Old Dominion University
ODU Digital Commons

STEMPS Faculty Publications | STEM Education & Professional Studies

Fall 2017

Curriculum for an Introductory Computer Science Course: Identifying Recommendations from Academia and Industry

Simon G. Sultana

Philip A. Reed
Old Dominion University

Follow this and additional works at: https://digitalcommons.odu.edu/stemps_fac_pubs

 Part of the Curriculum and Instruction Commons, Higher Education Commons, and the Science and Mathematics Education Commons

Original Publication Citation

Sultana, S. G., & Reed, P. A. (2017). Curriculum for an introductory computer science course: Identifying recommendations from academia and industry. *Journal of Technology Studies, 43*(2), 80-92. doi: 10.21061/jots.v43i2.a.3

This Article 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 Faculty Publications by an authorized administrator of ODU Digital Commons. For more information, please contact digitalcommons@odu.edu.

Curriculum for an Introductory Computer Science Course: Identifying Recommendations from Academia and Industry

By Simon G. Sultana and Philip A. Reed

ABSTRACT

The purpose of this study was to define the course content for a university introductory computer science course based on regional needs. Delphi methodology was used to identify the competencies, programming languages, and assessments that academic and industry experts felt most important. Four rounds of surveys were conducted to rate the items in the straw models, to determine the entries deemed most important, and to understand their relative importance according to each group. The groups were then asked to rank the items in each category and attempt to reach consensus as determined by Kendall's coefficient of concordance. 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. The industry experts did not reach consensus and showed low agreement on their recommendations for competencies, programming languages, and assessments.

Keywords: Curriculum Design, Delphi, Competencies, Assessments, Computer Science Education, Programming Languages, Introductory Course

INTRODUCTION AND BACKGROUND

As education aims to prepare a workforce for future jobs, it is of little surprise that the number of students in introductory computer science (CS) courses have continued to grow in colleges and universities. These courses can cover information systems, hardware and architecture, operating systems, software engineering (SE), programming, databases, among other topics (Anderson, Ferro, & Hilton, 2011; Wu, Hsu, Lee, Wang, & Sun, 2014). Additionally, instructors can select from several computer languages (Ali & Smith, 2014; Chang, 2014; Shein, 2015) to provide students an experience that is educational, motivating, and meets current industry practices. Likewise, there are several possibilities for assessment in these courses (Fulton & Schweitzer, 2011; Muñoz, Martínez, Cárdenas, & Cepeda, 2013; Shaw, 2010). The aim of this research was to provide suggestions for the competencies, programming languages, and assessments for an introductory CS course. The class, part of a new undergraduate SE program at a small private nonprofit university in Fresno County, California, will serve as a program gateway for students looking to major or minor in SE, and for others looking to develop some background in computing.

Sources of CS Curriculum Recommendations

Expert recommendations on computing curricula are found in professional associations, industry, academic institutions, and the literature. The Association for Computing Machinery (ACM) provided the first set of curriculum recommendations for undergraduate study in CS in 1965 and has published updates about once every decade, in recent years as part of the Joint Task Force on Computing Curricula (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 have often felt underserved by the documents (Liberal Arts Computer Science Consortium [LACS], 2007). The LACS last released a model curriculum almost ten years ago and based their suggestions on JTFCC's 2001 recommendations and included hours to focus on topics in introductory courses.

The computing industry includes businesses engaged in activities directly related to the disciplines of CS, computer engineering, information systems, information technology, and SE. Most of these distinct fields of study arose because of the individual skill sets required for these varied jobs and disciplines (Chand, 1974; Lunt, et al., 2005; Lutz, Naveda, & Vallino, 2014).

Industry defines the skills necessary for employment and education aims to teach them. Norton (1998) based the DACUM (Developing

a Curriculum) methodology on the premise that experts in industry best define their jobs and possess certain knowledge, skills, and aptitude with tools. Business practices are developed to improve effectiveness and efficiency and there arises a need for new employees who possess some knowledge of, and perhaps the ability to implement, them. There has been much written over the past few years on the reasons for teaching agile software development practices in the classroom (Guercio & Sharif; 2012; Lutz et al., 2014; Rajlich, 2013). The computing industry has thus shown that it serves a role in the curriculum definition of CS and related disciplines.

There are approximately 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 Education and industry, therefore, can both be regarded as sources of expertise that can be useful for the development of new computing curriculum. The findings in the literature, along with experts' recommendations, serve as rich sources to help a curriculum designer choose competencies, programming languages, and assessments.

