpha = 0.05$ are bolded. Results indicate for non-transfer students BDSI score differences between CS2, ORG and ADS are significant.

	CS2	ORG	ADS
CS2	1	**0.01**	**2.32e-08**
ORG	**0.01**	1	**0.01**
ADS	**2.32e-08**	**0.01**	1

Table 7.4: BDSI performance distribution details per course and subpopulation corresponding to the box plots in Figure 7.2. Statistically significant differences between subpopulations are bolded (for $\alpha = 0.05$ according to a Holm-Bonferroni-adjusted Mann-Whitney U test).

(a) Men and women.

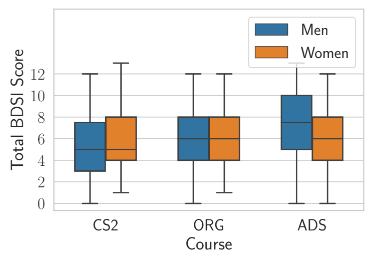
	CS2		ORG		ADS	
	M	F	M	F	M	F
n	147	71	122	64	210	98
mean	5.41	5.77	6.20	6.11	7.17	6.28
std	2.59	2.83	2.65	2.80	3.01	2.75
25%	3	4	4	4	5	4
50%	5	5	6	6	7.5	6
75%	7.5	8	8	8	10	8
M-W U	stat=4878.5 p=0.43		stat=3777.5 p=0.43		**stat=8415.0 p=0.01**	

(b) Transfer and non-transfer.

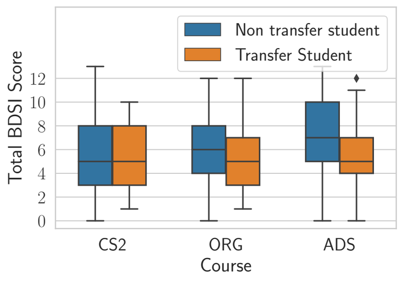
	CS2		ORG		ADS	
	NT	T	NT	T	NT	T
n	160	60	160	26	257	52
mean	5.56	5.45	6.33	5.15	7.16	5.50
std	2.75	2.47	2.66	2.75	2.93	2.68
25%	3	3	4	3	5	4
50%	5	5	6	5	7	5
75%	8	8	8	7	10	7
M-W U	stat=4714.5 p=0.42		**stat=1521.0 p=0.03**		**stat=4432.5 p=1.79e-04**	

(c) RG and URG students.

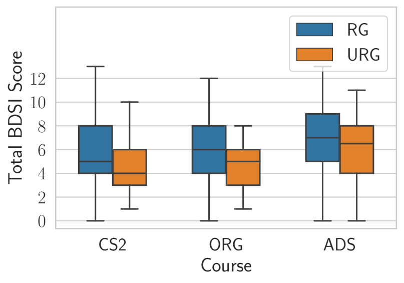
	CS2		ORG		ADS	
	RG	URG	RG	URG	RG	URG
n	199	22	169	17	282	28
mean	5.64	4.45	6.30	4.88	6.98	6.14
std	2.66	2.50	2.72	2.11	2.96	2.75
25%	4	3	4	3	5	4
50%	5	4	6	5	7	6.5
75%	8	6	8	6	9	8
M-W U	stat=1621.5 p=0.06		stat=998 p=0.06		stat=3313.0 p=0.08	



(a) Men and women.



(b) Transfer and non-transfer students. A clear upward trend is visible for the non-transfer students.



(c) RG and URG students.

Figure 7.2: Box plots of BDSI performance by gender, transfer status and URG status per course.

Table 7.5: BDSI performance distribution of transfer students in ADS who took CS2 at our institution and those who took CS2 at a different institution. The difference is not statistically significant for $\alpha = 0.05$.

	CS2 at our institution	CS2 not at our institution
n	27	17
mean	5.26	6.47
std	2.73	2.45
25%	3.5	5
50%	5	6
75%	7	8
	Mann-Whitney U stat=170.0, p=0.08	

Could it be the case that transfer students who took CS2 at our institution do better than those who took a basic data structures course elsewhere?

We asked ourselves this question after we noticed that the BDSI performance differences between transfer students and non-transfer students at the end of our CS2 course were not statistically significant. Thus we suspected that perhaps what mattered most for the difference in performance between transfer students and non-transfer students in the following courses was not caused by whether they were transfer students, but whether they had taken the same CS2 course.

However, contrary to our expectations, the transfer students in ADS who took CS2 at a different institution performed *better* on the BDSI than transfer students who had taken CS2 at our institution. This difference, though, was not statistically significant. Figure 7.3 and Table 7.5 show the details of the distributions. Students who did not report their CS2 institution were excluded from the analysis.

7.4.3 Gender

The differences in performance between men and women are not significant except in ADS, where we find men performed significantly better on the BDSI. Details about the performance distributions can be seen in Figure 7.2a and Table 7.4a. Section 7.5.2 discusses a possible explanation for why we may suddenly see this difference.

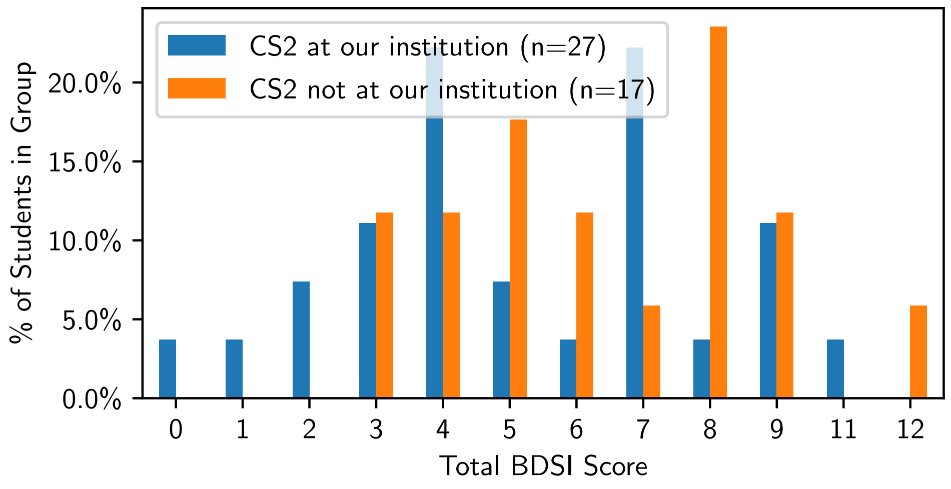


Figure 7.3: BDSI performance of transfer students in ADS who took CS2 at our institution plotted against those who took CS2 at a different institution.

Table 7.6: RG Students: Holm-Bonferroni-adjusted p-values resulting from a post-hoc Dunn's test for the BDSI performance of RG students. Significant values for $\alpha = 0.05$ are bolded. Results show that for RG students BDSI scores differ significantly between all three courses.

	CS2	ORG	ADS
CS2	1	**0.03**	**6.72e-07**
ORG	**0.03**	1	**0.03**
ADS	**6.72e-07**	**0.03**	1

7.4.4 Underrepresented Groups

The University of California San Diego defines students from represented groups in CS (RG) as those who are Caucasian (non Hispanic or Latinx) and Asian. Students from underrepresented groups in CS (URG) are defined as those who are Black, Hispanic or Latinx, Native American and Pacific Islander. Students who did not report their demographic background to the university were not included in the results pertaining to RGs and URGs. We find that, on average, RG students performed better than URG students in all courses. However, we were not able to establish statistical significance for these differences. Details can be found in Figure 7.2c and Table 7.4c.

