

## Old Dominion University ODU Digital Commons

---

STEMPS Theses & Dissertations

STEM Education & Professional Studies


---

Summer 2016

# Defining the Competencies, Programming Languages, and Assessments for an Introductory Computer Science Course

Simon Sultana  
*Old Dominion University*

Follow this and additional works at: [https://digitalcommons.odu.edu/stemps\\_etds](https://digitalcommons.odu.edu/stemps_etds)

 Part of the [Curriculum and Instruction Commons](#), [Educational Assessment, Evaluation, and Research Commons](#), and the [Programming Languages and Compilers Commons](#)

---

### Recommended Citation

Sultana, Simon. "Defining the Competencies, Programming Languages, and Assessments for an Introductory Computer Science Course" (2016). Doctor of Philosophy (PhD), dissertation, STEM and Professional Studies, Old Dominion University, DOI: 10.25777/sgra-pa16  
[https://digitalcommons.odu.edu/stemps\\_etds/10](https://digitalcommons.odu.edu/stemps_etds/10)

This Dissertation is brought to you for free and open access by the STEM Education & Professional Studies at ODU Digital Commons. It has been accepted for inclusion in STEMPS Theses & Dissertations by an authorized administrator of ODU Digital Commons. For more information, please contact [digitalcommons@odu.edu](mailto:digitalcommons@odu.edu).

DEFINING THE COMPETENCIES, PROGRAMMING LANGUAGES, AND  
ASSESSMENTS FOR AN INTRODUCTORY COMPUTER SCIENCE COURSE

by

Simon Sultana

B.S. July 1995, The University of Michigan

M.S. May 2000, Wayne State University

M.B.A. May 2003, Wayne State University

A Dissertation Submitted to the Faculty of  
Old Dominion University in Partial Fulfillment of the  
Requirements for the Degree of

DOCTOR OF PHILOSOPHY

EDUCATION

OLD DOMINON UNIVERSITY

August, 2016

Approved by:

Philip A. Reed (Director)

Darryl C. Draper (Member)

Ginger Watson (Member)

## ABSTRACT

### DEFINING THE COMPETENCIES, PROGRAMMING LANGUAGES, AND ASSESSMENTS FOR AN INTRODUCTORY COMPUTER SCIENCE COURSE

Simon Sultana  
Old Dominion University, 2016  
Director: Dr. Philip A. Reed

The purpose of this study was to define the competencies, programming languages, and assessments for an introductory computer science course at a small private liberal arts university. Three research questions were addressed that involved identifying the competencies, programming languages, and assessments that academic and industry experts in California's Central Valley felt most important and appropriate for an introduction to computer science course.

The Delphi methodology was used to collect data from the two groups of experts with various backgrounds related to computing. The goal was to find consensus among the individual groups to best define aspects that would best comprise an introductory CS0 course for majors and non-majors. The output would be valuable information to be considered by curriculum designers who are developing a new program in software engineering at the institution. The process outlined would also be useful to curriculum designers in other fields and geographic regions who attempt to address their local education needs.

Four rounds of surveys were conducted. The groups of experts were combined in the first round to rate the items in the straw models determined from the literature and add additional components when necessary. The academic and industry groupings were separated for the remainder of the study so that a curriculum designer could determine not only the items deemed most important, but also their relative importance among the two distinct groups. The experts

selected items in each of the three categories in the second round to reduce the possibilities for subsequent rounds. The groups were then asked to rank the items in each of the three categories for the third round. A fourth round was held as consensus was not reached by either of the groups for any of the categories as determined by Kendall's  $W$ . The academic experts reached consensus on a list of ranked competencies in the final round and showed a high degree of agreement on lists of ranked programming languages and assessments. Kendall's  $W$ , values, however, were just short of the required 0.7 threshold for consensus on these final two items. The industry experts did not reach consensus and showed low agreement on their recommendations for competencies, programming languages, and assessments.

This work is dedicated to my wife and children. First and foremost, I thank you Jenny for your support and understanding during the Ph.D. process. I know this was as much an undertaking for you as it was for me. I appreciate the sacrifices you made and the extra work you took on so that I could focus on my studies and research. I look forward to returning the favor as you further your education. Brittany, Nicholas, and Natalie, I hope I have inspired you to always pursue learning. Never stop. Also remember to always take time for what is most important in your lives.

## ACKNOWLEDGEMENTS

I offer thanks to God for blessing me during this process and throughout my life.

I want to thank all the faculty I have had the pleasure to meet at the School of Education at Old Dominion University. You have been valuable teachers and advisors and have taught me much. Your efforts helped me to persevere through the difficult undertaking of beginning a new job at a critical time in my doctoral studies. Thank you for all you do.

To my advisor, Dr. Phil Reed, I thank you for sharing your time so generously. You taught me from my first class in this program that distance learning can be viable only through the dedication of the instructor. I am especially grateful for your counsel as I transitioned from one job to another. To Dr. Deri Draper, thank you for your advice and encouragement as I tackled my new responsibilities of curriculum design. Dr. Ginger Watson, thank you for your contributions to this research and for your valuable insight.

I thank Dr. John Ritz. You are an inspirational educator and convinced me I wanted to be a Ph.D. for life. I would also like to express thanks to the students I met during course work and summer institutes. To Brian Preble, thanks for sharing ideas, counsel, and encouragement. Dr. Diana Cantu, thank you for your support and advice. I also want to thank Dr. Masud Mansuri for being a mentor and a friend.

Finally, I want to thank the administration at DeVry University for reimbursing my tuition expenses for my coursework, and those at Fresno Pacific University for the continued support from the comprehensive exam stage through the dissertation process. Thank you also for the opportunity to develop curriculum for two exciting programs I hope will help many students achieve their career goals.

## TABLE OF CONTENTS

	Page
LIST OF TABLES .....	x
Chapter	
I. INTRODUCTION .....	1
STATEMENT OF PROBLEM.....	3
RESEARCH QUESTIONS .....	4
BACKGROUND AND SIGNIFICANCE.....	5
LIMITATIONS.....	9
ASSUMPTIONS.....	10
PROCEDURES.....	11
DEFINITION OF TERMS .....	13
SUMMARY AND OVERVIEW .....	14
II. REVIEW OF LITERATURE.....	16
PREPARATION OF PROGRAMMERS AND SOFTWARE ENGINEERS.....	16
DEFINITION OF COMPUTER SCIENCE .....	16
OCCUPATIONS.....	17
BRIEF HISTORY OF UNDERGRADUATE STUDY OF COMPUTER SCIENCE.....	18
FORMAL EDUCATION IN COMPUTING.....	22
STAKEHOLDERS IN COMPUTER SCIENCE EDUCATION .....	24
INDUSTRY .....	25
ACADEMIA.....	26
INTRODUCTORY COMPUTER SCIENCE COURSES.....	30
GENERAL SKILLS .....	32
COMPETENCIES .....	35
TOPICS IDENTIFIED FROM PROFESSIONAL ASSOCIATIONS AND TEXTBOOKS.....	36
TOPICS IDENTIFIED IN JOURNAL ARTICLES.....	39

TOPICS IN PROGRAMMING .....	41
PROGRAMMING FUNDAMENTALS .....	42
DEVELOPING PROGRAMS .....	43
APPLICATIONS, TECHNIQUES, AND PROCESSES .....	44
TOPICS IN HARDWARE AND OTHER LEVELS OF THE OSI MODEL .....	46
PROFESSIONAL SKILLS.....	48
TEAMWORK AND COLLABORATION .....	48
PROBLEM SOLVING AND RELATED ATTRIBUTES.....	50
WAYS OF THINKING .....	51
OTHER PROFESSIONAL ATTRIBUTES .....	53
STRAW MODEL OF COMPETENCIES .....	54
PROGRAMMING LANGUAGES.....	57
OVERVIEW .....	57
LANGUAGE POPULARITY IN INDUSTRY .....	60
LANGUAGE POPULARITY IN ACADEMIA.....	61
STRAW MODEL OF PROGRAMMING LANGUAGES .....	66
ASSESSMENTS.....	68
SUMMARY .....	75
III. METHODOLOGY .....	78
PARTICIPANTS .....	78
DESIGN.....	80
RECRUIT AND IDENTIFY PARTICIPANTS .....	81
DEVELOP STRAW MODELS .....	83
DESIGN AND DISTRIBUTE ROUND 1 SURVEY TO BOTH GROUPS.....	84
ROUND 1 REVIEW AND ANALYSIS .....	85
DESIGN AND DISTRIBUTE ROUND 2 SELECTION SURVEY .....	85
ROUND 2 REVIEW AND ANALYSIS .....	86
DESIGN, DISTRIBUTE, AND ANALYZE ROUND 3 AND 4 SURVEYS .....	87
SUMMARY .....	88



IV. RESULTS .....	90
PARTICIPANTS AND DEMOGRAPHICS .....	90
ROUND 1 .....	92
COURSE COMPETENCIES .....	92
PROGRAMMING LANGUAGES.....	97
ASSESSMENTS.....	99
ROUND 2 .....	100
COURSE COMPETENCIES .....	101
PROGRAMMING LANGUAGES.....	101
ASSESSMENTS.....	105
ROUND 3 .....	105
COURSE COMPETENCIES .....	107
PROGRAMMING LANGUAGES.....	107
ASSESSMENTS.....	109
GROUP CONCORDANCE.....	110
ROUND 4 .....	112
COURSE COMPETENCIES .....	112
PROGRAMMING LANGUAGES.....	115
ASSESSMENTS.....	116
GROUP CONCORDANCE.....	117
V. CONCLUSIONS AND RECOMMENDATIONS .....	119
RECOMMENDATIONS FOR COURSE COMPETENCIES .....	120
RECOMMENDATIONS FOR PROGRAMMING LANGUAGES .....	125
RECOMMENDATIONS FOR ASSESSMENTS .....	129
OVERALL COURSE RECOMMENDATIONS AND FUTURE RESEARCH .....	133
REFERENCES .....	136

APPENDICES .....	166
APPENDIX A: INVITATION TO PARTICIPANTS .....	166
APPENDIX B: HUMAN SUBJECTS INFORMED CONSENT.....	167
APPENDIX C: SUMMARY OF THE STUDY.....	169
APPENDIX D: ROUND 1 QUESTIONNAIRE.....	171
APPENDIX E: ROUND 2 SURVEY .....	177
APPENDIX F: ROUND 3 SURVEY FOR ACADEMIC GROUP .....	180
APPENDIX G: ROUND 3 SURVEY FOR INDUSTRY GROUP .....	182
APPENDIX H: ROUND 4 SURVEY FOR ACADEMIC GROUP .....	184
APPENDIX I: ROUND 4 SURVEY FOR INDUSTRY GROUP .....	186
VITA .....	188

## LIST OF TABLES

Table		Page
1	United States Bureau of Labor Statistics Projected Job Outlook Over the Decade 2012 to 2022.....	2
2	State of California Employment Development Department Projected Job Outlook over the Decade 2012 to 2022 for Fresno County .....	6
3	California State University System Computer Science and Software Engineering Program Enrollment.....	7
4	United States Bureau of Labor Statistics Occupations Related to the Study of Computer Science .....	19
5	Coverage of Potential Topics for Introductory Computer Science.....	40
6	Introduction to Computer Science Competencies Straw Model.....	56
7	Programming Language Popularity Rankings in Software Industry .....	62
8	Programming Language Popularity Rankings in Academia.....	63
9	Programming Language Straw Model .....	67
10	Kendall's <i>W</i> Values Analyzed .....	88
11	Round 1 Expert Feedback on Competencies for Introductory Computer Science .....	93
12	Round 1 Expert Feedback on Programming Languages for Introductory Computer Science .....	98
13	Round 1 Expert Feedback on Assessments for Introductory Computer Science .....	99
14	Round 2 Selection Counts of Competencies for Introductory Computer Science.....	102
15	Round 2 Selection Counts of Programming Languages for Introductory Computer Science .....	104

16	Round 2 Selection Counts of Assessments for Introductory Computer Science.....	105
17	Round 3 Median Rankings of Competencies for Introductory Computer Science .....	108
18	Round 3 Median Rankings of Programming Languages for Introductory Computer Science .....	109
19	Round 3 Median Rankings of Assessments for Introductory Computer Science.....	110
20	Kendall's <i>W</i> Values for Round 3 .....	111
21	Round 4 Median Rankings of Competencies for Introductory Computer Science .....	114
22	Round 4 Median Rankings of Programming Languages for Introductory Computer Science .....	115
23	Round 4 Median Rankings of Assessments for Introductory Computer Science.....	117
24	Kendall's <i>W</i> Values for Round 4 .....	118
25	Top Recommended Competencies for Introductory Computer Science by Both Groups .....	121
26	Round 4 Interquartile Range Values for Suggested Primary and Secondary Competencies .....	125
27	Top Recommended Assessments for Introductory Computer Science by Both Groups	130

## CHAPTER I

### INTRODUCTION

Fields that deal with technology seem to undergo a constant redefinition of their identities. Technology education, for example, has its origins in manual training, which later became industrial arts before arriving at its current state of technology and engineering education (Foster, 1997). The study of computing appears to have some similarity in this respect. Computer science (CS) had its beginnings in mathematics. Humans devised calculating devices and practiced algorithmic thinking back in the days of antiquity (Keller & Volkov, 2014). The field was very much entrenched in science around the time the first computing academic programs were being offered (Denning, 2013), starting with the Diploma in Numerical Analysis and Automatic Computing at the University of Cambridge in 1953 (University of Cambridge, 2004). Purdue University and Stanford University formed CS departments nine years later (Denning, 2013) and the field of academic study was well on its way.

Since that time computing as an academic discipline has experienced changes in focus from science, to technology, and back to science (Denning, 2013). Subsequently, CS, along with electrical engineering and information systems, became three distinct options of formal study (The Joint Task Force for Computing Curricula 2005, 2006). The 1990s saw a great deal of change and computer engineering, software engineering (SE), and information technology (IT) were added to the other three academic program subjects. There is certainly overlap in the coverage of these programs and some debate persists on their exact definitions and boundaries (Kelly, 2007; Kelly, 2013; Lutz, Naveda, & Vallino, 2014).

In this short yet tumultuous history, the demand for computing professionals has continued to grow. Academic programs in CS and SE prepare students for occupations as

applications software developers, systems software developers, computer systems analysts, computer programmers, computer and information systems managers, web developers, and database administrators (Bureau of Labor Statistics [BLS], 2015). The BLS (2015) reports national increases in the demand for these occupations from 2012 to 2022 as seen in Table 1.

Table 1

*United States Bureau of Labor Statistics Projected Job Outlook Over the Decade 2012 to 2022*

Occupation Title	2012	2022	Change (%)
Software developers	1,018,000	1,240,600	22
Software developers, applications	613,000	752,900	23
Software developers, systems software	405,000	487,800	20
Computer systems analyst	520,600	648,400	25
Computer programmers	343,700	372,100	8
Computer & information systems managers	332,700	383,600	15
Web developers	141,400	169,900	20
Database administrators	118,700	136,600	15

*Note.* Adapted from Bureau of Labor Statistics, U.S. Department of Labor (2015).

It is no surprise that as a result of these projected trends, CS and SE have become popular choices as areas of study. SE is relatively new as an academic discipline as it was not until about four decades after the original CS programs in this country that Rochester Institute of Technology (2004) accepted the first students into a SE program in 1996. It is, therefore, sometimes more difficult to locate data on enrollment trends of students in SE programs. The Computing Research Association (2015) collects data on the Taulbee Survey to measure enrollment trends in CS, computer engineering, and information programs at PhD-granting institutions and their findings are summarized as follows:

1. Among U.S. schools that reported data in 2014 and 2013, enrollments in undergraduate CS programs rose 18.6% in 2014, marking the seventh straight year of increase. Overall enrollment, including schools that did not participate in the survey in 2013, increased 27.3%.

2. The number of Bachelor's degrees in CS awarded by the reporting institutions increased by 13.6% in 2014. Among schools who responded in 2014 and 2013, the increase was 14.2%.

3. The number of new Bachelor of Science majors rose 17.0% from 2013 to 2014 and 18.3% among those departments reporting both years.

Dye (2014) reported on findings from the Sparkroom Marketing Software student inquiry and enrollment database that CS program inquiries in the United States showed a general increase between the first quarters of 2010 and 2014. That increase was 20% from 2013 to 2014 (Dye, 2014). These data suggest increased interest nationally in undergraduate computing degrees.

### **Statement of Problem**

Students may begin formal computer science or SE study in a course entitled "Introduction to Computer Science," or a similar name. These introductory courses form the gateway to an academic program. They can have various goals including raising interest in the core field of study, introducing students to the different areas of the discipline, and exposing them to a project experience to practice the skills that will be needed in upcoming courses (Fulton & Schweitzer, 2011; Rolka & Remshagen, 2015; Shell & Soh, 2013). An introductory CS course may cover information systems, hardware and architecture, operating systems, SE, programming, databases, and several other topics (Anderson, Ferro, & Hilton, 2011; Wu, Hsu,

Lee, Wang, & Sun, 2014). Instructors can select from several computer languages (Ali & Smith, 2014; Chang, 2014; Shein, 2015) in trying to provide students with an experience that is educational, motivating, or applicable to current industry practices. Likewise, there are several possibilities for assessment in these courses (Fulton & Schweitzer, 2011; Muñoz, Martínez, Cárdenas, & Cepeda, 2013; Shaw, 2010; Zur, Vilner, & Shay, 2014).

Therefore, various possibilities and sources of information as to the desired content and best practices exist for the curriculum designer to consider for such a class. A strong approach to curriculum development is to take into account input from these different sources and to include the opinions of subject matter experts (SME) to help ensure the course's design meets the needs of all stakeholders (Brown & Green, 2011; Ornstein & Hunkins, 2013). Another possibility is to consider the essentials as dictated by local industry in designing undergraduate curriculum. (Bothe, Budimac, Cortazar, Ivanović, & Zedan, 2009; Bramwell & Wolfe, 2008; Levin, Cox, Cerven, & Haberler, 2010). The purpose of this study was to define the competencies, programming languages, and assessments for an introductory CS course at a small private liberal arts university that seeks to address the computing industry's needs of California's Central Valley.

### **Research Questions**

The author developed the following questions to guide the research:

RQ<sub>1</sub>: What competencies do subject matter experts recommend for students in California's Central Valley to master in an undergraduate introductory CS course?

RQ<sub>2</sub>: What programming languages do subject matter experts recommend for students in California's Central Valley to use in an undergraduate introductory CS course?



RQ<sub>3</sub>: What assessments do subject matter experts recommend for students in California's Central Valley to demonstrate mastery of competencies for an undergraduate introductory CS course?

### **Background and Significance**

California's Central Valley is typically known for its agriculture and it has begun to increasingly depend on computing technology (Pratt, 2015). Additionally, there has been an influx of small high technology companies looking to take advantage of low rent and costs and the available workforce offered by the region (Romero, 2014; Sheehan, 2014). As shown in Table 2, the State of California Employment Development Department (2015) identified software developer, computer systems analyst, computer programmer, and web developers as occupations that are expected to experience a growth of 30% or higher in Fresno County between the years of 2012 to 2022. The outlooks for other related occupations are also shown.

The increase in the projected number of jobs in computing in the region has been accompanied by substantial growth in the number of students in California pursuing degrees in CS and SE. The California State University (2015) awards the most bachelor's degrees in the state. The system's enrollment in CS and SE programs for the past five years is shown in Table 3. The number of students enrolled in a CS program has increased 95% and SE enrollment has grown 158% during the time period. Also included in Table 3 are enrollment numbers for these programs at two regional campuses of CSU in Fresno and San Luis Obispo. The Fresno campus has seen CS enrollment increase by 101% and the location in San Luis Obispo has seen its CS and SE program enrollments increase by 51% and 144%, respectively.

Table 2

*State of California Employment Development Department Projected Job Outlook Over the Decade 2012 to 2022 for Fresno County*

Occupation Title	Employment		Change (%)
	2012	2022	
Software developers	430	620	44
Software developers, applications	340	490	44
Software developers, systems software	90	130	44
Computer systems analyst	450	630	40
Computer programmers	230	300	30
Computer & information systems managers	270	290	7
Web developers	150	200	33
Database administrators	110	120	9

*Note.* Adapted from State of California Employment Development Department (2015).

The increased focus on computing appears to be enveloping this region. Sheehan (2014) reported on the efforts of Bitwise Industries to create a technology hub in downtown Fresno. The organization provides real estate so computer technology firms can be housed together in one location, trains students with technology skills with highly focused course offerings, and provides computer technology services for local industries. Bitwise CEO Jake Soberal made the case for a regional economy more focused on high technology in the traditionally agricultural area:

If we can get a critical mass of people here in Fresno who are competent and capable, national and global companies will choose to expand their operations here. The Silicon Valley and Boston and Portland will continue to grow. And so will Fresno—and Des Moines and Wichita. Software and tech have not been a zero-sum game. (Fallows, 2015 para. 12)

Table 3

*California State University System Computer Science and Software Engineering Program Enrollment*

Student Type	2010-11	2011-12	2012-13	2013-14	2014-15
Computer Science	4,752	5,807	6,703	7,909	9,283
Fresno	154	177	193	272	309
Cal Poly, San Luis Obispo	401	452	473	545	606
Software Engineering	231	273	323	449	597
Cal Poly, San Luis Obispo	95	114	143	204	232
Total	4,983	6,080	7,026	8358	9,880

*Note.* Adapted from California State University (2015b).

The context for this research is a new undergraduate SE program at a small private nonprofit university in Fresno County. An introductory course in CS will serve as a gateway to the program for those students looking to major or minor in SE, and for others looking to simply develop some background in computing. A consideration for the course will be those students who intend to remain and practice their skills in the local geographic region. It is, therefore, prudent to consider the needs of local industry.

As in any curriculum development there are many possibilities that exist for competencies, programming languages, and evaluation in an introductory computer science course. Curriculum designers consider information from stakeholders in the planning stage of development to ensure they meet students' needs (Kenny & Desmaris, 2012; Reigeluth, 1999). Professional associations, faculty, and industry members are sources of expertise who can provide valuable information on curriculum content and delivery. Also of consideration is the geographic location of a course as the more applicable areas of focus can vary from one location

to another. As a result, program curriculum should address these regional concerns and the proposed introductory course is no exception.

The competencies for the class are of primary concern. Ornstein and Hunkins (2013) state that curriculum development should begin with an “analysis of needs and tasks” (p. 190). The objectives or competencies defined will dictate the content to be covered. They will also serve as the central components for assessment of the learners’ abilities (Brown & Green, 2011) as well as for the instruction itself.

Several possibilities serve as sources of information for course competencies. Professional associations and accrediting bodies publish guides on what content academic programs of study should address. The Joint Task Force on Computing Curricula (JTFCC) (2014) has periodically published curricula guidelines for computer engineering, CS, information systems, IT, and SE since the 1960s. These recommendations include the areas of knowledge that higher education programs in these fields should address to properly prepare students for the world of work. The Accreditation Board for Engineering and Technology (ABET) and its Engineering Accreditation Commission specify student outcomes for engineering programs, including those in SE. These sources provide valuable input regarding the competencies and assessment devices for an introductory computing course.

Likewise, the literature provides rich insight into best practices and trends in computing education. Educators from all over the world conduct research on the use of instructional strategies, student motivation, assessment, resources, and choices of programming languages, for example. These reported findings become part of the body of pedagogical content knowledge and help instructors focus course offerings.

Though these resources seem to provide sufficient data, they typically lack a focus on the specific region in which instruction is offered. Industry emphases tend to vary from one location to another and it is not the norm that higher education is seen to meet those needs. Symonds, Schwartz, and Ferguson (2011) stated that “while the best community colleges are the most entrepreneurial, market-responsive institutions in higher education, they are the exception rather than the rule” (p. 28). Smaller institutions of higher learning can better serve their stakeholders by meeting local needs. A program that prepares graduates for a career in computing in the Central Valley will be in better synergy with the recent industry developments. It is, therefore, prudent to develop this curriculum and its introductory course with input from the discipline as well as local industry.

Finally, students might pursue an introductory course, and perhaps a minor in computing, to develop their skills for an occupation other than software development. Lazowska, Roberts, and Kurose (2014) reported that “students are figuring out that every 21st Century citizen needs to have facility with ‘computational thinking’ – problem analysis and decomposition (stepwise refinement), abstraction, algorithmic thinking, algorithmic expression, stepwise fault isolation (debugging), modeling – driving introductory course demand” (p. 16).

### **Limitations**

The researcher imposed the following limitations on this study:

1. The research will include feedback from a small number of experts in California’s Central Valley. Selection bias may be a potential issue (Mitchell, 1991). These persons will be selected based on their status as experienced industry professionals or faculty at the local state university and community colleges.

2. This survey research will capture the opinions and recommendations of the participants at a particular point in time. Computing can be a dynamic field and it is expected this feedback would vary if collected at a future timeframe.
3. The feedback of participants may be biased by their personal experiences, preferred learning and teaching styles, and disciplinary knowledge.
4. The written survey will communicate limited information as it will employ ratings and rankings. The reasons for differences in opinions are beyond the scope of this study.

### **Assumptions**

The researcher assumed the following points to be true throughout the data collection and analysis for this study:

1. Participants in the study would draw upon their experience and promote their informed opinions of the most important skills for inclusion in the introductory course. Their responses would constitute their personal views.
2. Industry professionals would make recommendations based on their content knowledge and the practices utilized at their place of work in the local industry. The recommendations of the instructors would consider pedagogical content knowledge (Shulman, 1986) to help promote the practicality and subject matter necessary to include in an introductory course in computing.
3. The recommendations of the instructors from the state university and community colleges would be equally important. Though their student populations may be different, their recommendations would apply equally to the target students of this introductory course.

4. The content and design of the introductory CS course in the local region may have emphases which differ from similar offerings outside this area. Participants will successfully represent the conditions in this region.

### **Procedures**

The researcher's goal was to design an introductory course in CS that meets the needs of local stakeholders. The first offering of the course was scheduled to take place in January 2016. The course had been approved by university committees before this study and would be run as previously designed for the first semester. The research would be conducted concurrently with the initial course offering and the results would be used to help design the second offering in August 2016. This descriptive study utilized surveys consisting of quantitative ratings and feedback to answer the research questions. Local industry members and college and university educators are experts who would likely provide disparate recommendations. Wilhelm (2001) suggested the Delphi technique as providing a method of getting "the relevant intuitive insights of experts and to use informed judgment as systematically as possible" (p. 6).

A panel of 23 experts were recruited for this study. Eleven members of industry were recruited from professionals holding programming or software development managerial roles with local firms for a period of at least five years (Guu, Lin, & Lee, 2014; Joyner & Smith, 2015). Another twelve individuals were educators with local universities and community colleges, who taught CS or related disciplines (i.e. computer engineering, information systems, IT, and SE) and held at least a Master's degree (Sami, 2007) in CS or a connected field. One research expert was recruited to help interpret the data from the Round 1 surveys. This individual was required to hold a Ph.D. degree and have experience with survey data.

The JTFCC's recommendations for introductory CS courses were consulted for guidance on competencies and assessment. A review of literature on trends and best practices regarding these items and programming languages available was then performed. The output from these sources were analyzed and served as the components for the straw models, which would then be used to clarify the undertaking and narrow the content for the participants in the Delphi procedure (Rotondi & Gustafson, 1996). The competencies, programming languages, and assessments identified from the literature were used as the topics for the first round survey.

The Round 1 questionnaire first asked for demographic data from the participants. This initial survey consisted of the potential competencies, programming languages, and assessment devices, and participants were asked to identify their importance or applicability for an introductory course in computing on a five-point Likert scale. Participants were also given the opportunity to add their own suggestions for each of the categories. The questionnaire was made available to the participants on the internet using SurveyMonkey (Elledge & McAleer, 2015) and an email was sent out to each individual with a link and instructions to complete the survey within one week. The researcher collected responses and sent out email reminders on the fourth and eighth days.

The results of the first survey were analyzed and items for competencies, programming languages, and assessments were ranked based on the responses. Additionally, the researcher collected suggestions of new items and added those that did not overlap with already existing options in the updated list. Any suggested items that garnered at least two independent suggestions would then be included as options in the questionnaire for Round 2. All open-ended responses were reviewed with a research SME and changes were made to the item lists as necessary.



The survey for Round 2 included the modified choices for competencies, programming languages, and assessments. All options and their median scores from Round 1 were indicated on the questionnaire. The two groups were then separated at this point into industry and academic experts constituting a panel design for Round 2 and after so they would function independently (Okoli & Pawlowski, 2004). Okoli and Pawlowski's (2004) recommendations were used so that each participant was given three lists representing the competencies, programming languages, and assessments and asked to select, not rank, at least ten of each of these topics. Those items that were selected by at least 50% of either the industry or academic group were kept for the subsequent round (Okoli & Pawlowski, 2004).

The updated lists were again submitted separately to each of the two groups in Round 3. Participants were asked to attribute importance or applicability of each item on a five-point Likert scale. The median score was computed for each topic and the overall Kendall's *W* for each category was calculated. Consensus was deemed to be achieved if the value was equal to 0.7 or greater (Okoli & Pawlowski, 2004; Schmidt, 1997). The sequence for Round 3 was repeated for one additional round if the *W* value was less than 0.7 (Okoli & Pawlowski, 2004).

The output from the Delphi study would thus consist of recommendations from two stakeholder groups: industry and academic experts. The median ranking scores would constitute the importance attributed to each item by each individual group. These itemized recommendations would then be made available to the curriculum developer to design the course accordingly.

### **Definition of Terms**

The following definitions are included to provide a clear understanding of terms used throughout the study:

*Competencies*: general objectives detailing the desired content and abilities students are expected to master as a result of learning (Koszalka, Russ-Eft, & Reiser, 2013); specifically, those used for the introductory CS course of concern in this research.

*Curriculum*: “a course of study organized by the content to be covered and the activities to be employed to cover them” (Brown & Green, 2011, pp. 101-102).

*Formative evaluation*: activities instructors use to gauge student learning to optimize the process (Ornstein & Hunkins, 2013).

*Resources*: items to aid learning in a course (e.g. textbook, computer language, embedded microcontroller, etc.).

*Pedagogical content knowledge*: knowledge of content that includes aspects “most germane to its teachability” (Shulman, 1986).

*Subject matter expert (SME)*: individual with experience or certification “who assists instructional designer by providing guidance on the scope and sequence of the content and tasks that need to be included in the instruction” (Brown & Green, 2011, p. 68). Individuals in this study who had at least five years’ experience in their occupations in industry or taught in academia with at least a master’s degree qualified as SMEs for this study.

*Summative evaluation*: means to assess “the overall quality of a produced and then taught curriculum” (Ornstein & Hunkins, 2013, p. 253).

### **Summary and Overview**

Administrators and faculty in colleges and universities spend a great deal of time and effort designing curriculum. It is imperative students are provided with content and delivery that will help prepare them for their future careers. The experience of an introductory CS course is crucially important as it can often be the student’s first exposure to a field of study. The

impressions of industry and academic experts should be considered during design and development to properly meet the needs of students and their future employers.

In Chapter II of this study a review of literature focuses on the different goals of introductory CS courses in terms of student competencies, programming languages, and assessment. Trends in all these aspects are analyzed and synthesized. Recommendations from professional associations are also presented and factored.

Chapter III presents the methods and procedures utilized to determine the optimal components for the introductory course. The target population of experts in the study and their recruitment are discussed. The author outlines the data to be gathered and the methods to be utilized in its collection. Details of the Delphi method and the design of each round are also presented.

Chapter IV presents the output from each round of the study and the conclusions reached by the researcher and research SME. The results are supported by the data which is presented for the reader's review.

Chapter V provides an evaluation of the study as a whole. The author uses the results to present conclusions and provide pertinent recommendations for the current course of interest and to educators responsible for the design of introductory courses in similar and related fields of study.

## **CHAPTER II**

### **REVIEW OF LITERATURE**

A survey course in CS can vary significantly from one institution to another in terms of topics covered and the level of mastery expected of students. Another area of discrepancy is the degree to which programming is covered and in the cases where it is a factor, the extent to which it is included and the programming languages used. The teaching approaches and assessments used in class are also sources of variation. The focus of this study is an introduction to CS course at a liberal arts university.

This review of literature will cover topics on the preparation of programmers, software engineers, the stakeholders of CS curriculum development, and introductory computing courses. The goal of this chapter is to provide the reader with a background in these areas and generate the straw models for the three subjects to be deliberated by the Delphi groups. A review of the education and training of programmers and software engineers is provided, including a definition of CS and a brief history of the schooling of students in the discipline. A concise analysis of the stakeholders in CS curriculum development in academia and industry is provided before the literature is reviewed on the competencies, programming languages, and assessments for introductory CS courses. The latter examination will serve to generate the straw models for the first round of the Delphi study

#### **Preparation of Programmers and Software Engineers**

##### **Definition of Computer Science**

Chapter 1 began with mention of the identity struggle that has been part of the history of CS. The most appropriate beginning would seem to be a definition of the field itself. Though the terms computing and CS are often used interchangeably, Denning (2013) used the term

‘computing’ to encompass the fields of “computer science, computational science, information science, computer engineering, and software engineering” (p. 35) and argued for its identity as a true science. The Association for Computing Machinery (ACM) Curriculum Committee on CS (1965) concentrated on the discipline’s relationship with information and stated that CS was “devoted to the *representation, storage, manipulation, and presentation* of information in an environment permitting automatic information systems” (p. 544). Over the next few decades the environment had become commonplace. Increased mention of algorithms, automata, data structures, and the handling of information found their way into updated definitions of the field (Gibbs & Tucker, 1986; May 1980). Brookshear (1997) seemingly simplified the description while including another element when he stated that CS was “the study of the theoretical foundations of information and computation” (p. 1). Six years later Brookshear (2003) elaborated to include “topics such as computer design, computer programming, information processing, algorithmic solutions of problems, and the algorithmic process itself” (p. 1). The emphasis on algorithms has seemingly become widespread. The authors of two recent texts on introductory CS directed their focus on algorithms as they defined the discipline (Dale & Lewis, 2016; Schneider & Gersting, 2016). It would appear then that coverage of CS must include how information is used to perform computations and construct algorithms.

### **Occupations**

In a text intended for young people thinking about their future careers, the editors at JIST (2010) offered the following occupations related to CS: computer network, systems, and database administrators; computer scientists; computer software engineers and computer programmers; computer support specialists; computer systems analysts; and mathematicians. Young persons reading the material may one day enroll in an introductory course in CS. As

mentioned in the first chapter, CS skills are associated with many occupations, including applications software developers, systems software developers, computer systems analysts, computer programmers, computer and information systems managers, web developers, and database administrators (BLS, 2015). Descriptions of these professions are provided in Table 4.

The BLS (2015) also lists related occupations that are associated with networking that are typically connected to degreed programs in that specific discipline. This subject is not offered as a field of study at the institution at which the introductory course will be offered. These networking occupations, therefore, are thus not considered here. The occupations that are listed in Table 4 require various skills, and a fundamental understanding of CS is important in all of them. Computer programmers and software developers need to be highly proficient in computer programming. Database administrators and web developers also rely on the ability to program. An introduction to programming in one or more languages will be highly beneficial for students in an introductory course who wish to pursue positions in these professions. Computer and information systems managers and systems analysts need a firm understanding of computer systems and how they can be used to help a business meet its goals. Students in an introductory course who are interested in these professions will benefit from a broad introduction to the discipline of CS and the systems, tools, and procedures associated with it.