Competencies

There are myriad topics in the CS discipline (JTFCC, 2001) so a consideration of disparate areas was required if experts were to be provided with a comprehensive list. The JTFCC (2013) identified potential topics and the LACS (2007) provided recommendations on areas of study. Three introductory CS course textbooks were also consulted: these were *Connecting with Computer Science* (2nd edition) (2011) by Anderson, Ferro, and Hilton, *Invitation to Computer Science* (7th edition) (2016) by Schneider and Gersting, and *Computer Science Illuminated* (6th edition) (2016) by Dale and Lewis.

A literature review was conducted to supplement the topics identified in these texts. A straw model was developed using the information on competencies gathered from these sources. Although identification of potential competencies from curriculum recommendations/textbooks and journal articles was done independently, 24 of the 26 topics in the former sources were found in the latter group. In all, 38 competencies were identified to form the straw list introduced to the experts in this study.

Programming Languages

Introductory CS courses include programming to varying degrees (Davies, Polack-Wahl, & Anewalt, 2011). There are reportedly up to 2,500 programming languages (Kinnersley, n.d.), though not all are actively used. Regardless, there are numerous languages available to introduce students to computer programming. Of utmost importance is accessibility for non-majors and beginners (Kelleher & Pausch, 2005; Malan & Leitner, 2007; Norman & Adams, 2015; Stefik & Gellenbeck, 2011) and perceived importance by majors (Forte & Guzdial, 2005).

Six sources were consulted to determine language use in industry; these included the TIOBE index, RedMonk, the PopularitY of Programming Language (PYPL) list, Trendy Skills, Black Duck Software, and IEEE Spectrum. Four sources were found that identified language popularity in academia. O'Grady (2013) reported on RedMonk's (2015) use of references of programming languages in the curriculum of leading colleges and universities to rank the top twenty languages, as did three additional sources from journal articles, which included popularity rankings (Ben Arfa Rabai, Cohen, & Mili, 2015; Davies et al., 2011; Guo, 2014). Using the guideline to include languages that were identified in at least three of the six industry sources, or in at least two of the four academic sources, a list of twenty languages was constructed. Additionally, three visual programming languages were thought to warrant inclusion (Alice, Greenfoot, and Scratch) as they have become increasingly popular in introductory courses (Davies et al., 2011; Malan & Leitner, 2007). In all, 23 programming languages were identified to form the straw list introduced to the experts in this study.

Assessments

The literature contained articles in which educators teaching computing courses shared their curriculum designs and explained assessments. Many researchers mentioned assessments they utilized in the classroom as evidence of student learning to demonstrate results. The authors reviewed reported on some

of the assessments used in introductory CS courses. Eleven distinct assessment devices were identified for academic and industry experts to consider for an introduction to a CS course. These items, in alphabetical order, were:

- Case studies
- Code reviews
- Concept questions
- Essays
- Final exams
- Interviews with professionals
- Lab exercises
- Online threaded discussions
- Quizzes
- Smaller programming activities
- Term projects

The goal of the study was to suggest competencies, programming languages, and assessments for an introductory CS course based on the recommendations of regional experts in academia and industry. This information could then be used by a curriculum developer to better meet the needs of students and other stakeholders in the region in which the introductory CS course was offered.

METHODS

The Delphi approach was used to collect data and surveys were distributed via SurveyMonkey. An email message with instructions and the appropriate link for each round was sent to the participants and they were asked to respond within one week. Follow-up emails were sent out during the week. This study's design was heavily based on the approach of Okoli and Pawlowski (2004) in that a panel structure was utilized, which divided the two expert groups as they selected items in Round 2 and ranked them in subsequent rounds. A major deviation from Okoli and Pawlowski's (2004) approach was to provide experts with straw models of initial items (Rotondi & Gustafson, 1996) for each of the three categories in Round 1.

Potential participants were identified using suggestions from professionals in higher education, graduates of academic programs, and research of organizations' web sites in California's Central Valley. All persons were invited to take part in the research by email. Phone calls were placed to those who did not initially respond. Snowball sampling was utilized to help increase exposure of the study to the expert population (Hays & Singh, 2012). Individuals who agreed to participate, therefore, were asked to suggest other candidates. 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.

One research subject matter expert was also recruited for this study to assist the researcher in reviewing participants' open responses from the first round to validate their identification. This individual was required to have a Ph.D. and have experience teaching in an information technology related discipline.