A Kruskal-Wallis H test shows that for RG students there is at least one strongly significant improvement from CS2 through ADS (26.99, p: 1.38e-06). A post-hoc Dunn's test (see Table 7.6) shows significant differences between the BDSI scores for all three courses for RG students.

We see slightly different results for students from URGs. Although, judging by the average scores, URGs do seem to increase their BDSI performance throughout CS2, ORG and ADS, and a Kruskal–Wallis H test shows near significance (5.80, p: 0.055), we did not find any significant differences between courses in a post-hoc Dunn's test.

7.5 Discussion

Our results led to several interesting findings that we will discuss in the context of our research questions.

7.5.1 RQ 1: BDSI Performance at Different Stages in Our CS Program

We find that general student performance on the BDSI improved throughout the progression of courses in our CS program. For example, the median score increased consistently from CS2 through ADS. Moreover, we find that the performance increases between the courses are statistically significant. These findings are encouraging as they imply students continue to improve their understanding of CS fundamentals as they progress through our program. However, we have some concern whether a 7 out of 13 in ADS represents mastery of the material. Unfortunately, the BDSI is a relatively new instrument and we know of no articles on student performance later in their CS careers against which we might compare.

7.5.2 RQ 2: Differences in Performance Between Subpopulations

Gender

We encouragingly find that men and women mostly perform equally well on the BDSI, despite the many sociocultural barriers that cause women to be less likely to gain access to computing education prior to university in our country [7, 40]. Only for ADS do we find a

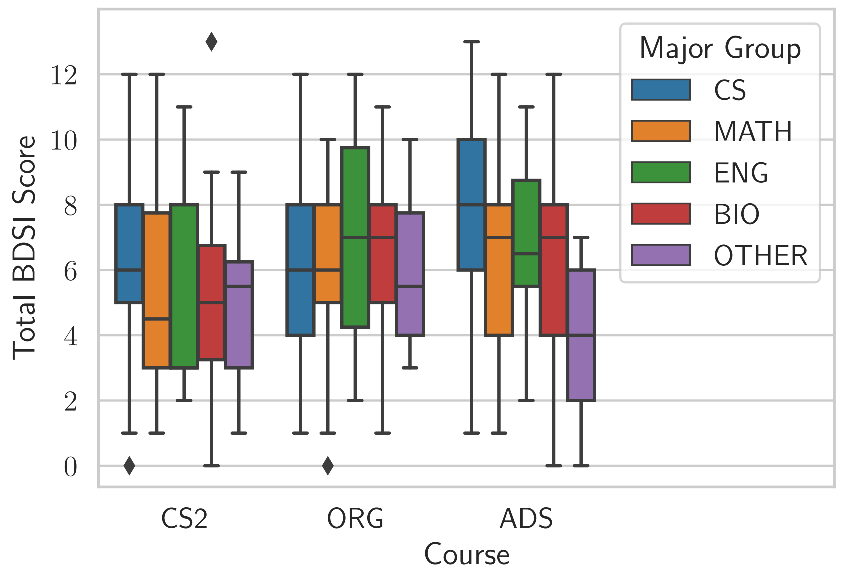


Figure 7.4: Student BDSI performance plotted per course and major group.

statistically significant difference in performance. However, we suspect this may be due to the fact that the women in our courses are less likely than the men to be CS majors (in ADS 69% of men were CS majors vs 49% of women). At our institution, other majors are allowed, at times, to bypass ORG to reach ADS and our department has found that those who take ADS without ORG are less likely to do well.

Indeed, we see in Figure 7.4 for ADS that CS majors scored substantially higher than other majors, whereas the differences between majors were not as substantial in the other courses. In ADS, 49% of women were Math and Biology-related majors, which both scored substantially lower on average than CS majors. As only 27% of men in ADS were Math and Biology-related majors, this could potentially explain why we are seeing a performance difference between men and women in ADS. Moreover, we encouragingly find no evidence that the performance difference between men and women is significant *within CS majors* in ADS.

Transfer Students

We find that transfer students perform significantly worse on the BDSI than non-transfer students in all courses except CS2. This concerning finding prompted us to investigate whether transfer students who took CS2 at our institution were at an advantage compared to their transfer student peers who had taken CS2 elsewhere. However, contrary to our expectations, transfer students who took CS2 at our institution do not seem to have an advantage over transfer students who took CS2 elsewhere. In fact, our data seems to suggest the opposite may be true. A possible explanation is that most of the community colleges that our transfer students attend are on a semester system, and we are on a quarter system. It is possible that the additional 4–5 weeks of content might explain the difference. However, we remain concerned that overall transfer students still perform worse than non-transfer students in our courses, and call for future research to determine the cause.

Underrepresented Groups

We find that students from URGs consistently score lower on average than students from RGs. However, the combination of small sample sizes of URGs and the small differences with RGs makes it difficult to attach any statistical significance to the observed differences. When we filter out transfer students and compare only non-transfer RGs and non-transfer URGs, the performance gap widens further.

Interestingly, in our preliminary investigations using self-reported URG status, we found much stronger effects than in our analysis on the official URG status data as provided by our institution. This suggests there may be merit in investigating the power of self-reported URG status when compared to an automatically assigned demographics-based status, especially in light of recent work on the impostor phenomenon and the importance of sense of belonging [96, 100].

7.5.3 RQ 3: Do Performance Differences Change Over Time?

We investigated whether the performance differences between the subpopulations change or remain the same throughout the different stages in the program. Our findings for this question are concerning.

Transfer students

For transfer students the performance gap compared to non-transfer students appears at all stages of the program that we measured. Moreover, the issue worsens over time, with the gap widening with each succeeding course. While non-transfer students significantly improve their understanding of data structures, transfer students appear to remain at the same level of proficiency as they progress through the program. This finding is deserving of future inquiry to see if it is consistent across multiple institutions and to determine potential sources of the problem.

Underrepresented Groups

Similarly, the performance difference between students from URGs and RGs appears in all three courses. Moreover, while we find that RGs improve significantly over time, we do not find statistically significant improvements between courses for URGs. However, we observe no clear trend that the difference between RGs and URGs increases or decreases over time.

We know from prior work that, compared to RGs, students from URGs have less exposure to computing before attending university [73]. We also know from prior work that prior experience (or lack thereof) causes performance gaps in CS courses throughout the entire curriculum [16]. Based on these findings one might wonder whether lack of prior experience causes a persisting gap in understanding of basic data structures concepts even for those who persevere in the major. This, too, is deserving of future inquiry.

7.6 Limitations and Threats

The students in our dataset all attend the University of California San Diego; our results may not generalize beyond that context. One limitation of our findings is that we did not measure performance over time for the same population of students. This would be difficult. For the "review sessions" recommended by the BDSI creators [92], the benefit to students is the subsequent discussion of the correct answers. Also, there is only one version of the BDSI. These factors limit our ability to administer the same exam to the same population more than once. We have assumed the populations of students are similar for our assessed courses, and believe this to be reasonable based on our institutional knowledge. However, a threat to validity remains that the populations of students are different.