### **Brief History of Undergraduate Study of CS**

To this day there exists debate on the identity of CS (Denning, 2013; Kelly, 2007; Kelly, 2013). Denning (2013) wrote that its “brief history suggests that computing began as science, morphed into engineering for 30 years while it developed technology, and then entered a science

Table 4

*United States Bureau of Labor Statistics Occupations Related to the Study of Computer Science*

Occupation	Description
Applications software developers	Design computer applications, such as word processors and games, for consumers. They may create custom software for a specific customer or commercial software to be sold to the general public.
Computer & information systems managers	Plan, coordinate, and direct computer-related activities in an organization. They help determine the information technology goals of an organization and are responsible for implementing computer systems to meet those goals.
Computer programmers	Write code to create software programs. They turn the program designs created by software developers and engineers into instructions that a computer can follow.
Computer systems analysts	Study an organization's current computer systems and procedures and design information systems solutions to help the organization operate more efficiently and effectively. They bring business and information technology together by understanding the needs and limitations of both.
Database administrators	Use specialized software to store and organize data, such as financial information and customer shipping records. They make sure that data are available to users and are secure from unauthorized access.
Systems software developers	Create the systems that keep computers functioning properly. These could be operating systems that are part of computers the general public buys or systems built specifically for an organization.
Web developers	Design and create websites. They are responsible for the look of the site. They are also responsible for the site's technical aspects, such as performance and capacity, which are measures of a website's speed and how much traffic the site can handle.

*Note.* Adapted from Bureau of Labor Statistics, U.S. Department of Labor (2015).

renaissance about 20 years ago" (p. 37). Koffman and Finerman (2004) wrote that CS had its origins as an area of academic study in the 1950s based on the need to educate users of computing machinery. IBM's sale of discounted computers to about 50 universities increased

this need (Gupta, 2007). The main offerings predominant in these early days were in the form of noncredit courses at computing centers (Gibbs & Tucker, 1986; Koffman & Finerman, 2004).

Programs in engineering and mathematics began to address the growing need for the content to be covered in formal education (Koffman & Finerman, 2004). It was not long, however, before the call for CS to be considered as a standalone discipline would start to be heard (Gorn, 1963; Keenan, 1964). Gorn (1963) seemingly began the questioning by asking “can such a rapidly growing discipline with clearly different interests and requirements continue indefinitely to be carried in an essentially different environment where accident has caused it to gestate” (p. 155)? Keenan (1964) for his part asked “should our colleges and universities adapt their educational plans to take into account an increasingly computerized society and, if so, how” (p. 207)? The answers became apparent as universities began to offer CS academic programs.

Undergraduate education, however, was predated by graduate study as there was a lack of definition as to what a program at the entry level would look like (Koffman & Finerman, 2004). Nonetheless, demand for education in the areas of computing grew. Pioneers in CS instruction presented content to be covered at the introductory level (e.g. Arden, 1964; Perlis, 1964). Rosser chaired the Committee on Uses of Computers (1966), which identified a need for increasing “capacity for education and research in computer science” (p. 1). Among the recommendations was the call to “increase as rapidly as possible the number of persons trained annually as computer specialists and the support of pioneering research into computer systems, computer languages, and specialized equipment” (Committee on Uses of Computers, 1966, p. 2).

Over the years, other academic disciplines branched out from CS. By 1969 Georgia State University had launched a graduate program in business information systems (BIS) (Chand, 1974). The Case Western Reserve University (2015) offered the first accredited computer



engineering program in the country in 1971. By the 1990s, IT and SE had become independent programs of study as well (Lunt et al., 2005; Rochester Institute of Technology, 2004).

The ACM provided the first set of curriculum recommendations for undergraduate study in CS in 1965. The ACM Curriculum Committee on CS (1965) described the discipline and particular courses as being attractive to students in different majors, especially those studying engineering. The organization released its first official curriculum guidelines for CS in 1968 and has published updates approximately once every decade thereafter (Dziallas & Fincher, 2015; Roach & Sahami, 2015). Dziallas and Fincher (2015) wrote that “these reports have become an institution; with each new iteration, chairs are chosen, task forces formed, disciplinary groups engaged, drafts produced and then posted on websites and presented at conferences to solicit community feedback” (p. 81). The ACM joined with the Institute of Electrical and Electronics Engineers (IEEE) to form the Joint Task Force on Computing Curricula (JTFCC) in 1988 (Koffman & Finerman, 2004; Turner, 1991), which would release the subsequent recommendations in 1991, 2001, 2008 and 2013 (Joint Interim Review Task Force, 2008; JTFCC 1990; JTFCC 2001; JTFCC 2013).

Though the JTFCC’s recommendations have provided much value to institutions offering CS programs over the years, educators at liberal arts colleges and universities were apparently underserved by the documents. Much has been written about the situation of these typically smaller schools and their education of students in CS (Bruce, Cupper, & Drysdale, 2010; Gibbs & Tucker, 1986; LaFrance & Roth, 1972; Lopez, Raymond, & Tardiff, 1977; Walker & Kelemen, 2010; Walker & Schneider, 1996). LaFrance and Roth (1972) conducted a workshop on CS education at liberal arts institutions and noted the goal of “educating the whole person” (p. 22) with “wide variation in the degree of commitment to vocational preparation” (p. 22) among

these colleges and universities. Lopez et al. (1977) performed a survey on curricula at liberal arts colleges and found that only 12% of those offering CS programs had instituted the ACM's recommendations from 1968. Because these institutions focus on offering students a broad education, compromise is often required. Walker and Schneider (1996) indicated that “a liberal arts education involves the investigation of a major at a reasonable level of detail [in which] the core areas of the discipline can be carefully and fully covered” (p. 86). It is apparent, therefore, that a wide variety of institutions have come to offer academic programs in CS and formal education has become increasingly important for students looking to obtain jobs in computing.

### **Formal Education in Computing**

According to the BLS, most of the aforementioned occupations related to computing require an academic degree. Computer programmers, computer systems analysts, database administrators, and software developers are all listed as requiring a bachelor's degree as entry level education while web developers require an associate's degree (BLS, 2015). While outlier examples like Bill Gates, Steve Jobs, Michael Dell, and Mark Zuckerberg have succeeded in creating technical ventures without completing an academic degree, it is increasingly rare to excel in the computing field without one (Aarts, 2015). Because the skills required in each of these occupations tends to be highly technical, employers seek applicants who have earned an academic degree and established a foundation of knowledge in the field. In 2006, 85% of computer software engineers and 73% of computer programmers aged 25-44 had earned at least a bachelor's degree (Liming & Wolf, 2008). O\*Net Online (2015) reported that as of 2009, 94% of systems software developers, 84% of applications software developers, 78% of computer programmers, and 60% of database administrators had earned at least a bachelor's degree while 67% of computer systems analysts had earned at least an associate's degree. Thus, education in

the form of an academic degree is predominantly required for occupations in the field. Another area of instruction achievement becoming increasingly visible is that of certification.

A prospective employee can show proof of aptitude with respect to computing products and systems of a certain producer (e.g. networked systems by Cisco Systems) by earning certification. Land and Reisman (2012) indicate that certification in computing has tended to be specific to vendor or domain and “helps bridge the knowledge gap between what new college graduates bring to the job versus what companies require” (p. 51). The IEEE Computer Society has offered various certifications since 2002, including Knowledge Area Certificates, Associate Software Engineer Certifications, Professional Competency Certifications, and Certificates of Achievement (Continuing Education) (IEEE Computer Society, 2015). These types of certifications are used to verify that prospective employees have been properly trained and have the necessary knowledge to help an employer meet software goals (Land & Reisman, 2012). This aptitude thus helps to ensure that new workers can contribute to an employer’s operations quickly and effectively and gives job seekers an advantage in competitive markets.

Government has also become involved in requiring proof of propensity by those employed in these fields. Though licensure is not a widespread requirement for those employed in the computing fields, it is becoming more commonplace. Land and Reisman (2012) reported that ten states “require software licensure for software engineers working on software systems that can affect public health, safety, and welfare” (p. 52). These forms of formal education can help those who hope to excel in a computing career obtain the background necessary to prepare them for their futures.

Though learning in places of formal education comprises a large part of the knowledge transfer taking place about computing, informal learning plays a significant role. Many people

work in fields outside of computing but rely on skills to develop scripts or programs in software applications to do their jobs (Dorn, 2011; Shein, 2014). Much learning takes place in the form of personal learning networks but most research has focused on what occurs in and around school campuses (Harding & Engelbrecht, 2015).

Moffitt (2012) discussed informal and formal learning of CS and SE noting that several opportunities existed for both. Nontraditional sources of computing education abound and include Code Academy ([www.codecademy.com](http://www.codecademy.com)), iTunes U (<http://www.open.edu/itunes/>), and various massively open online courses (MOOCs) including Coursera ([www.coursera.org](http://www.coursera.org)), Udemy ([www.udemy.com](http://www.udemy.com)), and offerings by Harvard University (<https://cs50.harvard.edu>) and Stanford University (<http://online.stanford.edu/courses>). Moffitt (2012) observed that formal education can help build a solid foundation for graduates pursuing careers in the industry because “software and web development can be a very complex field, which at times may require that detailed explanation or other points of view to get through certain situations” (p. 9). Education in computing has certainly come a long way over the last six decades. One consistency during this development, however, has been the list of stakeholders.

### **Stakeholders in Computer Science Education**

There are four major groups who are deeply involved in CS education; more specifically in an introduction to CS course. These include industry, academic institutions, professional associations, and students. Experts on computing education are found in the first three groups and these have primary roles in this study. Mention has already been made of the ACM and IEEE as the major professional associations. Students are the ultimate recipients of these efforts and have varied expectations and their aims to learn necessary skills has been discussed. This section will focus on the two main groups not yet examined.

## **Industry**

Firms engaged in business employ persons in varied occupations. The computing industry includes businesses engaged in activities directly related to the disciplines of CS, computer engineering, information systems, IT, and SE. Most of these distinct fields of study arose because of the needs of individual skill sets required by the computing industry (Chand, 1974; Lunt, et al., 2005; Lutz et al., 2014).

Industry directly defines the skills necessary for employment. Norton (1998) based the DACUM (Developing a Curriculum) methodology on the premises that experts in industry best define their jobs and occupations and successful employees possess certain knowledge, skills, and aptitude with tools used by these experts. He indicated that:

1. Expert workers can describe and define their job or occupation more accurately than anyone else ...
2. An effective way to define a job or occupation is to precisely describe the tasks that expert workers perform ...
3. All tasks, in order to be performed correctly, demand the certain knowledge/skills, tools and positive worker behaviors (Norton, 1998, pp. 1-2)

Industry experts in the computing field should therefore be regarded as primary stakeholders who possess deep knowledge on the skills required to successfully design and develop hardware and software computing solutions.

Industry continuously develops business practices with the aim to improve effectiveness and efficiency. One of the most significant events in the software development field over the past two decades has been the emergence of agile methods. Beck et al. (2001) called for emphasis on interactions, functional software, collaboration with customers, and response to change in their

Agile Manifesto. Because of these types of business practice developments, there arises a need for new employees who possess some knowledge of, and perhaps the ability to implement, them. Since industry needs graduates who have skills necessary to contribute to an organization's operations, the argument is often made that an academic program's curriculum needs to reflect this requirement (Lutz et al., 2014; Tan & Venables, 2010; Winberg, 2014). Universities and colleges often respond to these needs. There has been much written over the past few years on the reasons for and proper methods of teaching agile software development practices in the classroom (Guercio & Sharif; 2012; Lutz et al., 2014; Rajlich, 2013). Therefore, it can be safely assumed that industry does serve a role in the curriculum definition of CS and related disciplines.

Though this influence of industry on academia is apparent, it is not necessarily the dominant direction of impact in SE. Ben Arfa Rabai, Bai, and Mili (2011) developed a model that showed that the impact of successful adoption in academia on implementation in industry declines with time but is stronger than industry's effect on academic learning. It is obvious, therefore, that industry and academia influence one another and that universities and colleges are also primary stakeholders.

### **Academia**

There are around 1,300 academic institutions in the United States offering undergraduate programs in CS or related disciplines (U.S. News & World Report, 2015). Hambrusch, Libeskind-Hadas, and Aaron (2015) pointed to almost 800 such institutions in their study on the backgrounds of Ph.D. students majoring in CS. These institutions include community colleges, research and other four-year universities of the private and public, for-profit, and not-for-profit varieties. Gray and Herr (1998) indicated that two-year associate's level programs at community colleges have been attractive options for "high-skills/high-wage nonprofessional occupations" (p.

263). Though these varied institutions have different goals, there is dependence among them. It is perhaps no surprise that the ACM closed its 1978 curriculum recommendations with a discussion on the importance of articulation acknowledging that “transfer programs in community and junior colleges are often geared to programs at four-year institutions” (Austing, Barnes, Bonnette, Engel, & Stokes, 1978, p. 165). Even more recently there has been a trend for state governments to pass legislation to improve articulation between institutions of the two-year and four-year types.

The 2005 Career-Technical Credit Transfer (CT)<sup>2</sup> program in Ohio is one example that promoted flexibility for students. The legislation made it easier to transfer credits from approved career-technical or secondary career technical programs to four-year higher education institutions “without unnecessary duplication or institutional barriers” (“[CT]<sup>2</sup> Basic Information,” n.d., para. 1). State governments enacting this type of legislation can help address shortages of skilled workers within their borders. Sander (2008) pointed out that:

Ohio's move reflects the experience of many states that face shifting economies and a shrinking pool of jobs for workers without college credentials. Educators in those places are looking for ways to provide students greater access to credit-bearing courses and, ultimately, greater career potential. (A23)

The assertion here is that the needs of academic institutions, students, employers, and governments are better met by reducing the barriers of movement from one program to another.

Because CS and related disciplines provide education of technical skills, they are often a popular domain for these types of articulation efforts. Colson (2015) reported on the importance of the community college system in New Hampshire and that CS majors are among the most common students taking advantage of dual admission and transfer programs. The situation is

similar in California. In 2014-15, there were 1,883 students enrolled in information sciences programs in the CSU system who had transferred from the California Community College system (CSU, 2015). CSU (2015) reported that over 1,200 of these were enrolled in CS or CS and IT programs.

Community college administrators and faculty have found that collaboration with other institutions can be beneficial sources of both increased opportunities for students and enrollments for the institutions themselves. Levin et al. (2010) investigated best practices of programs within the California Community College system that have been successful at reducing the gap in success between majority and underrepresented minority (URM) groups; they found one of the top habits to be “the capacity of program personnel to develop and maintain linkages and relationships, both within the institution and to external parties, so that interdependence is both recognized and relied on to advance the interests of the program” (p. 53). These associations include various types of schools and a wide range of involvement.

One example is the joint program between CSU Monterey Bay and Hartnell College, which offers students the opportunity to graduate with a B.S. in CS in three years by taking courses offered at both institutions (CSin3, n.d.). *The Californian* reported that the CSin3 program has been highly successful at attracting traditionally URMs (“Matsui Foundation awarding over \$1M in scholarships,” 2015). Grandgenett, Thiele, Pensabene, and McPeak (2015) reported on the Midwest Center for Information Technology (MCIT) consortium of 10 community colleges that have been able to improve faculty professional development practices, increase the relevancy of curriculum, raise the number of female enrollments, and enhance articulation with both secondary schools and four-year institutions with their IT offerings. Bothe et al. (2009) reported on a collaborative effort involving individuals from seven universities in



five countries in Europe to create curriculum for a master's program in SE. Finally, Molnár, Toth, and Vincent-Finley (2014) recommended sharing of resources in curriculum development efforts and potentially allowing students at their institutions to enroll in courses at each of their different university's campuses. Collaboration among institutions of higher education appears to be a trend that is gaining in popularity as it offers opportunities for institutions to improve the quality and efficiency of program offerings.

Development of curriculum is an integral step in the creation of new academic programs. Content and delivery shape the student experience and curriculum designers look for resources to aid their efforts. Ornstein and Hunkins (2013) include science, society, moral doctrine, knowledge, and the learner as sources of information for the design of curriculum. The aforementioned collaboration efforts allow for sharing of knowledge and best practices along each of these domains. Franklin (2015) advocated including computing education research at the forefront of teaching CS to better understand how students learn concepts and what tools, languages, techniques, and themes are most appropriate. She stated that "researchers need deep expertise in computer science as well as a robust understanding of the types of questions and methods used in education" (Franklin, 2015, p. 35). The literature on CS education, therefore, provides a rich source of information for one tasked with developing curriculum. There has been much written on introductory courses throughout the history of CS. The findings in the literature serve as rich sources of information to create the straw models for experts to consider when judging the merits of different competencies, programming languages, and assessments. Before contemplating each of these areas, however, it is necessary to consider the breadth of the audience in introduction to CS courses and take into account their varying goals.

## **Introductory Computer Science Courses**

Introductory courses in CS serve many purposes to their various stakeholders. Program administrators and faculty might consider the course a gateway to a program, a foundation for further in-depth study, or a combination of these two (Ali & Smith, 2014; Dodds, Libeskind-Hadas, & Kuenning, 2008). Ali and Smith (2014) indicated that an introductory course is “useful in two ways: ... as a marketing tool for the teaching department to bring more students into their programs ... [and] as a prerequisite for other courses within the department and prepare student for advanced courses in the program” (p. 60). Urness and Manley (2011) concurred that these courses can be effective in attracting new students to CS. As such, development of curriculum for such a course must establish the purpose and degree to which these varied aims are to be met.

The topic of this study is an introductory course in CS that aims to provide a survey of the areas within the discipline as well as an introduction to programming. The goal of introducing the discipline of CS to a wide audience has given rise to what has been termed the CS0 course. The identifier alludes to the ACM’s naming of the original course suggestions mentioned in its 1978 curriculum recommendations, where Austing et al. (1978) identified courses in Computer Programming I and Computer Programming II as CS 1 and CS 2, respectively. Since then, the course title CS0 has come to represent a breadth-first introduction to the areas of the CS discipline (Bruce, Fowler, Guzdial, King, & Woszczyński, 2005; Forte & Guzdial, 2005; Huang, 2008; Urness & Manley, 2011), an introduction to programming (Ali, 2009; Guo, 2014), or a reduced or lack of emphasis on programming (Cheng, Jayasuriya, & Lim, 2010; Cortina, 2007; Enbody, Puch, & McCullen, 2009). Davies, Polack-Wahl, and Anewalt (2011) identified CS0 as “an introductory course with no prerequisites involving at least some programming that does not count towards the major” (p. 626) and a CS1 course as “the first

required course in the major programming sequence” (p. 626). They found wide variance in the emphasis on programming in CS0 courses (Davies et al., 2011). Hertz (2010) argued that the titles have become meaningless as he found little agreement on the topics of importance in CS1 and CS2 courses among educators. In spite of a lack of consensus on the exact definition of such courses, the terminology continues to be used and the focus for this study can therefore be termed a CS0 course.

It is, perhaps, no surprise that the first curriculum recommendations from the ACM Curriculum Committee on CS (1965) included special mention of introductory courses. What might be more of a revelation, however, is that the authors of these recommendations acknowledged from the beginning that introductory courses have a varied audience and that “the background of the students, the language appropriate to the subject, the pertinent exercises and examples, all differ, depending on the students' primary field” (p. 544). The importance of computing to different careers was, therefore, understood in these early days. That view continues today. Denning and Gordon (2015) indicated that “surveys show students are taking up computing ... because they perceive computer science as compatible with almost every other field” (p. 28). There is indeed a perception that CS has much to offer students of various disciplines with differing career aspirations. Introductory courses in CS have reportedly become the most popular offerings at university and college campuses all across the United States (Bernhard, 2014; Lazowska et al., 2014; Soper, 2014). Such courses serve varied purposes for the different students who enroll in them. CS is often viewed as an avenue to teach general and translatable skills, such as computational thinking and creativity.

## General Skills

There is debate among experts on what is meant by computational thinking (Committee for the Workshops on Computational Thinking & Computer Science and Telecommunications Board, 2010). Papert (1980) originally used the term and mentioned its importance in everyday life but did not elaborate on its meaning. Wing (2006) explained computational thinking as involving “solving problems, designing systems, and understanding human behavior, by drawing on the concepts fundamental to computer science” (p. 33). Shein (2014) seemingly agreed stating that it “helps people learn how to think abstractly and pull apart a problem into smaller pieces” (p. 17). Lazowska et al. (2014) identified these skills as “the problem analysis and decomposition (stepwise refinement), abstraction, algorithmic thinking, algorithmic expression, stepwise fault isolation (debugging), [and] modeling” (slide 16). These skills are ultimately of high importance to students studying CS (Dodds et al., 2008; Zhao, Su, & Wang, 2015) but offer much to others as well. Barr and Stephenson (2011) advocated for increased emphasis on computational thinking in K-12, stating:

Computational thinking is an approach to solving problems in a way that can be implemented with a computer. Students become not merely tool users but tool builders. They use a set of concepts, such as abstraction, recursion, and iteration, to process and analyze data, and to create real and virtual artifacts. Computational thinking is a problem solving methodology that can be automated and transferred and applied across subjects.

(p. 51)

Students will likely find it useful to develop this ability to solve problems in this manner, no matter their major.

Developing this attribute has seemingly become increasingly important as evidenced by its augmented coverage in the literature over the past few years. Czerkawski and Lyman (2015) agreed with this assertion stating “attaching computational thinking to our current list of discussions, however, and seeking deeper understanding of its outcomes will yield future benefits across multiple academic disciplines” (p. 64). That benefit appears to be increasingly acknowledged by students as evidenced by the aforementioned popularity of introductory computing courses. Another general attribute of courses in CS is creativity.

While creativeness is an emphasis in many academic courses, it has been discussed more as an outcome of studying computing. Forte and Guzdial (2004) posited that when considering the computer as a tool for communication and becoming literate, one “must acknowledge that literacy implies not only consumption, but also creation” (p. 1). The authors called for an increased emphasis on creativity in the CS classroom and that request has not diminished. Practices within the CS classroom can allow for opportunities for students to “exercise their creativity and have a sense of ownership over what they have created” (Cheng et al, 2010). A focus on creativity is often ignored in curriculum planning (Ornstein & Hunkins, 2013) and students may find a lack of opportunities to express themselves in their classes. Porter and Simon (2013) indicated that “CS is a creative endeavor” (p. 36) and students from all types of backgrounds may seek out opportunities to develop this skill in such classes.

Introductory courses, therefore, can attract a wide variety of students with potentially different desired outcomes. Forte and Guzdial (2004) stated that “the increase in non-majors who are required to take introductory computer science courses or who wish to improve their technical skills is drawing more attention to the strengths and weaknesses of traditional course implementations for a diverse student population” (p. 3). It can be challenging to address the

needs of different students and designing such courses may prove difficult. Those majoring in CS have different expectations than individuals who may be taking the course as an elective. Simon et al. (2009) performed a survey in which they asked students in CS1 courses at various institutions to define the programming experience and found non-majors to be less apt to provide positive responses but more prone to report it as useful and interesting; majors were more likely to use the words “fun” and “challenging.” Many institutions have taken to offering different introductory CS courses for learners based on their backgrounds or majors (Cohoon, Cohoon, & Sofa, 2013; Dodds et al., 2008; Forte & Guzdial, 2005; Guzdial, 2009; Norman & Adams, 2015). The curriculum in these types of courses can be tailored to the students enrolled allowing for a more focused approach.

This practice, however, can be prohibitive for smaller institutions or those with limited resources. Some introductory course offerings, therefore, must be designed to cover the breadth of the CS discipline to majors and non-majors alike. Many researchers have investigated best practices to meet the disparate needs of all students in one inclusive course (Bishop-Clark, Courte, Evans, & Howard, 2007; Malan & Leitner, 2007; Schneider, 2004). Schneider (2004) presented a three course introductory CS sequence and chose not to distinguish between majors and non-majors indicating “it is quite common for beginning students to have no idea of the field in which they will major or minor” (p. 41). The course of focus in this study is an example of one that will include students from different majors. The remainder of this literature review will aim to generate lists of competencies, programming languages, and assessments to consider for this CS0 course. This output will then serve as the three straw models for experts to deliberate in the Delphi study.

## Competencies

Since the early days of CS education in this country there has been dispute about what to cover in an introductory CS course. Gupta (2007) wrote about the development of CS curriculum in the 1960s and discussed the dispute about the objectives to be covered:

Perlis believed the introductory course should be more about problem solving and algorithms while Arden thought computing could be used to motivate students' ideas in mathematics. Hamming's view was that the course should focus more on the whole spectrum of applications including those that were not mathematical. (p. 43)

Before a discussion on the potential competencies of an introductory CS course, it is desirable to clarify the difference between competencies, objectives, and outcomes. Though the words are sometimes used interchangeably, they are applied to different ideas in education.

Hartell and Foegeding (2006) defined the terms as follows:

Competency: a general statement detailing the desired knowledge and skills of student graduating from [a] course or program.

Objective: a very general statement about the larger goals of the course or program.

Outcome: a very specific statement that describes exactly what a student will be able to do in some measurable way. A competency may have several specific learning outcomes so a course typically contains more outcomes than competencies. (p. 69)

Competencies, therefore, can be expected to be somewhat general and not specifically measurable. They do, however, define what a student will know or be able to do at some broad level.

The choice of competencies of a course defines its curriculum design. Barberà, Layne, and Gunawardena (2014) compared competencies for online courses from three different

disciplines (education, engineering, and business) in three different countries (Spain, United States, and Venezuela); they noted that instructional design was important to meet “the local context such as the needs and expectations of the learners, faculty perspectives, beliefs and values, and the needs of the institution, the community, and country” (p. 163). The authors, therefore, contend it is important to consider the needs of a locality in establishing competencies for a course of study. Barberà et al. (2014) went on to report that:

the type of competencies that the three disciplines from three countries [agreed] on: knowledge of the field, higher order cognitive processes such as critical thinking, analysis, problem solving, transfer of knowledge, oral and written communication skills, team work, decision making, leadership and management skills. (p. 163)

There are competencies, therefore, that will be deemed important regardless of location or major.

Starr, Manaris, and Stalvey (2008) provide a two-step process in which Bloom's taxonomy can be applied to establish outcomes: first “identify/select the topics to be covered... [and second] decide which is the highest level of mastery that all students should achieve *upon completion* of the course” (p. 263). The literature does provide clues, but not exactly defined guidance on the desired level of mastery in introductory CS0 courses. Their procedure to establish competencies was to first identify topics from the research and then apply the proper level of Bloom’s taxonomy to indicate the applicable level of ability.

### **Topics Identified from Professional Associations and Textbooks**

The identification of topics to cover in an introductory CS course is an arduous task. There are numerous subjects but of utmost consideration is the intended focus of the course. By definition, a breadth-first approach reflects a wide range of topics in the discipline (JTFCC, 2001) so a consideration of disparate areas is required.



Surakka (2007) used a literature review and group work to select 42 topics and skills important for software development professionals. He listed these in a questionnaire on commonly required CS skills, which he distributed to academic and industry specialists and students in Finland. The 19 topics that were rated at least a three on a four-point Likert scale by both the academic and industry groups were: data structures and algorithms, procedural programming, object-oriented programming, software architectures, operating systems, computer/data security, distributed systems, compilers, concurrent programming, computer architecture, design, implementation, requirements, test, concept exploration, version and configuration management, project management, and documentation. (Surakka, 2007).

Surakka (2007) found the industry group rated three items at least three out of four on a Likert scale that the academic group failed to rate as high (internet protocols, script programming, and systems programming). The academic group rated two items that the industry group did not rate at least a score of three (database management systems and implementing techniques of user interfaces) (Surakka, 2007). Because of the similarity in Surakka's research to this particular study, the topics he identified merit consideration for inclusion.

The JTFCC's recommendations from 2001 and 2013 were consulted for their direction on the desired competencies for students in introductory courses. While the JTFCC (2001) covered a multi-course introductory sequence, direction was gathered in the context of a single introduction to CS course.

The 2013 guidelines differed from those released in 2001, which had included specific topic suggestions for introductory classes. The JTFCC (2001) provided course descriptions for six different approaches (i.e. focus on imperatives, objects, functional, breadth, hardware, and algorithms). In the 2013 recommendations the JTFCC (2013) authors discussed introductory

pathways, focus on programming, programming languages, software development, parallel processing, and platform as topics to consider in introductory courses. They did not, however, make specific recommendations for introductory courses and left it to institutions to make decisions based on local needs.

The JTFCC's (2001) recommendations did, however, identify major topics for competencies. Specifically, suggestions were made for a breadth-first course, CS100B Preview of Computer Science covering the following topics: mathematical preliminaries, algorithms, algorithmic analysis, hardware realizations of algorithms, programming fundamentals, operating systems and virtual machines, networking and computer graphics, and social and professional issues (JTFCC, 2001). Topics in mathematics were eliminated as the institution that would host the course would present this material in a separate course.

The Liberal Arts Computer Science Consortium (LACS) released a model curriculum for liberal arts schools offering CS programs. The LACS (2007) based their suggestions on JTFCC's 2001 recommendations and included hours to focus on topics in introductory courses. They also offered objects-first and functional-first recommendations and topics from the first course from each of these sequences (CS1A and CS1B) were compared to the existing list. It was deemed that the subjects tracing, testing, and debugging and exceptions (LACS, 2007) were the only ones missing. These topics were combined and added to the existing list under the heading software development and engineering.

The JTFCC's 2013 curriculum recommendations were also analyzed for additional input. Specifically, the body of knowledge topics were reviewed to ensure all had corresponding coverage within the list identified for an introductory course. The topic of human-computer interaction was found to be missing and added to the list. Other topics were deemed to be

sufficiently covered by existing entries. As an example, platform-based development was thought to be addressed by the topics of information systems and World Wide Web and cloud computing.

Three textbooks that were deemed appropriate for a CS0 course were also consulted. The texts were *Connecting with Computer Science (CwCS)* (2nd edition) (2011) by Anderson, Ferro, and Hilton, *Invitation to Computer Science (ItCS)* (7th edition) (2016) by Schneider and Gersting, and *Computer Science Illuminated (CSI)* (6th edition) (2016) by Dale and Lewis. Dale and Lewis (2016) indicated in the preface to their text that they surveyed experts in CS and asked them to list four topics each for student mastery of CS0 and CS1 courses and four additional topics of the latter that required familiarity. Though they don't discuss their methodologies, they state the results, in conjunction with other research, formed the outline for the text. Tables of contents and indices of the texts were analyzed for treatment on topics and a list was developed. The lists from the five sources were then merged as necessary. The results are shown in Table 5.

This table lists 26 topics and the sources that give considerable coverage (more than a simple mention in the textbooks). All the subjects listed are covered in at least two of the sources except file structures. Anderson et al. (2011) dedicate an entire chapter to this area but similar attention was not found in the other sources.

### **Topics Identified in Journal Articles**

Additional resources in the literature discuss topics of concern in introductory CS courses. Many authors review topics in varying levels of detail while writing about an introductory course's design (Alvarado & Dodds, 2010; Cortina, 2007; Dodds et al., 2008; Huang, 2008; Kelly, 2007; Muñoz et al., 2013; Schneider, 2004; Whitfield, 2003). Other

Table 5

*Coverage of Potential Topics for Introductory Computer Science*

Topic	<i>CSI</i> (2016)	<i>ItCS</i> (2016)	<i>CwCS</i> (2011)	<i>JTFCC</i> (2001, 2013) & <i>LACS</i> (2007)
Algorithmic analysis	X	X		X
Algorithms and problem solving	X	X	X	X
Artificial intelligence	X	X		X
Basic computability	X	X		X
Binary values and number systems	X	X	X	X
Digital logic and digital systems	X	X	X	X
File structures			X	
Fundamental data structures	X	X	X	X
Fundamental programming constructs	X	X	X	X
Hardware realizations of algorithms	X	X	X	X
History of computing	X	X	X	X
Human computer interaction			X	X
Information systems	X	X	X	X
Language translation and compilers	X	X		X
Networking	X	X	X	X
Operating systems and virtual machines	X	X	X	X
Overview of programming languages	X	X	X	X
Parallel and distributed computing	X	X	X	X
Programming fundamentals	X	X	X	X
Recursion	X	X		X
Security of information and networks	X	X	X	X
Simulation, modeling, graphics, and gaming	X	X		X
Social and professional issues	X	X	X	X
Software engineering			X	X
Software verification and validation	X	X	X	X
World Wide Web and cloud computing	X	X	X	X

researchers simply mention subjects in CS when discussing a particular study that took place in the context of an introductory course (Bishop-Clark et al., 2007; Enbody et al., 2009; Norman & Adams, 2015; Wang, Su, Ma, Wang, & Wang, 2011). The topics identified from these other

sources can be divided into three main categories: programming, hardware and lower levels of the Open Source Interconnection (OSI) model (International Organization for Standardization [ISO]/International Electrotechnical Commission [IEC], 1994), and professional skills.

### **Topics in programming**

As previously mentioned, one of the more elementary topics in programming is algorithms. Authors have often written on the need for students to be able to read, write, and explain algorithms (Cortina, 2007; desJardins & Littman, 2010; Goldman et al., 2008; LaFrance & Roth, 1972; Schneider, 2004; Schulte & Bennedsen, 2006; Surakka, 2007; Walker & Kelemen, 2010; Walker & Schneider, 1996; Zhao et al., 2015). Goldman et al. (2008) discussed the necessity of exhibiting this skill to conceptualize problems and design solutions. Some authors specifically mentioned the need for students to have a working comprehension of classical CS algorithms, such as searching and sorting (desJardins & Littman, 2010; Schneider, 2004; Walker & Schneider, 1996). Students are thus expected to be able to create algorithms to solve problems they encounter.

Related to the actual creation of algorithms is the ability for students to analyze existing ones. Several authors mentioned the ability to analyze the efficiency of algorithms as being particularly desirable (Cortina, 2007; Roach & Sahami, 2015; Schneider, 2004; Schulte & Bennedsen, 2006). Students with this knowledge would be expected to be able to compare algorithms and identify higher quality solutions to problems. Skills related to algorithms can be learned independent of programming language. The understanding of programming essentials, however, are often taught in the context of a particular language or a paradigm.

### *Programming fundamentals*

The topic of programming fundamentals can receive varying levels of emphasis in introductory courses; whether they be of the CS0 or CS1 types. Tew (2010) attempted to identify the key concepts in a CS1 course using the CC2001 and four textbooks as sources to reduce the topics she had obtained from the literature. She focused only on the areas of fundamentals and object-oriented programming and discarded concepts in other categories (including SE, algorithms and complexity, etc.). Tew (2010) grouped concepts into a table arranged under headings of: expressions, control structures, functions/methods, data types and structures, and object-oriented programming; also included were the items variable, simple I/O, and recursion, which were not grouped under a dedicated heading (Tew, 2010). Goldman et al. (2008) also mentioned variables and recursion as a topic for an introductory course.

Several of the headings listed by Tew (2010) could be grouped under the heading of programming fundamentals and many have received distinct mention in the literature. These areas include expressions (Tew, 2010); control structures (Norman & Adams, 2015; Schneider, 2004; Tew, 2010), conditionals (Alvarado & Dodds, 2010; Walker & Schneider, 1996), functions and methods (Schneider, 2004; Tew, 2010), and the previously mentioned topic of recursion (Alvarado & Dodds, 2010; Cortina, 2007; Dorn, 2011; Schulte & Bennedsen, 2006; Walker & Schneider, 1996; Winter, 2014). These items could be considered as suggested competency topics for a CS1 course emphasizing programming and warranted some consideration in a CS0 course that includes programming.