In the first survey, each participant was asked to provide demographics (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. The second set of questions asked the participants to rate the applicability of the competencies from the straw model on a five-point Likert-type scale (very important = 5, important = 4, moderately important = 3, of little importance = 2, unimportant = 1). The subsequent sections provided a list of programming languages and assessments. Blank entries were also available for optional contributions to each of the three categories.

The results of the surveys were downloaded into Microsoft Excel. Statistics were computed for the age, years of experience, number of programming languages in which the participants were fluent, gender, employment, and highest education were computed using various built-in functions. Responses to each of the three content categories were also copied into Excel and quantified according to the anchors as previously identified. Newly suggested items by participants were checked for individuality and inserted into the lists. The newly suggested items were reviewed with the subject matter expert and changes to the surveys for the next round were made. Any item selected by at least two participants was added to the list of competencies, programming languages, or assessments.

The rated lists of items and their median weight scores were added to the survey for the second

round. The median was computed as these data were Likert-type in nature (Boone, H. N. Jr. & Boone, D. A., 2012) and this value in the questionnaires would communicate the perceived importance attributed to each item. The participants were instructed to determine whether 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 potentially allow experts to come to consensus more quickly and would allow recommendations from each group to be distinguished for final decision making by the curriculum designer/researcher.

Feedback was collected from participants on their selected items from each of the three categories. Those items selected by at least half of each expert group were chosen to be included for Round 3 (Okoli & Pawlowski, 2004) for that group. The findings from this point would be independent for each group.

The steps in Rounds 3 and 4 were identical. The lists of items as selected by the experts from the previous round were added to the survey. Participants were asked to rank each item in each of the three categories of competencies, programming languages, and assessments. The lists were imported into SurveyMonkey as two questionnaires in keeping with separate panels.

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). Schmidt (1997) identified a value of at least 0.7 to indicate strong agreement so this threshold was used to determine whether 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* value would, therefore be computed six times for Round 3. Each *W* value would be analyzed independently and only those topics that failed to meet the minimum 0.7 threshold value were included in a Round 4 survey for each individual panel.

It was decided that a maximum of four rounds would be considered as it has been found that major fluctuations are typically not expected after a fourth round (Wilhelm, 2001) and participant fatigue can become a concern (Schmidt, 1997; Sitlington, 2015). Two ranked lists of suggested competencies, programming languages, and assessments 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. See Figure 1 for an overview of the study's design methodology.

Participants

The target members for experts were experienced industry and academic professionals in California's Central Valley. Since the opinion of experts in these positions was sought, a minimum of five years' experience was required for potential industry participants (Guu, Lin, & Lee, 2014; Joyner & Smith, 2015). Educators who held at least a Master's Degree in their field (Surakka, 2007) were approached about their interest in participating as academic experts. The researcher directly invited 85 experts from California's Central Valley; 48 individuals (56%) were from higher education; and 37 (44%) were from industry. 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.

RESULTS

Round 1

Eleven academic (92%) and eleven industry (100%) experts completed the Round 1 survey, including twenty males and 2 females (one from academia and one from industry). The second section of the survey asked participants to rate potential competencies for an introductory CS course. It was noteworthy that four competencies, those dealing with procedural programming, teamwork/interpersonal group skills, problem solving, and critical thinking, received median scores of 5 (very important) and the latter three items received minimum rating values no lower than 3.

The next section of the survey asked participants to rate 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