The BDSI instrument we used assesses just one subject that has a unique position in the typical CS curriculum. The results may not generalize to other subjects. Furthermore, working in a field of practice leads students to constantly advance their knowledge by themselves (e.g., studying for interviews), so good performance on the BDSI cannot necessarily be attributed to what was taught nor the quality of instruction.

A threat to external validity is that the BDSI was originally designed for in-person sessions, whereas, due to the ongoing COVID-19 pandemic, we administered the BDSI over video calls. This may have had an impact on student performance, so our results may not be comparable to studies where the BDSI was administered in person. We attempted to minimize this risk by telling students to treat the online BDSI session like a test on paper in a lecture hall, and by asking them to not discuss or search for answers online.

Future work could explore a wider array of institutions, graduating students, and evaluate knowledge on topics beyond basic data structures. Access to student grades in CS2 could generate useful additional insight. Likewise, learning students' interview preparation strategies could shed further light on the sources of graduating student knowledge.

7.7 Summary

This study investigated the basic data structures proficiency of several student subpopulations at various stages in a CS program. Encouragingly, we find that the average student proficiency with basic data structures increases for courses later in the CS program. This evidence suggests that students, on average, continue to learn core concepts like basic data structures as they progress through the program.

Concerningly, we find that transfer students perform significantly worse than non-transfer students in both ORG and ADS. Furthermore, we find that this performance gap widens as students progress further in their studies. Similarly to transfer students, students from URGs in CS do not perform as well on the BDSI as students from RGs, but this difference is not statistically significant and we do not find any evidence for a widening or closing of this performance gap over time. The performance for both transfer students and students from URGs (despite a visible trend in Figure 7.2c for URGs) does not statistically significantly improve in courses over time, whereas performance of non-transfer students and students from RGs increased significantly.

These findings are concerning for the diversity, equity, and inclusion of computing courses, particularly in light of the socioeconomic diversity of transfer students. It is possible that the systemic societal problems in our country that limit access to computing before college for students with lower socioeconomic status (often transfer students) and those from URGs [73] continues to impact student performance even after many courses in our major. This unsettling finding would be consistent with the work of Alvarado et al. who found that students who took CS in high school outperform those who did not, all the way through the CS curriculum [16]. We believe these findings are a call to action for more research into the sources and extent of the problem as well as possible interventions that aim to close the performance gap.

7.8 Acknowledgements

We thank instructors Greg Miranda, Bryan Chin, Niema Moshiri and Miles Jones for agreeing to let us run this study in their courses. This work was supported in part by NSF award DUE-1712508. This material is based upon work supported by the National Science Foundation Graduate Research Fellowship Program under Grant No. DGE-1650112. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

Chapter 7, in full, is a reprint of the material as it appears in the Proceedings of the 2021 ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE 2021). Sander Valstar, Sophia Krause-Levy, Adrian Salguero, Leo Porter and William G. Griswold. "Proficiency in Basic Data Structures among Various Subpopulations of Students at Different Stages in a CS Program". The dissertation author was the first author of this paper.

Chapter 8

Conclusions and Future Work

The academia-industry gap has persisted for at least two decades despite substantial attention from the academic community. This attention has come in the form of studies conducted to further understand the gap's dimensions as well as attempts to address it (see Table 1.1 and Chapter 5).

Encouragingly, Chapters 3 and 4 show that a majority of faculty are willing to help close the gap. However, we also uncovered that faculty experience a wide variety of barriers that keep them from addressing the gap in their own courses by themselves. Additionally, there is unfortunately still a sizeable minority of faculty who disagree with the notion that preparing students for industry should be a primary goal of undergraduate CS education. It is unclear how much influence this dissenting group has on the design of CS curricula.

We found in Chapter 5 that for each gap dimension, several educational innovations to address it have already been proposed in academic literature. Moreover, we uncovered that the adoptability of many of these innovations could be substantially improved by simple changes such as providing relevant course materials with the publications.

Regarding student learning, we found in Chapter 6 that students on average are not as proficient with prerequisite materials as expected coming into a more advanced later course.

However, we also found that students on average do improve their proficiency with prerequisite materials during that later course. Moreover, findings in Chapter 7 show that students on average continue to improve their understanding of CS fundamentals throughout their CS program and not just until the end of the course that taught the concepts. Unfortunately we also found that there are substantial performance disparities between certain groups of students with especially worrying data for transfer students.

With these findings, the claims in my thesis statement as outlined in Section 1.4 have all been addressed.

8.1 Limitations

The research presented in this dissertation has several limitations. For example, most of the data is from the University of California San Diego. And while the survey in Chapter 4 was administered to CS faculty world-wide, participants were still predominantly from North American universities. Thus the presented findings may be specific to this geographical location.

Moreover, the faculty who participated in our studies were not incentivized to participate. This could mean that our sample of faculty is not representative of the general population of CS faculty due to self-selection bias. We have tried to mitigate this by framing our work in terms of the goals of CS education in general instead of in terms of the academia-industry gap when recruiting study participants.

In the analysis of existing educational innovations for presence of barriers to adoption that was discussed in Chapter 5 we were limited to the information presented in the papers. Some authors may have made additional efforts to increase the adoptability of their work outside of the paper. Such efforts were not captured in our analysis.

8.2 Future Work

Studies presented in this dissertation uncovered that many faculty experience barriers in adopting educational innovations into their own courses. Future work should focus on alleviating some of these barriers such that it becomes easier for faculty who are willing to help close the academia-industry gap to adopt relevant innovations into their own courses.

8.2.1 A Call on the Computing Education Research Community

While there are already many promising educational innovations that could help close the gap, the findings presented in Chapter 5 show that there is much that can be improved in terms of making these innovations easier to adopt. Some possible improvements, such as providing relevant course materials, should be relatively easy to achieve. Especially since conference guidelines already contain phrases about adoption and reproduction. For example, for experience reports the SIGCSE 2021 reviewer guidelines say: "All papers in this track should provide enough detail so that others could adopt the new innovation." And for research track papers, the reviewer guidelines say that the data analysis and methodology should be sufficiently described so that the reader could reproduce the study [105]. Unfortunately it appears these requirements are not yet specific enough to incentivize authors to provide the relevant course materials for their educational innovations.

Future work should explore how to best approach this issue. Preferably in a coordinated structural way such as explicitly requiring publication of all relevant course materials in the author and reviewer guidelines of conferences and journals. Moreover, because of page count limits for papers, conferences and journals should suggest suitable methods for CS education researchers to publish these resources. In the future, the community should look at standardizing the publication method for these external resources as it is important that they remain accessible in perpetuity.

8.2.2 DevContainers and Online Codespaces

Another challenge which should be addressed is how to share tech stacks between innovators and adopters. For example, if an innovation includes custom software or scripts, then it should be possible for the innovator to share it in such a way that it is still easy for potential adopters to get started without first having to install and configure a large number of tools.

Just like potential adopters, students may experience similar setup difficulties with programming assignments. I noticed during my teaching at the University of California San Diego that if assignments come with complex tooling, some students do not bother to setup and configure the suggested tools. As a result those students may be making their assignments much more challenging for themselves. For example, if students did not configure the debugger, they would not have any debugging aids available to them during the assignment. This issue can be especially challenging when the instructor-suggested tools are not available for all common operating systems.