In addition to these subtopics of programming fundamentals were subjects related to data. Many authors included data types and structures, and arrays and lists (Alvarado & Dodds, 2010; Cortina, 2007; Roach & Sahami, 2015; Schneider, 2004; Schulte & Bennedsen, 2006; Surakka,

2007; Tew, 2010; Walker & Kelemen, 2010; Walker & Schneider, 1996; Whitfield, 2003).

Whitfield (2003) specified in a learning outcome that students “list the scalar data types supported by the core language, identifying the domain of and the operations defined for each type” (p. 215). Students would be expected to describe the role and functionality of data structures in computer programs.

A final consideration related to programming fundamentals is the choice of whether or not to focus on a paradigm. The JTFCC (2001) provided for three distinct programming-first models in their curriculum recommendations (imperative-first, objects-first, and functional-first). The object-oriented paradigm warrants particular mention as it has often been mentioned in the context of introductory courses (Ali & Mensch, 2008; Alvarado & Dodds, 2010; Goldman et al., 2008; Norman & Adams, 2015; Schulte & Bennedsen, 2006; Surakka, 2007; Tew, 2010). The choice of a particular paradigm and potential focus on fundamentals related to object-oriented programming in a CS0 course should be considered. Another topic for a potential competency is an overview of programming languages (Cortina, 2007; Walker & Schneider, 1996). These fundamentals all form a foundation for the act of programming, which is another potential competency for students in an introductory course.

### *Developing programs*

Writing programs is often the main focus of an introductory CS course (Alvarado & Dodds, 2010; Ali & Smith, 2014; Baldwin, Brady, Danyluk, Adams, & Lawrence, 2010; Bishop-Clark et al., 2007; Dodds et al., 2008; Forte & Guzdial, 2005; Schneider, 2004; Wang et al., 2011; Whitfield, 2003). Schneider (2004) included the skills of compiling, testing, and debugging as being elementary to programming. Wang et al. (2011) specified that students have the ability to “write, type in, correct and run programs” (p. 220). Baldwin et al. (2010) noted these skills were

taught in an introduction to CS course at a liberal arts college. Programming can be an intensive competency and can be the sole focus of introduction courses, especially those of the CS1 variety. It can also, however, be associated with related subtopics and various outcomes.

Kelleher and Pausch (2005) defined programming as “the act of assembling a set of symbols representing computational actions” (p. 83). This ability, however, is tied to other related competencies, including “how to express instructions to the computer (e.g., syntax), how to organize these instructions (e.g., programming style), and how the computer executes these statements” (Kelleher & Pausch, 2005, p. 86). Whitfield (2003) specified a competency that called for students to “identify and write code containing various statements supported by the core language, including assignment, input, output, selection, iteration, and function call” (p. 215). The foundations topics mentioned previously are of utmost importance for students to properly develop the ability to program. Additionally, other topics in CS related to programming have also been mentioned for primary application in introductory courses.

### *Applications, techniques, and processes*

The applications, techniques, and processes of programming identify additional subjects for potential competencies to be considered. Applications of programming include artificial intelligence, information systems, computer graphics, and internet protocols, for example. Authors have written about artificial intelligence as a topic of study in an introductory CS course (Bishop-Clark et al., 2007; desJardins & Littman, 2010; Huang, 2008). Huang (2008) shared a design for an introductory CS course emphasizing the principles of artificial intelligence as a major theme. He stated that

Regardless of whether they take future CS courses, students will be exposed to the challenge, power, and beauty of designing and implementing algorithms, as well as the



wide-reaching impact of computer science and the myriad opportunities available for those who study it. (Huang, 2008, p. 101)

A topic as seemingly as complex as artificial intelligence can, therefore, be considered in a breadth-first introduction course.

The subject of information systems also warrants consideration. Many non-majors, including those studying business for example, might find the subject especially useful. Authors have included mention of information systems and structured query language (SQL) databases (Cortina, 2007; Poulouva & Klimova, 2015; Surakka, 2005; Surakka, 2007). Surakka (2005) identified database management systems as one of the important subjects for graduates in software systems programs in Finland. Interestingly the academic group in the Delphi portion of his study placed more interest on this area than did the industry experts (Surakka, 2007). Other applications of programming include the World Wide Web and internet protocols (Surakka, 2007), computer graphics (desJardins & Littman, 2010), and modeling for simulation (Norman & Adams, 2015).

Techniques of programming include those skills that programmers rely on in their activities. These topics include information, or data, representation (Fulton & Schweitzer, 2011; Walker & Schneider, 1996), documentation (Surakka, 2007), user-interface techniques (i.e. human-computer interaction) (Surakka, 2007), and basic computability (desJardins & Littman, 2010). The topic of binary numbers, or numbering systems in general, (desJardins & Littman, 2010; Goldman et al., 2008) could be considered a subset of data representation. A major related topic is SE, which has become increasingly popular in recent years.

Software development and its associated processes has garnered significant interest in introductory courses (Forte & Guzdial, 2005; Kelly, 2007; Muñoz, et al., 2013; Poulouva &

Klimova, 2015; Schulte & Bennedsen, 2006; Vitkutė-Adžgauskienė, & Vidžiūnas, 2012; Walker & Schneider, 1996; Zhao et al., 2015). Agile methods, such as pair programming are popular practices in early programming courses and these have sometimes been found to improve student persistence (Barker, McDowell, & Kalahar, 2009; Bishop-Clark et. al, 2007; Guercio & Sharif, 2012; Horton, Craig, Campbell, Gries, & Zingaro, 2014; McDowell, Werner, Bullock, & Fenald, 2006; Porter, Guzdial, McDowell, & Simon, 2013; Rubio, Romero-Zaliz, Mañoso, de Madrid Teague & Roe, 2007).

Including topics and methodologies from SE can help shape the curriculum for an introductory course. Vitkutė-Adžgauskienė and Vidžiūnas (2012) promoted “shifting the focus from teaching programming paradigms towards concentrating on main software engineering concepts” (p. 280). Goldman et al. (2008) identified topics such as debugging/exception handling and designing tests as among the most important related to program design. Surakka (2007) identified several topics that were important to both academics and industry professionals including design, requirements, test, version and configuration management, project management, etc. These topics in SE, therefore, should also be considered by the experts in this Delphi study.

### **Topics in hardware and other levels of the OSI model**

The topics mentioned thus far reside in the programming and applications layers of the OSI model (Dale & Lewis, 2016). Several topics have been identified for introductory CS courses that deal with hardware and computing activities associated with other levels of the OSI model (OSI/IEC, 1994). The subject areas related to the study of hardware include digital circuits and Boolean logic (desJardins & Littman, 2010) and computer architecture and organization (Alvarado & Dodds, 2010; Bishop-Clark et al., 2007; desJardins & Littman, 2010; Poulova &

Klimova, 2015; Surakka, 2007). The study of computer architecture introduces students to the hardware systems and components that are used in computing. A related topic that has identified as increasingly important is parallel and distributed systems and programming (Roach & Sahami, 2015; Surakka, 2007; Winter, 2014).

Other associated areas sometimes encompass multiple layers of the OSI model (OSI/IEC, 1994). These include operating systems, (Surakka, 2007); computer networks (Bishop-Clark et al., 2007; Poulouva & Klimova, 2015), and compilers (Bishop-Clark et al., 2007; Cortina, 2007; Surakka, 2007). Operating systems form the interface between a student's programming and hardware (Dale & Lewis, 2016; Schneider & Gersting, 2016). Compilers are used to translate a program into executable source code (Dale & Lewis, 2016; Schneider & Gersting, 2016). The topic of networks deals with aspects of communication and associated hardware in computing. Two additional subtopics that are related to networks are data compression (desJardins & Littman, 2010) and computer and data security (desJardins & Littman, 2010; Fulton & Schweitzer, 2011; Surakka, 2007; Whitfield, 2003). This latter area has also received increasing attention in recent years (JTFCC, 2013).

A final subject related to computer hardware and programming is the history of computing. This topic has also been mentioned as an area of focus for introductory courses (Cortina, 2007; Walker & Schneider, 1996) and allows students to see the development of computer artifacts and processes over the past seven decades. These areas constitute a breadth of topics that are of particular interest for CS majors, and of debatable importance to non-majors. Of less question is the relevance of professional skills to both groups.

### **Professional skills**

Professional skills are often emphasized both for non-majors and majors in introductory CS courses (Muñoz, et al., 2013). Sometimes referred to as soft skills, they involve the mechanics one utilizes to perform job activities. While this identifier can mean different specifics for majors and non-majors, it is nonetheless an important component to any introductory CS course which houses both groups. Roach and Sahami (2015) stated that

any CS curriculum should prepare graduates to succeed in a rapidly changing field; thus, it must prepare students for lifelong learning and include professional practice elements—communication skills, working in teams, ethics, and so on—as components of the undergraduate experience. (p. 116)

The authors referred to the curriculum of a program as a whole but their statement can easily be put into the context of an introductory course. These skills range from those that can be developed in various courses and subjects, such as teamwork and communication, to others in which CS specifically has been found to be effective and unique context for their development.

An introduction to CS course for majors and non-majors alike provides interdisciplinary opportunities to develop such skills. Several researchers have discussed the development of teamwork, interpersonal, and group skills (Guercio & Sharif, 2012; Muñoz et al., 2013; Soper, 2014; Walker & Kelemen, 2010; Whitfield, 2003). Soper (2014) wrote that according to Ed Lazowska, of the University of Washington, teamwork is one of the most important competencies to develop in a CS course.

### ***Teamwork and collaboration***

Even though computer programming is often associated with individual work in which one works alone at a computer terminal (Teague & Roe, 2007), it heavily requires working with

others. It has been found that CS programs have work to do to dispel these misconceptions and show students that understanding can be developed effectively in team situations (Lewis, Jackson, & Waite, 2010). Muñoz et al. (2013) specified the need to work both autonomously and in interdisciplinary teams. An introductory CS course can be designed to offer opportunities for both experiences to students.

An emphasis on teamwork seemingly has other benefits. Law, Lee, and Yu (2010) studied the key factors that motivate students to learn in computer programming courses and found only “social pressure and competition” (p. 226) correlated with efficacy. These experiences provided in a CS course, therefore, can help students to develop self-worth. Additionally, Barker et al. (2009) found increased student-to-student interaction was the most significant determining factor of CS majors in an introductory course to persist in their program. The previously mentioned method of pair programming provides occasions to develop these skills. McDowell et al., (2006) found pair programming to specifically improve student persistence in CS and quality of programs, and to increase enjoyment and confidence of students in an introductory course; they suggested the learning technique as a potential solution to improving the performance of URMs.

An emphasis on developing teamwork capabilities has perceived benefits for students. Sometimes this focus can provide students with chances to develop other competencies. Teague and Roe (2007) stated that:

Encapsulating collaboration into learning to program can effectively utilise the resources already available, encourage more vigorous and active engagement by students; encourage them to think aloud and verbalise every step of their problem solving process, as well as satisfy their intense need for interaction and support. (p. 17)

An ability mentioned here, problem solving, is another highly important professional skill that can be developed in an introduction to CS course.

*Problem solving and related attributes*

Problem solving was one of the most often mentioned competencies for introductory CS courses in the reviewed literature (Barberà et al., 2014; Cheng et al., 2010; Cortina, 2007; Dodds et al., 2008; Enbody et al., 2009; Fulton & Schweitzer, 2011; Guercio & Sharif, 2012; LACS, 2007; Muñoz et al., 2013; Norman & Adams, 2015; Roach & Sahami, 2015; Schneider, 2004; Schulte & Bennedsen, 2006; Shein, 2014; Sonnier, 2013; Walker & Schneider, 1996; Walker & Kelemen, 2010; Wang et al., 2011; Whitfield, 2003; Zhao et al., 2015). This competency has been linked with other benefits, some that are especially important to majors. Teague and Roe (2007) stated that “the basic problem solving process template has been instrumental in highlighting the advantages of good documentation in the early stages of program design, especially for more challenging exercises by novice programmers” (p. 11).

Other authors have repeated the importance of structure in the problem solving approach and considered other benefits, which are desirable to students from all disciplines. Poulova and Klimova (2015) argued for the importance of problem analysis as a key competency. They identified it as the “ability to approach the problem broadly and consider connections, ability to structure the problem, its generalization or on the contrary, its specification” (Poulova & Klimova, 2015, p. 1999). Tackling a problem broadly relies on a capability to properly define it. The ability to seek and analyze information from different sources is related and has also been mentioned (Enbody et al., 2009; Muñoz et al., 2013).

It is no wonder that problem solving, therefore, receives considerable mention. Sonnier (2013) reasoned about the role of a CS program in liberal arts institutions, stating the intent “to

tie classical logic to digital logic and integrate an understanding of modern digital concepts into the traditional liberal arts, the program should have a balance of theory and application and focus on problem solving” (p. 119). Thus the ability to solve problems can indeed be viewed as a central emphasis for any CS course. One can consider, though, that to be able to solve problems effectively, students must develop alternate ways of thinking.

### ***Ways of thinking***

The study of CS has been linked with methods of thinking that can be viewed as professional skills. These include systems, computational, algorithmic, critical, and creative thinking. Poulouva and Klimova’s (2015) previous mention of “an ability to approach a problem broadly” (p. 1999) is related to a capacity to view the problem’s environment as a system. Muñoz et al. (2013) also mentioned systems thinking as a skill to be developed by an introduction to CS course. The ability to maximize understanding of one’s environment is one that is translatable to varied domains; so too are algorithmic and computational thinking.

The typical focus on algorithms in CS courses has already been mentioned. Algorithmic thinking has been identified as a desirable attribute to be gained by students studying CS (Courte & Howard, 2005; Fulton & Schweitzer, 2011; Gupta, 2007; Katai, 2014; Kiss, 2013; Lazowska et al., 2014; Norman & Adams, 2015; Schneider, 2004). Katai (2014) argued for algorithmic thinking as an important skill, pointing out that “many fields of modern life involve the processes of following procedures, applying protocols or implementing techniques, all of which can be viewed as human-processed algorithms” (p. 287). Liberal arts colleges and universities have been known to emphasize this skill in CS programs. Baldwin et al. (2010) presented examples of five CS programs at liberal arts institutions and differentiated them from offerings at other colleges and universities. They specifically noted an increased emphasis on algorithms, stating

that there is “a general consensus among the liberal arts programs to require more work in the algorithms and complexity area than required by the ACM/Computer Society model, and to treat material required by the ACM/Computer Society model in areas such as net-centric computing, graphics and visual computing, and intelligent systems as desirable but optional” (Baldwin et al., 2010, p. 27).

Related to algorithmic thinking is the aforementioned general skill of computational thinking, which has also received much attention as a desirable skill (Czerkawski & Lyman, 2015; Dorn, 2011; Forte & Guzdial, 2004; Franklin, 2015; Shein, 2014; Syslo, 2015; Walker & Kelemen, 2010; Wing, 2006; Zhao et al., 2015). Wing’s (2006) previously mentioned definition included recursion, abstraction and decomposition and heuristic reasoning as attributes for students to learn and utilize. Syslo (2015) considered computational thinking as an extension of algorithmic thinking while Shein (2014) indicated it “helps people learn how to think abstractly and pull apart a problem into smaller pieces” (p. 17). This capability has been increasingly referred to as a general skill that is necessary for all (Czerkawski & Lyman, 2015; Wing, 2006). Critical thinking is yet another related skill.

The ability to think critically is one that is often mentioned as a desirable professional skill that can be developed in CS courses (Barberà et al., 2014; LACS, 2007; Muñoz, et al., 2013; Voskoglou & Buckley, 2012; Whitfield 2003). Voskoglou and Buckley (2012) pointed out that though a universal definition is elusive, critical thinking is a foundational skill for computational thinking and “plays a central role in knowledge acquisition and creation” (p. 41). The authors warn, however, that although acquiring knowledge is important, it is overshadowed by the ability to think creatively (Voskoglou & Buckley, 2012).



Creativity has already been mentioned as a translatable general skill of potentially high importance for majors and non-majors alike. This attribute has obvious implications for one's ability to perform varied tasks, including solving problems. Creative thinking has often been referenced as a desirable attribute for CS students (Cheng et al., 2010; Forte & Guzdial, 2004; Lewis et al., 2010; Poulouva & Klimova, 2015). As with any of the aforementioned attributes, it can receive differing emphasis in one course of study versus another. Lewis et al. (2010) suggested that "faculty must consider ways to move students toward the idea that 'the work you do in computer science in the real world requires a lot of creativity,' rather than away from it" (p. 85). These ways of thinking have received considerable attention as targeted competencies for students and there is some level of overlap between them. The potential list of competencies that can be identified as professional skills includes others not yet mentioned.

#### *Other professional attributes*

Additional competencies outside those of problem solving and ways of thinking are also beneficial to CS majors and non-majors alike. One that is often mentioned in the context of CS study is acting ethically and exhibiting responsibility and accountability (Guercio & Sharif, 2012; Schulte & Bennedsen, 2006; Muñoz et al., 2013; Walker & Kelemen, 2010; Walker & Schneider, 1996; Whitfield, 2003). Related to these behaviors is an understanding of the societal impact of CS (Bishop-Clark et al., 2007; Huang, 2008; Roach & Sahami, 2015). These competencies are important as computing relies on technology. As with any other technology, there is strong interplay with society, which therefore requires a solid comprehension of this relationship (Pearson & Young, 2002).

Also related are skills that help students to use computer technology effectively in their careers. Digital literacy and working with computers, their systems, and software, have been

listed among desirable competencies for introductory CS students (Enbody et al., 2009; LaFrance & Roth, 1972; Schneider, 2004). LaFrance and Roth (1972) discussed an introductory CS course for liberal arts institutions and alluded to digital literacy pointing out that “foundational to the program is a service course enabling persons from all departments to learn how to make effective use of the computer in their discipline” (p. 22). The importance of these skills has only grown since the statement was made four decades ago and there is benefit to all students. Related to the concept of digital literacy is media computation (Forte & Guzdial, 2004; Forte & Guzdial, 2005; Porter et al., 2013). Porter et al. (2013) indicated that media computation “explained how digital media are manipulated” (p. 35).

Other professional skills have been identified as being noteworthy. These include communicating orally and in writing (Guercio & Sharif, 2012; Muñoz et al., 2013; Whitfield, 2003), self-learning and self-assessment (Muñoz et al., 2013), managing time and resources (Guercio & Sharif, 2012; Muñoz et al., 2013), exhibiting entrepreneurship (Muñoz et al., 2013), and meeting specifications with a designed solution (Poulova & Klimova, 2015; Whitfield, 2003). Finally, Muñoz et al. (2013) mentioned career planning as being beneficial to cover in introductory courses for CS majors specifically. This list of competencies identified from sources outside professional associations and textbooks is indeed sizable but there is overlap.

### **Straw Model of Competencies**

A straw model was developed using the information on competencies gathered from this review of literature. Topics from association curriculum recommendations and the three textbooks were synthesized with the other sources from the literature to form a comprehensive list of topics. Action verbs from Bloom’s taxonomy (Anderson & Krathwohl, 2001) were applied

to topics to express the intent of student mastery in an introductory CS course as called out in the literature.

Although identification of potential competencies from the curriculum recommendations and textbooks and journal articles was done independently, 24 of the 26 topics in the former sources were found in the latter group. Only the topics of file structures and verification and validation were not identified from analysis of the journal articles. The second topic could, however, be implied in the subject of SE. In addition to these topics, the authors in the journal articles identified competencies associated with the act of programming, including writing procedural and object-oriented programs and documenting them; and several professional skills. The act of writing programs was found to encompass the competencies concerning data types and structures; programming fundamentals; and expressions, control structures, functions, and methods so these were combined accordingly. Finally, the topic of social and professional issues was separated due to its broad scope. In all, 38 competencies were identified and these are listed in Table 6. Note that sources are coded as text, for textbooks and curriculum recommendations; article, for items identified in professional journal articles; or both, when the competency was identified in both these types of sources.

The straw model in Table 6 was to be provided to the experts from academia and industry for their consideration in this study. Eighteen of these competencies were identified from curriculum recommendations, texts, and articles. Another 16 competencies originated from analysis of journal articles. Only two of the items originated in only either curriculum recommendations or the textbooks. The experts would be asked to deliberate on the importance of these items and provided the opportunity to add any competencies deemed to be lacking.

Table 6

*Introduction to Computer Science Competencies Straw Model*

Competency	Source(s)
Analyze algorithms for effectiveness and efficiency	Both
Illustrate concepts in artificial intelligence	Both
Summarize basic computability, theory of computation, and its limits	Both
Describe different types of data representation (e.g. graphics, binary numbers, etc.)	Both
Illustrate the use of Boolean logic and basic combinational digital circuits	Both
Describe basic computer architecture and organization	Both
Summarize the history of computing and its ramifications to implementation today	Both
Explain the factors contributing to human-computer interaction in computing	Both
Illustrate the use of databases and apply SQL	Both
Explain the operation of compilers	Both
Discuss the operation of networks and related practices (e.g. data compression, etc.)	Both
Explain the functionality of operating systems and provide examples	Both
Describe common programming languages and their popular uses	Both
Describe benefit and operation of parallel and distributed systems and programming	Both
Demonstrate use of recursion in a program	Both
Describe the need for computer and data security and identify best practices	Both
Explain the role of modeling and simulation in computing	Both
Describe societal impact of computing	Both
Describe the World Wide Web and select internet protocols	Both
Describe process and practices in software engineering	Both
Plan a career in CS	Article
Write functional object-oriented programs employing programming fundamentals	Article
Write functional procedural programs employing programming fundamentals	Article
Implement good documentation practices in programming	Article
Demonstrate teamwork and interpersonal group skills	Article
Demonstrate algorithmic thinking.	Article
Demonstrate computational thinking	Article
Demonstrate problem solving	Article
Demonstrate critical thinking and reasoning	Article
Demonstrate systems thinking	Article
Demonstrate creativity in programming	Article
Demonstrate time and resource management skills in a project	Article
Exhibit entrepreneurship in computing	Article
Communicate effectively orally and in writing	Article
Describes self-learning and assesses self	Article
Exhibit digital literacy	Article

Table 6 Continued

Competency	Source(s)
Explain and choose from different file structures	Text
Explain and utilize effective procedures in software verification and validation	Text

## Programming Languages

### Overview

Though CS1 courses typically have a significant emphasis on computer programming, CS0 courses include this element to varying degrees (Davies et al., 2011). Some institutions may choose to offer an introductory course with a reduced emphasis on programming (Cortina, 2007; desJardins and Littman, 2010) but even these have included it in the curriculum. There are reportedly up to 2,500 programming languages that have been developed (Kinnersley, n.d.), though not all of these are still actively used. Regardless, there are several languages that could potentially be utilized to introduce students to computer programming. Shein (2015) quoted Shriram Krishnamurthi, a CS professor from Brown University, who stated that “ever since (Blaise) Pascal introduced the idea of ‘one programming language for introductory programming education,’ the community has been stuck in a rut of trying to find one and then arguing about it” (p. 21). Though there have been conflicting findings on whether the choice of language in this context significantly affects performance (Kunkle, 2010; Watson & Li, 2014), many researchers have analyzed the factors influencing student comprehension.

While the choice of programming language for an introductory course will depend to an extent on the desired student competencies, languages can be assessed on their own general merit. Of utmost importance in an introductory course is a student’s ability to learn how to use it. Student motivation and accessibility and utility of the language warrant consideration.

Forte and Guzdial (2005) surveyed students in three tailored introductory CS course offerings at Georgia Institute of Technology and learned that engineering students found choice of programming language to be highly important because they were “eager to learn a language that [would] help them perform their jobs” (p. 251). Other students, who were more interested in communication, were not as motivated by choice of a particular programming language and were more concerned with an avenue to express themselves meaningfully and creatively (Forte & Guzdial, 2005).

Another factor is accessibility to a programming language. Students typically begin their collegiate studies with little to no experience with programming (Winter, 2014). Hurdles, such as syntax and semantics, can often make it difficult for beginner programmers (Kelleher & Pausch, 2005; Malan & Leitner, 2007; McIver, 2001; Norman & Adams, 2015; Stefik & Gellenbeck, 2011). Syntax, for example, can prevent a programmer from seeing results because of problems with something as trivial as punctuation.

This barrier to learning has long been recognized by the computing education community and languages have been developed to help address these issues to some extent. Kelleher and Pausch (2005) developed a taxonomy of programming languages and identified three approaches that have been used to make languages more approachable, including “1) simplifying the language, 2) tailoring the language for a specific, small domain of programming problems, and 3) preventing syntax errors” (p. 88). Powers, Ecott, and Hirshfield (2007) pointed out that eliminating frustration due to errors in syntax can help improve student confidence. These efforts have resulted in a plethora of languages available for introductory courses in CS. The JTFCC (2013) acknowledged that most introductory courses emphasize programming and commented on the popularity of certain programming languages:

There does, however, appear to be a growing trend toward 'safer' or more managed languages (for example, moving from C to Java) as well as the use of more dynamic languages, such as Python or JavaScript. Visual programming languages, such as Alice and Scratch, have also become popular choices to provide a 'syntax-light' introduction to programming; these are often (although not exclusively) used with non-majors or at the start of an introductory course. (p. 42)

These trends appear to indicate a desire to increase accessibility of all students, majors and non-majors alike, to programming. Several researchers have found a positive link between using visual languages (e.g. Alice, Scratch, Greenfoot) and increased confidence, enjoyment, and understanding of programming (Bishop-Clark et al., 2007; Daly, 2011; Powers et al., 2007). Students who are not overly concerned with syntax can focus on solving problems creatively and develop important competencies.

Arguments have been made, however, for not shielding students from syntax. Zhao et al. (2015) debated that knowledge of syntax helps students to understand the process of software design and development, master the basic methods of constructional and object-oriented programming, ... understand computational thinking on how to describe and solve specific problems by computers, as well as the foundational methodologies of software system design and implementation. (p. 196)

Certainly these skills are desirable as well. The question of which programming language is best to learn in an introductory CS course for both majors and non-majors can be difficult to answer.

CS majors especially want to learn to use languages that are applicable to their future careers (Forte & Guzdial, 2005) though different communities will vary on their preferred

programming languages (Meyerovich & Rabkin, 2013). Meyerovich and Rabkin (2013) argue it is desirable for majors to learn multiple languages to improve their versatility. Curriculum designers, therefore, have many options to choose from depending on the needs of the course and their students. The many attributes of the languages available help determine their usage in both academia and industry. Their popularity, however, is not necessarily the same in both domains.

### **Language Popularity in Industry**

Determining the usage of certain programming languages over others can be challenging. There are valuable sources of information, however, that provide input. The monthly TIOBE Programming Community index is one such resource. The TIOBE index for December 2015 (2015) ratings are based on the numbers of skilled professionals using languages according to information from web search engines (e.g. Google, Yahoo!, Wikipedia, Amazon, YouTube, etc.). The authors, however, warn that the “index is not about the best programming language or the language in which most lines of code have been written” (TIOBE Index for December 2015, 2015, para. 2). Ben Arfa Rabai, Cohen, and Mili (2015) compared the TIOBE programming index to other resources and found it a valuable indicator of programming language use in industry.

Another source for programming language popularity is RedMonk. The software developer analyst firm releases its rankings twice each year and aims to “correlate language discussion (Stack Overflow) and usage (GitHub) in an effort to extract insights into potential future adoption trends” (RedMonk, 2015a, para. 1). A similar resource is the PYPL Popularity of Programming Language list. Carbonelle (2015) explained the PYPL rankings as being based on online searches for tutorials, making it a leading index indicating future use. The Trendy Skills resource uses job advertisements online (e.g. Monster.com, etc.) to gauge the software



industry's need for particular programming languages in 13 countries and ranks the top ten based on the results. Black Duck Software (2015) tracks open source projects and the relative popularity of languages used. Finally, IEEE Spectrum also provides a ranked list of languages and its methodology is similar to that used in the TIOBE index, though with some different primary sources. The "rankings are created by weighting and combining 12 metrics from 10 sources" (Diakopoulos & Cass, 2015, para. 1). Table 7 lists the rankings obtained from these six resources. Only languages that were listed in at least two of the six sources were included. The data in Table 7 were based upon usage or projected use in industry. It is interesting to note that Java, C, C++, Python, C#, PHP, and JavaScript are ranked in each of the six lists. It can be safely assumed that these are highly popular languages in industry. Perl and Ruby are also highly considered as they are ranked in five of the six resources referenced and Visual Basic, Swift, Objective C, MATLAB, R, and Scala are mentioned in four of six. Assembly language, PL/SQL, and Shell were all listed in half of the sources. The list is important to consider as a source of programming language options for industry members to consider in this study. While these languages also warrant consideration by academic experts, it is expected that industry members will have familiarity with these languages.

### **Language Popularity in Academia**

Data for academic use of programming languages was more difficult to determine. One reason may be the tendency for language use to have a less uniform distribution in academia than in industry (Ben Arfa Rabai et al., 2015). Four sources, however, were identified that listed recent data. Red Monk (2015b) used mentions in the curriculum of the Top Ten Forbes Colleges and Universities to rank the top twenty languages. Guo (2014) used U.S. News and World Report's top 39 CS departments and their use of the top seven programming languages in CS0

Table 7

*Programming Language Popularity Rankings in Software Industry*

Language	TIOBE	RedMonk	PYPL	Trendy Skills	Black Duck	IEEE Spectrum
Java	1	2	1	1	4	1
C	2	9	6	6	2	2
C++	3	5	5	7	3	3
Python	4	4	2	9	10	4
C#	5	5	4	3	11	5
PHP	6	3	3	5	5	7
JavaScript	8	1	7	2	1	8
Perl	9	11	15	--	14	15
Ruby	10	5	12	--	7	9
Assembly Language	11	--	--	--	12	13
Visual Basic	12	19	13	--	--	16
Delphi/Object Pascal	13	--	--	--	--	29
Swift	14	18	9	--	--	19
Objective C	15	10	8	--	--	20
MATLAB	16	17	11	--	--	10
R	18	13	10	--	--	6
PL/SQL	19	--	--	--	13	12
Fortran	22	--	--	--	--	28
D	23	--	--	--	--	25
Groovy	24	19	--	--	--	--
SAS	27	--	--	--	--	26
Scala	28	14	16	--	--	18
Lisp	29	--	--	--	--	27
Shell	--	12	--	--	13	11
Go	--	15	--	--	--	14
Haskell	--	15	--	--	--	30
Lua	--	--	17	--	--	24

and CS1 classes. Ben Arfa Rabai et al. (2015) conducted a survey on programming language

usage in CS1 courses at 134 U.S. academic institutions in 2013 and included the top 12 in their

article. Davies et al., (2011) surveyed over 200 U.S. institutions offering CS0 courses and listed 11 languages that were used at multiple institutions. Table 8 lists the top ten (seven from Guo's work) languages based upon use in higher academia.

Table 8

*Programming Language Popularity Rankings in Academia*

Language	RedMonk (2014)	Guo (2014)	Ben Arfa Rabai et al. (2013)	Davies et al., (2011)
C	1	4	4	7
Java	2	2	1	4
C++	3	5	2	6
Python	4	1	3	2
MATLAB	5	3	5	--
JavaScript	6	--	9	5
ML	7	--	--	--
Objective-C	8	--	--	8
C#	9	--	6	--
Haskell	10	--	7	--
PHP	--	--	8	--
Scheme	--	6	10	--
Scratch	--	7	--	10
Alice	--	--	--	1
Visual Basic	--	--	--	3
Lisp/Scheme	--	--	--	8

One observation about this list is that the top five languages according to the most recent three sources are the same. The most popular languages as used in academia appear to be Java, C, C++, and Python as these four appear in each of the four lists. MATLAB appears in all but the research provided by Davies et al., mainly because their work focused on CS0 and this language

is associated with more technically focused courses in engineering. JavaScript appeared in three sources, but did not make Guo's (2014) top seven. C#, Haskell, Scheme, and Scratch appeared in two of the four sources. Davies et al. (2011) remarked on the previously mentioned "novice focused environments" (p. 627) like Alice, and noted that Scratch and Greenfoot were only beginning to be implemented in introductory courses. These three languages warrant particular attention as they have been developed to teach beginner programmers.

Visual languages allow beginning users to develop programs by clicking and dragging potential commands as opposed to typing them out. These languages were originally designed to instruct younger or at-risk CS students (Chang, 2014; Cooper, 2010). Urness and Manley (2011) noted they "make programming accessible and immerse the programmer in a media-rich environment, which is appealing to larger audiences who might otherwise disregard computer science because of preconceived perceptions about programming" (p. 272). These environments have garnered significant attention in recent years and researchers have presented their advantages and disadvantages.

As previously mentioned, it can be highly desirable for instructors to use languages that introduce programming concepts without the hurdles associated with syntax. The Alice programming language was developed at Carnegie Mellon University (Kelleher & Pausch, 2005) and has been identified as a means for students to develop creativity in programming without having to focus on syntax issues (Ali & Mensch, 2008; Ali & Smith, 2014). Bishop-Clark et al. (2007) performed a mixed methods study in which 154 students, most of which were non-majors, in an introductory computing course participated in a 2.5-week unit using Alice to introduce fundamental programming concepts. They found that students experienced increased confidence, enjoyment, and understanding of programming but were frustrated with its

limitations (Bishop-Clark et al., 2007). Courte and Howard (2005) used Alice in a non-majors CS course and found students experienced increased enjoyment and positive attitudes toward programming. While Moskal, Cooper, Munson, and Dann (2008) did not observe an effect on students' attitudes as a result of using Alice, they did find a positive effect on conceptual knowledge.

Scratch was developed by educators at the Massachusetts Institute of Technology in 2003 and launched four years later (Maloney, Resnick, Rusk, Silverman, & Eastmond, 2010). Maloney et al. (2010) pointed to Scratch's strengths as allowing tinkering for beginners, making program execution directly visible, a lack of error messages, and reduced data types. They did, however, acknowledge the deficiency of a lack of support for procedures (Maloney et al., 2010). It is interesting to note that Scratch was the only visual programming language that made at least two lists in Tables 7 and 8. One significant disadvantage is potential frustration and dissatisfaction on the part of experienced programmers using Scratch (Tanrikulu & Schaefer, 2011).

Greenfoot, developed at the University of Kent, has the reported strengths of illustrating object-oriented concepts, scalability, and ease to begin programming but potential drawbacks with error handling and reporting and the use of a two-dimensional system (Kölling, 2010). The language and environment have been used in research efforts aimed at improving student performance in introductory CS courses (Rolka & Remshagen, 2015) and educators at the postsecondary level have found students to enjoy working with it (e.g. Zur et al., 2014).

Researchers have observed favorable results with the use of these languages as gateways to other languages. Malan and Leitner (2007) found 76% of students believed Scratch was a positive influence on their ability to later learn Java. Daly (2011) found "students in [a]

Alice/Java course had higher levels of confidence at the end of the course in all of the categories when compared to the course using pure Java programming” (p. 28).

On the negative side, such languages may not be as popular with CS majors or may cause issues for those moving onto other languages in later courses. Ali and Smith (2014) warned that “majors may not find [Alice] very challenging or interesting” (p. 7). Cooper (2010) pointed to Alice’s lack of dynamic object creation, burdensome visual arrays, and the promotion of trial-and-error troubleshooting techniques. Powers et al., (2007) cautioned “the object model in Alice can easily lead to misconceptions, and although the lack of syntax errors can raise students’ confidence while programming in Alice, it can be detrimental when these same students transition to C++ or Java” (p. 217).

### **Straw Model of Programming Languages**

A list of programming languages to be considered by experts in industry and academia should include data from both sources. Sources of data from industry were more plentiful than from academia. Using the guideline to include languages that were identified in at least three of the six industry sources in Table 7, or in at least two of the four sources in Table 8, and the three visual programming languages that warranted inclusion, the straw model of programming languages shown in Table 9 was constructed.