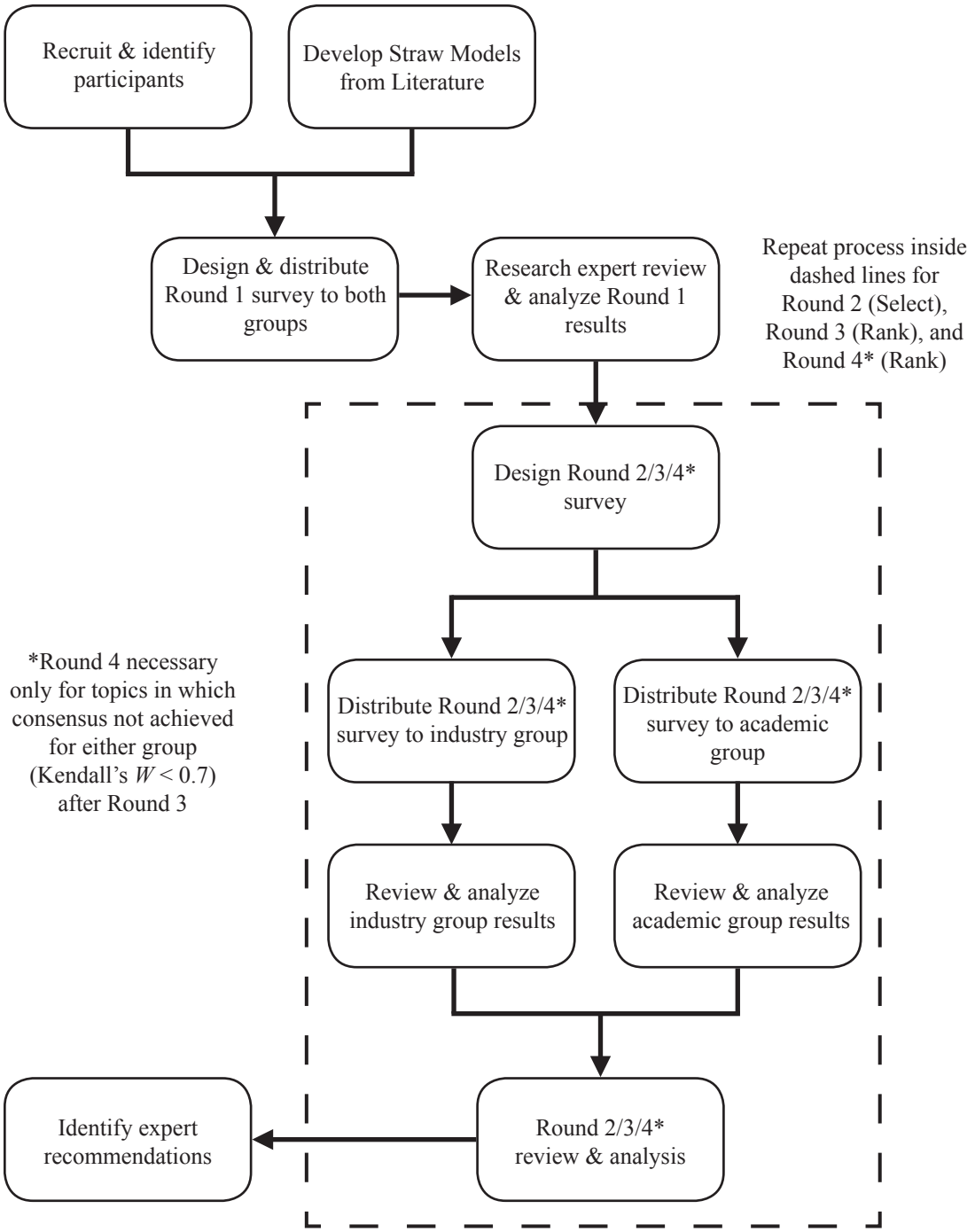

Figure 1. Study design methodology

as 0 points. 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 (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 but only HTML5 (Hypertext Markup Language) was mentioned in two responses. Though not typically considered a true programming language, HTML5 was added to the list for Round 2 because concepts in CS could be taught using this markup language.

The final section of the Round 1 survey asked participants to rate 11 potential assessments. The rating scores available were identical to

those used with the course competencies. The experts provided only four open-ended responses to the list of assessments to be considered. Team programming assignments were recommended by two individuals so this assessment was added for Round 2.

Round 2

The median ratings of the competencies, programming languages, and assessments were recorded into the survey for Round 2 to communicate the importance attributed to each item by the overall group. 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 rating score from Round 1. This instruction enabled participants to utilize deliberation as characterized by the Delphi approach without meeting with other experts in person.

Eight programming languages were selected by at least half of the experts in the academic group. The industry group elected to include 12 languages. All eight languages selected by at least half the experts in the academic group were also chosen by the industry group. The sole programming language chosen by all industry experts was JavaScript. No academic expert chose Greenfoot and no industry professional included Alice, Greenfoot, MATLAB, Scala, or Scratch.

Finally, the groups ranked 11 assessments. Because of the low number of assessments, the narrowing effect was expected to be minimal. Only essays were not chosen to be carried over into Rounds 3 and 4 and this omission was true for both groups.

The detailed data from Rounds 1 and 2 are not included in this article but are available in Sultana (2016).

Round 3

The third round provided experts the opportunity to rank the items 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 for majors and non-majors. 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 in their group selecting it in Round 2. 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%).

The academic group ranked 15 competencies and the industry experts ranked 12 competencies as shown in Table 1. The interquartile range (IQR) was calculated to identify the dispersion of the middle half of these data. The IQR values for the rankings of the top five competencies varied from 3.0 to 5.5 for the academic group and from 5.3 to 7.5 for the industry group.