In order to make it simpler for students to get started with their programming assignments without having to worry about installing and configuring a large number of different tools, I have experimented with a promising technology called "DevContainers" and published an experience report about my findings [127]. DevContainers are an integration of Microsoft Visual Studio Code [79] and Docker [1]. When implemented in a programming assignment, a DevContainer allows students to open a fully configured development environment with the click of a single button. The only limitation is that students will still have to install Visual Studio Code and Docker on their machines. However, GitHub is working on a new initiative called "Codespaces" [46]. It will allow users to open a GitHub project in a DevContainer running in their web browser with the click of a button, eliminating the need for students to install tools such as Docker (which can be difficult on certain operating systems). At the time of writing, GitHub Codespaces is still in a closed beta phase. However, I expect this to become widely used in CS classrooms in the future as it seems to effectively address the issue of configuring student programming environments. Future work should investigate whether DevContainers and GitHub Codespaces can in a similar way be used by educational innovators to share their tech stacks with potential adopters.

8.2.3 Iterating on Existing Innovations

Appendix A.1 lists three educational innovations per area of the academia-industry gap as defined by Radermacher and Walia. I encourage CS educators to attempt to adopt any relevant innovations into their own courses. Since many of the innovations on this list contain several barriers to adoption, adopting them might be an intensive task. Thus I encourage educators who take up this challenge to publish about their findings and to publish resources that would make it easier for future potential adopters to adopt these innovations into their own course. The checklist presented in Appendix A.2 may be beneficial for helping determine what resources to publish.

Appendix A

Materials for Chapter 5

A.1 Analyzed Solutions

Gap Dimension/Title	**DOI Link**
Written Communication	
"Writing for computer science: a taxonomy of writing tasks and general advice"	10.5555/1127442.1127468
"Integrating communication skills into the computer science curriculum"	10.1145/2157136.2157248
"Communication skills in the CS curriculum"	10.5555/1516546.1516560
Oral communication	
"Technically speaking: fostering the communication skills of computer science and mathematics students"	10.1145/1227504.1227375
"It is time to stand up and communicate [computer science courses]"	10.5555/1253528.1254284
"A Curriculum Model Featuring Oral Communication Instruction and Practice"	10.1145/3017680.3017775
Project Management	

"Integrating Project Based Learning and Project Management for Software Engineering Teaching: An Experience Report"	10.1145/3159450.3159599
"Teaching design and project management with lego RCX robots"	10.1145/366413.364551
"Re-imagining a course in software project management"	10.1145/3183377.3183379

Software Tools

"Teaching Git on the Side: Version Control System as a Course Platform"	10.1145/2729094.2742608
"Distributed version control in the classroom"	10.1145/1953163.1953342
"Pushing Git & GitHub in undergraduate computer science classes"	10.5555/3015220.3015251

Testing

"Rethinking computer science education from a test-first perspective"	10.1145/949344.949390
"Teaching software testing concepts using a mutation testing game"	10.1109/ICSE-SEET.2017.1
"Test-driven learning: intrinsic integration of testing into the CS/SE curriculum"	10.1145/1124706.1121419

Teamwork

"Teaching teamwork in engineering and computer science"	10.1109/FIE.2011.6143000
"A Method to Analyze Computer Science Students Teamwork in Online Collaborative Learning Environments"	10.1145/2793507
"Towards best practices in software teamwork"	10.5555/948785.948797

Problem Solving & Critical Thinking

"Critical thinking and computer science: implicit and explicit connections"	10.5555/1127389.1127423
"Critical thinking in an introductory programming course"	10.5555/2184451.2184473
"Using pseudocode to teach problem solving"	10.5555/1089053.1089088

Programming

"Evaluating programming ability in an introductory computer science course"	10.1145/331795.331857
"A short unit to introduce multi-threaded programming"	10.5555/1734797.1734800
"Effects of Course-Long Use of a Program Visualization Tool"	10.5555/1862219.1862234

Requirements Engineering

"Facing the challenges of teaching requirements engineering"	10.1145/2889160.2889200
"Teaching requirements engineering to an unsuspecting audience"	10.1145/1124706.1121475
"Teaching requirements engineering to undergraduate students"	10.1145/1953163.1953207

Personal Skills

"Teaching computer science soft skills as soft concepts"	10.1145/2445196.2445219
"Examining Classroom Interventions to Reduce Procrastination"	10.1145/2729094.2742632
"Nature of Creativity in Computer Science Education. Designing Innovative Workshops for CS Students"	10.1145/2643572.2643580

Ethics

"How to teach computer ethics through science fiction"	10.1145/3154485
"Ethics Education in Context: A Case Study of Novel Ethics Activities for the CS Classroom"	10.1145/3159450.3159573
"A learner-centered approach to teaching ethics in computing"	10.1145/1124706.1121505

Leadership

"Learning to love computer science: peer leaders gain teaching skill, communicative ability and content knowledge in the CS classroom"	10.1145/1953163.1953225

"Increasing technical excellence, leadership and commitment of computing students through identity-based mentoring"	10.1145/1734263.1734320
"Teaching communication, leadership, and the social context of computing via a consulting course"	10.1145/1734263.1734291

Software Design

"A Collaborative Approach to Teaching Software Architecture"	10.1145/3017680.3017737
"An instructional scaffolding approach to teaching software design"	10.5555/1127442.1127472
"Teaching software architecture to undergraduate students: an experience report"	10.5555/2819009.2819079

Databases

"Teaching database security and auditing"	10.1145/1539024.1508954
"Teaching database in an integrated oracle environment"	10.1145/1189215.1189174
"Animated database courseware: using animations to extend conceptual understanding of database concepts"	10.5555/1409823.1409855

Development & Improvement Processes

"TEAMSCOPE: measuring software engineering processes with teamwork telemetry"	10.1145/3197091.3197107
"Making students read and review code"	10.1145/343048.343090
"Providing a baseline in software process improvement education with lego scrum simulations"	10.1145/3183377.3183378

A.2 Barriers to Adoption Checklist – Version 1.0

Checklist for Identifying Barriers to Adoption of a CS Educational Innovation Paper

Paper Title:

____

Rating: Yes ☑, No/Unsure/Cannot Assess ☐
The rater is advised to take notes on why they chose to select or not select the items

1. **Project type**
   - ☐ Curricular change
   - ☐ Course change
   - ☐ New course
   - ☐ Technology
   - ☐ Other: ________________________
2. **Course components mentioned in the paper**
   - ☐ Course topic schedule
   - ☐ Lectures
   - ☐ Readings (e.g. assigned readings)
   - ☐ In-class activities (e.g. peer instruction, student presentations, discussions)
   - ☐ Videos (e.g. for flipped classrooms)
   - ☐ Individual assignments (e.g. homework, programming assignments)
   - ☐ Group projects
   - ☐ Labs (e.g. to introduce technologies or to take a test on a computer)
   - ☐ Required software (e.g. web application, LMS, custom software)
   - ☐ Other: ________________
3. **Actual course materials or equivalently complete examples publicly available in English (in the paper or linked in the paper)**
   A title is sufficient if googling the title leads to the resource
   Rate "No" for materials NOT selected under 2
   - ☐ Course topic schedule (e.g. course topics by week)
   - ☐ Lecture materials (e.g. slides)
   - ☐ Reading materials
   - ☐ In-class activity materials (e.g. clicker questions)
   - ☐ Video materials
   - ☐ Assignment materials
   - ☐ Project materials
   - ☐ Lab materials
   - ☐ Grading strategies (e.g. how to autograde the proposed assignments/projects/labs)
   - ☐ All required software
   - ☐ User manuals/documentation for software
   - ☐ Other: _________________
4. **Expertise requirements for implementing this innovation**
   (look for expertise mentioned in the paper and on the author's university profile page)
   - ☐ The author has (or collaborates with someone who has) a level of expertise in _____________________________, beyond what is typically taught in a Bachelor CS degree.
   - ☐ The author provides or links information in the paper to help adopters learn about the topic
5. **Monetary requirements for implementing this innovation. The paper mentions the author uses**
   (google the products mentioned in the paper)
   - ☐ Products/services that the author or students had to pay for themselves (e.g. books, software)
   - ☐ Other monetary requirements: _________________