This list of 23 programming languages was meant to include those that could faithfully represent current use in academia and industry. Eleven of the languages (C, C#, C++, Java, JavaScript, MATLAB, Objective-C, Perl, Python, and Scala) made this list based on their popularity in both academic and industrial environments. Eight languages were included based on their popularity in industry only (Assembly language, PHP, PL/SQL, R, Ruby, Shell, Swift, and Visual Basic). The academic community researched in the literature contributed another five

languages (Alice, Greenfoot, Haskell, Scheme, and Scratch). This study's participants would be given the opportunity to add to the list in the Delphi's first round in the event that anyone felt a worthy language had been excluded.

Table 9

*Programming Language Straw Model*

Language	Source(s)
Alice	Academia
Assembly Language	Industry
C	Both
C#	Both
C++	Both
Greenfoot	Academia
Haskell	Academia
Java	Both
JavaScript	Both
MATLAB	Both
Objective-C	Both
Perl	Both
PHP	Industry
PL/SQL	Industry
Python	Both
R	Industry
Ruby	Industry
Scala	Both
Scheme	Academia
Scratch	Academia
Shell	Industry
Swift	Industry
Visual Basic	Industry

## Assessments

The third category of items the industry and academic groups would be asked to consider were the assessments for an introductory CS course. Differentiation should be made between the terms assessment and evaluation. Brown and Green (2011) defined evaluation as “the process for determining the success level of an individual or product based on data” and assessment as the “procedures or techniques used to obtain [that] data” (p. 138). Assessment can be of formative or summative types. Summative assessment tends to measure learning having taken place at the end of a course, or its major units, whereas formative assessment takes place throughout the teaching and learning process (Brown & Green, 2011; Ornstein & Hunkins, 2013). Xiang and Ye (2009) discussed the issues with relying solely on summative assessment to gauge student learning and presented a framework making extensive use of formative assessment.

Both types of evaluation can be used to gauge the success of not only the learner, but of the curriculum itself (Brown & Green, 2011; Ornstein & Hunkins, 2013). Bransford, Brown, and Cocking (1999) stated this information is valuable to students and teachers alike. Whitfield (2003) wrote about embedding assessment in a CS1 class and pointed out that good practice helped to “identify both the strengths and weaknesses of the course” p. 219). The assessment devices used, therefore, provide not only valuable information about what the student has learned, but also about the learning environment itself.

Curriculum designers can choose from several options when planning an introductory course in CS. Barker et al., (2009) looked at factors contributing to CS majors’ program persistence and indicated that “teaching concepts in appealing contexts and relating material to students’ prior knowledge and interests is positively associated with retention” (p. 155). Urness and Manley (2011) suggested that “to support student interest, it is critical that the assignments



are relevant, manageable, not trivial, and highlight the concepts stressed in the classroom” (p. 271). The optimal choices depend on the target audience, which in this case also included non-majors. It is also helpful to consider best practices.

Determining a straw model for assessment types to provide for expert groups to consider can be challenging because of these varied considerations. The research contained articles in which educators teaching computing courses shared their course designs and explained assessments (Cheng et al, 2010; Cortina, 2007; desJardins & Littman, 2010; Muñoz et al., 2013; Wang et al., 2011; Whitfield, 2003; Zhao et al., 2015). Many researchers mentioned assessments they utilized in the classroom in their research and sometimes used these as evidence of student learning to demonstrate results (Bishop-Clark et al., 2007; McDowell et al., 2006; Norman & Adams, 2015; Rubio et al., 2015). The authors of the literature reviewed for this study reported on some of the assessments used in introductory CS courses.

Eleven distinct types of assessment devices were identified from the literature. First, there were several mentions of laboratory exercises or smaller programming activities (Bishop-Clark et al., 2007; Cheng, et al., 2010; Cohoon et al., 2013; Cortina, 2007; Horton, et al., 2014; Malan & Leitner, 2007; Moura & van Hattum-Janssen, 2011; Muñoz et al., 2013; Norman & Adams, 2015; Wang et al., 2011; Zhao et al., 2015). Sometimes these assignments had specific requirements or components (e.g. Malan & Leitner, 2007) or could be open-ended (e.g. Moura & van Hattum-Janssen, 2011). These types of activities give students the opportunity to investigate concepts by actively engaging in them.

Students are sometimes asked to write essays or papers to demonstrate or develop their knowledge on a topic. Essays of various types were mentioned in the literature as assessments used in introductory CS courses (Bishop-Clark et al., 2007; Cortina, 2007; desJardins & Littman,

2010; Moura & van Hattum-Janssen, 2011). Moura and van Hattum-Janssen (2011) mentioned a midterm essay assignment consisting of “software analysis, design, and development activities” (p. 476). DesJardins and Littman (2010) stated their students, who included non-majors, were required to write a research paper in which they investigated computing applications in an area of interest. Cortina (2007) described a term paper in which students pursuing various majors addressed the interplay between computers and society in an introductory course. Essays can thus focus on a number of different topics and provide students the opportunity to expand their knowledge on the various areas of CS.

Class discussions can also be used as assessment devices in introductory CS courses (Barker et al., 2009; Harding & Engelbrecht, 2015; Lan, Tsai, Yang, and Hung, 2012; Muñoz et al., 2013; Riabov, 2013). The discussions mentioned in the literature were typically those used in the online modality. Riabov (2013) examined the benefits of a project-based approach to CS graduate student motivation and learning in the online modality. Lan et al. (2012) found students in an introduction to CS course in Taiwan were more motivated to participate and performed better in online threaded discussions if mobile technology support was available. This type of added technology is sometimes included so students have more accessibility to these discussions.

Threaded discussions allow students to exchange ideas and learn from one another and the course instructor may assume a facilitator role. A benefit of participation in these forums is that students get exposure to perspectives and knowledge of their peers. Harding and Engelbrecht (2015) studied personal learning network clusters and found students to appreciate insight into the perceptions of others. Barker et al. (2009) stated that “the strong relationship between collaborative environments and classroom climate suggests that faculty engineer student-student interaction by setting clear expectations for student peer involvement in their classrooms and

labs through shared assignments, group problem solving, group discussions, and other methods” (p. 156). This call for the increased collaboration provided by both lab exercises and threaded discussions, therefore, warranted their inclusion.

Code reviews are assessment devices that originated in industry. Participants review a program together and identify strengths and weaknesses and look to improve a software product when possible. In an academic setting, these are sometimes not graded but the activities share similarity with threaded discussions because students exchange ideas and perspectives.

Hauswirth and Adamoli (2013) wrote that students’ curiosity about the performance of their peers helped to motivate learning. Cohoon et al. (2013) had students examine code artifacts, which involves similar activities to those encountered in a code review. Students “discuss, inspect, and modify programming artifacts” (Cohoon et al., 2013, p. 53). Law et al. (2010) stated that certain assessment experiences, such as code reviews, give students an opportunity to compete with one another and evaluate best practices in programming. These reviews give students important practice in reading the code of others (Malan & Leitner, 2007). These activities would, therefore, be especially beneficial for CS majors intending to work as programmers.

Quizzes and tests were often mentioned as assessments in introductory CS courses (Cheng et al., 2010; Fulton & Schweitzer, 2011; McDowell et al., 2006; Norman & Adams, 2015). These tests can be of varied designs, including short answer, true/false, and multiple choice questions (Norman & Adams, 2015). Moura and van Hattum-Janssen (2011) recommended the use of short weekly quizzes to “assess ... understanding of CS fundamentals and ... ability to solve simple programming exercises that require these fundamentals” (p. 482). Bälter, Enström, and Klingenberg (2013) conducted a study involving two introductory

programming classes and found quizzes after exposure to content helped students improve study habits. Horton et al. (2014) used quizzes in an inverted learning approach but graded students only on their participation; they found students performed significantly better on a final exam than students learning via a traditional approach. Quizzes and tests, therefore, have been shown to have important benefits for students and are often used in introductory CS courses.

According to the literature, another popular assessment in these classes is the use of concept questions (Cortina, 2007; Fulton & Schweitzer, 2011; Horton et al., 2014; Muñoz et al., 2013; Whitfield, 2003). Though there was not much elaboration on these assessments it can be assumed these involve having students answer questions on content they are studying. Muñoz et al. (2013) reported that students found concept questions to be “thought-provoking and helpful for critical thinking development” (p. 31). There were also two additional assignments mentioned. Muñoz et al. (2013) also referred to the use of case studies and student interviews with professionals as assessments in introductory CS courses. Though these assessments were only mentioned in one source, they merit consideration.

The assignments mentioned thus far have mostly been of the formative type. Researchers referenced assessment devices in the literature that were mainly summative in nature, including final exams and term programming projects. Final exams are typical in many undergraduate courses and introductory CS classes are no exception. Some authors made simple mention of final exams in their writing (Cortina, 2007; Dodds et al., 2008) whereas others used final exams as a dependent variable in some type of experimental research because they were good indicators of student learning of course content (Fulton & Schweitzer, 2011; Horton et al., 2014; Norman & Adams, 2015).

Exams and tests can vary widely with regard to substance. Often final examinations in CS courses will include a programming activity (Horton et al., 2014; McDowell et al., 2006; Moura & van Hattum-Janssen, 2011; Wang et al., 2011). Exams have been called important components in computing courses that can supplement assessment information obtained from programming assignments (Whitfield, 2003). In discussing tests in CS courses, Scott (2003) suggested that questions should assess learning at all six levels of Bloom's taxonomy to properly evaluate the knowledge of the student and the curriculum environment itself. Starr et al. (2008) reiterated the importance of utilizing Bloom's taxonomy when aligning assessment with expected student outcomes.

Even though exams have been associated with student anxiety (Gerwing, Rash, Gerwing, Bramble, & Landine, 2015), they have not been found to be a leading cause for angst in introductory programming classes. Hawi (2010) looked at the major causes of anxiety as reported by undergraduate business computing students and though it was one of the causes identified for success or failure in an introductory level programming course, it was not one of the most often mentioned.

The other major summative assessment often mentioned in the literature was a programming project (Dodds et al., 2008; Fulton & Schweitzer, 2011; Horton et al., 2014; Muñoz et al., 2013; Zhao et al., 2015). These assignments were usually on a larger or longer term scale and authors described courses in which students completed two (e.g. Muñoz et al., 2013; Zhao et al., 2015) or three (Fulton & Schweitzer, 2011; Horton et al., 2014). Sultana (2015) interviewed four hiring managers in the software industry in the geographic region of this study and found project experience to be a commonly reported attribute desired of potential employees.

The inclusion of projects in CS curriculum has some history. Gupta (2007) noted that the ACM's 1968 curriculum guidelines recommended "true-to-life programming projects" (p. 58). Lutz et al. (2014) reported on their SE program at the Rochester Institute of Technology and stated that "all of the software engineering courses incorporate team projects as significant graded components" (p. 54). Muñoz et al. (2013) reported that "students appreciate working on medium- to large- size projects that are challenging and well-structured" (p. 31). Moura and van Hattum-Janssen (2011) had students lead a presentation of a term project, which they found helped to promote individual and learning accountability. Projects, therefore, are an important assessment device in introductory CS courses.

The straw model of assessments for academic and industry experts to consider for an introduction to CS course consist of these eleven items in alphabetical order:

- case studies,
- code reviews,
- concept questions,
- essays,
- final exams,
- online threaded discussions,
- interviews with professionals,
- lab exercises
- quizzes,
- smaller programming activities, and
- term projects.

The experts would again be provided the occasion to add to the list if they determined an item had been left out.

### **Summary**

This chapter presented a synopsis of CS education by outlining the academic preparation of professionals in the field, examining stakeholders, and researching introductory courses. Many undergraduates study CS in pursuit of careers as software developers, systems analysts, programmers, systems managers, web developers, or database administrators. Others simply take an introductory course to develop skills that will be beneficial for other types of occupations. Though undergraduate study of CS was predated by graduate programs, it has become highly defined thanks to curriculum recommendations from professional societies and sharing of best practices by institutions in academia and industry over the past seven decades.

Formal education has become the norm for most occupations related to computing. Most positions require a bachelor's degree in CS, SE, information systems, computer engineering, or IT; depending on a student's desired area of specialization. Certification and licensure have started to become a standard for some, depending on geographic location and specialization.

The major stakeholders of CS education, like those in any other field of study, have distinct interests and much interaction. Members of the computing industry need employees who have the proficiencies required to develop products effectively and efficiently. Industry contributes much to CS education by helping to identify and define these skills, though the influence is not necessarily a dominant one. Academic institutions provide the future workforce for industry by preparing students with the aptitudes needed. Scholastic preparation of CS students takes place in various types of institutions, ranging from community colleges to research universities. Liberal arts institutions have increasingly offered programs in CS as its study has

come to be recognized as presenting highly translatable skills. Representatives from these diverse academic institutions sometimes collaborate in the development or offerings of programs in the hopes of maximizing their quality.

Introduction to CS courses have become highly popular and attract majors and non-majors alike. Colleges and universities have responded in differing ways, including offering distinct courses to account for those with different levels of experience or future career goals. Liberal arts institutions, and others that might have more limited resources, may typically offer a single introductory course for majors and non-majors. The identification of student competencies, programming languages, and assessments to consider for such a context becomes increasingly difficult.

The curriculum recommendations from organizations such as the JTFCC and LACS offer valuable input as do scholars who research CS education. These sources were used to create the straw models for the competencies, programming languages, and assessments to consider for an introductory CS course. The models identified from these sources were then provided to experts from academia and industry to consider their relative importance and applicability.

The competencies of an introductory CS course aim to identify the topics of importance for students as they are provided a breadth-first overview of the field. Additionally, these competencies generally state the level of mastery expected. The proficiencies identified here deal with topics in programming and related areas, hardware and lower level architecture, and professional skills.

Various programming languages were identified that warranted consideration for an introduction to CS course. These varied from visual to text-based in nature and were reported to have significantly divergent use in industry and academia. There has been much debate about



how to teach programming at the introductory level and the 23 languages identified in the straw model have distinct characteristics for academic and industry professionals to consider.

Assessments can often define an academic experience for students. At a cursory level the assessments to consider for an introduction to computing course might appear to be of interest to academic experts only. However, industry professionals are concerned with the skill base of potential employees and would likely have valuable input. These straw models would be provided to experts from academia and industry so they could help identify the competencies, programming languages, and assessments most important and applicable to the goals of a survey course in CS.

The next chapter will convey the research methods utilized in this study. Special attention is paid to the Delphi methodology, which would play the primary role in the determination of inputs to consider for the introductory course's design. A description of the populations involved is presented, along with an explanation of the methods planned to collect and analyze data.

## **CHAPTER III**

### **METHODOLOGY**

The overall goal of this research was to identify the competencies, programming languages, and assessments recommended by experts for an introductory CS class at a private nonprofit liberal arts university in Fresno County, California. The literature and input from experts in industry and higher education were used toward these ends. This chapter specifies the methodology implemented in this study. Detail is provided about the participants and their selection and the design and procedures used to gather and analyze data.

#### **Participants**

Zhao (n.d.) wrote about curriculum in U.S. schools stating that content conveyed should be representative of the society in which it is taught. This suggestion has merit for development at the postsecondary level. It is, therefore, prudent to learn about the target content for a curriculum by those deemed experts in a given area.

This study, therefore, utilized a Delphi approach. One of the limitations of this methodology is the lack of representative sampling methods in the recruitment of participants (Beech, 1999). The goal, however, is not to form a group who is necessarily a cross-section of a population (Okoli & Pawlowski, 2004). What is important, however, is that participants are “experts or at least informed advocates” (Goodman, 1987, p. 730) who can be impartial, provide current input, and are interested in the research (Hasson, Keeney, & McKenna, 2000).

A major area of concern regarding Delphi studies is the number of participants to include. Wilhelm (2001) stated that statistics do not play a role in determining a sample size. Participant pool sizes vary widely with this type of research. Skulmoski, Hartman, and Krahn (2007) identified several Delphi studies in the information sciences and IT fields and found the number

of participants ranged from nine to 126. They also looked at Delphi studies outside of the information sciences and IT and found numbers as few as three and as high as 171.

Two main groups constituted the participants for this part of the study: industry and academic professionals. The goal was to identify regional experts' recommendations for an introductory CS course as these individuals would be best able to identify the most important concepts and practices in the geographical area. The target members for industry experts were, therefore, experienced computing professionals in Fresno County, California. According to the State of California Employment Development Department (2015), there were 920 total persons employed as computer programmers, database administrators, applications and systems software developers, and web developers in Fresno County in 2012. Since the opinion of experts in these positions was sought, a minimum of five years' experience was required for potential participants (Guu et al., 2014; Joyner & Smith, 2015).

The experience level of the individuals employed in these positions in Fresno County was unknown but a modest participant pool was expected. This situation was not without precedent. Brungs and Jamieson (2010) looked at legal issues in computer forensics in Australia and used a heterogeneous sample of 11 persons representing four different stakeholder groups from a limited population of approximately 30 experts. Surakka (2007) sought out 10 to 20 persons when trying to identify the most important topics for software development students in Finland to study. Due to the similarity of this study, and the limited population from which to draw, 10 to 20 experienced professionals were targeted. Hasson et al. (2000) recommended involving participants who were strongly interested in the topic of study. Therefore, industry members who were actively or previously involved with local higher education through program advisory committees or similar activities were recruited.

The second group involved in this study were instructors in CS or related fields in higher education. Four community colleges and a state university offer courses in CS in the county and immediate surrounding area. The department web sites for these institutions were used to determine a total pool of 39 professors of CS or related fields. Since it was desired that the academic and industry groups were comparable in size, 10 to 20 participants from higher education were targeted. Educators who held at least a Master's Degree in their field (Surakka, 2007) at these institutions were approached about their interest in participating in this research.

### **Design**

This research was descriptive in nature as the goal was to survey a situation rather than identify causes or institute change (Leedy & Ormrod, 2014). Focus groups and interviews were considered for this study as both provided opportunities to solicit feedback from experts and allowed for direct and immediate contact between interviewer and participants (Hays & Singh, 2012). The focus group, however, can suffer from issues of participant conformity and input from a few dominant voices, whereas interviews require a skilled and experienced questioner and can seem more intrusive to participants (Hays & Singh, 2012). The use of both a focus group and separate interviews was considered to help offset some of these deficiencies but it was deemed the commitment might be seen as too intensive on the part of potential participants. Because there was concern about the ability to recruit a sufficient number of experts willing to commit time to the study, the Delphi methodology was chosen.

The RAND Corporation first developed the Delphi technique and the use of questionnaires and feedback to determine expert consensus for the U.S. Air Force (Linstone & Turoff, 2011). Linstone and Turoff (1975) described the Delphi approach “as a method for structuring a group communication process so that the process is effective in allowing a group of

individuals, as a whole, to deal with a complex problem” (p. 3). Dalkey and Helmer (1963) described the first implementation by RAND and indicated that the group aimed to achieve consensus “by a series of intensive questionnaires interspersed with controlled opinion feedback” (p. 458). An important distinction of the approach is the anonymity of the participants (Linstone & Turoff, 1975; Strauss & Zeigler, 1975; Wilhelm, 2001). This factor can help ensure that participants provide their true and unadulterated opinions in an environment that can reduce apprehensions. The Delphi methodology was thus viewed as the best approach to solicit feedback from the experts identified for this research.

Among the many popular uses for the Delphi method is consensus on frameworks for a field of study (Bacon & Fitzgerald, 2001) and objectives for curriculum (Brungs & Jamieson, 2010; Elledge & McAleer, 2015; Mamelok, 2013; Surakka, 2007). Several approaches and variations have been utilized and the specific design was modified to fit the context of this study per methodologies found in the research literature. The design is shown in Figure 1.

### **Recruit and Identify Participants**

Potential participants were located using suggestions from professionals in higher education, graduates of academic programs, and research of organizations’ web sites. All participants were invited to take part in the research by email. A copy of the message is included in Appendix A. Phone calls were placed one week after the emails were sent to those who had not yet responded. Snowball sampling was utilized as individuals who agreed to participate were asked to suggest other candidates for the study (Hays & Singh, 2012). The participants expressing interest were questioned about their backgrounds in the fields of computing and software development to verify they met the criterion of a minimum of five years’ experience.

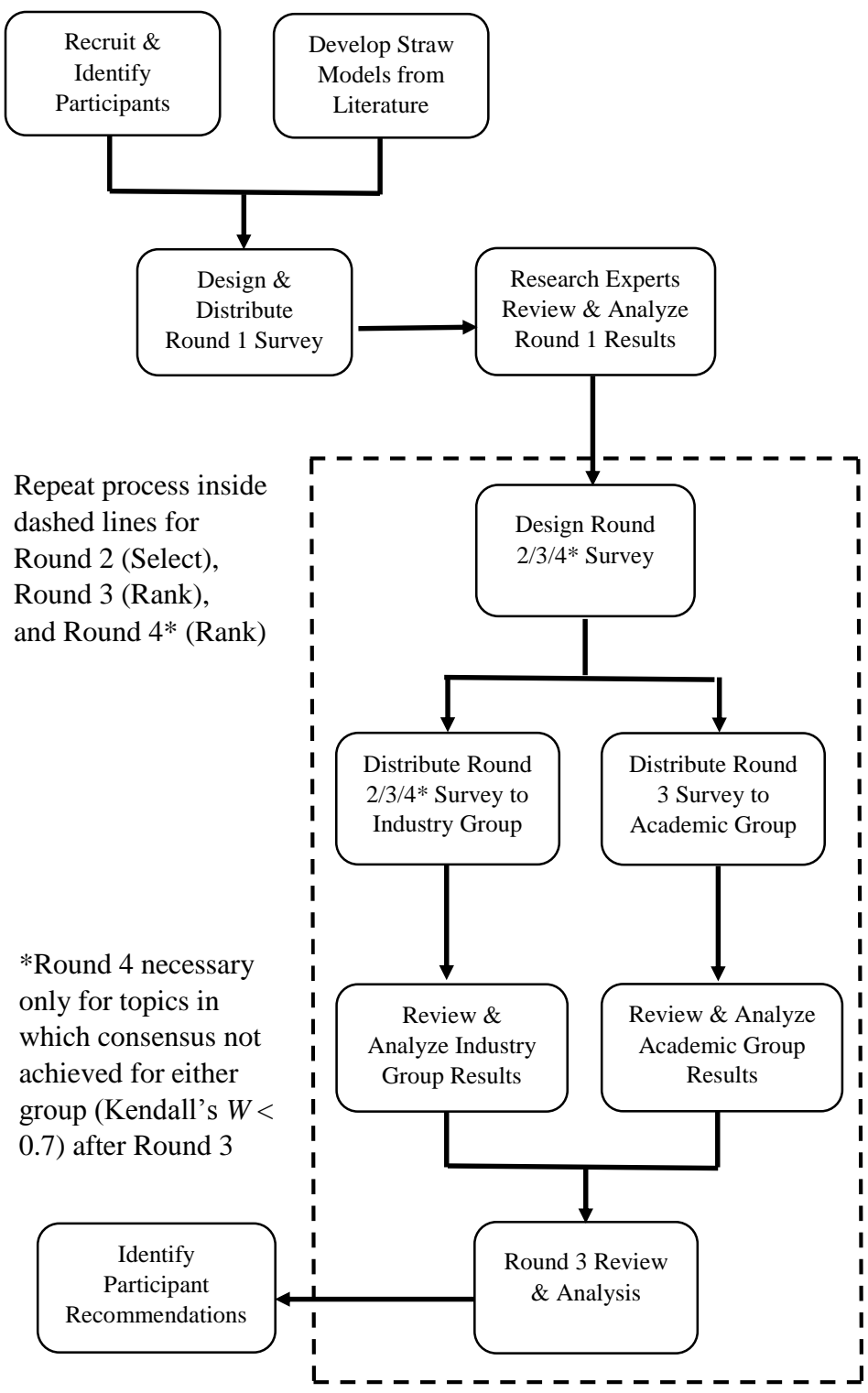


Figure 1. Delphi design methodology.

The plan was to continue this process for each of the groups until ten to twenty individuals were located or no additional candidates could be identified.

One research SME was also recruited for this study. This individual would be responsible for assisting the researcher in reviewing participants' open responses from the first round to validate their identification. This individual was required to have a PhD in any field and have experience teaching in any IT-related field.

A preliminary phone call was placed to each participant and they were provided a brief overview of the study and the research goals. Each member was informed the surveys would be administered electronically and asked to complete a human subjects consent form. This document, which is included in Appendix B, was sent electronically, along with a summary of the study, shown in Appendix C. The summary was provided so that each participant would be fully aware of their role. Subjects were asked to read and sign the consent form and return it electronically to the researcher's email address.

### **Develop Straw Models**

The review of literature was used to develop straw models, which then served to give the participants starting reference points for the most relevant topics (Rotondi & Gustafson, 1996) to be considered as competencies, programming languages, and assessments for an introductory CS course. The exclusion of input from the experts into the initial models was deemed acceptable to ensure making the most efficient use of all participants' time (Brace-Govan, Farrelly, Joy, Luxton, & Davey, 2001; Eskandari, Sala-Diakanda, & Furterer, 2007). The experts would be allowed to add their own suggestions in the first round of the survey to ensure that all important possibilities were included in the analysis.

### **Design and Distribute Round 1 Survey to Both Groups**

The first survey included four sections and is presented in Appendix D. The initial set of questions asked participants to provide their demographic information including gender, age, current employment, years of experience, highest education earned in CS or a related field, and the number of programming languages in which the individual was fluent. This information was collected to describe the background and expertise of the group. The second set of questions asked the participants to rate the applicability of the competencies from the straw model on a five-point Likert scale. Each item was to be categorized as very important, important, moderately important, of little importance, or unimportant (Siegle, 2010). The subsequent sections provided a list of programming languages and assessments to be used in an introductory CS course and the same rating categories were provided. Blank entries were also available for optional contributions by the individuals for each of the three categories. Instructions were included to write any additional topics and weight their importance. A field was also available for an explanation of the optional entry so participants could clarify their suggestion to the rest of the group.

The Round 1 survey was constructed in SurveyMonkey. An email was sent to each participant inviting them to the first round of the study along with a link to the survey. Responses were requested within one week of the email being sent. Because the survey was confidential, the identity of those completing it was not known. Individuals who had not sent emails to indicate they had completed the survey by the deadline were sent email reminders on the fourth and eighth days. Any individuals who indicated they could no longer participate were removed from the study but those who did not respond were retained in the distribution for future rounds.



### **Round 1 Review and Analysis**

The results of the surveys were downloaded into Microsoft Excel. The mean age, years of experience, and number of programming languages in which the participants were fluent were computed using the “MEAN” function. Minimum, maximum, and standard deviation values for each category were determined using the “MIN,” “MAX,” and “STDEV.P” functions, respectively. Counts for individuals by gender, employment, and highest education were computed using the “COUNT” function.

Responses to each of the three content categories were copied into Excel and quantified as follows: very important = 5, important = 4, moderately important = 3, of little importance = 2, unimportant = 1. Newly suggested items by participants were checked for individuality and inserted into the lists ranked according to the weight given. The processing for newly suggested items was reviewed with the research SME and changes to the surveys for the next round were made as necessary.

Any item selected by at least two participants was added to the list of competencies, programming languages, or assessments to be used in subsequent rounds. Okoli and Pawlowski (2004) asked participants in a Delphi study for open suggestions and included items appearing on at least 50% of their surveys. This research utilized a strategy that added to the questionnaire any item suggested more than once, thus indicating recommendation by at least 10% of the participants. This approach would be more inclusive to input from the experts and address the lack of an open-ended brainstorming approach in the first round.

### **Design and Distribute Round 2 Selection Survey**

The ranked lists of items and their median weight scores were added to the survey for the second round. Reference the questionnaire in Appendix E. The median was computed instead of

the mean as these data were Likert-type in nature (Boone Jr. & Boone, 2012). The inclusion of this value in the questionnaires would communicate the perceived importance attributed to each item. The instructions for this questionnaire were different than those used in the first round. First, the questions on demographics were omitted as this information had already been collected. Second, there were no blank fields for optional entries. Finally, participants were instructed to determine whether or not each of the items should be included for the introductory CS course by choosing to select at least ten topics for each of the three categories (Okoli & Pawlowski, 2004). The items were imported into SurveyMonkey as two equivalent questionnaires for the academic and industry groups.

At this stage the study took on a panel structure (Okoli & Pawlowski, 2004). The industry and academic groups were given separate links so analysis of their feedback could be done independently. This design would allow each group to come to a consensus more quickly and would allow recommendations from each group to be distinguished for final decision making by the curriculum designer.

An email with instructions and the appropriate link were sent to all participants, including those who may have not submitted a survey in Round 1 but had initially expressed interest in the research. Participants were asked to respond within one week. Follow-up emails were sent on the fourth and eighth days.

### **Round 2 Review and Analysis**

Feedback was collected from each participant on their selected items from each of the three categories. Those items that were selected by at least half the participants were chosen to be included for Round 3 (Okoli & Pawolowski, 2004). Those not selected by half the experts were omitted from further consideration by that group, but not necessarily the other panel. The

findings from this point would be independent for each group. In this manner the subsequent surveys would be more focused on the suggestions for each individual panel.

### **Design, Distribute, and Analyze Round 3 and 4 Surveys**

The steps in Round 3, and Round 4 if necessary, were identical. The lists of items as selected by the experts from the previous round were added to the survey. The questionnaires for the industry and academic groups are included in Appendices F, G, H, and I. Participants were asked to rank each item in each of the three categories of competencies, programming languages, and assessments in terms of their use in an introductory CS course. The lists were again imported into SurveyMonkey as two questionnaires in keeping with separate panels. The email message with instructions and the appropriate link were sent to the participants, who were once again asked to respond within one week. Follow-up emails were again used as necessary.

The coefficient of concordance, Kendall's  $W$ , was used to determine the level of agreement among the participants' ranked lists for each panel. Kendall's  $W$  ranges from zero to one to indicate a scale of increasing unanimity between rankings (Field, 2009). Israel (2008) identified the requirements to use Kendall's  $W$  to check agreement among ranked lists as working with ordinal data and a sample size, between 3 and 300, that is equal between the groups to be compared. Schmidt (1997) indicated a value of at least 0.7 indicates strong agreement. This threshold was used to determine whether or not any of the lists of competencies, programming languages, or assessments needed to be submitted in a fourth round to either of the panels. The  $W$  would, therefore be computed six times after Round 3 as shown in Table 10. Each  $W$  value would be analyzed independently and only those topics that failed to meet the 0.7 threshold value were included in a Round 4 survey for each individual panel.

Table 10

*Kendall's W Values Analyzed*

Expert Group	Levels of Agreement		
	Competencies	Programming Languages	Assessments
Academic	$W_{AC}$	$W_{AL}$	$W_{AA}$
Industry	$W_{IC}$	$W_{IL}$	$W_{IA}$

It was decided that a maximum of four rounds would be considered for the Delphi portion of this research. It has been found that major fluctuations are typically not expected after a fourth round (Scheibe, Skutsch, & Schofer, 1975; Wilhelm, 2001) and participant fatigue can become a concern (Schmidt, 1997; Sitlington, 2015).

The output from the final round consisted of ranked lists of the most important competencies, programming languages, and assessments to be utilized in an introductory course. Two lists were available as the industry and academia experts would likely have different preferences. These data would then be used in the curriculum development of the introductory class to the extent desired by the course designer.

### **Summary**

This chapter outlined the methods and procedures used in this research. The study involved a Delphi technique to determine the recommendations of two local groups of experts on the competencies, programming languages, and assessments to use for an introductory CS course.

The participants for the Delphi portion of the research represented professionals from the computing industry and academia in the area surrounding the university in which the course

would be taught. Twenty to forty participants were targeted with half coming from each of these two expert groups.

The Delphi approach used was a modified adaptation, which began with straw models developed from the literature. Round 1 was used to rank the initial items and request suggestions from the experts. The groups were then separated so that each could function independently and provide their specific recommendations. The experts were asked to select the items that were deemed most important for each of the three categories of competencies, programming languages, and assessments in Round 2. Round 3 was used to determine preferences and consensus from the participants on each of the three topics; again with each group acting individually. Kendall's  $W$  was used to gauge agreement and any topics that did not have consensus between either of the groups were carried over to a fourth and final round.

Chapter IV will present the findings from this study. The recommendations for the competencies, programming languages, and assessments to be used for the introductory CS course will be presented from each of the expert groups. These items will then be available as recommendations for the development of the course's curriculum.

## CHAPTER IV

### RESULTS

This chapter presents the results of this research. The four rounds of the online Delphi study were conducted with academic and industry experts in CS during the months of April and May 2016. This chapter reviews the steps undertaken throughout the four rounds and the feedback provided by the experts.

#### **Participants and Demographics**

Professional acquaintances of the researcher and web searches of regional industry organizations and academic institutions were used to identify participants. Potential contributors to this study were invited by email and asked to nominate other experts who met the criterion of five years' experience in industry or teaching CS or related disciplines with at least a master's degree. Phone calls were also placed to individuals who did not respond by email. The researcher directly invited 85 experts from California's Central Valley; 48 individuals (56%) were from higher education and 37 (44%) were from industry. Invitations were sent via email between March 14 and April 8, 2016. These persons were also free to forward the invitation to others so it is unknown exactly how many total persons were contacted. A total of 23 individuals (27% of those directly invited) agreed to participate in the study. There were 11 persons (48%) in the industry group and 12 persons (52%) in the academic group.

The link to the survey for Round 1 was sent by email on April 11, 2016. Participants were given eight days to complete the survey. Email reminders were sent out on the fourth day and again on the final day. There were 22 experts who participated in Round 1 and these were evenly distributed between the industry and academic groups. There were 20 males and 2 females (one from academia and one from industry). The participants were asked to identify their

ethnicity and the group consisted of 15 Caucasian, 4 Asian, 2 Hispanic, and 1 African American. Their average age was 45.9 years, ranging from a minimum of 31 years to a maximum of 59 years. They had an average of 18.1 years of experience with values spanning five to 36 years. Their formal educational attainment consisted of four doctoral degrees, 10 master's degrees, five bachelor's degrees, and one associate's degree. Two individuals reported having some postsecondary experience. These degrees came from the disciplines of CS (eight individuals, including two persons with degrees in CS and engineering), information systems (five individuals), engineering (one electrical and one chemical), digital animation (one individual), management (one individual), psychology (one individual), computational linguistics (one individual), and mathematics (one individual).

The experts identified themselves as being fluent in an average of 4.4 programming languages; ranging from zero to eight languages. The industry experts worked in areas of focus including consulting and software development, sports, statistical data processing, government, healthcare, education, web development, information systems, and higher education. Faculty taught courses at the undergraduate and graduate levels in introductory and intermediate programming concepts and methodology, discrete mathematics, computer organization/architecture, operating systems, mathematical programming, programming for scientists and engineers, SE, applications programming, client and server side scripting, computer concepts and literacy, network systems management, databases, enterprise resource planning, management information systems, and web development. Finally, six participants held certifications with titles including Microsoft Certified Application Developer, Microsoft Certified IT Professional, Certified Scrum Master, Certified Scrum Product Owner, Cisco Certified Academy Instructor, Java 2 Certified Programmer, and Java 2 Certified Developer; and

in areas such as Microsoft Certified Systems Engineering, COMPTIA A+, COMPTIA Strata IT, Bureau for Postsecondary and Vocational Education, and Oracle Implementation.

Two participants in the academic group did not hold master's degrees in a computer-related field. These individuals had degrees in chemical engineering, with an emphasis on computer applications, and in mathematics, with an emphasis on CS. They were, however, included in the study as each had extensive experience teaching introductory CS courses for 10 and 26 years, respectively.

## **Round 1**

### **Course Competencies**

The goal of the first round was to solicit opinions from the experts on the importance of the course competencies, programming languages, and assessments for an introductory undergraduate CS course. The survey included the straw models developed from the literature and provided participants the opportunity to suggest any items they may have felt were missing.

Eleven academic (92%) and eleven industry (100%) experts completed the Round 1 survey. Participants were first prompted to answer demographic questions and then asked to rate the potential competencies for an introductory CS course on a five-point Likert scale. Responses were weighted as Very Important (5), Important (4), Moderately Important (3), Of Little Importance (2), and Not Important (1). See Table 11 for an overview of the responses submitted by the 21 experts participating in Round 1. It was noteworthy that four of the competencies listed received median scores of five (Very Important); these included "writing functional procedural programs employing programming fundamentals," "demonstrating teamwork and interpersonal group skills," "demonstrating problem solving," and "demonstrating critical thinking and reasoning." Also of note was the competency of "exhibiting entrepreneurship in computing,"



Table 11

*Round 1 Expert Feedback on Competencies for Introductory Computer Science*

Competency	Median Rating	Minimum Rating	Maximum Rating
Analyze algorithms for effectiveness and efficiency	4	1	5
Illustrate concepts in artificial intelligence	3	1	4
Summarize basic computability, theory of computation, and its limits	3	1	5
Describe different types of data representation (e.g. graphics, binary numbers, etc.)	3	2	5
Illustrate the use of Boolean logic and basic combinational digital circuits	3	2	5
Describe basic computer architecture and organization	4	2	5
Summarize the history of computing and its ramifications to implementation today	3	1	5
Explain the factors contributing to human-computer interaction in computing	4	1	5
Illustrate the use of databases and apply SQL	4	1	5
Explain the operation of compilers	3	1	5
Discuss the operation of networks and related practices (e.g. data compression, etc.)	3	2	4
Explain the functionality of operating systems and provide examples	3	2	5
Describe common programming languages and their popular uses	4	2	5
Describe benefit and operation of parallel and distributed systems and programming	3	1	5
Demonstrate use of recursion in a program	4	1	5
Describe the need for computer and data security and identify best practices	4	2	5
Explain the role of modeling and simulation in computing	3	1	5
Describe societal impact of computing	3	1	5
Describe the World Wide Web and select internet protocols	3	1	5
Describe process and practices in SE	4	1	5
Plan a career in CS	3	1	5
Write functional object-oriented programs employing programming fundamentals	4	1	5
Write functional procedural programs employing programming fundamentals	5	1	5
Implement good documentation practices in programming	4	1	5

Table 11 Continued

Competency	Median Rating	Minimum Rating	Maximum Rating
Demonstrate teamwork and interpersonal group skills	5	3	5
Demonstrate algorithmic thinking.	4	1	5
Demonstrate computational thinking	4	1	5
Demonstrate problem solving	5	3	5
Demonstrate critical thinking and reasoning	5	3	5
Demonstrate systems thinking	3	1	5
Demonstrate creativity in programming	3	1	5
Demonstrate time and resource management skills in a project	3	1	5
Exhibit entrepreneurship in computing	2	1	5
Communicate effectively orally and in writing	4	2	5
Describes self-learning and assesses self	4	1	5
Exhibit digital literacy	4	2	5
Explain and choose from different file structures	3	2	5
Explain and utilize effective procedures in software verification and validation	4	1	5

*Note.* N = 22. Median calculated for a five-point Likert scale (5 – Very Important, 4 – Important, 3 – Moderately Important, 2 – Of Little Importance, 1 – Not Important).

which received the lowest median rating of two (Of Little Importance). Additionally, two competencies failed to receive a maximum score of five by any of the participants; these were “illustrating concepts in artificial intelligence” and “discussing the operation of networks and related practices.” Only three items received minimum rating values no lower than three. These were the competencies involving teamwork, problem solving, and critical thinking.

The participants were provided the opportunity to append to the list additional competencies that they felt were important for an introductory CS course. These open-ended responses were reviewed with a third-party research expert who held a Ph.D. in an engineering field and had experience teaching introductory programming. The research expert reviewed the open-ended answers independently then met with the researcher to make suggestions on changes to the original straw models. Consensus was reached on all necessary changes.

Seventeen open-ended responses were provided to this optional question. These responses varied widely on content including:

- Effective workflow and process analysis - very important. Need to be able to understand and analyze workflows/processes BEFORE designing an application or programming a solution.
- Working on a Team in a Project - because that's what they will do.
- I would refine the "societal impact" category to be more specific to things like cyberbullying, privacy, intellectual property.
- Object Oriented Design - Important to tie real world objects into the thinking really early in the learning process. Makes it relevant.
- Write programs employing functional programming fundamentals. Functional programming is a paradigm that is gaining greater importance especially in

distributed and parallel programming fields. Functional programming is also a very different paradigm from procedural and object oriented code and therefore provides its users with an additional tool set for problems solving.

- Write programs using Generic Programming concepts and techniques. These concepts feature in modern revisions of most programming languages.
- Early industry exposure or experience.
- When solving a programming problem, it is important to know why to use a certain programming language.
- I believe that debugger competency would merit consideration. Too many engineers do not properly know how to effectively use one.
- Under software validation, principals like unit testing and mocking/facades as well as integration testing would be really beneficial. This is very important
- Ability to figure out difficult tasks - critical thought processes - You aren't always told what to do...
- I would refine the "data security" category to be more like "designing, implementing, and verifying" hacker-resistant safe code.
- How will the training translate to a job in the marketplace?
- Code portability and reuse. There are many design and distribution strategies to be able to use code written once in multiple use cases (SOLID, packages like NPM or Nuget).
- Ability to be critical effectively - you have to understand how to criticize appropriately for effectiveness and to be heard.
- DevOps, large category but inclusive of Continuous Integration and Deployment.

- Web development, because the world is surrounded by it.

There were no competencies suggested by at least two individuals. Therefore, no new competencies were added for Round 2. However, two suggestions to clarify existing competencies were deemed beneficial by the researcher and third party expert and were used to modify the entries for Round 2. The competency related to societal impact was modified to include examples of topics, such as cyberbullying, intellectual property, and privacy. Likewise, the item on data security was modified to include the actions of designing, implementing, and verifying hacker-resistant safe code.

### **Programming Languages**

The next section of the survey asked participants to rate 23 programming languages in terms of their importance for an introductory CS course. The rating scale was similar to the one used for course competencies with the inclusion of an option titled “Unfamiliar,” which was weighted as zero points. Table 12 presents the results of this portion of the survey.

It was noteworthy that only 5 of the 23 languages were known to all the participants, including assembly language, C, C++, Java, and Visual Basic. Six languages achieved median scores of zero, indicating unfamiliarity by more than half the group. These languages included Alice, Greenfoot, Haskell, R, Scheme, and Scratch. Five languages were rated as being “Very Important” according to their median rankings (C#, C++, Java, JavaScript, and Python).

The experts provided six open-ended responses to the optional questions about additional programming languages not listed. The recommendations included Elm (a functional language based on Haskell), Clojure (a LISP dialect), Extensible Application Markup Language (XAML), CSS (a stylesheet language), and HTML5 (Hypertext Markup Language). The last three recommendations were for markup or stylesheet languages, which are not typically identified as programming languages (van der Spuy, 2012). However, it was considered prudent to add

HTML5 to the Round 2 survey as it was recommended by two experts. Though it is not a true programming language, concepts in CS could be taught using HTML5.

Table 12

*Round 1 Expert Feedback on Programming Languages for Introductory Computer Science*

Programming Language	Median Rating	Minimum Rating	Maximum Rating
Alice	0	0	4
Assembly Language	2	1	4
C	3	1	5
C#	4	0	5
C++	4	1	5
Greenfoot	0	0	2
Haskell	0	0	4
Java	4	1	5
JavaScript	4	0	5
MATLAB	1	0	5
Objective-C	2	0	4
Perl	1	0	3
PHP	2	0	4
PL/SQL	3	0	5
Python	4	0	5
R	0	0	4
Ruby	1	0	4
Scala	1	0	5
Scheme	0	0	3
Scratch	0	0	5
Shell	2	0	5
Swift	1	0	4
Visual Basic	1	1	5

*Note.* N = 22. Median calculated for a five-point Likert scale (5 – Very Important, 4 – Important, 3 – Moderately Important, 2 – Of Little Importance, 1 – Not Important, 0 – Unfamiliar).

## Assessments

The final section of the Round 1 survey asked participants to rate 11 potential assessments for an introductory CS course. The rating scores available were identical to those used with the course competencies, ranging from “Most Important (5)” to “Not Important (1).” The results are presented in Table 13. The assessments deemed most important according to the median rating scored by the participants were lab exercises, smaller programming activities, and term projects. Only one assessment device, essays, was rated below “Moderately Important” in terms of its median score.

Table 13

### *Round 1 Expert Feedback on Assessments for Introductory Computer Science*

Assessments	Median Rating	Minimum Rating	Maximum Rating
Case Studies	3	2	5
Code Reviews	4	1	5
Concept Questions	4	2	5
Essays	2	1	5
Final Exams	4	1	5
Online Threaded Discussions	3	1	5
Interviews with Professionals	3	1	5
Lab Exercises	5	4	5
Quizzes	3	1	5
Smaller Programming Activities	5	1	5
Term Projects	5	1	5

*Note.* N = 22. Median calculated for a five-point Likert scale (5 – Very Important, 4 – Important, 3 – Moderately Important, 2 – Of Little Importance, 1 – Not Important).

Finally, the experts provided only four open-ended responses to the list of assessments to be considered. Team programming assignments were recommended by at least two individuals so this assessment was included for Round 2.

The lists used in the first round and the two items suggested by multiple experts were copied over into a new survey. The median ratings provided in Tables 10 to 12 were recorded into the survey for Round 2 to communicate the importance attributed to each item by the overall group.

## **Round 2**

The goal of the second round was to give experts the opportunity to narrow down the lists they would rank in Rounds 3 and 4 (Okoli & Pawlowski, 2004). Participants were instructed to select no fewer than 10 items from each of the lists of competencies, programming languages, and assessments. They were also advised to consider their opinions on each item in relation to the importance attributed by the overall group as indicated by the median score from Round 1. This instruction enabled participants to utilize deliberation as characterized by the Delphi approach without meeting with other experts in person.

The survey links for Round 2 were sent by email on April 25, 2016. Identical surveys with duplicate instructions were provided to the academic and industry groups. Participants were again given eight days to complete the survey. Email reminders were sent out on the fourth and on the eighth days. There were 21 experts who participated in the second round with eleven in the industry group (100%) and 10 in the academic group (83%). At least five selections for an item were required for it to be carried over into the final rounds of the study for the academic group. This number was six for the industry group.



## **Course Competencies**

The selection counts for competencies are shown in Table 14. The academic group selected 16 competencies compared to the industry group, which chose 12 competencies. Nine of the competencies were identical between the groups, including those concerning the topics of algorithms, computer architecture, SE, object-oriented programming, functional programming, documentation, teamwork, problem solving, and critical thinking. These shared competencies were highly focused on programming and professional skills.

The academic group also included seven competencies that were not chosen by the industry experts. These competencies involved computability, operating systems, recursion, computer and data security, modeling and simulation, algorithmic thinking, and computational thinking. The industry group selected only three items not chosen by the academic experts, including those dealing with data representation, databases and SQL, and common programming languages. No competency was selected by all the experts in either group and only “exhibiting entrepreneurship in computing” was not selected by anyone.

## **Programming Languages**

The second round instructions for programming languages were identical in that participants were instructed to select no fewer than ten. The number of programming languages was augmented with HTML5 per the results of the first round. Each group was given the opportunity to narrow their respective lists and the results are shown in Table 15.

Eight programming languages were selected by at least half of the experts in the academic group. The industry group selected 12 languages for inclusion in the latter rounds. All eight languages selected by at least half the experts in the academic group were also chosen by the industry group; these were C, C#, C++, Java, JavaScript, PHP, Python, and Ruby. The

Table 14

*Round 2 Selection Counts of Competencies for Introductory Computer Science*

Competency	Academic Group Selections	Industry Group Selections
Analyze algorithms for effectiveness and efficiency	5	8
Illustrate concepts in artificial intelligence	0	1
Summarize basic computability, theory of computation, and its limits	7	3
Describe different types of data representation (e.g. graphics, binary numbers, etc.)	3	7
Illustrate the use of Boolean logic and basic combinational digital circuits	3	5
Describe basic computer architecture and organization	7	6
Summarize the history of computing and its ramifications to implementation today	2	3
Explain the factors contributing to human-computer interaction in computing	2	4
Illustrate the use of databases and apply SQL	3	7
Explain the operation of compilers	3	1
Discuss the operation of networks and related practices (e.g. data compression, etc.)	1	2
Explain the functionality of operating systems and provide examples	6	4
Describe common programming languages and their popular uses	4	9
Describe benefit and operation of parallel and distributed systems and programming	1	1
Demonstrate use of recursion in a program	5	3
Describe the need for computer and data security and identify best practices	5	4
Explain the role of modeling and simulation in computing	5	3
Describe societal impact of computing	2	3
Describe the World Wide Web and select internet protocols	1	2
Describe process and practices in SE	6	7
Plan a career in CS	4	2
Write functional object-oriented programs employing programming fundamentals	8	8
Write functional procedural programs employing programming fundamentals	9	8
Implement good documentation practices in programming	6	6

Table 14 Continued

Competency	Academic Group Selections	Industry Group Selections
Demonstrate teamwork and interpersonal group skills	8	8
Demonstrate algorithmic thinking.	6	5
Demonstrate computational thinking	6	5
Demonstrate problem solving	8	6
Demonstrate critical thinking and reasoning	7	8
Demonstrate systems thinking	4	2
Demonstrate creativity in programming	4	2
Demonstrate time and resource management skills in a project	0	3
Exhibit entrepreneurship in computing	0	0
Communicate effectively orally and in writing	4	4
Describes self-learning and assesses self	1	3
Exhibit digital literacy	2	4
Explain and choose from different file structures	2	2
Explain and utilize effective procedures in software verification and validation	3	4

*Note.* N = 10 for academic group and N = 11 for industry group.

industry group also included assembly language, HTML5, PL/SQL, and Shell. All Round 2 participants in the academic group selected C++, Java, and Python. The sole programming language chosen by all industry experts was JavaScript. No academic expert selected Greenfoot and no industry professional selected Alice, Greenfoot, MATLAB, Scala, or Scratch.

Table 15

*Round 2 Selection Counts of Programming Languages for Introductory Computer Science*

Programming Language	Academic Group Selections	Industry Group Selections
Alice	1	0
Assembly Language	4	6
C	9	6
C#	8	9
C++	10	9
Greenfoot	0	0
Haskell	1	2
HTML5	4	9
Java	10	9
JavaScript	8	11
MATLAB	4	0
Objective-C	2	4
Perl	2	1
PHP	5	6
PL/SQL	2	7
Python	10	10
R	1	1
Ruby	6	7
Scala	1	0
Scheme	2	1
Scratch	3	0
Shell	4	7
Swift	2	4
Visual Basic	3	4

*Note.* N = 10 for academic group and N = 11 for industry group.

## Assessments

Finally, the groups were asked to select no fewer than ten of twelve assessments. Because of the low number of assessments, the narrowing effect was expected to be minimal. Only one item, essays, was not chosen to be carried over into Rounds 3 and 4. The results of the selection counts by each group for assessments are shown in Table 16. Items selected by at least half of each group were copied over into the Round 3 survey for that group. The number of selections for each of the categories, as shown in Tables 14 to 16 were also included on the surveys.

Table 16

### *Round 2 Selection Counts of Assessments for Introductory Computer Science*

Assessments	Academic Group Selections	Industry Group Selections
Case Studies	9	10
Code Reviews	9	11
Concept Questions	10	10
Essays	3	1
Final Exams	10	9
Online Threaded Discussions	8	8
Interviews with Professionals	5	11
Lab Exercises	10	11
Quizzes	10	9
Smaller Programming Activities	10	11
Team Programming Assignments	9	10
Term Projects	9	11

*Note.* N = 10 for academic group and N = 11 for industry group.

## Round 3

The third round of the study provided experts the opportunity to rank the competencies, programming languages, and assessments selected in the previous round. The participants were

instructed to rank the items in each of the lists according to their importance for an introductory CS course. They were again advised to consider their opinions on each entry in relation to the importance attributed by the overall group as indicated by the number of experts selecting it in Round 2. Participants were given instructions on how to rank the items. They had the opportunity to click a radio button next to each item and select the numerical ranking. A second option was for participants to click and drag each item into order to indicate ranking.

The survey links for Round 3 were distributed by email on May 9, 2016. The panel structure was again utilized so that academic and industry experts provided feedback separately. The surveys for each group included duplicate instructions but consisted of the itemized lists as selected by each group in the previous round. Participants were again given eight days to complete the survey. Email reminders were sent out on the fourth and on the eighth days. There were 19 total experts who participated in the third round with 10 in the industry group (91%) and nine in the academic group (75%).

Each participant's rankings were copied into a Microsoft Excel workbook. One worksheet was used for the academic group and another for the industry group. Competencies, programming languages, and assessments were copied into rows and individual expert rankings were copied from SurveyMonkey into Microsoft Excel columns. No identifiers were used as the information submitted was confidential. The median rankings for each of the items was computed via the MEDIAN function. The interquartile range (IQR) was calculated to identify the dispersion of the middle half of these data. Field (2009) stated the median is often excluded in this calculation so the QUARTILE.exc function was used to define the first and third quartiles.

## **Course Competencies**

The academic group ranked 15 competencies and the results are presented in Table 17. Overall, the group chose “write functional procedural programs employing programming fundamentals” as the most important competency. “Write functional object-oriented programs employing programming fundamentals” and “demonstrate problem solving” were next in terms of importance, followed by “demonstrate critical thinking and reasoning” and “demonstrate algorithmic thinking.” The industry experts’ results are also shown in Table 17. They ranked 12 competencies and collectively selected “demonstrate problem solving” as their most important. The competency related to critical thinking and reasoning placed second in terms of importance. The next three competencies for the industry group were “describe processes and practices in SE,” “write functional procedural programs employing programming fundamentals,” and “demonstrate teamwork and interpersonal group skills.” The IQR values for these highly ranked items varied from 3.0 to 5.5 for the academic group and from 5.3 to 7.5 for the industry group.

The lowest ranked competencies were “describe the need for computer and data security and identify best practices” for the academic group and “illustrate the use of databases and apply SQL” for the industry group. Both groups exhibited comparatively low IQR values for their selections for the lowest ranked competencies.

## **Programming Languages**

The ranked programming languages for both groups are presented in Table 18. The academic group ranked eight programming languages and chose Java as their most important and C++ as the next highest ranked. The IQR values for the rankings of these two languages were 1.5

Table 17

*Round 3 Median Rankings of Competencies for Introductory Computer Science*

Competency	Academic Group Ranking		Industry Group Ranking	
	Median	IQR	Median	IQR
Analyze algorithms for effectiveness and efficiency	9.0	4.0	7.0	5.3
Describe different types of data representation (e.g. graphics, binary numbers, etc.)	-	-	7.0	5.5
Describe basic computer architecture and organization	12.0	5.5	6.5	9.0
Illustrate the use of databases and apply SQL	-	-	9.5	4.5
Explain the functionality of operating systems and provide examples	12.0	4.5	-	-
Describe common programming languages and their popular uses	-	-	7.5	6.3
Demonstrate use of recursion in a program	12.0	3.0	-	-
Describe the need for computer and data security and identify best practices	14.0	2.0	-	-
Explain the role of modeling and simulation in computing	12.0	6.5	-	-
Describe process and practices in SE	11.0	4.5	5.0	6.0
Write functional object-oriented programs employing programming fundamentals	3.0	4.5	7.0	3.8
Write functional procedural programs employing programming fundamentals	1.0	4.5	5.5	5.3
Implement good documentation practices in programming	7.0	7.5	8.5	7.3
Demonstrate teamwork and interpersonal group skills	8.0	6.5	6.0	7.5
Demonstrate algorithmic thinking	5.0	5.5	-	-
Demonstrate computational thinking	6.0	3.0	-	-
Demonstrate problem solving	3.0	3.5	2.5	7.5
Demonstrate critical thinking and reasoning	5.0	3.0	3.0	7.3

*Note.* N = 9 for academic group (16 items ranked) and N = 10 (12 items ranked) for industry group.



and 2.0, respectively; indicating low variability. The industry experts ranked 12 languages and selected JavaScript and Python as their most important, though there was less variability on the former's ranking (IQR = 2.3) than that of the latter (IQR = 5.3). The academic experts chose JavaScript as the least important language on their list but the IQR value of 3.5 pointed to some variability in this low ranking among the group. The industry group, meanwhile, selected Assembly language as the least important with a corresponding IQR value of 4.3.

Table 18

*Round 3 Median Rankings of Programming Languages for Introductory Computer Science*

Programming Language	Academic Group Ranking		Industry Group Ranking	
	Median	IQR	Median	IQR
Assembly Language	-	-	10.0	4.3
C	4.0	2.5	7.0	4.8
C#	6.0	2.5	4.5	5.8
C++	2.0	2.0	5.5	4.0
HTML5	-	-	5.5	5.8
Java	1.0	1.5	4.5	6.8
JavaScript	7.0	3.5	3.0	2.3
PHP	6.0	2.5	6.0	7.3
PL/SQL	-	-	8.0	4.3
Python	4.0	2.5	3.0	5.3
Ruby	6.0	2.0	9.5	3.3
Shell	-	-	9.0	4.8

*Note.* N = 9 for academic group (8 items ranked) and N = 10 for industry group (12 items ranked).

### Assessments

Finally, the groups ranked 11 assessments as shown in Table 19. Both groups selected smaller programming activities among their highest ranked items and did so with little variability

as indicated by the low IQR values of 1.5 for the academic group and 2.3 for the industry group. The academic group also selected lab exercises as a top assessment and again did so with a low variability as evidenced by the IQR value (2.0). The industry group also selected term projects as tied for the most important assessments but with a high IQR value (8.3).

Table 19

*Round 3 Median Rankings of Assessments for Introductory Computer Science*

Assessment	Academic Group Ranking		Industry Group Ranking	
	Median	IQR	Median	IQR
Case Studies	9.0	4.5	6.5	2.3
Code Reviews	6.0	3.0	4.5	2.0
Concept Questions	5.0	2.5	6.0	5.8
Final Exams	7.0	3.0	8.0	3.0
Online Threaded Discussions	10.0	1.5	9.0	4.3
Interviews with Professionals	10.0	1.5	8.5	4.0
Lab Exercises	2.0	2.0	4.0	6.8
Quizzes	6.0	3.5	8.5	7.5
Smaller Programming Activities	2.0	1.5	3.0	2.3
Team Programming Assignments	4.0	3.5	6.0	5.5
Term Projects	6.0	4.5	3.0	8.3

*Note.* N = 9 for academic group (11 items ranked) and N = 10 for industry group (11 items ranked).

### **Group Concordance**

Kendall's  $W$  was chosen to analyze the conformity among the rankings of the three categories by the expert groups. Computation of this statistic requires the deviations of the individual elements to be analyzed, the number of items, and the number of rankings to be compared. The  $k$  value, indicating the number of items ranked was computed via Microsoft

Excel's COUNT function of the rows used to indicate competencies, programming languages, and assessments. The  $m$  value, indicating the number of participants, was computed via the COUNT function of the columns with rankings. Zaiontz (2013) suggested using Microsoft Excel to calculate Kendall's  $W$  and the approach was used for the three categories for both the academic and industry groups. The results are presented in Table 20.

Table 20

*Kendall's W Values for Round 3*

Expert Group	Levels of Agreement		
	Competencies	Programming Languages	Assessments
Academic	$W_{AC} = 0.57$	$W_{AL} = 0.63$	$W_{AA} = 0.53$
Industry	$W_{IC} = 0.13$	$W_{IL} = 0.10$	$W_{IA} = 0.20$

*Note.* N = 10 for academic group and N = 11 for industry group.

Linear transformations of the Kendall's  $W$  were performed to describe the corresponding correlations ( $r$ ) so the level of agreement for each of the categories by the groups could be identified (Zaiontz, 2013). P-values were calculated to determine significance. Neither group reached the consensus threshold of  $W = 0.7$ , as recommended by Schmidt (1997), on any of the three categories in Round 3. Even so, the academic experts apparently agreed more on each of the three categories than the industry experts did. Kendall's coefficient of concordance ( $W$ ) tests showed statistically significant agreement by the academic group on the competencies ( $W_{AC} = 0.57$ ,  $r_{AC} = 0.52$ ,  $p < 0.001$ ), programming languages ( $W_{AL} = 0.63$ ,  $r_{AL} = 0.58$ ,  $p < 0.001$ ), and assessments ( $W_{AA} = 0.53$ ,  $r_{AA} = 0.48$ ,  $p < 0.001$ ). The industry experts exhibited more disparity in

their rankings but achieved statistical significance in their agreement for their assessments ( $W_{IA} = 0.20$ ,  $r_{IA} = 0.11$ ,  $p = 0.03$ ). Their agreement levels for the competencies ( $W_{IC} = 0.13$ ,  $r_{IC} = 0.03$ ,  $p = 0.21$ ) and languages ( $W_{IL} = 0.10$ ,  $r_{IL} = 0.00$ ,  $p = 0.43$ ), however, lacked statistical significance.

#### **Round 4**

Because of the lack of consensus among either group on any of the three categories, the Round 3 surveys were copied for Round 4. The coefficient of concordance values for each category were included and explained to the participants so they would have information on the level of consensus they had achieved. The median rank values were also provided so the experts could weigh their preferences against those of the rest of the group. The items for each category were listed in order of their median rankings from Round 3.

The links for the final round's surveys were distributed by email on May 23, 2016. Participants were this time given nine days to complete the survey because of a holiday. The structure was identical to that utilized in the previous round with academic and industry experts providing split feedback. Email reminders were sent out on the fourth and on the ninth days. There were 20 experts who participated in the final round. All eleven industry members participated (100%) and nine of the twelve academic experts (75%) completed their surveys. The same process was used to analyze the data. Ranking values were downloaded to an Excel spreadsheet and the same formulas were used as in Round 3.

#### **Course Competencies**

Round 4 rankings for competencies by both groups are presented in Table 21. The academic group made few changes to their rankings for competencies from Round 3. The order of the competencies' rankings was highly similar to that in the previous round with only slight exceptions. "Write functional procedural programs employing programming fundamentals" was

again the most important competency and “describe the need for computer and data security and identify best practices” the least important. Overall, the ordered rankings were more defined in Round 4 as there were only two competencies with equivalent rankings. The competencies dealing with operating systems and recursion in programming both exhibited median rankings of 13.0 whereas both had achieved median rankings of 12.0 in the previous round. Other than this minor difference, the ordered rankings of the competencies were identical in Round 4 for the academic group. Additionally, the IQR values decreased for all but one of the competencies. The sole exception was the item dealing with computer and data security, which had an IQR value of 2.5 compared with 2.0 in the previous round. The lower IQR values indicated less variation in the final round of item rankings in this category.

The situation was similar for the industry group’s rankings, though to a reduced extent. “Demonstrate critical thinking” was promoted to a median ranking of 2.0, joining “demonstrate problem solving” as the most important competency. “Illustrate the use of databases and apply SQL” was again the lowest ranked item. A notable change in the ordered list of competencies for the industry professionals was that “implement good documentation practices in programming” was promoted ahead of “write functional object-oriented programs employing programming fundamentals” and “describe common programming languages and their popular uses.” Other than this minor difference, the list was generally comparable to what was generated by the group in Round 3. Most items experienced a decrease in IQR, again pointing to less variation in the ranked values of competencies. The two exceptions were the competencies dealing with object-oriented programming (3.8 to 5.0) and common programming languages (6.3 to 7.0).

Table 21

*Round 4 Median Rankings of Competencies for Introductory Computer Science*

Competency	Academic Group Ranking		Industry Group Ranking	
	Median	IQR	Median	IQR
Analyze algorithms for effectiveness and efficiency	9.0	2.0	7.0	3.0
Describe different types of data representation (e.g. graphics, binary numbers, etc.)	-	-	7.0	4.0
Describe basic computer architecture and organization	11.0	3.0	6.0	7.0
Illustrate the use of databases and apply SQL	-	-	11.0	2.0
Explain the functionality of operating systems and provide examples	13.0	2.5	-	-
Describe common programming languages and their popular uses	-	-	9.0	7.0
Demonstrate use of recursion in a program	13.0	2.0	-	-
Describe the need for computer and data security and identify best practices	15.0	2.5	-	-
Explain the role of modeling and simulation in computing	14.0	1.5	-	-
Describe process and practices in SE	10.0	1.0	3.0	5.0
Write functional object-oriented programs employing programming fundamentals	2.0	0.5	9.0	5.0
Write functional procedural programs employing programming fundamentals	1.0	1.0	6.0	5.0
Implement good documentation practices in programming	7.0	2.5	8.0	5.0
Demonstrate teamwork and interpersonal group skills	8.0	2.5	6.0	3.0
Demonstrate algorithmic thinking	4.0	4.0	-	-
Demonstrate computational thinking	6.0	0.5	-	-
Demonstrate problem solving	3.0	1.0	2.0	2.0
Demonstrate critical thinking and reasoning	5.0	2.0	2.0	5.0

*Note.* N = 9 for academic group (16 items ranked) and N = 11 for industry group (12 items ranked).

## Programming Languages

The Round 4 results for programming languages are shown in Table 22. The academic group changed little in their rankings from Round 3 to Round 4. Java remained the top language, (Median = 1.0, IQR = 2.0), followed by C++ (Median = 2.0, IQR = 1.5). The experts gave JavaScript a median ranking of 6.0 in Round 4 compared with a 7.0 in Round 3, which had then represented the least important language. This change came at the expense of C#, which went from a ranking of 6.0 to 8.0 (last in Round 4).

Table 22

### *Round 4 Median Rankings of Programming Languages for Introductory Computer Science*

Programming Language	Academic Group Ranking		Industry Group Ranking	
	Median	IQR	Median	IQR
Assembly Language	-	-	11.0	9.0
C	4.0	4.0	7.0	5.0
C#	8.0	3.0	4.0	3.0
C++	2.0	1.5	6.0	4.0
HTML5	-	-	6.0	4.0
Java	1.0	2.0	3.0	7.0
JavaScript	6.0	3.5	3.0	4.0
PHP	6.0	2.0	9.0	5.0
PL/SQL	-	-	8.0	4.0
Python	4.0	1.0	3.0	4.0
Ruby	6.0	1.0	9.0	6.0
Shell	-	-	8.0	3.0

*Note.* N = 9 for academic group (8 items ranked) and N = 11 for industry group (12 items ranked).

The industry group had a few more noteworthy changes in their rankings of programming languages in the final round. Java (median rank = 3.0, IQR = 2.0), joined Python (median rank =

3.0, IQR = 1.0) and JavaScript (median rank = 3.0, IQR = 4.0) as the most important languages. Assembly language held its position as last in the list (median rank = 11.0) but experienced a sizable increase in variability (IQR = 9.0) among its rankings. It was notable that two industry experts chose assembly language as the most important language on their lists while another five ranked it least important. The only other major change involved the group's dropping of PHP from a ranking of 6.0 in Round 3 (IQR = 7.3) to 9.0 in Round 4 (IQR = 5.0).

### **Assessments**

The final round rankings for assessments by each group are shown in Table 23. Again, the academic group exhibited little difference in their ranked lists. Lab exercises were deemed the most important assessment device by the group (Median rank = 1.0, IQR = 1.5), followed by smaller programming activities (Median rank = 2.0, IQR = 1.0). The least important assessment chosen by the group was again interviews with professionals (Median rank = 11.0, IQR = 3.0). The only other noteworthy change in the rankings was the equivalent ranking of final exams (Median rank = 8.0, IQR = 3.5) and case studies (Median rank = 8.0, IQR = 2.5). The former had been ranked just ahead of the latter in the previous round.

The industry group ranked assessments slightly differently than they had in Round 3. Smaller programming activities (Median rank = 1.0, IQR = 4.0) was still chosen as the most important assessment device, though on its own in Round 4. The other previously top ranked assessment device, term projects, was demoted to third on the list (Median rank = 4.0, IQR = 5.0), surpassed by lab exercises (Median rank = 3.0, IQR = 2.0). Online threaded discussions (Median rank = 11.0, IQR = 3.0) were again ranked as the least important assessments by the group.



Table 23

*Round 4 Median Rankings of Assessments for Introductory Computer Science*

Assessment	Academic Group Ranking		Industry Group Ranking	
	Median	IQR	Median	IQR
Case Studies	8.0	2.5	7.0	1.0
Code Reviews	6.0	3.0	5.0	4.0
Concept Questions	4.0	3.5	5.0	3.0
Final Exams	8.0	3.5	9.0	3.0
Online Threaded Discussions	10.0	1.0	11.0	3.0
Interviews with Professionals	11.0	3.0	9.0	4.0
Lab Exercises	1.0	1.5	3.0	2.0
Quizzes	6.0	2.5	8.0	8.0
Smaller Programming Activities	2.0	1.0	1.0	4.0
Team Programming Assignments	3.0	2.5	6.0	4.0
Term Projects	6.0	3.0	4.0	5.0

*Note.* N = 9 for academic group (11 items ranked) and N = 11 for industry group (11 items ranked).

### Group Concordance

Kendall's coefficient of concordance ( $W$ ) tests were again computed in Round 4. The results are presented in Table 24. Consensus, as defined by the threshold of  $W = 0.7$ , was only achieved by the academic group on the rankings for competencies ( $W_{AC} = 0.84$ ,  $r_{AC} = 0.82$ ,  $p < 0.001$ ). Though concordance values increased for both groups on each of the three categories, the academic experts again showed higher conformity than the experts from industry. Kendall's  $W$  values again showed statistically significant agreement by the academic group on the competencies, programming languages ( $W_{AL} = 0.63$ ,  $r_{AL} = 0.58$ ,  $p < 0.001$ ), and assessments ( $W_{AA} = 0.67$ ,  $r_{AA} = 0.62$ ,  $p < 0.001$ ). The concordance values for the industry group again revealed less agreement in their rankings but this time achieved statistical significance in their agreement for both competencies ( $W_{IC} = 0.32$ ,  $r_{IC} = 0.25$ ,  $p < 0.001$ ) and assessments ( $W_{IA} =$

0.37,  $r_{IA} = 0.31$ ,  $p < 0.001$ ). The industry group, however, displayed little agreement on programming languages ( $W_{IL} = 0.12$ ,  $r_{IL} = 0.02$ ,  $p = 0.25$ ).

Table 24

*Kendall's W Values for Round 4*

Expert Group	Levels of Agreement		
	Competencies	Programming Languages	Assessments
Academic	$W_{AC} = 0.84$	$W_{AL} = 0.63$	$W_{AA} = 0.67$
Industry	$W_{IC} = 0.32$	$W_{IL} = 0.12$	$W_{IA} = 0.38$

*Note.* N = 9 for academic group and N = 11 for industry group.

The synopsis of these findings are presented in Chapter 5. The implications of the results from the rankings by the academic and industry experts are discussed. This information is useful to the curriculum designer of introductory courses in CS, and to those who may wish to replicate the study elsewhere.