6. **Hardware requirements for implementing this innovation. The paper mentions the author uses**
   - ☐ Raspberry Pi, Arduino or similarly priced hardware
   - ☐ Generic web servers
   - ☐ Self-designed or self-assembled hardware
   - ☐ Other hardware: ___________
7. **Teaching assistant requirements for implementing this innovation. The paper reports**
   - ☐ The author uses ___ teaching assistants per ___ students
   - ☐ The author's teaching assistants have a level of expertise in ___________________, beyond what is typically taught in a CS degree
8. **Collaboration requirements for implementing this innovation. The paper reports the author collaborates with ____________________ *(do NOT count TAs as collaborators)*.**
   - ☐ A professor in the their own department (e.g. guest speaker, topic expert)
   - ☐ A professor from the ___________________ department at their university
   - ☐ ________________ at their university (e.g. writing center, library, administrators etc.)
   - ☐ Another university
   - ☐ Industry (e.g. guest speaker, co-op program, local business as customer, etc.)
   - ☐ The open source community (e.g. students contribute to a popular open source project)
   - ☐ Other: ______________
9. **The paper reports the innovation has been used**
   - ☐ On a maximum class size of _____ students
   - ☐ On a minimum class size of _____ students
   - ☐ In multiple offerings of the same course
   - ☐ In multiple different courses
   - ☐ By multiple instructors
   - ☐ At multiple institutions
   - ☐ In multiple countries
10. **The paper reports statistical evidence**
    - ☐ That shows the innovation is beneficial
11. **Potential scalability issues**
    - ☐ Class size is **NOT** mentioned in paper
    - ☐ The paper does **NOT** report the innovation was used on a large class (≥200 students)
    - ☐ Requires manual grading
    - ☐ Requires a significant amount of reading for grading (e.g. grading reports)
    - ☐ Requires students to give presentations
    - ☐ Requires in-class interaction between students (e.g. in-class group work or discussions)
    - ☐ Requires all students to work at location at the same time (e.g. timed computer lab assessment)
    - ☐ Requires students to have real employment (e.g. internship, co-op program)
    - ☐ Requires students to interact with a real customer or product owner (e.g. local business or organization, open source project maintainer)
    - ☐ Not every student can participate due to limited availability of spots (e.g. tutor program)
12. **Other potential barriers to adoption:**

____

____

Bibliography

[1] Docker. URL `https://www.docker.com/`.

[2] Faculty survey. URL `https://paperdata.page.link/faculty-survey`.

[3] Faculty interview questions. URL `https://paperdata.page.link/faculty-interview-questions`.

[4] Solutions. URL `https://paperdata.page.link/solutions`.

[5] Times higher education ranking. URL `https://www.timeshighereducation.com/world-university-rankings/2020/subject-ranking/computer-science`.

[6] Course and professor evaluations (cape). URL `http://cape.ucsd.edu/`.

[7] Statistics for advanced placement (ap) computer science students in high schools. URL `https://code.org/promote/ap`.

[8] Prerequisite surveys. URL `https://paperdata.page.link/surveys`.

[9] University data – data usa. 2019. URL `https://datausa.io`.

[10] Mikel Aickin and Helen Gensler. Adjusting for multiple testing when reporting research results: the bonferroni vs holm methods. *American journal of public health*, 1996.

[11] Gerlese S. Åkerlind. Variation and commonality in phenomenographic research methods. *Higher Education Research & Development*, 2005.

[12] Ahmed Al-Zubidy, Jeffrey C. Carver, Sarah Heckman, and Mark Sherriff. A (updated) review of empiricism at the sigcse technical symposium. In *ACM SIGCSE Technical Symposium on Computer Science Education (SIGCSE)*, 2016.

[13] Nurul Ezza Asyikin Mohamed Almi, Najwa Abdul Rahman, Durkadavi Purusothaman, and Shahida Sulaiman. Software engineering education: The gap between industry's requirements and graduates' readiness. In *IEEE Symposium on Computers & Informatics*, 2011.

[14] Vicki L. Almstrum, Peter B. Henderson, Valerie Harvey, Cinda Heeren, William Marion, Charles Riedesel, Leen-Kiat Soh, and Allison Elliott Tew. Concept inventories in computer science for the topic discrete mathematics. *SIGCSE Bulletin*, 2006.

[15] Amnah Alshahrani, Isla Ross, and Murray I. Wood. Using social cognitive career theory to understand why students choose to study computer science. In *ACM Conference on International Computing Education Research (ICER)*, 2018.

[16] Christine Alvarado, Gustavo Umbelino, and Mia Minnes. The persistent effect of pre-college computing experience on college cs course grades. In *ACM SIGCSE Technical Symposium on Computer Science Education (SIGCSE)*, 2018.

[17] Mark Ardis, David Budgen, Gregory W Hislop, Jeff Offutt, Mark Sebern, and Willem Visser. Se 2014: Curriculum guidelines for undergraduate degree programs in software engineering. *Computer*, 2015.

[18] Gloria Honny Asirifi, Vida Doku, Sarah Morrison, and Augustina Sackeley Sackey. The gap between the hospitality education and hospitality industry. *Journal of Education and Practice (JEP)*, 2013.

[19] Alan D Baddeley and Alan D Baddeley. *Your memory: A user's guide*. Carlton Books New York, NY, USA:, 2004.

[20] Kathy Beckman, Neal Coulter, Soheil Khajenoori, and Nancy R Mead. Collaborations: closing the industry-academia gap. *IEEE Software*, 1997.

[21] Andrew Begel and Beth Simon. Novice software developers, all over again. In *ACM Conference on International Computing Education Research (ICER)*, 2008.

[22] Andrew Begel and Beth Simon. Struggles of new college graduates in their first software development job. In *ACM SIGCSE Technical Symposium on Computer Science Education (SIGCSE)*, 2008.

[23] Emma-Jane Berridge, Della Freeth, Judi Sharpe, and C. Michael Roberts. Bridging the gap: supporting the transition from medical student to practising doctor. *Medical Teacher*, 2007.

[24] Maura Borrego, Stephanie Cutler, Michael Prince, Charles Henderson, and Jeffrey E. Froyd. Fidelity of implementation of research-based instructional strategies (rbis) in engineering science courses. *Journal of Engineering Education (JEE)*, 2013.

[25] Eric Brechner. Things they would not teach me of in college: What microsoft developers learn later. In *Companion of SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, 2003.