## CHAPTER V

### CONCLUSIONS AND RECOMMENDATIONS

The aim of this research was to define the competencies, programming languages, and assessments for an introductory CS course recommended by experts in academia and industry in California's Central Valley. This chapter will examine the findings presented in the previous one and present conclusions and recommendations for curriculum designers in this geographic region and those individuals who have an interest in teaching introductory CS.

Roberts (2011) pointed to the dependence of our economy on persons trained in CS. The increased national emphasis on the teaching of CS at the K-12 level (Fisher, 2016) and initiatives such as Hour of Code, which recently boasted almost 200,000 events worldwide (Doplick, 2015), point to continued growth in the popularity of introductory courses in the field. Creating courses to meet this increased demand can be difficult as there are myriad topics and tools available for curriculum designers to consider.

This research identified recommendations for some major components for an introductory course in CS. Competencies help to determine the scope of an instructional course, which in turn helps to define curriculum (Brown & Green, 2011). Assessments are sources of student and instruction evaluation and also play an instrumental role in a course's design (Brown & Green, 2011; Ornstein & Hunkins, 2013). Though the possibilities for what to include in a course in CS are wide (Gupta, 2007), programming tends to be included as a significant component to introductory CS courses (JTFCC, 2013).

Recommendations of experts in academia and industry were solicited using a Delphi approach. Elements were used from Okoli and Pawlowski's (2004) design to obtain feedback on potential competencies, programming languages, and assessments. The industry group ranked 12

competencies, 12 programming languages, and 11 assessments whereas the academic group provided feedback on 15 competencies, 8 programming languages, and 11 assessments.

### **Recommendations for Course Competencies**

The competencies for the introductory CS course help to identify the topics to be covered and the level of mastery required by students (Koszalka, Russ-Eft, & Reiser, 2013). The first research question addressed course competencies:

RQ<sub>1</sub>: What competencies do subject matter experts recommend for students in California's Central Valley to master in an undergraduate introductory CS course?

A primary concern is the scope of the content in a course's design. Herlo (2015) recommended that learning outcomes should be "limited in number" (p. 38) and based on competencies. Barker et al. (2009) found perceived "pace and workload" (p. 156) to be significant predictors of CS0 students' intent to pursue a CS major. It is desirable, therefore, to narrow the focus the competencies for an introductory course.

There was a high degree of commonality reached between the academic and industry groups though they ranked items separately. Nine competencies were selected by both groups in Round 2. See Table 25 for this list of competencies. Of these nine, only "describe basic computer architecture and organization" failed to place in the top ten competencies for each of the groups; as it came in eleventh for the academic group. Similarities were strong and the experts from academia and industry appear to give a somewhat unified recommendation to an introductory CS course designer.

For example, there was strong emphasis on professional skills, programming, and the software development process to teach the fundamentals of CS. The professional skills of problem solving, critical thinking and reasoning, teamwork and interpersonal group skills are

typical CS program level outcomes and a focus can often be found in introductory courses (Whitfield, 2003). The industry group ranked solving problems as the most important competency in Rounds 3 and 4. The academic group selected it as the third most important competency in the final round. Both groups also achieved a high degree of consensus on their rankings of problem solving as evidenced by their low corresponding IQR values.

Table 25

*Top Recommended Competencies for Introductory Computer Science by Both Groups*

---

Competency

---

Demonstrate problem solving

Demonstrate critical thinking and reasoning

Write functional procedural programs employing programming fundamentals

Describe process and practices in SE

Demonstrate teamwork and interpersonal group skills

Write functional object-oriented programs employing programming fundamentals

Implement good documentation practices in programming

Analyze algorithms for effectiveness and efficiency

Describe basic computer architecture and organization

---

The importance attributed to problem solving in introductory computing courses is often found in other research (Schneider, 2004; Schulte & Bennedsen, 2006; Wang et al., 2011; Whitfield, 2003) and this focus is certainly not limited to the United States. The ability to solve problems has been identified as one of the most important required by business and CS industry in the Czech Republic (Poulova & Klimova, 2015), though it is often overlooked as an educational outcome in undergraduate programming courses (Krpan, Mladenović, & Rosić, 2015).

Another professional skill that was given high priority by both groups was critical thinking. The industry group rated the competency to “demonstrate critical thinking and reasoning” as tied for most important while the academic group included it in their top five. The skill has often been identified as one of the most recognizable goals for students in a CS program (Muñoz et al., 2013; Tasneem, 2012).

The experts in this study attributed high importance to students learning how to program. Though introductory CS courses don’t always rely on programming they typically focus on this activity (JTFCC, 2013; Marling & Juedes, 2016; Stamey & Sheel, 2010). CS0 courses, specifically, vary widely in their focus on programming (Davies et al., 2011). The professionals in this study recommended an emphasis on programming and software development skills over other topics, such as networking, operating systems, or databases, to introduce students to the field of CS.

The competency to “write functional procedural programs employing programming fundamentals” was among the highest ranked in this study. The academic group chose it as the top competency and with minimal variation and the industry experts ranked it in their top five. The writing of functional object-oriented programs was also important to both groups but was ranked lower than the competency related to procedural programming in each case. The academic group ranked object-oriented programming second most important and the industry experts placed it in their top ten. Object-oriented programming has been emphasized in introductory computing courses (Elarde & Fatt-Fei, 2011; Tew, 2010). The inclusion of both procedural and object-oriented paradigms is consistent with the findings of Goldman et al. (2008), who also found topics in both approaches to be among the most important for CS1 courses. Surakka (2005) also reported that procedural and object-oriented programming were

among the most important topics as reported by academic and industry professionals. Schulte and Bennedsen (2006) found that introductory programming courses shared many of the same learning objectives, regardless of the use of the procedural or objects paradigm. The recommendation from the experts in this study appears to be to focus on procedural programming and include elements of object-oriented programming in an introductory CS course.

Two competencies ranked in the top five of one group and in the top ten of the other. The competency “demonstrate teamwork and interpersonal group skills” was ranked in the top five of the industry group and in the top ten of the academic group. The ability to work well with other students is typically a desired competency for introductory CS courses (McDowell et al., 2006; Moura and van Hattum-Janssen, 2011; Muñoz et al., 2013; Soper, 2014) and the experts in this research agreed. Additionally, the competency to “describe process and practices in SE” was ranked third by the industry experts and tenth by the academic group. It was of little surprise the industry group rated this skill prominently for an introductory course as it is of primary importance in software development (Lutz et al., 2014; Surakka, 2007). This recognition by the academic experts was noteworthy. Introduction to CS course designers should consider giving students opportunities to work in teams and utilize SE processes in their development of code.

The two groups also included two additional competencies in their top ten rankings. The ability to “implement good documentation practices in programming” is covered as a recommended skill by the JTFCC (2013) but has been noted as being often overlooked in introductory courses (McMaster & Zastre, 2011). The skill acquired attention by both academic and industry experts in this case. Additionally, the ability to “analyze algorithms for effectiveness and efficiency” was also ranked in the top ten of both groups. This competency,

suggested by the JTFCC (2013) and LACS (2007) curriculum recommendations is also found as an area of focus in the literature for introductory courses (Dodds et al., 2008; Gal-Ezer, Vilner, & Zur, 2003; Gorn, 1963; Schulte & Bennedsen, 2006) as well as more advanced courses in computing (Enbody et al., 2009).

The final competency selected by both groups in Round 2 and placing in the top ten of the industry group was related to computer architecture. This topic was covered extensively in each of the three textbooks reviewed in the literature (Anderson et al., 2011, Schneider & Gersting, 2016; Dale & Lewis, 2016). Surakka (2005) found that separate groups of software developers and professors in Finland considered the topic important for software development as they both rated the topic as at least a 3.0 on a 4-point Likert scale. It was notable that the industry group ranked this competency as tied for fourth along with the competencies related to procedural programs and teamwork. This fact, coupled with the academic group ranking it just outside the top ten provides justification for including it as one of the top competencies to consider for introductory courses in CS. It is important to note, however, that both groups had the most dispersed rankings for this competency among those listed in Table 25. The introductory CS course designer, might be advised to pay attention to the level of focus given to topics in computer architecture. A compromise would be to include this competency as a secondary one to consider for the course and this is the suggestion here.

Another nine competencies were selected by only one of the two groups in Round 2. Competencies related to algorithmic and computational thinking were among the most important to the academic group, ranking fourth and sixth, respectively; but were not selected by the industry group. The competency related to data representation was likewise considered important by the industry group (Median rank = 7) but not selected by the academic group. Because these



competencies were not considered important by both groups, a curriculum designer might consider using them as enabling objectives or as areas to cover in other courses. A reduced emphasis on these secondary topics will allow for more focus on those considered most important by both academic and industry experts.

Additional support for this suggestion comes from the level of consensus among both groups on the competencies listed in Table 25 as compared to these other ten. The IQR values for the suggested primary and secondary competencies are shown in Table 26 for both groups. Goldman et al. (2008) found CS topics that were outliers or controversial garnered weak consensus and this situation is confirmed here.

Table 26

*Round 4 Interquartile Range Values for Suggested Primary and Secondary Competencies*

Expert Group	Primary	Secondary
Academic	1.56	2.29
Industry	4.13	5.00

*Note.* N = 8 for primary competencies and N = 9 for secondary competencies.

### **Recommendations for Programming Languages**

The next aspect of the introductory CS course's design to be researched was the selection of programming languages. It was clear from the experts' preferences for competencies relating to programming and software development that the language would play an important role in the course. The related research question read as follows:

RQ2: What programming languages do subject matter experts recommend for students in California's Central Valley to use in an undergraduate introductory CS course?

The choice of language to teach programming concepts has been researched and discussed extensively (Dodds et al., 2008; Forte & Guzdial, 2005; Guzdial, 2009; Settle, Lalor, & Steinbach, 2015; Schulte & Bennedsen, 2006; Shein, 2015). Academic institutions typically use one language in introductory courses (Lewis, Blank, Bruce, & Osera, 2016) though there are instances in which as many as three languages have been utilized (Mahmoud, Dobosiewicz, & Swayne, 2004). The top four languages recommended by the academic experts in this study were Java, C++, Python, and C. These top four perfectly matched the results reported by Ben Arfa Rabai et al. (2015). The industry group, however, were less uniform in their selection and rankings of languages, again consistent with the findings of Ben Arfa Rabai et al. (2015). They selected twelve languages, compared to eight by the academic group, and their rankings exhibited an average IQR value of 4.83, compared to 2.25 for those of the academic experts. The reason was not investigated as part of this study but may have been a result of the developers' emphases and company sizes (Meyerovich & Rabkin, 2013).

The academic and industry groups both recommended Java as the top language for an introductory CS course. Java earned the top ranking by the academic group in both the final two rounds. This result was consistent with the findings of Ben Arfa Rabai et al. (2015). The industry group also promoted it to be tied with two other languages (JavaScript and Python) deemed most important in the final round. This preference for Java was somewhat expected as the language has been among the most popular for use in introductory programming courses for some time (Ben Arfa Rabai et al., 2015; Guo, 2014; Schulte & Bennedsen, 2006). The language was ranked as most popular in four of the six resources used to gauge national industry preferences presented in Chapter II. Davies et al. (2011) conducted a survey of undergraduate CS departments and found that over 40% used Java in their CS1 and CS2 courses. It is worth noting, however, that

the industry experts exhibited high variability in their final ranking of Java (IQR = 7.0) and that only assembly language had higher variation in its rank. There was thus wide discrepancy in the opinions of the industry professionals on the importance of Java for introductory CS courses. It can be inferred that Java is still considered highly important in several domains of the software development industry (TIOBE Index for December 2015), but there is recognition the language is more difficult for beginners to learn than other options (Ali & Mensch, 2008; Ali & Smith, 2014; Cheng et al., 2010).

The preference for Java by academic and industry experts was, however, at odds with both groups' higher ranking of the competency related to procedural programming over the one dealing with object-oriented programming. It might have been more expected that a procedural language would have been recommended higher by both groups. The industry group also selected JavaScript, a language that spans several paradigms, and Python, which can also include object-oriented elements and support multiple paradigms (Agarwal, Agarwal, & Fife, 2012), as top ranked choices.

JavaScript has been used in introductory courses (Baldwin et al., 2010; Elarde & Fatt-Fei, 2011; Mahmoud et al., 2004; Schneider, 2004) and has been used in CS0 and first computing courses (Ben Arfa Rabai et al., 2015; Davies et al., 2011). It is more widely known, however, for its use in industry (Guo, 2014; Meyerovich & Rabkin, 2013; Shein, 2015). The academic group also ranked JavaScript in their list (Median rank = 6.0), but did so below Python (Median rank = 4.0), which equated to third most popular and tied with C.

There was an indication in this research that Python is becoming increasingly important in the eyes of industry, or that there is a recognition of its value in teaching introductory programming. This situation confirms what has been reported about the continuous growth of its

popularity nationwide (Ben Arfa Rabai et al., 2015; Davies et al., 2011; Guo, 2014; JTFCC, 2013; Meyerovich & Rabkin, 2013; Shein, 2015). Many curriculum designers have chosen to use Python for initial computing courses (Agarwal et al., 2012; Ben Arfa Rabai et al., 2015; Cheng et al., 2010; Davies et al., 2011; Guzdial, 2009; Norman & Adams, 2015) and lead up to using Java or C++ in later programming classes (Davies et al., 2011; Dodds et al., 2008; Enbody et al., 2009). Python's attractiveness in introductory courses is a result of its relative ease to learn (Enobdy et al., 2009; Meyerovich & Rabkin, 2013; Shein, 2015).

The C++ programming language was also highly ranked by both groups. The academic group ranked it second, again with a high degree of consensus, whereas the industry group ranked it in their top five, with some level of agreement. The language is still highly popular in industry but has been more associated with CS1 and CS2 courses than CS0 courses (Ben Arfa Rabai et al., 2015; Davies et al., 2011).

Visual programming languages, such as Alice and Scratch, have been recommended as gateways to introduce students to programming (Ali & Mensch, 2008; Ali & Smith, 2014; desJardins & Littman, 2010; JTFCC, 2013; Malan & Leitner, 2007; Tanrikulu & Schaefer, 2011) or as remedial aids for introductory programmers (Chang, 2014). Davies et al. (2011) found Alice to be the most widely used language in CS0 courses. The experts in this study, however, did not choose any of the three visual programming languages available and this omission was likely due to lack of awareness. For example, 17 of 21 participants in Round 1 were not familiar with Greenfoot. Scratch and Alice were not known by 12 of the experts. Ben Arfa Rabai et al. (2015) lamented at the lack of adoption of visual programming languages to teach programming in academia. Despite this lack of awareness by the experts in this study, these languages merit consideration as they have been shown to be successful tools to teach students programming

skills (Malan & Leitner, 2007; Rizvi, Humphries, Major, Jones, & Lauzun, 2011) and preferred by non-majors (Elarde & Fatt-Fei, 2011).

There was consistency between the recommendations of experts in California's Central Valley to those located nationwide and beyond. The preferences from both groups of participants in this research coupled with the literature appear to call for the use of Java in an introductory CS course. It is apparent the language is valued in industry and has a consistent track record in use in academia. Java could also be supplemented with a visual programming language, such as Scratch or Alice, to introduce programming topics early in the course. Another strong possibility would be to use Python instead of Java to introduce topics in programming. This suggestion would appear to have increased merit especially if accessibility was a concern. Python also seems to better match the preference for competencies relating to both procedural and object-oriented programming. It would again be a good idea to ease into teaching the course with a visual programming language.

### **Recommendations for Assessments**

The final components of the introductory CS course studied were assessments. These elements would help to define the level of student mastery of the course competencies. The research question guiding this portion of the study was:

RQ<sub>3</sub>: What assessments do subject matter experts recommend for students in California's Central Valley to demonstrate mastery of competencies for an undergraduate introductory CS course?

The participants in this research also provided preferences for the assessments to be used in an introductory CS course. A review of the syllabi referenced or presented in the literature (desJardins & Littman, 2010; Fulton & Schweitzer, 2011; Kunkle, 2010; Muñoz et al., 2013;

Xiang & Ye, 2009; Zhao et al., 2015) and select course syllabi used at Harvard University (Malan, 2012), Massachusetts Institute of Technology (Kaelbling et al., 2011), and Stanford University (Parlante, 2014) identified the use of two to six distinct categories of assessments used in introductory CS classes.

The academic and industry experts had some level of similarity in the assessments they recommended. Five of the suggested assessments were placed in the top five of both groups. These items are shown in Table 27. Smaller programming activities were ranked as the top assessment by the industry group and second by the academic group. The academic group had more uniformity in its placement of smaller programming activities (IQR = 1.0) than did the industry group (IQR = 4.0). It is noteworthy that these types of exercises strongly relate to the emphasis on programming exhibited by the experts and their preferred competencies.

Table 27

*Top Recommended Assessments for Introductory Computer Science by Both Groups*

---

Assessment

---

Smaller programming assignments

Lab exercises

Concept questions

Term projects

Code reviews

Team programming assignments

---

Lab exercises were also ranked highly by academic and industry experts. The faculty members ranked this assessment type at the top of their list while the industry professionals

ranked it second most important. The academic and industry groups were both in high agreement of their rankings of this assessment yielding IQR values of 1.5 and 2.0, respectively.

The major conclusion on the preference for these two items is that experts believe that students need to learn by doing. There has been an increased call for active learning opportunities in engineering and CS classes (Crawley, Malmqvist, Östlund, Brodeur, & Eström, 2014) and the experts in this research appear to agree.

Concept questions were often mentioned in the literature as being important components of introductory CS courses (Cortina, 2007; Fulton & Schweitzer, 2011; Horton et al., 2014; Muñoz et al., 2013; Whitfield, 2003). Another observation is that concept questions, along with lab exercises and smaller programming activities are more formative in nature. It appears the experts also preferred shorter and more periodic assessment for an introductory CS course. This preference has also been reflected in the literature (Cardona, Vélez, Tobón, 2014; Bälter et al., 2013) and a CS course curriculum designer should accordingly emphasize these types of experiences.

Term projects were also highly recommended as they placed tied for fifth in the academic group's rankings and third in those of the industry group. Term projects are summative experiences and encompass more time during a semester. They have been noted as important components in introductory CS courses emphasizing active learning (Moura & van Hattum-Janssen, 2011).

Finally, code reviews also received notable support from academic and industry professionals. The academic group ranked them as fifth most important (tied with quizzes) and the industry group attributed a fourth place ranking (Median rank = 5.0, tied with concept questions). Code reviews or artifact examinations (Cohoon et al., 2013) mimic activities in a

professional setting and allow students to review programs together. These types of assessments have been shown to be highly beneficial to student learning (Harding & Engelbrecht, 2015; Law et al., 2010). Hauswirth and Adamoli (2013) noted that “having to evaluate their peer’s solutions (after submitting their own) increases the chance that [students] will recognize their knowledge gaps and that they will become interested in filling those gaps by asking questions during the subsequent discussion phase” (pp. 511-512). Code reviews, therefore, appear to potentially play an important role in an introductory CS course and merit consideration.

Another item, team programming assignments received strong support from the groups, placing in the top five of one and in the top ten of the other. Team programming assignments not only emphasize the competencies relating to programming, but also the one dealing with teamwork. The academic group ranked the item as third most important whereas the industry group ranked it just outside their top five (Median rank = 6.0). These types of experiences, and pair programming specifically, have been associated with increased proficiency and confidence of programmers and increased representation of underrepresented groups (McDowell et al., 2006). Although the addition of a sixth type of assessment might appear to lead to a crowded syllabus, team aspects could certainly be added to other assessments recommended by the experts (e.g. lab exercises, smaller programming assignments, etc.).

The remaining assessments were not as highly regarded by the participants in this study. More traditional activities, such as quizzes, case studies, and final exams were not deemed as important to assess learning in introductory CS courses as those previously mentioned. These items, though still popular (Fulton & Schweitzer, 2011; Horton et al., 2014; Muñoz et al., 2013; Rolka & Remshagen, 2015; Wang et al., 2011), were not valued as much by either group of experts. One possible explanation might be the aforementioned desire for more active learning



experiences associated with the higher ranked assessment selections. It is important to note, however, that quizzes constitute opportunities for formative assessment and could help at improving student outcomes (Bälter et al., 2013).

Additionally, online threaded discussions were likely seen as not as important as they are associated with the online and blended modalities (Harmon, Alpert, & Lambrinos, 2014; Singleton, 2013) and some participants may not have been familiar with these approaches to learning. Interviews with professionals were also ranked low on both academic and industry lists. These assessments would likely be more useful for courses with more major students than those with a considerable non-major population.

### **Overall Course Recommendations and Future Research**

The overall goal of this study was to identify regional experts' recommendations to help better design an introductory CS course for majors and non-majors. While there are diverse needs to address in such a course, curriculum designers must do their best to prepare a positive learning experience. Professionals in academia and industry can provide invaluable input on the content for such a course. Though their interests are varied, there can be consensus on course components such as competencies, programming languages, and assessments. The experts in this study recommended a CS course that provides students with a focus on programming and SE process along with training in so-called soft skills; such as problem solving, critical thinking, and teamwork. Assessments should be based on the opportunity to learn by doing; in the form of smaller and team programming activities, lab exercises, and term projects, and more traditional concept questions. Code reviews should also be used to help students learn best practices and build their own knowledge. These recommendations seemingly point more to an introductory

course in SE, than one in CS. This change would be more in line with the intended course as part of an SE program.

The choice of programming languages to use in introductory CS courses will likely remain a contentious one. A curriculum designer is well advised to use a language like Java, which continues to thrive in the classroom and in industry. It is important, however, to consider the audience and keep a close eye on the dynamic programming field. Python continues to increase in popularity and its accessibility and versatility make it a strong choice, especially for courses with non-majors. Visual programming languages like Alice, Greenfoot, and Scratch should continue to be considered to introduce concepts in programming before transitioning to a language like Java or Python.

This research identified the recommendations of stakeholders and two major groups were considered: academic and industry professionals. The context for this study was an introductory CS course at a small private university launching a new SE program. It would be beneficial to obtain feedback on course content from graduates and their employers. While this situation was not conducive to that type of information, it is the intent to solicit the opinions of both of these groups at a future date. Course designers at other institutions are encouraged to seek out the opinions of various stakeholders and continuously update curriculum. With these types of activities academic professionals can be more confident that they provide students with the opportunities they need to develop the required skills for their future careers.

A final suggestion for future research would be to include focus groups or one-on-one interviews with academic and industry professionals. The online Delphi approach used in this study was successful in that 20 academic and industry professionals remained engaged through four rounds and provided valuable information. Alternate designs, however, would allow for the

study of the differences between the groups. Separate interviews would help to identify the reasons for experts' choices and help the curriculum designer make more informed decisions.

Most academic programs in computing have industry advisory groups and their members would be excellent candidates to provide this level of detail and for development efforts aimed at continuous improvement.

## References

- Aarts, D. (2015). The dropout founder myth. *Canadian Business*, 88(14), 24.
- Agarwal, K. K., Agarwal, A., & Fife, L. (2012). Python and Visual Logic: A good combination for CS0. *Journal of Computing Sciences in Colleges*, 27(4), 22-27.
- Ali, A. (2009). A conceptual model for learning to program in introductory programming courses. *Issues in Informing Science and Information Technology*, 6, 517-529.
- Ali, A., & Mensch, S. (2008). Issues and challenges for selecting a programming language in a technology update course. *Information Systems Education Journal*, 7(85), 1-10.  
Retrieved from <http://isedj.org/7/85/ISEDJ.7%2885%29.Ali.pdf>
- Ali, A., & Smith, D. (2014). Teaching an introductory programming language in a general education course. *Journal of Information Technology Education: Innovations in Practice*, 13, 57-67. Retrieved from <http://www.jite.org/documents/Vol13/JITEv13IIPp057-067Ali0496.pdf>
- Alvarado, C., & Dodds, Z. (2010). Women in CS: An evaluation of three promising practices. In *Proceedings of the 41st ACM Technical Symposium on Computer Science Education*, Milwaukee, WI (pp. 57-61). New York: ACM. doi:10.1145/1734263.1734281
- Anderson, G., Ferro, D., & Hilton, R. (2011). *Connecting with computer science* (2nd ed.). Boston, MA: Course Technology.
- Anderson, L. W., & Krathwohl, D. R. (2001). *A taxonomy for learning, teaching, and assessing: A revision of Bloom's taxonomy of educational objectives*. Boston, MA: Allyn & Bacon.
- Arden, B. W. (1964). On introducing digital computing. *Communications of the ACM*, 7(4), 212-214.

- Association for Computing Machinery Curriculum Committee on CS. (1965). An undergraduate program in computer science--preliminary recommendations. *Communications of the ACM*, 8(9), 543-552.
- Austing, R. H., Barnes, B. H., Bonnette, D. T., Engel, G. L., & Stokes, G. (Eds.). (1978). Curriculum '78: Recommendations for the undergraduate program in computer science. *Communications of the ACM*, 22(3), 147-166.
- Bacon, C. J., & Fitzgerald, B. (2001). A systematic framework for the field of information systems. *The DATA BASE for Advances in Information Systems*, 32(2), 46-67.
- Baldwin, D., Brady, A., Danyluk, A., Adams, J., & Lawrence, A. (2010). Case studies of liberal arts computer science programs. *ACM Transactions on Computing Education*, 10(1), 1-30. doi:10.1145/1731041.1731045
- Bälter, O., Enström, E., & Klingenberg, B. (2013). The effect of short formative diagnostic web quizzes with minimal feedback. *Computers & Education*, 60, 234-242.
- Barberà, E., Layne, L., & Gunawardena, C. N. (2014). Designing online interaction to address disciplinary competencies: A cross-country comparison of faculty perspectives. *The International Review of Research in Open and Distributed Learning*, 15(2), 142-169.
- Barker, L. J., McDowell, C., & Kalahar, K. (2009). Exploring factors that influence computer science introductory course students to persist in the major. *ACM SIGCSE Bulletin*, 41(1), 153-157.
- Barr, V., & Stephenson, C. (2011). Bringing computational thinking to K-12: What is involved and what is the role of the computer science education community? *ACM Inroads*, 2(1), 48-54.

- Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., ... Thomas, D. (2001). *Manifesto for Agile Software development*. Retrieved from <http://www.agilemanifesto.org/iso/en/>
- Beech, B. (1999). Go the extra mile – Use the Delphi technique. *Journal of Nursing Management*, 7(5), 281-288.
- Ben Arfa Rabai, L., Bai, Y. Z., & Mili, A. (2011). A quantitative model for software engineering trends. *Information Sciences*, 181(22), 4993-5009.
- Ben Arfa Rabai, L., Cohen, B., & Mili, A. (2015). Programming language use in US academia and industry. *Informatics in Education*, 14(2), 143-160.
- Bernhard, M. P. (2014, September 11). CS50 logs record-breaking enrollment numbers. *The Harvard Crimson*. Retrieved from <http://www.thecrimson.com/article/2014/9/11/cs50-breaks-enrollment-records/?page=single>
- Bishop-Clark, C., Courte, J., Evans, D., & Howard, E. V. (2007). A quantitative and qualitative investigation of using Alice Programming to improve confidence, enjoyment and achievement among non-majors. *Journal of Educational Computing Research*, 37(2), 193-207.
- Black Duck Software. (2015). This year's language use. Retrieved from <https://www.blackducksoftware.com/resources/data/this-years-language-use>
- Boone, Jr. H. N., & Boone, D. A. (2012). Analyzing Likert data. *Journal of Extension*, 50(2), 1-5. Retrieved from <http://www.joe.org/joe/2012april/tt2.php>
- Bothe, K., Budimac, Z., Cortazar, R., Ivanović, M., & Zedan, H. (2009). Development of a modern curriculum in software engineering at master level across Countries. *Computer Science & Information Systems*, 6(1), 1-21.

- Brace-Govan, J., Farrelly, F., Joy, S., Luxton, S., & Davey, I. (2001). Delphi revisited: A concise method for industry consultation on curriculum. *Australian and New Zealand Journal of Vocational Education Research*, 9(1), 1-19.
- Bramwell, A., & Wolfe, D. A. (2008). Universities and regional economic development: The entrepreneurial University of Waterloo. *Research Policy*, 37(8), 1175-1187.
- Bransford, J., Brown, A., & Cocking, R. (1999). How people learn: Brain, mind experience and school. Washington, DC: National Research Council. Retrieved from <http://www.nap.edu/catalog/6160/how-people-learn-brain-mind-experience-and-school>
- Brookshear, G. (1997). *Computer science: An overview*. (5th ed.). Boston, MA: Addison-Wesley.
- Brookshear, G. (2003). *Computer science: An overview*. (7th ed.). Boston, MA: Addison-Wesley.
- Brown, A., & Green, A. (2011). *The essentials of instructional design; connecting fundamental principles with process and practice*. (2nd ed.). Boston, MA: Pearson.
- Bruce, K. B., Cupper, R. D., & Drysdale, R. L. S. (2010). A history of the liberal arts computer science consortium and its model curricula. *ACM Transactions on Computing Education*, 10(1), 1-12.
- Bruce, R., Fowler, C., Guzdial, M., King, M. S., & Woszczyński, A. (2005). CS0/CS1 - Filter or funnel: Recruitment, retention, and student success. In *Proceedings of the 43rd Annual Southeast Regional Conference*, Kennesaw, GA (pp. 29-30). New York, NY: ACM. doi:10.1145/1167350.1167370
- Brungs, A., & Jamieson, R. (2010). Identification of legal issues for computer forensics. *Journal of Digital Forensic Practice*, 3(2-4), 140-149.

- Bureau of Labor Statistics, U.S. Department of Labor. (2015). *Occupational outlook handbook, 2014-15 edition*. Retrieved from <http://www.bls.gov/ooh/computer-and-information-technology/home.htm>
- California State University. (2015a). In the spotlight. Retrieved from <http://www.calstate.edu/>
- California State University. (2015b). CSU student enrollment in degree programs, analytic studies. Retrieved from [http://www.calstate.edu/as/stat\\_reports/degree.shtml](http://www.calstate.edu/as/stat_reports/degree.shtml)
- Carbonelle, P. (2015). PYPL PopularitY of programming language. Retrieved from <http://pypl.github.io/PYPL.html>
- Cardona, S., Vélez, J., Tobón, S. (2014). Towards a model for the development and assessment of competences through formative projects. *CLEI Electronic Journal*, 17(3), 1-16.
- Case Western Reserve University. (2015). History. Retrieved from <https://engineering.case.edu/eecs/about/history>
- Chand, D. R. (1974). Computer science education in business schools. *SIGCSE Bulletin*, 6(3), 91-97.
- Chang, C-K. (2014). Effects of using Alice and Scratch in an introductory programming course for corrective instruction. *Journal of Educational Computing Research*, 51(2), 185-204.
- Cheng, T. K., Jayasuriya, M., & Lim, J. (2010). Removing the fear factor in programming. *The Python Papers Monograph*, 2, 1-9.
- Clark, C. B., Courte, J., Evans, D., & Howard, E. V. (2007). A quantitative and qualitative investigation of using Alice Programming to improve confidence, enjoyment and achievement among non-majors. *Journal of Educational Computing Research*, 37(2), 193-207.



- Cohoon, J. P., Cohoon, M., & Soffa, M. L. (2013). Educating diverse computing science students at the University of Virginia. *Computer*, 46(3), 52-55.
- Colson, N. (2015). The two-to-four year plan. *Business NH Magazine*, 32(8), 36-38.
- Committee for the Workshops on Computational Thinking & Computer Science and Telecommunications Board. (2010). *Report of a workshop on the scope and nature of computational thinking*. Washington, D.C.: The National Academies Press. Retrieved from [http://www.nap.edu/download.php?record\\_id=12840](http://www.nap.edu/download.php?record_id=12840)
- Committee on Uses of Computers. (1966). *Digital computer needs in universities and colleges*. Washington, D.C.: National Academy of Sciences, National Research Council.
- Computing Research Association. (2015). *The Taulbee survey*. Retrieved from <http://cra.org/resources/taulbee-survey/>
- Cooper, S. (2010). The design of Alice. *ACM Transactions on Computing Education*, 10(4), 1-16. doi:10.1145/1868358.1868362
- Cortina, T. J. (2007). An introduction to computer science for non-majors using principles of computation. *ACM SIGCSE Bulletin*, 39(1) 218-222.
- Courte, J., & Howard, E. V. (2005). Using Alice in a computer science survey course. *ACM SIGCSE Bulletin*, 39(1), 213-217.
- Crawley, E. F., Malmqvist, J., Östlund, S., Brodeur, D. R., & Eström, K. (2014). *Rethinking engineering education: The CDIO approach* (2nd ed.). New York, NY: Springer.
- CSin3. (n.d.). Retrieved from <https://sites.google.com/site/csitin3/>
- (CT)<sup>2</sup> basic information. (n.d.). OhioHigherEd. Retrieved from <https://www.ohiohighered.org/transfer/ct2/basicinfo>

- Czerkawski, B., & Lyman, E. (2015). Exploring issues about computational thinking in higher education. *Tech Trends: Linking Research & Practice to Improve Learning*, 59(2), 57-65.
- Dale, N., & Lewis, J. (2016). *Computer science illuminated* (6th ed.). Burlington, MA: Jones & Bartlett Learning.
- Dalkey, N., & Helmer, O. (1963). An experimental application of the Delphi method to the use of experts. *Management Science*, 9(3), 458-467.
- Daly, T. (2011). Minimizing to maximize: An initial attempt at teaching introductory programming using Alice. *Journal of Computing Sciences in Colleges*, 26(5), 23-30.
- Davies, S., Polack-Wahl, J. A., & Anewalt, K. (2011). A snapshot of current practices in teaching the introductory programming sequence. In *Proceedings of the 42nd ACM Technical Symposium on Computer Science Education* Dallas, TX (pp. 625-630). New York, NY: ACM. doi:10.1145/1953163.1953339
- Denning, P. J. (2013). The science in computer science. *Communications of the ACM*, 56(5), 35-38.
- Denning, P. J., & Gordon, E. E. (2015). A technician shortage. *Communications of the ACM*, 58(3), 28-30.
- desJardins, M., & Littman, M. (2010). Broadening student enthusiasm for computer science with a great insights course. In *Proceedings of the 41st ACM technical symposium on computer science education* Milwaukee, WI (pp. 157-161). New York, NY: ACM. doi:10.1145/ 1734263.1734317
- Diakopoulos, N., & Cass, S. (2015, July 20). Interactive: The top programming languages 2015. Retrieved from <http://spectrum.ieee.org/static/interactive-the-top-programming-languages-2015>

- Dodds, Z., Libeskind-Hadas, R., Alvarado, C., & Kuenning, G. (2008). Evaluating a breadth-first CS1 for scientists. *ACM SIGCSE Bulletin*, 40(1), 266-270
- Dopplick, R. (2015, December 7). Celebrate Computer Science Education Week, December 7-13. [Web log comment]. Retrieved from <https://techpolicy.acm.org/?p=5379>
- Dorn, B. (2011). Education reaching learners beyond our hallowed halls. *Communications of the ACM*, 54(5), 28-30.
- Dye, E. (2014, September 22). 2014 year-to-date trends in higher education marketing. Retrieved from <http://sparkroom.com/blog/tag/marketing-analytics/>
- Dziallas, S., & Fincher, S. (2015). ACM curriculum reports: A pedagogic perspective. In *Proceedings of the 11th Annual International Conference on International Computing Education Research*, Omaha, NE (pp. 81-89). New York, NY: ACM. doi:10.1145/2787622.2787714
- Elarde, J. V., & Fatt-Fei, C. (2011). Introductory computing course content: Educator and student perspectives. In *Proceedings of the 2011 Conference on Information Technology Education* West Point, NY (pp. 55-60). New York, NY: ACM. doi:10.1145/2047594.2047610
- Elena, B., Layne, L., & Gunawardena, C. N. (2014). Designing online interaction to address disciplinary competencies: A cross-country comparison of faculty perspectives. *International Review of Research in Open & Distance Learning*, 15(2), 142-169.
- Elledge, R. O. C., & McAleer, S. (2015). Planning the content of a brief educational course in maxillofacial emergencies for staff in accident and emergency departments: A modified Delphi study. *British Journal of Oral & Maxillofacial Surgery*, 53(2), 109-113.