[26] Emanuelle Burton, Judy Goldsmith, and Nicholas Mattei. How to teach computer ethics through science fiction. *Communications of the ACM*, 2018.

[27] Tracy Camp, W. Richards Adrion, Betsy Bizot, Susan Davidson, Mary Hall, Susanne Hambrusch, Ellen Walker, and Stuart Zweben. Generation cs: The growth of computer science. *ACM Inroads*, 2017.

[28] Angela Carbone, Linda Mannila, and Sue Fitzgerald. Computer science and it teachers' conceptions of successful and unsuccessful teaching: A phenomenographic study. *Computer Science Education*, 2007.

[29] Gail Carmichael, Christine Jordan, Andrea Ross, and Alison Evans Adnani. Curriculum-aligned work-integrated learning: A new kind of industry-academic degree partnership. In *ACM SIGCSE Technical Symposium on Computer Science Education (SIGCSE)*, 2018.

[30] Secil Caskurlu, Iryna Ashby, and Marisa Exter. The alignment between formal education and software design professionals' needs in industry: Faculty perception. In *ASEE Annual Conference & Exposition*, 2017.

[31] Allan Collins, John Seely Brown, and Ann Holum. Cognitive apprenticeship: Making thinking visible. *American educator*, 1991.

[32] Michelle Craig, Phill Conrad, Dylan Lynch, Natasha Lee, and Laura Anthony. Listening to early career software developers. *Journal of Computing Sciences in Colleges*, 2018.

[33] Eugène J.F.M. Custers. Long-term retention of basic science knowledge: A review study. *Advances in Health Sciences Education*, 2010.

[34] Holger Danielsiek, Wolfgang Paul, and Jan Vahrenhold. Detecting and understanding students' misconceptions related to algorithms and data structures. In *ACM SIGCSE Technical Symposium on Computer Science Education (SIGCSE)*, 2012.

[35] Benjamin J. Drury, John Oliver Siy, and Sapna Cheryan. When do female role models benefit women? the importance of differentiating recruitment from retention in stem. *Psychological Inquiry*, 2011.

[36] Richard O. Duda, Peter E. Hart, and Nils J. Nilsson. Subjective bayesian methods for rule-based inference systems. In *National Computer Conference and Exposition*, 1976.

[37] Elisabeth Dunne and Mike Rawlins. Bridging the gap between industry and higher education. *Innovations in Education and Training international (IETI)*, 2000.

[38] Stephen H. Edwards. Rethinking computer science education from a test-first perspective. In *Companion of ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, 2003.

[39] Maria Lydia Fioravanti, Bruno Sena, Leo Natan Paschoal, Laíza R. Silva, Ana P. Allian, Elisa Y. Nakagawa, Simone R.S. Souza, Seiji Isotani, and Ellen F. Barbosa. Integrating project based learning and project management for software engineering teaching. In *ACM SIGCSE Technical Symposium on Computer Science Education (SIGCSE)*, 2018.

[40] Allan Fisher and Jane Margolis. Unlocking the clubhouse: The carnegie mellon experience. *SIGCSE Bulletin*, 2002.

[41] Kunze Florian, Boehm Stephan, and Bruch Heike. Age, resistance to change, and job performance. *Journal of Managerial Psychology*, 2013.

[42] Karen A. Frenkel. Women and computing. *Communications of the ACM*, 1990.

[43] Kirti Garg and Vasudeva Varma. A study of the effectiveness of case study approach in software engineering education. In *International Conference on Software Engineering Education & Training (CSEE&T)*, 2007.

[44] Sharon Gedye, Elizabeth Fender, and Brian Chalkley. Students' undergraduate expectations and post-graduation experiences of the value of a degree. *Journal of Geography in Higher Education*, 2004.

[45] E. Giangrande. Communication skills in the cs curriculum. *Journal of Computing Sciences in Colleges*, 2009.

[46] GitHub. Codespaces. URL `https://github.com/features/codespaces`.

[47] Tina Götschi, Ian Sanders, and Vashti Galpin. Mental models of recursion. In *ACM SIGCSE Technical Symposium on Computer Science Education (SIGCSE)*, 2003.

[48] Mark Guzdial. Exploring hypotheses about media computation. In *ACM Conference on International Computing Education Research (ICER)*, 2013.

[49] Mark Guzdial. Learner-centered design of computing education: Research on computing for everyone. *Synthesis Lectures on Human-Centered Informatics*, 2015.

[50] David C Haak, Janneke HilleRisLambers, Emile Pitre, and Scott Freeman. Increased structure and active learning reduce the achievement gap in introductory biology. *Science*, 2011.

[51] Hisham Haddad. Post-graduate assessment of cs students: Experience and position paper. *Journal of Computing Sciences in Colleges*, 2002.

[52] Dianne Hagan. Employer satisfaction with ict graduates. In *Australasian Conference on Computing Education*, 2004.

[53] Sally Hamouda, Stephen H. Edwards, Hicham G. Elmongui, Jeremy V. Ernst, and Clifford A. Shaffer. A basic recursion concept inventory. *Computer Science Education*, 2017.

[54] Qiang Hao, David H Smith IV, Naitra Iriumi, Michail Tsikerdekis, and Andrew J Ko. A systematic investigation of replications in computing education research. *ACM Transactions on Computing Education (TOCE)*, 2019.

[55] C. Richard G. Helps, Robert B. Jackson, and Marshall B. Romney. Student expectations of computing majors. In *Proceedings of the 6th Conference on Information Technology Education*, 2005.

[56] Charles Henderson and Melissa H Dancy. Barriers to the use of research-based instructional strategies: The influence of both individual and situational characteristics. *Physical Review Special Topics-Physics Education Research*, 2007.

[57] Charles Henderson and Melissa H Dancy. Increasing the impact and diffusion of stem education innovations. In *National Academy of Engineering*, 2011.

[58] Charles Henderson, Andrea Beach, and Noah Finkelstein. Facilitating change in undergraduate stem instructional practices: An analytic review of the literature. *Journal of research in science teaching*, 2011.

[59] Geoffrey L. Herman, Michael C. Loui, and Craig Zilles. Students' misconceptions about medium-scale integrated circuits. *IEEE Transactions on Education*, 2011.

[60] Christopher Lynnly Hovey, Lecia Barker, and Vaughan Nagy. Survey results on why cs faculty adopt new teaching practices. In *ACM SIGCSE Technical Symposium on Computer Science Education (SIGCSE)*, 2019.

[61] Association for Computing Machinery (ACM) Joint Task Force on Computing Curricula and IEEE Computer Society. *Computer Science Curricula 2013: Curriculum Guidelines for Undergraduate Degree Programs in Computer Science*. ACM, 2013.

[62] Amanpreet Kapoor and Christina Gardner-McCune. Understanding cs undergraduate students' professional development through the lens of internship experiences. In *ACM SIGCSE Technical Symposium on Computer Science Education (SIGCSE)*, 2019.

[63] Raina Khatri, Charles Henderson, Renée Cole, Jeffrey E Froyd, Debra Friedrichsen, and Courtney Stanford. Designing for sustained adoption: A model of developing educational innovations for successful propagation. *Physical Review Physics Education Research*, 2016.

[64] Päivi Kinnunen and Beth Simon. Phenomenography and grounded theory as research methods in computing education research field. *Computer Science Education*, 2012.

[65] Jean Lave and Etienne Wenger. *Situated learning: Legitimate peripheral participation*. Cambridge university press, 1991.

[66] Wolfgang Lehmann. University as vocational education: working-class students' expectations for university. *British Journal of Sociology of Education*, 2009.

[67] Timothy C. Lethbridge. The relevance of software education: A survey and some recommendations. *Annals of Software Engineering*, 1998.

[68] Clayton Lewis. Attitudes and beliefs about computer science among students and faculty. *SIGCSE Bulletin*, 2007.

[69] Clayton Lewis, Michele H. Jackson, and William M. Waite. Student and faculty attitudes and beliefs about computer science. *Communications of the ACM*, 2010.

[70] Alex Lishinski, Aman Yadav, Jon Good, and Richard Enbody. Learning to program: Gender differences and interactive effects of students' motivation, goals, and self-efficacy on performance. In *ACM Conference on International Computing Education Research (ICER)*, 2016.

[71] Lauri Malmi, Judy Sheard, Roman Bednarik, Juha Helminen, Ari Korhonen, Niko Myller, Juha Sorva, and Ahmad Taherkhani. Characterizing research in computing education: a preliminary analysis of the literature. In *International workshop on Computing education research*, 2010.

[72] Marcia A. Mardis, Jinxuan Ma, Faye R. Jones, Chandrahasa R. Ambavarapu, Heather M. Kelleher, Laura I. Spears, and Charles R. McClure. Assessing alignment between information technology educational opportunities, professional requirements, and industry demands. *Education and Information Technologies*, 2018.

[73] Jane Margolis, Rachel Estrella, Joanna Goode, Jennifer Jellison Holme, and Kim Nao. *Stuck in the shallow end: Education, race, and computing*. MIT press, 2017.

[74] Marshall A Martin. Course prerequisites and undergraduate student performance. *NACTA Journal*, 1989.

[75] Ference Marton. Phenomenography — describing conceptions of the world around us. *Instructional Science*, 1981.

[76] Eric Mazur. Peer instruction: Getting students to think in class. *AIP Conference Proceedings*, 1997.

[77] Monica M McGill, Adrienne Decker, and Zachary Abbott. Improving research and experience reports of pre-college computing activities: A gap analysis. In *ACM SIGCSE Technical Symposium on Computer Science Education (SIGCSE)*, 2018.

[78] Peter McHardy and Teresa Allan. Closing the gap between what industry needs and what he provides. *Education + Training*, 2000.

[79] Microsoft. Visual studio code. URL `https://code.visualstudio.com/`.

[80] MIT. The missing semester of your cs education. URL `https://missing.csail.mit.edu/`.

[81] Karthik Muralidharan, Abhijeet Singh, and Alejandro J Ganimian. Disrupting education? experimental evidence on technology-aided instruction in india. Technical report, National Bureau of Economic Research, 2016.

[82] Stephen E Newstead, Arlene Franklyn-Stokes, and Penny Armstead. Individual differences in student cheating. *Journal of Educational Psychology*, 1996.

[83] Pan-Wei Ng and Shihong Huang. Essence: A framework to help bridge the gap between software engineering education and industry needs. In *International Conference on Software Engineering Education and Training (CSEE&T)*, 2013.

[84] C. Norton and T. Martini. Perceived benefits of an undergraduate degree. *The Canadian Journal for the Scholarship of Teaching and Learning*, 2017.

[85] ACM/IEEE-CS Joint Task Force on Computing Curricula. Computer science curricula 2013. 2013. URL `http://cra.org/data/Generation-CS/`.

[86] Karen Owens. Community college transfer students' adjustment to a four-year institution: A qualitative analysis. *Journal of The First-Year Experience & Students in Transition*, 2010.

[87] Miranda C Parker, Mark Guzdial, and Shelly Engleman. Replication, validation, and use of a language independent cs1 knowledge assessment. In *ACM Conference on International Computing Education Research (ICER)*, 2016.

[88] Jacob Perrenet. Differences in beliefs and attitudes about computer science among students and faculty of the bachelor program. In *ACM SIGCSE Conference on Innovation and Technology in Computer Science Education (ITiCSE)*, 2009.

[89] Anne-Kathrin Peters, Anders Berglund, Anna Eckerdal, and Arnold Pears. Second year computer science and it students' experience of participation in the discipline. In *Koli Calling Conference on Computing Education Research*, 2015.

[90] R. Pham, S. Kiesling, L. Singer, and K. Schneider. Onboarding inexperienced developers: struggles and perceptions regarding automated testing. *Software Quality Journal*, 2016.

[91] Leo Porter, Saturnino Garcia, Hung-Wei Tseng, and Daniel Zingaro. Evaluating student understanding of core concepts in computer architecture. In *ACM SIGCSE Conference on Innovation and Technology in Computer Science Education (ITiCSE)*, 2013.

[92] Leo Porter, Daniel Zingaro, Soohyun Nam Liao, Cynthia Taylor, Kevin C. Webb, Cynthia Lee, and Michael Clancy. Bdsi: A validated concept inventory for basic data structures. In *ACM Conference on International Computing Education Research (ICER)*, 2019.

[93] Alex Radermacher and Gursimran Walia. Gaps between industry expectations and the abilities of graduates. In *ACM SIGCSE Technical Symposium on Computer Science Education (SIGCSE)*, 2013.

[94] Alex Radermacher, Gursimran Walia, and Dean Knudson. Investigating the skill gap between graduating students and industry expectations. In *IEEE/ACM International Conference on Software Engineering (ICSE)*, 2014.

[95] Vennila Ramalingam, Deborah LaBelle, and Susan Wiedenbeck. Self-efficacy and mental models in learning to program. *SIGCSE Bulletin*, 2004.

[96] Adam Rosenstein, Aishma Raghu, and Leo Porter. Identifying the prevalence of the impostor phenomenon among computer science students. In *ACM SIGCSE Technical Symposium on Computer Science Education (SIGCSE)*, 2020.

[97] Mehran Sahami and Steve Roach. Computer science curricula 2013 released. *Communications of the ACM*, 2014.

[98] Adrian Salguero, Julian McAuley, Beth Simon, and Leo Porter. A longitudinal evaluation of a best practices cs1. In *ACM Conference on International Computing Education Research (ICER)*, 2020.

[99] Kate Sanders, Judy Sheard, Brett A. Becker, Anna Eckerdal, Sally Hamouda, and Simon. Inferential statistics in computing education research: A methodological review. In *ACM Conference on International Computing Education Research (ICER)*, 2019.

[100] Linda J. Sax, Jennifer M. Blaney, Kathleen J. Lehman, Sarah L. Rodriguez, Kari L. George, and Christina Zavala. Sense of belonging in computing: The role of introductory courses for women and underrepresented minority students. *Social Sciences*, 2018.

[101] Zalia Shams. Automated assessment of students' testing skills for improving correctness of their code. In *Companion publication for conference on Systems, programming, & applications: software for humanity*, 2013.

[102] Judy Sheard, Angela Carbone, Raymond Lister, Beth Simon, Errol Thompson, and Jacqueline L. Whalley. Going solo to assess novice programmers. In *ACM SIGCSE Conference on Innovation and Technology in Computer Science Education (ITiCSE)*, 2008.