- Enbody, R., J., Puch, W. F., McCullen, M. (2009). Python CS1 as preparation for C++ CS2. *ACM SIGCSE Bulletin*, 41(1), 116-120.
- Eskandari, H., Sala-Daikanda, S., & Ruterer, S. (2007). Enhancing the undergraduate industrial engineering curriculum: Defining desired characteristics and emerging topics. *Education & Training*, 49(1), 45-55.
- Fallows, J. (2015, March 14). California's centers of technology: Bay Area, L.A., San Diego, and ... Fresno? *The Atlantic*. Retrieved from <http://www.theatlantic.com/national/archive/2015/03/californias-centers-of-technology-bay-area-la-san-diego-and-fresno/387808/>
- Field, A. (2009). *Discovering statistics using SPSS* (3rd ed.). Thousand Oaks, CA: Sage Publications Inc.
- Fisher, L. M. (2016). A decade of ACM efforts contribute to computer science for all. *Communications of the ACM*, 59(4), 25-27.
- Forte, A., & Guzdial, M. (2004). Computers for communication, not calculation: Media as a motivation and context for learning. In *Proceedings from 37th Hawaiian International Conference of Systems Sciences*, Big Island, HI (pp. 1-10). Retrieved from <http://www.andreaforte.net/ForteGuzdialCommNotCalc.pdf>
- Forte, A., & Guzdial, M., (2005). Motivation and nonmajors in computer science: Identifying discrete audiences for introductory courses. *IEEE Transactions on Education*, 48(2), 248-253.
- Foster, P. (1997). Lessons from history: Industrial arts/technology education as a case. *Journal of Vocational and Technical Education*, 13(2), 5-15.

- Franklin, D. (2015). Putting the computer science in computing education research. *Communications of the ACM*, 58(2), 34-36.
- Fulton, S., & Schweitzer, D. (2011). Impact of giving students a choice of homework assignments in an introductory computer science class. *International Journal for the Scholarship of Teaching and Learning*, 5(1), 1-12.
- Gal-Ezer, J., Vilner, T., & Zur, E. (2003). Teaching algorithm efficiency in a CS1 course: A different approach. In *Proceedings of the 8th Annual Conference on Innovation and Technology in Computer Science Education* Thessaloniki, Greece (p. 256). New York, NY: ACM. doi:10.1145/961511.961616
- Gerwing, T. G., Rash, J. A., Gerwing, A. M. A., Bramble, B., & Landine, J. (2015). Perceptions and incidence of test anxiety. *Canadian Journal for the Scholarship of Teaching & Learning*, 6(3), 1-14. Retrieved from [http://ir.ib.uwo.ca/cjsotl\\_rcacea/vol6/iss3/3](http://ir.ib.uwo.ca/cjsotl_rcacea/vol6/iss3/3)
- Gibbs, N. E., & Tucker, A. B. (1986). A model curriculum for a liberal arts degree in computer science. *Communications of the ACM*, 29(3), 202-210.
- Goldman, K., Gross, P., Heeren, C., Herman, G., Kaczmarczyk, L., Loui, M. C., & Zilles, C. (2008). Identifying important and difficult concepts in introductory computing courses using a Delphi process. *ACM SIGCSE Bulletin*, 40(1), 256-260.
- Goodman, C. M. (1987). The Delphi technique: A critique. *Journal of Advanced Nursing*, 12(6), 729-734.
- Gorn, S. (1963). The computer and information sciences: A new basic discipline. *SIAM Review*, 5(2), 150-155.
- Grandgenett, N., Thiele, L., Pensabene, T., & McPeak, B. (2015). It takes a village to raise an information technology project: Suggestions on collaboration from our 10-community-

- college consortium. *Community College Journal of Research and Practice*, 39(7), 647-658.
- Gray, K. C., & Herr, E. L. (1998). *Workforce education*. Needham Heights, MA: Allyn & Bacon.
- Guercio, A., & Sharif, B. (2012). Being Agile in computer science classrooms. *AURCO Journal*, 18, 41-62.
- Guo, P. (2014, July 7). Python is now the most popular introductory teaching language at top U.S. universities. [Web log comment]. Retrieved from <http://cacm.acm.org/blogs/blog-cacm/176450-python-is-now-the-most-popular-introductory-teaching-language-at-top-us-universities/fulltext>
- Gupta, G. K. (2007). Computer science curriculum developments in the 1960s. *IEEE Annals of the History of Computing*, 29(2), 40-64.
- Guu, Y. H., Lin, K-Y., & Lee, L-S. (2014). Identifying professional competencies of the flip-chip packaging engineer in Taiwan. *Turkish Online Journal of Educational Technology*, 13(4), 61-70.
- Guzdial, M. (2009). Teaching computing to everyone. *Communications of the ACM*, 52(5), 31-33.
- Hambruch, S., Libeskind-Hadas, R., & Aaron, E. (2015). Understanding the U.S. domestic computer science Ph.D. pipeline. *Communications of the ACM*, 58(8), 29-32.
- Harding, A., & Engelbrecht, J. (2015). Personal learning network clusters: A comparison between mathematics and computer science students. *Educational Technology & Society*, 18(3), 173-184.
- Harmon, O. R., Alpert, W. T., & Lambrinos, J. (2014). Testing the effect of hybrid lecture delivery on learning outcomes. *Journal of Online Learning & Teaching*, 10(1), 112-121.

- Hartell, R. W., & Foegeding, E. A. (2006). Learning: Objectives, competencies, or outcomes? *Journal of Food Science Education*, 3(4), 69-70.
- Hasson, F., Keeney, S., & McKenna, Hugh. (2000). Research guidelines for the Delphi survey technique. *Journal of Advanced Nursing*, 32(4), 1008-1015.
- Hauswirth, M., & Adamoli, A. (2013). Teaching Java programming with the Informa clicker system. *Science of Computer Programming*, 78(5), 499-520.
- Hawi, N. (2010). Causal attributions of success and failure made by undergraduate students in an introductory-level computer programming course. *Computers & Education*, 54, 1127-1136.
- Hays, D.G. & Singh, A.A. (2012) *Qualitative inquiry in clinical and educational settings*. New York, New York: Guilford.
- Herlo, D. (2015). New trends in curriculum design process for higher education. *Journal Plus Education / Educatia Plus*, 13(2), 36-41.
- Hertz, M. (2010). What do “CS1” and “CS2” mean?: Investigating differences in the early courses. In *Proceedings of the 41st ACM Technical Symposium on Computer Science Education* Milwaukee, WI (pp. 199-203). New York, NY: ACM. doi:10.1145/1734263.1734335
- Horton, D., Craig, M., Campbell, J., Gries, P., & Zingaro, D. (2014). Comparing outcomes in inverted and traditional CS1. In *Proceedings of the 2014 Conference on Innovation & Technology in Computer Science Education* Uppsala, Sweden (pp. 261-266). New York, NY: ACM. doi: 10.1145/2591708.2591752

- Huang, T. (2008). Restructuring the introductory computer science course with topics from AI. In *Papers from the AAAI Spring Symposium: Using AI to Motivate Greater Participation in Computer Science* (pp. 100-101). Menlo Park, CA: AAAI Press.
- IEEE Computer Society. (2015). IEEE Computer Society Certification and Credential Program. Retrieved from <http://www.computer.org/web/education/certifications>
- International Organization for Standardization [ISO]/International Electrotechnical Commission [IEC]. (1994). *ISO International Standard ISO/IEC 7498-1 – Information technology – Open systems interconnection – Basic reference model: The basic model*. Geneva, Switzerland: International Organization for Standardization (ISO). Retrieved from <http://www.ecma-international.org/activities/Communications/TG11/s020269e.pdf>
- Israel, D. (2008). *Data analysis in business research: A step-by-step nonparametric approach*. Thousand Oaks, CA: SAGE Publications, Inc.
- Joint Interim Review Task Force. (2008). *Computer science curriculum 2008: An interim revision of CS 2001*. New York: Author. Retrieved from <http://www.acm.org/education/curricula/ComputerScience2008.pdf>
- Joint Task Force on Computing Curricula. (1990). Computing curricula 1991. *Communications of the ACM*, 34(6), 68-84.
- Joint Task Force on Computing Curricula. (2001). *Computing curricula 2001*. New York: Author. Retrieved from <http://www.acm.org/sigcse/cc2001>
- Joint Task Force for Computing Curricula 2005. (2006). *Computing Curricula 2005: The overview report covering undergraduate programs in computer engineering, computer science, information systems, information technology, software engineering*. Los



- Alamitos, CA: Author. Retrieved from  
[http://www.acm.org/education/curric\\_vols/CC2005-March06Final.pdf](http://www.acm.org/education/curric_vols/CC2005-March06Final.pdf)
- Joint Task Force on Computing Curricula. (2013). *Computer science curricula 2013: Curriculum guidelines for undergraduate degree programs in computer science*. New York: Author. Retrieved from <https://www.acm.org/education/CS2013-final-report.pdf>
- Joint Task Force on Computing Curricula. (2014). *Software engineering 2014: Curriculum guidelines for undergraduate degree programs in software engineering*. New York: Author. Retrieved from <https://www.acm.org/education/se2014.pdf>
- Joyner, H. S., & Smith, D. (2015). Using Delphi surveying techniques to gather input from non-academics for development of a modern dairy manufacturing curriculum. *Journal of Food Science Education*, 14(3), 88-115.
- Kaelbling, L., White, J., Abelson, H., Freeman, D. Lozano-Pérez, T., & Chuang, I. (2011). *Introduction to Electrical Engineering and Computer Science I*, (Massachusetts Institute of Technology: MIT OpenCourseWare) [Course Syllabus]. Cambridge, MA: Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology. Retrieved from <http://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-01sc-introduction-to-electrical-engineering-and-computer-science-i-spring-2011/Syllabus/>
- Katai, Z. (2014). The challenge of promoting algorithmic thinking of both sciences- and humanities-oriented learners. *Journal of Computer Assisted Learning*, 31(4), 287-299.
- Keenan, T. A. (1964). Computers and education. *Communications of the ACM*, 7(4), 205-209.

- Kelleher, C. & Pausch, R. (2005). Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers. *ACM Computing Surveys*, 37(2), 83-137.
- Keller, A., & Volkov, A. (2014). Mathematics education in oriental antiquity and Middle Ages. History of mathematical education in ancient, medieval and pre modern India. In *Handbook on the history of mathematics* (pp. 70-83). New York, NY: Springer. Retrieved from <https://halshs.archives-ouvertes.fr/halshs-01006132/document>
- Kelly, D. F. (2007). A software chasm: Software engineering and scientific computing. *IEEE Software*, 24(6), 120-+.
- Kelly, D. (2013). Software engineering and scientific software - Farther apart than ever. *Software Practitioner*, 23(3), 4-6.
- Kenny, N., & Desmaris, S. (2012). *A guide to developing and assessing learning outcomes at the University of Guelph*. Retrieved from <http://www.uoguelph.ca/vpacademic/avpa/pdf/LearningOutcomes.pdf>
- Kinnersley, B. (n.d.). Collected information on about 2500 computer languages, past and present. Retrieved from <http://people.ku.edu/~nkidders/LangList/Extras/langlist.htm>
- Kiss, G. (2013, November 26). Teaching programming in the higher education not for engineering students. *Procedia - Social and Behavioral Sciences*, 103, 922-927. doi:10.1016/j.sbspro.2013.10.414
- Koffman, E. B., & Finerman, A. (2004). Education in computer science. In E.D. Reilly (Ed.), *Concise encyclopedia of computer science* (289-292). Hoboken, NJ: John Wiley & Sons, Inc.

- Kölling, M. (2010). The Greenfoot programming environment. *ACM Transactions on Computing Education*, 10(4), 1-21. doi:10.1145/1868358.1868361
- Koszalka, T. A., Russ-Eft, D. F., Reiser, R., & Senior, F. A. (2013). *Instructional designer competencies: The standards* (4th ed.). Charlotte, NC: Information Age Publishing, Inc.
- Krpan, D., Mladenović, S., & Rosić, M. (2015). Undergraduate programming courses: Students' perception and success. *Procedia - Social and Behavioral Sciences*, 174, 3868-3872. doi:10.1016/j.sbspro.2015.01.1126
- Kunkle, W. M. (2010). *The impact of different teaching approaches and languages on student learning of introductory programming concepts* (Doctoral Dissertation. Retrieved from ProQuest. (3430595)
- LaFrance, J., & Roth, R. W. (1972). Computer science for liberal arts colleges. *ACM SIGCSE Bulletin*, 4(4), 22-31.
- Lan, Y-F., Tsai, P-W., Yang, S-H., & Hung, C-L. (2012). Comparing the social knowledge construction behavioral patterns of problem-based online asynchronous discussion in e/m-learning environments. *Computers & Education*, 59(4), 1122-1135.
- Land, S. K., & Reisman, S. (2012). Software engineering certification in today's environment. *IT Professional*, 14(3), 50-54.
- Law, K. M. Y., Lee, V. C. S., & Yu, Y. T. (2010). Learning motivation in e-learning facilitated computer programming courses. *Computers & Education*, 55(1), 218-228.
- Lazowska, E., Roberts, E., & Kurose, J. (2014, May). *Tsunami or sea change? Responding to the explosion of student interest in computer science* [PowerPoint slides]. Retrieved from <http://lazowska.cs.washington.edu/NCWIT.pdf>

- Leedy, P.D., & Ormrod, J.E. (2014). *Practical research: Planning and design* (10th ed.). Upper Saddle River, NJ: Pearson Education.
- Levin, J. S., Cox, E. M., Cerven, C., & Haberler, Z. (2010). The recipe for promising practices in community colleges. *Community College Review*, 38(1), 31-58.
- Lewis, C., Jackson, M. H., & Waite, W. M. (2010) Student and faculty attitudes and beliefs about computer science. *Communications of the ACM*, 53(5), 78-85.
- Lewis, M. C., Blank, D., Bruce, K., & Osera, P-M. (2016). Uncommon teaching languages. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education* Memphis, TN (pp. 492-493). New York, NY: ACM. doi:10.1145/2839509.2844666
- Liberal Arts Computer Science Consortium (LACS). (2007). A 2007 model curriculum for a liberal arts degree in computer science. *ACM Journal on Educational Resources in Computing*, 7(2), 1-35. doi:10.1145/1240200.1240202
- Liming, D., & Wolf, M. (2008). Job outlook by education, 2006-16. *Occupational Outlook Quarterly*, 52(3), 2-29.
- Linstone, H. A., & Turoff, M. (1975). Introduction. In Linstone, H. A., & Turoff, M. (Eds.), *The Delphi method: Techniques and applications* (3-12). Reading, Massachusetts: Addison-Wesley Publishing Company.
- Linstone, H. A., & Turoff, M. (2011). Delphi: A brief look backward and forward. *Technological Forecasting & Social Change*, 78(9), 1712-1719.
- Lopez, A. A., Raymond, R., & Tardiff, R. (1977). A survey of computer science offerings in small liberal arts colleges, *Communications of the ACM*, 20(2), 902-906.

- Lunt, B., Ekstrom, J., Lawson, E. A., Kamali, R., Miller, J., Gorke, S., & Reichgelt, H. (2005). Defining the IT curriculum: The results of the past 3 years. *Issues in Informing Science and Information Technology Education*, 2, 259-270.
- Lutz, M. J., Naveda, J. F., & Vallino, J. R. (2014). Undergraduate software engineering. *Communications of the ACM*, 57(8), 52-58.
- Mahmoud, Q. H., Dobosiewicz, W., & Swayne, D. (2004). Redesigning introductory computer programming with HTML, JavaScript, and Java. *ACM SIGCSE Bulletin*, 36(1), 120-124.
- Malan, D. J. (2012). *This is CS50* [Course Syllabus]. Cambridge, MA: Computer Science Department, Harvard University. Retrieved from <https://cs50.harvard.edu/docs/syllabus.pdf>
- Malan, D. J., & Leitner, H. H. (2007). Scratch for budding computer scientists. *ACM SIGCSE Bulletin*, 39(1), 223-227.
- Maloney, J., Resnick, M., Rusk, N., Silverman, B., & Eastmond, E. (2010). The Scratch programming language and environment. *ACM Transactions on Computing Education*, 10(4), 1-15. doi:10.1145/1868358.1868363
- Mamelok, J. (2013). Achieving a consensus on educational objectives and assessments for extended specialty training programmes for licensing in general practice. *Education for Primary Care*, 24(4), 258-265.
- Marling, C., & Juedes, D. (2016). CS0 for computer science majors at Ohio University. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education* Memphis, TN (pp. 138-143). New York, NY: ACM.

Matsui Foundation awarding over \$1M in scholarships. (2015, July 27). *The Californian*.

Retrieved from <http://www.thecalifornian.com/story/news/education/2015/07/27/matsui-foundation-awarding-scholarships/30752057/>

May, K. (1980). Historiography: A perspective for computer scientists. In N. Metropolis (Ed.), *A History of Computing in the Twentieth Century* (pp. 11-18). New York, NY: Academic Press.

McDowell, C., Werner, L., Bullock, H. E., & Fernald, J. (2006). Pair programming improves student retention, confidence, and program quality. *Communications of the ACM*, 49(8), 90-95.

McIver, L. (2001). *Syntactic and semantic issues in introductory programming education* (Doctoral dissertation). Retrieved from <http://www.csse.monash.edu.au/~lindap/papers/LindaMcIverThesis.pdf>

McMaster, G., & Zastre, M. (2011). More concepts for teaching introductory programming. In *Proceedings of the 16th Western Canadian Conference on Computing Education* Prince George, BC, Canada (pp. 7-11). New York, NY: ACM. doi:10.1145/1989622.1989625

Meyerovich, L. A., & Rabkin, A. (2013). Empirical analysis of programming language adoption. Paper presented at the 2013 ACM SIGPLAN International Conference on Object Oriented Programming Systems Languages & Applications, Indianapolis, IN. Retrieved from <http://sns.cs.princeton.edu/docs/asr-oopsla13.pdf>

Mitchell, V. W. (1991). The Delphi technique: an exposition and application. *Technology Analysis & Strategic Management*, 3(4), 333–358

Moffitt, C. (2012, March 30). Formal education not always desired for computer programmers. *Business Journal Serving Fresno & the Central San Joaquin Valley*, pp. 8, 9.

- Molnár, P., Toth, D. M., Vincent-Finley, R. E. (2014). Development of undergraduate and graduate programs in computational science. *Concurrency & Computation: Practice & Experience*, 26(13), 2329-2335.
- Moskal, B., Cooper, S., Munson, A., & Dann, W. (2008, June). *The impact of the Alice curriculum on community college students' attitudes and learning with respect to computer science*. Paper presented at the 2008 Annual Conference & Exposition, Pittsburgh, PA. <https://peer.asee.org/3306>
- Moura, I. C., & van Hattum-Janssen, N. (2011). Teaching a CS introductory course: An active approach. *Computers & Education*, 56(2), 475-483.
- Muñoz, M, Martínez, C., Cárdenas, C., & Cepeda, M. (2013). Active learning in first-year engineering courses at Universidad Católica de la Santísima Concepción, Chile. *Australasian Journal of Engineering Education*, 19(1), 27-38.
- Norman, V., & Adams, J. (2015). Improving non-CS major performance in CS1. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*, Kansas City, MO (pp. 558-562). New York, NY: ACM. doi:10.1145/2676723.2677214
- Norton, R. E. (1998). *Quality instruction for the high performance workplace: DACUM*. Retrieved from <http://files.eric.ed.gov/fulltext/ED419155.pdf>
- Okoli, C., & Pawlowski, S. D. (2004). The Delphi method as a research tool: an example, design considerations and applications. *Information & Management*, 42(1), 15-29.
- O\*NET Online. (2015). Summary report for: 15-1121.00 - computer systems analysts, 15-1131.00 - computer programmers, 15-1132.00 - applications software developers, 15-1133.00 - systems software developers, & 15-1141.00 - database administrators. In O\*Net online. Retrieved from <http://www.onetonline.org/link/summary/15-1131.00>

- Ornstein, A., & Hunkins, F. (2013). *Curriculum foundations, principles and theory* (6th ed.). Boston, MA: Allyn and Bacon.
- Papert, S. (1980). *Mindstorms: Children, computers, and powerful ideas*. New York, NY: Basic Books, Inc.
- Parlante, N. (2014). *Computer Science 101* [Course Syllabus]. Stanford, CA: Computer Science Department, Stanford University. Retrieved from <https://web.stanford.edu/class/cs101/syllabus.html>
- Pearson, G. & Young, A. T. (Eds.). (2002). *Technically speaking*. Washington, DC: National Academy Press.
- Perlis, A. J. (1964). Programming of digital computers. *Communications of the ACM*, 7(2), 210-211.
- Porter, L., Guzdial, M., McDowell, C., & Simon, B. (2013). Success in introductory programming: What works? *Communications of the ACM*, 58(8), 34-36.
- Porter, L., & Simon, B. (2013). Fostering creativity in CS1 by hosting a computer science art show. *ACM Inroads*, 4(1), 29-31.
- Poulova, P., & Klimova, B. (2015). Education in computational sciences. *Procedia Computer Science*, 51, 1996-2005.
- Powers, K., Ecott, S., & Hirshfield, L. M. (2007). Through the looking glass: Teaching CS0 with Alice. *ACM SIGCSE Bulletin*, 39(1), 213-217.
- Pratt, M. K. (2015). Technology nourishes the food chain. *Computerworld Digital Magazine*, 2(1), 20-26.



- Rajlich, V. (2013). Teaching developer skills in the first software engineering course. In *Proceedings of the 2013 International Conference on Software Engineering San Francisco, CA* (pp. 1109-1116). Piscataway, NJ: IEEE Press.
- RedMonk. (2015a). The RedMonk programming language rankings: June 2015. Retrieved from <https://redmonk.com/sogrady/category/programming-languages/>
- RedMonk (2015b). Academia and programming language preferences. Retrieved from <http://redmonk.com/sogrady/2013/04/04/academia-and-programming-languages/>
- Reigeluth, C. M. (Ed.). (1999). In *Instructional-design theories and models* (Vol. 2, pp. 1-29). Mahwah, NJ: Lawrence Erlbaum Associates.
- Riabov, V. V. (2013). Tools and methodologies for teaching online computer-science courses in LMS environment. In Y. Kats (Ed.), *Learning management systems and instructional design: Best practices in online education* (144-177). Hershey, PA: IGI Publishing.
- Rizvi, M., Humphries, T., Major, D., Jones, M., & Lauzun, H. (2011). A CS0 course using Scratch. *Journal of Computing Sciences in Colleges*, 26(3), 19-27.
- Roach, S., & Sahami, M. (2015). CS2013: Computer science curricula 2013. *Computer*, 48(3), 114-116.
- Roberts, E. S. (2011). Meeting the challenges of rising enrollments. *ACM Inroads*, 2(3), 4-6.
- Rochester Institute of Technology. (2004). College of computing & information sciences timeline. *RIT History*. Retrieved from <https://www2.rit.edu/175/timelineGCCIS.html>
- Rolka, C., & Remshagen, A. (2015). Showing up is half the battle: Assessing different contextualized learning tools to increase the performance in introductory computer science courses. *International Journal for the Scholarship of Teaching & Learning*, 9(1), 1-18.

Romero, E. D. (2014). Not just ag: Growing tech in the Central Valley. *The California Report*.

Retrieved from <http://audio.californiareport.org/archive/R201408221630/d>

Rotondi, A., & Gustafson, D. (1996). Theoretical, methodological and practical issues arising out of the Delphi method. In Adler, M., & Ziglio, E. (Eds.), *Gazing into the Oracle: The Delphi Method and Its Application to Social Policy and Public Health* (34-55). Bristol, UK: Jessica Kingsley Publishers, Ltd.

Rubio, M. A., Romero-Zaliz, R., Mañoso, C., de Madrid, A. P. (2015). Closing the gender gap in an introductory programming course. *Computers & Education*, 82, 409-420.

Sami, S. (2007). What subjects and skills are important for software developers?

*Communications of the ACM*, 50(1), 73-78.

Sander, L. (2008). For work-force training, a plan to give college credit where it's due. *Chronicle of Higher Education*. 54(39), A22-A23.

Scheibe, M., Skutsch, M., & Schofer, J. (1975). Experiments in Delphi methodology. In Linstone, H. A., & Turoff, M. (Eds.), *The Delphi method: Techniques and applications* (257-281). Reading, Massachusetts: Addison-Wesley Publishing Company.

Schmidt, R. C. (1997). Managing Delphi surveys using nonparametric statistical techniques, *Decision Sciences*, 28(3), 763-744.

Schneider, G. M. (2004). A model for a three course introductory sequence. *ACM SIGCSE Bulletin*, 36(2), 40-43.

Schneider, G. M., & Gersting, J. L. (2016). *Invitation to computer science* (7th ed.). Boston, MA: Cengage Learning.

- Schulte, C., & Bennedsen, J. (2006). What do teachers teach in introductory programming? In *Proceedings of the Second International Workshop on Computing Education Research* Canterbury, UK (pp. 17-28). New York, NY: ACM Press.
- Scott, T. (2003). Bloom's taxonomy applied to testing in computer science classes. *Journal of Computing Sciences in Colleges*, 19(1), 267-274.
- Settle, A., Lalor, J., & Steinbach, T. (2015, March 4-7). Reconsidering the impact of CS1 on novice attitudes. Paper presented at the 46th ACM Technical Symposium on Computer Science Education, Kansas City, MO (229-234). New York, NY: ACM.
- Shaw, A. (2010). Modifying computer programming education courses to support Web 2.0 and social computing paradigms. *Journal of Information Systems Technology & Planning*, 3(6), 54-60.
- Sheehan, T. (2014, June 26). Tech growth spurs Bitwise Industries expansion in downtown. *The Fresno Bee*. Retrieved from <http://www.fresnobee.com/2014/06/25/3996542/tech-growth-spurs-bitwise-industries.html>
- Shein, E. (2014). Should everybody learn to code? *Communications of the ACM*, 57(2), 16-18.
- Shein, E. (2015). Python for beginners. *Communications of the ACM*, 58(3), 19-21.
- Shell, D. F., & Soh, L. (2013). Profiles of motivated self-regulation in college computer science courses: Differences in major versus required non-major courses. *Journal of Science Education & Technology*, 22(6), 899-913.
- Shulman, L. S. (1986). Those who understand: Knowledge growth in teaching. *Educational Researcher*, 15(2), 4-14.
- Sheehan, T. (2014, June 26). Tech growth spurs Bitwise Industries expansion in downtown. *The Fresno Bee*.

- Siegle, D. (2010, November 24). Importance. *Likert Scale*. Retrieved from <http://www.gifted.uconn.edu/siegle/research/instrument%20reliability%20and%20validity/likert.html>
- Simon, B., Hanks, B., McCauley, R., Morrison, B., Murphy, L., & Zander, C. (2009). For me, programming is.... In *Proceedings of the 5th International Computing Education Research Workshop* Berkeley, CA (pp. 105-116). New York, NY: ACM. doi:10.1145/1584322.1584335
- Singleton, D. M. (2013). Transitioning to blended learning: The importance of communication and culture. *Journal of Applied Learning Technology*, 3(1), 12-15.
- Sitlington, H. (2015). Using the Delphi technique to support curriculum development. *Education + Training*, 57(3), 306-321.
- Skulmoski, G. J., Hartman, F. T., & Krahn, J. (2007). The Delphi method for graduate research. *Journal of Information Technology Education*, 6, 1-21.
- Sonnier, D. L. (2013). Computer science in a liberal arts school: Convincing the skeptic. *Journal of Computing Sciences in Colleges*, 28(5), 115-121.
- Soper, T. (2014, June 6). Analysis: The exploding demand for computer science education, and why America needs to keep up. [Web log comment]. Retrieved from: <http://www.geekwire.com/2014/analysis-examining-computer-science-education-explosion/>
- Sprinthall, R. C. (2012). *Basic statistical analysis*. Boston, MA: Allyn & Bacon.
- Stamey, J., & Sheel, S. (2010). A boot camp approach to learning programming in a CS0 course. *Journal of Computing Sciences in Colleges*, 25(5), 34-40.

- Starr, C. W., Manaris, B., & Stalvey, R. H. (2008). Bloom's Taxonomy Revisited: Specifying Assessable Learning Objectives in Computer Science. *ACM SIGCSE Bulletin*, 40(1), 261-265.
- State of California Employment Development Department. (2015). *Employment projections*. Retrieved from <http://www.labormarketinfo.edd.ca.gov/data/employment-projections.html#Long>
- Stefik, A., & Gellenbeck, E. (2011). Empirical studies on programming language stimuli. *Software Quality Journal*, 19(1), 65-99.
- Strauss, H. J., & Zeigler, L. H. (1975). Delphi technique and its uses in social science research. *Journal of Creative Behavior*, 9(4), 253-259.
- Sultana, S. G. (2015). *Computer science education needs in Fresno*. Unpublished manuscript.
- Surakka, S. (2005). *Needs assessment of software systems graduates* (Doctoral dissertation). Retrieved from <http://lib.tkk.fi/Diss/2005/isbn9512279517/>
- Surakka, S. (2007). What subjects and skills are important for software developers? *Communications of the ACM*, 50(1), 73-78.
- Symonds, W. C., Schwartz, R.B., Ferguson, R. F. (2011). *Pathways to prosperity: meeting the challenge of preparing young Americans for the 21st Century*. Cambridge, MA: Pathways to Prosperity Project, Harvard Graduate School of Education.
- Syslo, M. M. (2015). From algorithmic to computational thinking: On the way for computing for all students. In *Proceedings of the 2015 ACM Conference on Innovation and Technology in Computer Science Education* Vilnius, Lithuania (p. 1). New York, NY: ACM. doi:10.1145/2729094.2742582

- Tan, G., & Venables, A. (2010). Designing a network and systems computing curriculum: The stakeholders and the issues. *Journal of Information Technology*, 9, 103-112.
- Tanrikulu, E., & Schaefer, B. C. (2011). The users who touched the ceiling of scratch. *Procedia - Social and Behavioral Sciences*, 28, 764-769.
- Tasneem, S. (2012). Critical thinking in an introductory programming course. *Journal of Computing Sciences in Colleges*, 27(6), 81-83.
- Teague, D., & Roe, P. (2007). Learning to program: Going pair-shaped. *Innovations in Teaching and Learning in Information and Computer Sciences*, 6(4), 4-22.
- Tew, A. E. (2010). *Assessing fundamental introductory computing concept knowledge in a language independent manner* (Doctoral Dissertation). Retrieved from ProQuest. (3451304)
- The Editors at JIST (Ed.). (2010). *Young person's occupational outlook handbook* (7th ed.). Indianapolis, IN: JIST Publishing.
- The United States Bureau of Labor Statistics. (2015). *Quarterly census of employment and wages*. Retrieved from [http://www.bls.gov/cew/apps/data\\_views/data\\_views.htm#tab=Tables](http://www.bls.gov/cew/apps/data_views/data_views.htm#tab=Tables)
- TIOBE Index for December 2015. (2015). Retrieved from <http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html>
- Trendy Skills. (2015). About Trendy Skills. Retrieved from <http://trendyskills.com/about>
- Turner, A. J. (1991). Report of the joint ACM/IEEE-CS curriculum task force. In *Proceedings of the 19th Annual Conference on Computer Science* San Antonio, TX (p. 707). New York, NY: ACM. doi:10.1145/327164.328873

University of Cambridge. (2004, August 11). The history of the computer lab. Retrieved from

<http://www.cl.cam.ac.uk/relics/history.html>

Urness, T., & Manley, E. (2011). Building a thriving CS program at a small liberal arts college.

*Journal of Computing Sciences in Colleges*, 26(5), 268-274.

U.S. News & World Report. (2015). University Directory. Retrieved from

<http://www.usnewsuniversitydirectory.com/>

van der Spuy, R. (2012). *Foundation game design with HTML5 and JavaScript*. New York, NY:

Springer.

Voskoglou, M. G., & Buckley, S. (2012). Problem solving and computers in a learning

environment. *Egyptian Computer Science Journal*, 36(4), 28-46.

Vitkutė-Adžgauskienė, D. & Vidžiūnas, A. (2012). Problems in choosing tools and methods for

teaching programming. *Informatics in Education*, 11(2), 271-282.

Walker, H. M., & Kelemen, C. (2010). Computer science and the liberal arts: A philosophical

examination. *ACM Transactions on Computing Education*, 10(1), 1-10. doi:10.1145

/1731041.1731043

Walker, H. M., & Schneider, G. M. (1996). A revised model curriculum for a liberal arts degree

in computer science. *Communications of the ACM*, 39(12), 85-95.

Wang, T., Su, X. Ma, P., Wang, Y., & Wang, K. (2011). Ability-training-oriented automated

assessment in introductory programming. *Serious Games, Computers & Education*,

56(1), 220-226.

Watson, C., & Li, F. W. B. (2014). Failure rates in introductory programming revisited. In

*Proceedings of the 2014 Conference on Innovation & Technology in Computer Science*

- Education* Uppsala, Sweden (pp. 39-44). New York, NY: ACM. doi:10.1145/2591708.2591749
- Whitfield, D. (2003). From university wide outcomes to course embedded assessment of CS1. *Journal of Computing Sciences in Colleges*, 18(5), 210-220.
- Wilhelm, W. J. (2001). Alchemy of the oracle: The Delphi technique. *Delta Pi Epsilon Journal*, 43(1), 6-26.
- Winberg, S. (2014). 'Responsiveness' and 'responsibility': Determining what matters in a computer engineering curriculum. *South African Journal of Higher Education*, 28(3), 983-102.
- Wing, J. (2006). Computational thinking. *Communications of the ACM*, 49(3), 33-35.
- Winter, V. (2014). Bricklayer: An authentic introduction to the functional programming language SML. In J. Caldwell, P. Hölzenspies, & P. Achten (Eds.), *Electronic Proceedings in Theoretical Computer Science at 3rd International Workshop on Trends in Functional Programming in Education*, Soesterberg, the Netherlands (pp. 33-49). doi:10.4204/EPTCS.170.3
- Wu, H-T., Hsu, P-C., Lee, C-Y., Wang, H-J., & Sun, C-K. (2014). The impact of supplementary hands-on practice on learning in introductory computer science course for freshmen. *Computers & Education*, 701-8.
- Xiang, J., & Ye, L. (2009). A general software framework based on reform in formative assessment. *Journal of Software*, 4(10), 1076-1083.
- Zaiontz, C. (2013). Real statistic using Excel. Retrieved from <http://www.real-statistics.com/kendalls-w/>



Zhao, Y. (n.d.). What knowledge has the most worth? Retrieved from <http://www.aasa.org/SchoolAdministratorArticle.aspx?id=6032>

Zhao, L., Su, X., & Wang, T. (2015). Bring CS2013 recommendations into a C programming course. *Procedia - Social and Behavioral Sciences*, 176, 194-199.

Zur, E., Vilner, T., & Shay, T. (2014). Integrating Greenfoot into CS1: A case study. *Mathematics & Computer Education*, 48(1), 29-42.