[103] Kim Bartel Sheehan. E-mail Survey Response Rates: a Review. *Journal of Computer-Mediated Communication*, 2001.

[104] Gary Siegel, James E. Sorensen, Thomas Klammer, and Sandra B. Richtermeyer. The ongoing preparation gap in accounting education: A call to action. *Management Accounting Quarterly*, 2010.

[105] SIGCSE. Reviewer guidelines, 2021. URL `https://sigcse2021.sigcse.org/reviewers/paper-review-guidelines/`.

[106] Simon and Judy Sheard. Academic integrity and computing assessments. In *Australasian Computer Science Week Multiconference*, 2016.

[107] Simon, Michael de Raadt, Ken Sutton, and Anne Venables. The distinctive role of lab practical classes in computing education. In *Koli Calling Baltic Sea Conference on Computing Education Research*, 2006.

[108] Elliot Soloway. Learning to program= learning to construct mechanisms and explanations. *Communications of the ACM*, 1986.

[109] Sulayman K. Sowe, Ioannis Stamelos, and Ignatios Deligiannis. A framework for teaching software testing using f/oss methodology. In *Open Source Systems*, 2006.

[110] Marilyne Stains and Trisha Vickrey. Fidelity of implementation: An overlooked yet critical construct to establish effectiveness of evidence-based instructional practices. *CBE—Life Sciences Education*, 2017.

[111] Courtney Stanford, Renée Cole, Jeff Froyd, Debra Friedrichsen, Raina Khatri, and Charles Henderson. Supporting sustained adoption of education innovations: The designing for sustained adoption assessment instrument. *International Journal of STEM Education*, 2016.

[112] Courtney Stanford, Renee Cole, Jeff Froyd, Charles Henderson, Debra Friedrichsen, and Raina Khatri. Analysis of propagation plans in nsf-funded education development projects. *Journal of Science Education and Technology*, 2017.

[113] Bjarne Stroustrup. Viewpoint: What should we teach new software developers? Why? *Communications of the ACM*, 2010.

[114] Bjarne Stroustrup. What should we teach new software developers? why? vol. 53 № 1. *Communications of the ACM*, 2010.

[115] Leigh Ann Sudol and Ciera Jaspan. Analyzing the strength of undergraduate misconceptions about software engineering. In *International Workshop on Computing Education Research*, 2010.

[116] Darryl K. Taft. Programming grads meet a skills gap in the real world, 2007. URL `https://www.eweek.com/development/programming-grads-meet-a-skills-gap-in-the-real-world/`.

[117] Cynthia Taylor, Daniel Zingaro, Leo Porter, Kevin C Webb, Cynthia Bailey Lee, and Mike Clancy. Computer science concept inventories: past and future. *Computer Science Education*, 2014.

[118] Cynthia Taylor, Jaime Spacco, David P. Bunde, Zack Butler, Heather Bort, Christopher Lynnly Hovey, Francesco Maiorana, and Thomas Zeume. Propagating the adoption of cs educational innovations. In *ITiCSE Companion*, 2018.

[119] Cynthia Taylor, Jaime Spacco, David P. Bunde, Thomas Zeume, Zack Butler, Martina Barnas, Heather Bort, Francesco Maiorana, and Christopher Lynnly Hovey. Promoting the adoption of educational innovations. In *ACM SIGCSE Conference on Innovation and Technology in Computer Science Education (ITiCSE)*, 2018.

[120] Matthew F. Tennyson and Marc Beck. A study of knowledge retention in introductory programming courses. *Journal of Computing Sciences in Colleges*, 2018.

[121] Allison Elliott Tew and Mark Guzdial. The fcs1: A language independent assessment of cs1 knowledge. In *ACM SIGCSE Technical Symposium on Computer Science Education (SIGCSE)*, 2011.

[122] Increase the Impact. Design for sustained adoption assessment instrument (dsaai), . URL `http://www.increasetheimpact.com/resources.html`.

[123] Increase the Impact. Sustainable innovation, . URL `http://www.increasetheimpact.com/publications.html`.

[124] Elli J Theobald, Mariah J Hill, Elisa Tran, Sweta Agrawal, E Nicole Arroyo, Shawn Behling, Nyasha Chambwe, Dianne Laboy Cintrón, Jacob D Cooper, Gideon Dunster, et al. Active learning narrows achievement gaps for underrepresented students in undergraduate science, technology, engineering, and math. *Proceedings of the National Academy of Sciences*, 2020.

[125] Chandra Turpen, Melissa Dancy, and Charles Henderson. Perceived affordances and constraints regarding instructors' use of peer instruction: Implications for promoting instructional change. *Physical Review Physics Education Research*, 2016.

[126] Smrithi Rekha V and V. Adinarayanan. An open source approach to enhance industry preparedness of students. In *International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, 2014.

[127] Sander Valstar, William G. Griswold, and Leo Porter. Using devcontainers to standardize student development environments: An experience report. In *ACM Conference on Innovation and Technology in Computer Science Education*, 2020.

[128] JD Vitale, SP Wanger, and DC Adams. Explaining student performance in an undergraduate agricultural economics classroom. *NACTA Journal*, 2010.

[129] Lev Vygotsky. Interaction between learning and development. *Readings on the development of children*, 1978.

[130] Henry M. Walker. Prerequisites: Shaping the computing curriculum. *ACM Inroads*, 2010.

[131] Emily M. Walter, Charles R. Henderson, Andrea L. Beach, and Cody T. Williams. Introducing the postsecondary instructional practices survey (pips). *CBE—Life Sciences Education*, 2016.

[132] Christopher Watson, Frederick W.B. Li, and Jamie L. Godwin. No tests required: Comparing traditional and dynamic predictors of programming success. In *ACM SIGCSE Technical Symposium on Computer Science Education (SIGCSE)*, 2014.

[133] Kevin C. Webb and Cynthia Taylor. Developing a pre- and post-course concept inventory to gauge operating systems learning. In *ACM SIGCSE Technical Symposium on Computer Science Education (SIGCSE)*, 2014.

[134] Linda L Werner, Brian Hanks, and Charlie McDowell. Pair-programming helps female computer science students. *Journal on Educational Resources in Computing (JERIC)*, 2004.

[135] Carl Wieman and Sarah Gilbert. The teaching practices inventory: A new tool for characterizing college and university teaching in mathematics and science. *CBE—Life Sciences Education*, 2014.

[136] Anna Wilson, Susan Howitt, Pam Roberts, Gerlese Åkerlind, and Kate Wilson. Connecting expectations and experiences of students in a research-immersive degree. *Studies in Higher Education*, 2013.

[137] Brenda Cantwell Wilson and Sharon Shrock. Contributing to success in an introductory computer science course: a study of twelve factors. *SIGCSE Bulletin*, 2001.

[138] Daniel Zingaro and Leo Porter. Impact of student achievement goals on cs1 outcomes. In *ACM SIGCSE Technical Symposium on Computer Science Education (SIGCSE)*, 2016.

[139] Daniel Zingaro, Michelle Craig, Leo Porter, Brett A Becker, Yingjun Cao, Phill Conrad, Diana Cukierman, Arto Hellas, Dastyni Loksa, and Neena Thota. Achievement goals in CS1: Replication and extension. In *ACM SIGCSE Technical Symposium on Computer Science Education (SIGCSE)*, 2018.