**APPENDIX A**  
**INVITATION TO PARTICIPANTS**

March 14, 2016

Dear \_\_\_\_\_:

You have been identified as a subject matter expert in computer science, software engineering, or a related information technology field. You meet the criterion of having at least five years of experience working as a computing professional or teaching in the field with at least a master's degree. Your participation is requested in a three to four round Delphi study that will focus on identifying the competencies, programming languages, and assessments to be used for an introductory course in computer science that will be offered as part of a software engineering program, minor, and as an elective at Fresno Pacific University. You will be asked to select from suggested items for these three fields or rate them on a Likert-type scale. All involvement will take place online via the SurveyMonkey web site.

Your participation in this process will help to shape an introductory course that will equip students with an understanding of the fields of computer science and software engineering and with an elementary ability to develop basic code. Though your input would be a valuable contribution to this process, your participation in this study will be totally voluntary throughout the duration of the study. Should you choose to be involved in this study for any length of time, your identification will remain completely confidential.

If you choose to agree to participate, you will receive a survey that will include basic questions on your professional background. You will be asked to return the survey within one week. You will be reminded throughout this process that your involvement is completely voluntary and that you can feel free to depart the study at any time. You will receive no direct benefit by participating. If you decide to agree to contribute your time and input to this study, please respond by March 27, 2016.

Please feel free to forward this message to anyone who might be interested.

Thank you for your consideration,

Simon Sultana  
Ph.D. Candidate, Education  
  
Old Dominion University

Philip Reed  
Associate Professor, STEM Education &  
Professional Studies  
  
Old Dominion University

## **APPENDIX B**

### **HUMAN SUBJECTS INFORMED CONSENT**

Researcher: Simon Sultana

Study: Local experts' recommendations for competencies, content, and delivery in an introduction to computer science course

I am asking for your voluntary participation in this research study being conducted as part of a dissertation. Please reference the information concerning this study below. If you agree to participate, please sign your name below.

Purpose of the study: The purpose of this study is to define the competencies, programming languages, and assessments for an introductory computer science course at a small private nonprofit university that seeks to address the industrial needs of California's Central Valley.

Your involvement: You will be asked to participate in three to four rounds of a Delphi study by completing surveys, which will be made available over the internet. The Round 1 survey will ask you for demographic information, including age, current employment, years of experience, highest education earned in computer science or a related field, and the number of programming languages in which you are fluent. Round 1 questions will ask participants to rate the applicability of course competencies, programming languages, and assessment methods on a five-point Likert scale. You will also be able to submit your own suggestions for items you deem worthy of inclusion. In Round 2 you will be provided ranked lists of each item analyzed with their median scores from Round 1. You will be asked to select at least ten competencies, programming languages, and assessment methods provided. In Round 3 you will be asked to rank an updated list of items for each of the three categories on a five-point Likert scale. If agreement is reached by the panel, the study will end. If not, one more round, identical to Round 3 will be conducted.

Potential risks to you: There are no identifiable risks to you as a result of your participation.

Benefits: There are no foreseeable benefits to you as a result of your participation in this study.

Confidentiality procedures: Your name will not be recorded in the research data but will be replaced by an untraceable identifier which will be tied only to your campus location. All data will be kept in a password-protected database existing only on the researcher's laptop. This computer will also be password-protected.

Participation and withdrawal: Your participation in this study is completely voluntary. If you choose to participate in this study, you may withdraw at any time by notifying the researcher.

If at any time you feel pressured to participate, or if you have any questions about your rights or this form, then you should contact Dr. Edwin Gomez, Chair of the Darden College of Education Human Subjects Review Committee, Old Dominion University, at [egomez@odu.edu](mailto:egomez@odu.edu).

If you have any questions, please feel free to contact:

Simon Sultana  
Principal Researcher  
(559) 453-5501  
Simon.sultana@fresno.edu

Dr. Philip Reed  
Responsible Project Investigator  
(757) 683-4307  
preed@odu.edu

Consent: I completely understand all the information presented about my voluntary participation in this study and agree to my involvement.

\_\_\_\_\_ Subject printed name

\_\_\_\_\_ Subject signature/date

Investigator's Statement: I certify that I have explained to this participant the nature and purpose of this research, including benefits, risks, costs, and any experimental procedures. I have described the rights and protections afforded to human subjects and have done nothing to pressure, coerce, or falsely entice this subject into participating. I am aware of my obligations under state and federal laws, and promise compliance. I have answered the participant's questions and have encouraged him/her to ask additional questions at any time during the course of this study. I have witnessed the above signature(s) on this consent form.

\_\_\_\_\_ Researcher printed name

\_\_\_\_\_ Researcher signature/date

## APPENDIX C

### SUMMARY OF THE STUDY

March 30, 2016

Dear \_\_\_\_\_:

Thank you for agreeing to share your insight in this study on identifying the competencies, programming languages, and assessment methods recommended for an introductory computer science course. We are appreciative of your time and efforts and feel that your input will be most valuable in this endeavor. If you know of others who might also qualify as subject matter experts in software development and have at least five years of experience as a professional in industry or academics, please consider sending contact information to the researcher at [simon.sultana@fresno.edu](mailto:simon.sultana@fresno.edu).

#### Overview of the Study

#### EXPERTS' RECOMMENDATIONS FOR COMPETENCIES, CONTENT, AND DELIVERY IN AN INTRODUCTION TO COMPUTER SCIENCE COURSE

**Purpose:** Students who enroll in an introduction to computer science course do so for various reasons. First, students may be enrolled in a bachelor's level software engineering program that requires the course as an introduction to computer science and software development. Second, students may be pursuing a minor in software engineering and the course is among those required. Finally, students from other majors may want to take the course as a general elective to learn about the computing field and obtain an elementary ability to write computer programs.

The competencies for a course are general objectives detailing the desired content and abilities students are expected to master as a result of learning (Koszalka, Russ-Eft, & Reiser, 2013). Competencies help to identify the focus of the curriculum content. This particular course is intended to provide a survey of the areas within computer science and give students to develop an entry level ability to program a computer. There are several possible areas of focus for this type of course and various different programming languages might be considered. Numerous assessments and experiences are also available. Assessments include those that are formative in nature, whereby students and instructors can gauge their learning during a course; or those that are summative, which provide an overall appraisal of learning (Ornstein & Hunkins, 2013).

**Instructions:** This research will consist of three to four rounds of a Delphi study. You will access each round via the SurveyMonkey web site. The link for the Round 1 questionnaire will be made available to you by email. You will first be asked to provide some demographic information about yourself. You will not be asked for your name. This information is being collected to summarize the background and expertise of the professionals in this study. Remember that all identifiable information collected will be kept strictly confidential.

The second part of the Round 1 survey will present three lists of potential course competencies, programming languages, and assessment experiences for an introduction to computer science course. You will be asked to rate the applicability of each of these items in each of the three

categories on a five-point Likert scale. You will also be given the opportunity to submit your own items for any of the three categories, in the event that you feel an important topic is missing. Insert any additional suggestions in the blank fields provided. If you would like to submit more than three, please send an email message to [ssult004@odu.edu](mailto:ssult004@odu.edu). You will also be asked to rate the applicability of any new entries using the Likert scale and to explain your entry so your suggestion can be described to other participants in this study.

When you have completed the study, please submit your responses. The SurveyMonkey resource will keep your responses anonymous. Once all questionnaires have been submitted by the study's participants, the responses will be aggregated and the researcher will consult with a research subject matter expert for verification of the results. Once there is agreement on the results and new entries provided by participants, the Round 2 survey will be created and distributed in the same way as in Round 1. In Round 2 you will be asked to select at least ten items from each category (competencies, programming languages, and assessments). Items that are selected by at least half of the group you are in (academic or industry) will be retained for Round 3. In the third round you will be asked to rank the items in each category from most important to least important. Each group's rankings will be checked for consensus for each category (competencies, programming languages, and assessments). Categories that reach consensus will be deemed complete whereas those that are not in agreement in Round 3 will be carried over into a final Round 4.

Thank you again for agreeing to participate in this study. Your time and input are very much valued. Should you have any issues or concerns, please feel free to contact Simon Sultana at [ssult004@odu.edu](mailto:ssult004@odu.edu) or by cell phone at (734) 765-7340.

Sincerely,

Simon Sultana  
Ph.D. Candidate, Education

Old Dominion University  
[ssult004@odu.edu](mailto:ssult004@odu.edu)

Philip Reed  
Associate Professor, STEM Education &  
Professional Studies  
Old Dominion University  
[preed@odu.edu](mailto:preed@odu.edu)

#### References

- Koszalka, T. A., Russ-Eft, D. F., Reiser, R., & Senior, F. A. (2013). *Instructional designer competencies: The standards* (4th ed.). Charlotte, NC: Information Age Publishing, Inc.
- Ornstein, A., & Hunkins, F. (2013). *Curriculum foundations, principles and theory* (6th ed.). Boston, MA: Allyn and Bacon.

**APPENDIX D**  
**ROUND 1 SURVEY**

April 1, 2016

Dear \_\_\_\_\_:

Thank you for agree to share your insight in this study on identifying the competencies, programming languages, and assessment methods recommended for an introductory computer science course. We are appreciative of your time and efforts and feel that your input will be most valuable in this endeavor. Below you will find the survey for Round 1. Part 1 asks some questions about your background. This information is being collected to characterize the experience level of the group and will be reported in aggregate only. Your information will be kept confidential. Part 2 of the questionnaire lists competencies, programming languages, and assessments to be used in an introduction to computer science course. Students enrolling in this course may be pursuing a bachelor's degree in software engineering, a minor in software engineering, or taking the class as a general elective. The topics in each of the three sections are to be weighted in terms of their applicability for such a course.

Part 1 – Demographic Data

1. Age: \_\_\_\_\_
  2. Gender: \_\_\_\_\_
  3. Ethnicity origin (e.g. African American, Asian, Caucasian, Hispanic, Native American, etc.): \_\_\_\_\_
  4. How many years of experience do you have as a professional teaching or developing computer systems or software? \_\_\_\_\_
  5. What is the highest level of formal education you have in a field related to computer science (including information technology, information systems, computer engineering, software engineering, etc.)? \_\_\_\_\_
  6. In what area of study is this degree? \_\_\_\_\_
- Experience questions (if any of the below questions do not apply, please indicate "N/A")
7. In how many programming languages are you fluent? \_\_\_\_\_
  8. If you are employed in industry what is the focus of the company at which you are employed: \_\_\_\_\_
  9. If you are a faculty member, please indicate the courses you teach:  
\_\_\_\_\_

10. List any professional certifications you have related to computing:

---



## Part 2 – Content for an Introduction to Computer Science Course

- A. Rate the following competencies in terms of their importance in an introduction to computer science course as very important, important, moderately important, of little importance, or unimportant.

	Competency	very important	important	moderately important	of little importance	unimportant
1	Analyze algorithms for effectiveness and efficiency					
2	Illustrate concepts in artificial intelligence					
3	Summarize basic computability, theory of computation, and its limits					
4	Describe different types of data representation (e.g. graphics, binary numbers, etc.)					
5	Illustrate the use of Boolean logic and basic combinational digital circuits					
6	Describe basic computer architecture and organization					
7	Summarize the history of computing and its ramifications to implementation today					
8	Explain the factors contributing to human-computer interaction in computing					
9	Illustrate the use of databases and apply SQL					
10	Explain the operation of compilers					
11	Discuss the operation of networks and related practices (e.g. data compression, etc.)					
12	Explain the functionality of operating systems and provide examples					
13	Describe common programming languages and their popular uses					
14	Describe benefit and operation of parallel and distributed systems and programming					
15	Demonstrate use of recursion in a program					
16	Describe the need for computer and data security and identify best practices					
17	Explain the role of modeling and simulation in computing					
18	Describe societal impact of computing					
19	Describe the World Wide Web and select internet protocols					

	Competency	very important	important	moderately important	of little importance	unimportant
20	Describe process and practices in software engineering					
21	Plan a career in CS					
22	Write functional object-oriented programs employing programming fundamentals					
23	Write functional procedural programs employing programming fundamentals					
24	Implement good documentation practices in programming					
25	Demonstrate teamwork and interpersonal group skills					
26	Demonstrate algorithmic thinking.					
27	Demonstrate computational thinking					
28	Demonstrate problem solving					
29	Demonstrate critical thinking and reasoning					
30	Demonstrate systems thinking					
31	Demonstrate creativity in programming					
32	Demonstrate time and resource management skills in a project					
33	Exhibit entrepreneurship in computing					
34	Communicate effectively orally and in writing					
35	Describes self-learning and assesses self					
36	Exhibit digital literacy					
37	Explain and choose from different file structures					
38	Explain and utilize effective procedures in software verification and validation					

Identify any competencies not included here that you feel would be important to include in an introduction to computer science course. Also indicate the rating for this topic.

1. \_\_\_\_\_
2. \_\_\_\_\_
3. \_\_\_\_\_

Comment on why you feel these competencies merit consideration.

---



---



---

- B. Rate the following programming languages in terms of their importance in an introduction to computer science course as very important, important, moderately important, of little importance, or unimportant.

	Programming Language	very important	important	moderately important	of little importance	unimportant
1	Alice					
2	Assembly Language					
3	C					
4	C#					
5	C++					
6	Greenfoot					
7	Haskell					
8	Java					
9	JavaScript					
10	MATLAB					
11	Objective-C					
12	Perl					
13	PHP					
14	PL/SQL					
15	Python					
16	R					
17	Ruby					
18	Scala					
19	Scheme					
20	Scratch					
21	Shell					
22	Swift					
23	Visual Basic					

Identify any programming languages not included here that you feel would be important to include in an introduction to computer science course. Also indicate the rating for this topic.

1. \_\_\_\_\_
2. \_\_\_\_\_
3. \_\_\_\_\_

Comment on why you feel these competencies merit consideration.

---



---



---

- C. Rate the following assessments in terms of their importance in an introduction to computer science course as very important, important, moderately important, of little importance, or unimportant.

	Assessment	very important	important	moderately important	of little importance	unimportant
1	Case studies					
2	Code reviews					
3	Concept questions					
4	Essays					
5	Final Exams					
6	Online threaded discussions					
7	Interviews with professionals					
8	Lab exercises					
9	Quizzes					
10	Smaller programming activities					
11	Term projects					

Identify any assessments not included here that you feel would be important to include in an introduction to computer science course. Also indicate the rating for this topic.

1. \_\_\_\_\_
2. \_\_\_\_\_
3. \_\_\_\_\_

Comment on why you feel these competencies merit consideration.

---



---



---

**APPENDIX E**  
**ROUND 2 SURVEY**

April 15, 2016

Dear \_\_\_\_\_:

Thank you for agree to share your insight in this study on identifying the competencies, programming languages, and assessment methods recommended for an introductory computer science course. We are appreciative of your time and efforts and feel that your input will be most valuable in this endeavor. Below you will find the questionnaire for Round 2. The questionnaire lists the competencies, programming languages, and assessments to be used in an introduction to computer science course. Their median rank values are included as insight to how the study's group assessed their importance. Remember that students enrolling in this course may be pursuing a bachelor's degree in software engineering, a minor in software engineering, or as a general elective.

You are to select the topics in each of the three categories that should be included for consideration in subsequent rounds. You may select no more than ten items in each of the three categories.

Content for an Introduction to Computer Science Course

- A. Select competencies from the following in terms of their importance in an introduction to computer science course. Select **at least ten** competencies.

	Competency	Select
1	Analyze algorithms for effectiveness and efficiency	
2	Illustrate concepts in artificial intelligence	
3	Summarize basic computability, theory of computation, and its limits	
4	Describe different types of data representation (e.g. graphics, binary numbers, etc.)	
5	Illustrate the use of Boolean logic and basic combinational digital circuits	
6	Describe basic computer architecture and organization	
7	Summarize the history of computing and its ramifications to implementation today	
8	Explain the factors contributing to human-computer interaction in computing	
9	Illustrate the use of databases and apply SQL	
10	Explain the operation of compilers	
11	Discuss the operation of networks and related practices (e.g. data compression, etc.)	
12	Explain the functionality of operating systems and provide examples	
13	Describe common programming languages and their popular uses	
14	Describe benefit and operation of parallel and distributed systems and programming	
15	Demonstrate use of recursion in a program	
16	Describe the need for computer and data security and identify best practices	
17	Explain the role of modeling and simulation in computing	
18	Describe societal impact of computing	

19	Describe the World Wide Web and select internet protocols	
20	Describe process and practices in software engineering	
21	Plan a career in CS	
22	Write functional object-oriented programs employing programming fundamentals	
23	Write functional procedural programs employing programming fundamentals	
24	Implement good documentation practices in programming	
25	Demonstrate teamwork and interpersonal group skills	
26	Demonstrate algorithmic thinking.	
27	Demonstrate computational thinking	
28	Demonstrate problem solving	
29	Demonstrate critical thinking and reasoning	
30	Demonstrate systems thinking	
31	Demonstrate creativity in programming	
32	Demonstrate time and resource management skills in a project	
33	Exhibit entrepreneurship in computing	
34	Communicate effectively orally and in writing	
35	Describes self-learning and assesses self	
36	Exhibit digital literacy	
37	Explain and choose from different file structures	
38	Explain and utilize effective procedures in software verification and validation	

B. Select programming languages from the following in terms of their importance in an introduction to computer science course. Select **at least ten** languages.

	Programming Language	Select
1	Alice	
2	Assembly Language	
3	C	
4	C#	
5	C++	
6	Greenfoot	
7	Haskell	
8	Java	
9	JavaScript	
10	MATLAB	
11	Objective-C	
12	Perl	
13	PHP	
14	PL/SQL	
15	Python	
16	R	
17	Ruby	
18	Scala	
19	Scheme	

20	Scratch	
21	Shell	
22	Swift	
23	Visual Basic	

C. Select competencies from the following in terms of their importance in an introduction to computer science course. Select **at least ten** competencies.

	Assessment	Select
1	Case studies	
2	Code reviews	
3	Concept questions,	
4	Essays	
5	Final exams,	
6	Online threaded discussions	
7	Interview with software professional	
8	Lab exercises or smaller programming activities	
9	Quizzes	
10	Smaller programming activities	
11	Term projects	

## APPENDIX F

### ROUND 3 SURVEY FOR ACADEMIC GROUP

May 9, 2016

Welcome to Round 3! Thank you for sharing your insight in this study on experts' recommendations for the competencies, programming languages, and assessment methods for an introductory computer science course. We are appreciative of your time and efforts and feel that your input will be most valuable in this endeavor.

Below you will find the survey for Round 3. Please be sure to answer the three questions on the three separate pages. Items selected by at least five of the ten experts participating in Round 2 are included in Round 3. You are asked to rank these items for each of the three categories. Consider your thoughts in relation to the importance attributed to it by the overall group of experts in Round 2 (via the indicated number of experts selecting each item in the previous round).

The rankings for competencies, programming languages, and assessments will be checked for conformity among the group. Categories that have achieved conformity will be deemed complete while those that do not achieve agreement will be carried over to a final Round 4. As before your information will be kept confidential. This survey will close on Monday, May 16, 2016 so thank you for completing it before then.

This page lists potential competencies to be used in an introductory computer science survey course. Please note: Students enrolling in this course may be pursuing a bachelor's degree in software engineering, a minor in software engineering, or taking the class as a general elective.

#### Content for an Introduction to Computer Science Course

Q1: Rank the following competencies for an introduction to computer science course. You can click and drag or numerically rank items from most important (1) to least important (15).

Analyze algorithms for effectiveness and efficiency (Selected by 5 of 10 in Round 2)	
Describe basic computer architecture and organization (Selected by 7 of 10 in Round 2)	
Explain the functionality of operating systems and provide examples (Selected by 6 of 10 in Round 2)	
Demonstrate use of recursion in a program (Selected by 5 of 10 in Round 2)	
Describe the need for computer and data security and identify best practices (designing, implementing, and verifying hacker-resistant safe code) (Selected by 5 of 10 in Round 2)	
Explain the role of modeling and simulation in computing (Selected by 5 of 10 in Round 2)	
Describe process and practices in software engineering (Selected by 6 of 10 in Round 2)	
Write functional object-oriented programs employing programming fundamentals (Selected by 8 of 10 in Round 2)	
Write functional procedural programs employing programming fundamentals (Selected by 9 of 10 in Round 2)	



Implement good documentation practices in programming (Selected by 6 of 10 in Round 2)	
Demonstrate teamwork and interpersonal group skills (Selected by 8 of 10 in Round 2)	
Demonstrate algorithmic thinking (Selected by 6 of 10 in Round 2)	
Demonstrate computational thinking (Selected by 6 of 10 in Round 2)	
Demonstrate problem solving (Selected by 8 of 10 in Round 2)	
Demonstrate critical thinking and reasoning (Selected by 7 of 10 in Round 2)	

Q2: Rank these programming languages to potentially be used in an introduction to computer science course. You can click and drag or numerically rank items from most important (1) to least important (8).

C (Selected by 9 of 10 in Round 2)	
C# (Selected by 8 of 10 in Round 2)	
C++ (Selected by 10 of 10 in Round 2)	
Java (Selected by 10 of 10 in Round 2)	
JavaScript (Selected by 10 of 10 in Round 2)	
PHP (Selected by 5 of 10 in Round 2)	
Python (Selected by 10 of 10 in Round 2)	
Ruby (Selected by 6 of 10 in Round 2)	

Q3: Rank the following assessments to potentially be used in an introduction to computer science course. You can click and drag or numerically rank items from most important (1) to least important (11).

Case studies (Selected by 9 of 10 in Round 2)	
Code reviews (Selected by 9 of 10 in Round 2)	
Concept questions (Selected by 10 of 10 in Round 2)	
Final exams (Selected by 10 of 10 in Round 2)	
Online threaded discussions (Selected by 8 of 10 in Round 2)	
Interviews with professionals (Selected by 5 of 10 in Round 2)	
Lab exercises (Selected by 10 of 10 in Round 2)	
Quizzes (Selected by 10 of 10 in Round 2)	
Smaller programming activities (Selected by 10 of 10 in Round 2)	
Team programming assignments (Selected by 9 of 10 in Round 2)	
Term projects (Selected by 9 of 10 in Round 2)	

## APPENDIX G

### ROUND 3 SURVEY FOR INDUSTRY GROUP

May 9, 2016

Welcome to Round 3! Thank you for sharing your insight in this study on experts' recommendations for the competencies, programming languages, and assessment methods for an introductory computer science course. We are appreciative of your time and efforts and feel that your input will be most valuable in this endeavor.

Below you will find the survey for Round 3. Please be sure to answer the three questions on the three separate pages. Items selected by at least six of the eleven experts participating in Round 2 are included in Round 3. You are asked to rank these items for each of the three categories. Consider your thoughts in relation to the importance attributed to it by the overall group of experts in Round 2 (via the indicated number of experts selecting each item in the previous round).

The rankings for competencies, programming languages, and assessments will be checked for conformity among the group. Categories that have achieved conformity will be deemed complete while those that do not achieve agreement will be carried over to a final Round 4. As before your information will be kept confidential. This survey will close on Monday, May 16, 2016 so thank you for completing it before then.

This page lists potential competencies to be used in an introductory computer science survey course. Please note: Students enrolling in this course may be pursuing a bachelor's degree in software engineering, a minor in software engineering, or taking the class as a general elective.

#### Content for an Introduction to Computer Science Course

Q1: Rank the following competencies for an introduction to computer science course. You can click and drag or numerically rank items from most important (1) to least important (12).

Analyze algorithms for effectiveness and efficiency (Selected by 8 of 11 in Round 2)	
Describe different types of data representation (e.g. graphics, binary numbers, etc.) (Selected by 7 of 11 in Round 2)	
Describe basic computer architecture and organization (Selected by 6 of 11 in Round 2)	
Illustrate the use of databases and apply SQL (Selected by 7 of 11 in Round 2)	
Describe common programming languages and their popular uses (Selected by 9 of 11 in Round 2)	
Describe process and practices in software engineering (Selected by 7 of 11 in Round 2)	
Write functional object-oriented programs employing programming fundamentals (Selected by 8 of 11 in Round 2)	
Write functional procedural programs employing programming fundamentals (Selected by 8 of 11 in Round 2)	

Implement good documentation practices in programming (Selected by 6 of 11 in Round 2)	
Demonstrate teamwork and interpersonal group skills (Selected by 8 of 11 in Round 2)	
Demonstrate problem solving (Selected by 6 of 11 in Round 2)	
Demonstrate critical thinking and reasoning (Selected by 8 of 11 in Round 2)	

Q2: Rank these programming languages to potentially be used in an introduction to computer science course. You can click and drag or numerically rank items from most important (1) to least important (12).

Assembly Language (Selected by 6 of 11 in Round 2)	
C (Selected by 6 of 11 in Round 2)	
C# (Selected by 9 of 11 in Round 2)	
C++ (Selected by 9 of 11 in Round 2)	
HTML5 (Selected by 9 of 11 in Round 2)	
Java (Selected by 9 of 11 in Round 2)	
JavaScript (Selected by 11 of 11 in Round 2)	
PHP (Selected by 6 of 11 in Round 2)	
PL/SQL (Selected by 7 of 11 in Round 2)	
Python (Selected by 10 of 11 in Round 2)	
Ruby (Selected by 7 of 11 in Round 2)	
Shell (Selected by 7 of 11 in Round 2)	

Q3: Rank the following assessments to potentially be used in an introduction to computer science course. You can click and drag or numerically rank items from most important (1) to least important (11).

Case studies (Selected by 10 of 11 in Round 2)	
Code reviews (Selected by 11 of 11 in Round 2)	
Concept questions (Selected by 10 of 11 in Round 2)	
Final exams (Selected by 9 of 11 in Round 2)	
Online threaded discussions (Selected by 8 of 11 in Round 2)	
Interviews with professionals (Selected by 11 of 11 in Round 2)	
Lab exercises (Selected by 11 of 11 in Round 2)	
Quizzes (Selected by 9 of 11 in Round 2)	
Smaller programming activities (Selected by 11 of 11 in Round 2)	
Team programming assignments (Selected by 10 of 11 in Round 2)	
Term projects (Selected by 11 of 11 in Round 2)	

## APPENDIX H

### ROUND 4 SURVEY FOR ACADEMIC GROUP

Please read before starting.

Welcome to Round 4, the final round of the survey! The group did not achieve conformity on any of the three categories in Round 3 so we will proceed with one final round of rankings. The level of conformity of the rankings in Round 3 was computed using Kendall's  $W$ , which ranges from 0 (zero agreement) to 1 (full agreement). You will see the value achieved by the group in each of the three questions. Please be sure to answer the three questions on the three separate pages. You are again asked to rank these items for each of the three categories. The items are listed according to the results in Round 3. Consider your thoughts in relation to the importance attributed to it by the overall group of experts (via the median ranking) in Round 3.

As before your information will be kept confidential. This survey will close on Tuesday, May 31, 2016 so thank you for completing it before then.

This page lists potential competencies to be used in an introductory computer science survey course. Please remember: Students enrolling in this course may be pursuing a bachelor's degree in software engineering, a minor in software engineering, or taking the class as a general elective. Content for an Introduction to Computer Science Course

Q1: Rank the following competencies for an introduction to computer science course. You can click and drag or numerically rank items from most important (1) to least important (15). The level of conformity (as indicated by Kendall's  $W$  ranging from 0-no agreement to 1-full agreement) for this category was 0.56 (moderate agreement) in Round 3.

Competency	Rank
Write functional procedural programs employing programming fundamentals (Round 3 median rank = 1)	
Write functional object-oriented programs employing programming fundamentals (Round 3 median rank = 3)	
Demonstrate problem solving (Round 3 median rank = 3)	
Demonstrate algorithmic thinking (Round 3 median rank = 5)	
Demonstrate critical thinking and reasoning (Round 3 median rank = 5)	
Demonstrate computational thinking (Round 3 median rank = 6)	
Implement good documentation practices in programming (Round 3 median rank = 7)	
Demonstrate teamwork and interpersonal group skills (Round 3 median rank = 8)	
Analyze algorithms for effectiveness and efficiency (Round 3 median rank = 9)	
Describe process and practices in software engineering (Round 3 median rank = 11)	
Describe basic computer architecture and organization (Round 3 median rank = 12)	
Explain the functionality of operating systems and provide examples (Round 3 median rank = 12)	
Demonstrate use of recursion in a program (Round 3 median rank = 12)	
Explain the role of modeling and simulation in computing (Round 3 median rank = 12)	

Describe the need for computer and data security and identify best practices (designing, implementing, and verifying hacker-resistant safe code) (Round 3 median rank = 14)	
---	--

Q2: Rank these programming languages to potentially be used in an introduction to computer science course. You can click and drag or numerically rank items from most important (1) to least important (8). The level of conformity (as indicated by Kendall's  $W$  ranging from 0-no agreement to 1-full agreement) for this category was 0.63 (moderate agreement) in Round 3.

Programming Language	Rank
Java (Round 3 median rank = 1)	
C++ (Round 3 median rank = 2)	
C (Round 3 median rank = 4)	
Python (Round 3 median rank = 4)	
C# (Round 3 median rank = 6)	
PHP (Round 3 median rank = 6)	
Ruby (Round 3 median rank = 6)	
JavaScript (Round 3 median rank = 7)	

Q3: Rank the following assessments to potentially be used in an introduction to computer science course. You can click and drag or numerically rank items from most important (1) to least important (11). The level of conformity (as indicated by Kendall's  $W$  ranging from 0-no agreement to 1-full agreement) for this category was 0.53 (weak agreement) in Round 3.

Assessment	Rank
Lab exercises (Round 3 median rank = 2)	
Smaller programming activities (Round 3 median rank = 2)	
Team programming assignments (Round 3 median rank = 4)	
Concept questions (Round 3 median rank = 5)	
Code reviews (Round 3 median rank = 6)	
Quizzes (Round 3 median rank = 6)	
Term projects (Round 3 median rank = 6)	
Final exams (Round 3 median rank = 7)	
Case studies (Round 3 median rank = 9)	
Online threaded discussions (Round 3 median rank = 10)	
Interviews with professionals (Round 3 median rank = 10)	

## APPENDIX I

### ROUND 4 SURVEY FOR INDUSTRY GROUP

Please read before starting

Welcome to Round 4, the final round of the survey! The group did not achieve conformity on any of the three categories in Round 3 so we will proceed with one final round of rankings. The level of conformity of the rankings in Round 3 was computed using Kendall's  $W$ , which ranges from 0 (zero agreement) to 1 (full agreement). You will see the value achieved by the group in each of the three categories. Please be sure to answer the three questions on the three separate pages. You are again asked to rank these items for each of the three categories. The items are listed according to the results in Round 3. Consider your thoughts in relation to the importance attributed to it by the overall group of experts (via the median ranking) in Round 3.

As before your information will be kept confidential. This survey will close on Tuesday, May 31, 2016 so thank you for completing it before then.

This page lists potential competencies to be used in an introductory computer science survey course. Please remember: Students enrolling in this course may be pursuing a bachelor's degree in software engineering, a minor in software engineering, or taking the class as a general elective.

Q1: Rank the following competencies for an introduction to computer science course. You can click and drag or numerically rank items from most important (1) to least important (12). The level of conformity (as indicated by Kendall's  $W$  ranging from 0-no agreement to 1-full agreement) for this category was 0.13 (little to no agreement) in Round 3.

Competency	Rank
Demonstrate problem solving (Round 3 median rank = 2.5)	
Demonstrate critical thinking and reasoning (Round 3 median rank = 3)	
Describe process and practices in software engineering (Round 3 median rank = 5)	
Write functional procedural programs employing programming fundamentals (Round 3 median rank = 5.5)	
Demonstrate teamwork and interpersonal group skills (Round 3 median rank = 6)	
Describe basic computer architecture and organization (Round 3 median rank = 6.5)	
Analyze algorithms for effectiveness and efficiency (Round 3 median rank = 7)	
Describe different types of data representation (e.g. graphics, binary numbers, etc.) (Round 3 median rank = 7)	
Write functional object-oriented programs employing programming fundamentals (Round 3 median rank = 7)	
Describe common programming languages and their popular uses (Round 3 median rank = 7.5)	
Implement good documentation practices in programming (Round 3 median rank = 8.5)	
Illustrate the use of databases and apply SQL (Round 3 median rank = 9.5)	

Q2: Rank these programming languages to potentially be used in an introduction to computer science course. You can click and drag or numerically rank items from most important (1) to least important (12). The level of conformity (as indicated by Kendall's  $W$  ranging from 0-no agreement to 1-full agreement) for this category was 0.11 (little to no agreement) in Round 3.

Programming Language	Rank
JavaScript (Round 3 median rank = 3)	
Python (Round 3 median rank = 3)	
C# (Round 3 median rank = 4.5)	
Java (Round 3 median rank = 4.5)	
C++ (Round 3 median rank = 5.5)	
HTML5 (Round 3 median rank = 5.5)	
PHP (Round 3 median rank = 6)	
C (Round 3 median rank = 7)	
PL/SQL (Round 3 median rank = 8)	
Shell (Round 3 median rank = 9)	
Ruby (Round 3 median rank = 9.5)	
Assembly Language (Round 3 median rank = 10)	

Q3: Rank the following assessments to potentially be used in an introduction to computer science course. You can click and drag or numerically rank items from most important (1) to least important (11). The level of conformity (as indicated by Kendall's  $W$  ranging from 0-no agreement to 1-full agreement) for this category was 0.20 (little to no agreement) in Round 3.

Assessment	Rank
Smaller programming activities (Round 3 median rank = 3)	
Term projects (Round 3 median rank = 3)	
Lab exercises (Round 3 median rank = 4)	
Code reviews (Round 3 median rank = 4.5)	
Concept questions (Round 3 median rank = 6)	
Team programming assignments (Round 3 median rank = 6)	
Case studies (Round 3 median rank = 6.5)	
Final exams (Round 3 median rank = 8)	
Interviews with professionals (Round 3 median rank = 8.5)	
Quizzes (Round 3 median rank = 8.5)	
Online threaded discussions (Round 3 median rank = 9)	

## VITA

Simon Sultana

Darden College of Education  
Old Dominion University  
Norfolk VA, 23529

### Academic Degrees

M.B.A. Wayne State University	2003 Business Administration
M.S. Wayne State University	2000 Electrical Engineering
B.S. The University of Michigan	1995 Electrical Engineering

### Professional Experience

2015-Present	Fresno Pacific University, Faculty Software Engineering & Computer Information Systems
2006-2015	DeVry University, Program Dean College of Engineering & Information Sciences
2005-2006	ASI Systems Integrators, Senior Engineering Consultant
2004-2005	Motorola, Inc., Senior Engineer
1995-2004	Chrysler Corporation, Senior Engineer

### Affiliations

2015-Present	Association for Computing Machinery
2012, 2014-2015	American Society for Engineering Education
2014-2015	Association of Technology, Management, and Applied Engineering
2008-2009	Institute of Electrical and Electronics Engineers

### Publications

Sultana, S. (2015). [Review of the book *Rethinking engineering: The CDIO approach* by Crawley, Malmqvist, Östlund, Brodeur, & Eström]. *Journal of Technology Education*, 26(2), 74-79.

Sultana, S., Amer, H., Johnson, R., Soderlund, T., & Draper, D. (2015). Acme portable motion projector: Wile E. Coyote's learning device? *Tech Trends*, 59(3), 15-16.

Sultana, S. (2015). *Impressions of engineering technology faculty on student mathematics comprehension*. Unpublished Manuscript.

Sultana, S. (2015). *Efforts to improve the mathematics ability of online students in an electronic circuits course*. Unpublished Manuscript.