The ranked programming languages from Round 3 for both groups are presented in Table 2. The academic group ranked eight programming languages and chose Java as their most important and C++ as the next highest ranked. The industry experts ranked 12 languages and selected JavaScript and Python as their most important.

Finally, the groups ranked 11 assessments. 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 experts also selected lab exercises as a top assessment and again did so with a low variability (IQR = 2.0). The industry group also selected term projects as tied for the most important assessments but with a high IQR value (8.3).

Kendall's W was calculated to analyze the conformity among the rankings of the three categories by the expert groups. 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 did the industry experts. Kendall's coefficient of concordance (W) tests were statistically significant, yet lacked full agreement, for 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 ($W_{AA} = 0.53$, $r_{AA} = 0.48$, $p < 0.001$).

TABLE 1: Rounds 3 & 4 Median Rankings of Competencies for Introductory Computer Science

Competency	Round 3				Round 4			
	Academic Group		Industry Group		Academic Group		Industry Group	
	Median	IQR	Median	IQR	Median	IQR	Median	IQR
Analyze algorithms for effectiveness and efficiency	9.0	4.0	7.0	5.3	9.0	2.0	7.0	3.0
Describe different types of data representation	-	-	7.0	5.5	-	-	7.0	4.0
Describe basic computer architecture and organization	12.0	5.5	6.5	9.0	11.0	3.0	6.0	7.0
Illustrate the use of databases and apply SQL	-	-	9.5	4.5	-	-	11.0	2.0
Explain the functionality of operating systems with examples	12.0	4.5	-	-	13.0	2.5	-	-
Describe common programming languages and popular uses	-	-	7.5	6.3	-	-	9.0	7.0
Demonstrate use of recursion in a program	12.0	3.0	-	-	13.0	2.0	-	-
Describe best practices for computer and data security	14.0	2.0	-	-	15.0	2.5	-	-
Explain the role of modeling and simulation in computing	12.0	6.5	-	-	14.0	1.5	-	-
Describe process and practices in SE	11.0	4.5	5.0	6.0	10.0	1.0	3.0	5.0
Write functioning object-oriented programs	3.0	4.5	7.0	3.8	2.0	0.5	9.0	5.0
Write functioning procedural programs	1.0	4.5	5.5	5.3	1.0	1.0	6.0	5.0
Implement good documentation practices in programming	7.0	7.5	8.5	7.3	7.0	2.5	8.0	5.0
Demonstrate teamwork and interpersonal group skills	8.0	6.5	6.0	7.5	8.0	2.5	6.0	3.0
Demonstrate algorithmic thinking	5.0	5.5	-	-	4.0	4.0	-	-
Demonstrate computational thinking	6.0	3.0	-	-	6.0	0.5	-	-
Demonstrate problem solving	3.0	3.5	2.5	7.5	3.0	1.0	2.0	2.0
Demonstrate critical thinking and reasoning	5.0	3.0	3.0	7.3	5.0	2.0	2.0	5.0

Note. $N = 9$ for academic group and $N = 10$ for industry group in Round 3, and $N = 9$ for academic group and $N = 11$ for industry group in Round 4.

TABLE 2: Round 3 & 4 Median Rankings of Programming Languages for Introductory Computer Science

Programming Language	Round 3				Round 4			
	Academic Group		Industry Group		Academic Group		Industry Group	
	Median	IQR	Median	IQR	Median	IQR	Median	IQR
Assembly Language	-	-	10.0	4.3	-	-	11.0	9.0
C	4.0	2.5	7.0	4.8	4.0	4.0	7.0	5.0
C#	6.0	2.5	4.5	5.8	8.0	3.0	4.0	3.0
C++	2.0	2.0	5.5	4.0	2.0	1.5	6.0	4.0
HTML5	-	-	5.5	5.8	-	-	6.0	4.0
Java	1.0	1.5	4.5	6.8	1.0	2.0	3.0	7.0
JavaScript	7.0	3.5	3.0	2.3	6.0	3.5	3.0	4.0
PHP	6.0	2.5	6.0	7.3	6.0	2.0	9.0	5.0
PL/SQL	-	-	8.0	4.3	-	-	8.0	4.0
Python	4.0	2.5	3.0	5.3	4.0	1.0	3.0	4.0
Ruby	6.0	2.0	9.5	3.3	6.0	1.0	9.0	6.0
Shell	-	-	9.0	4.8	-	-	8.0	3.0

Note. $N = 9$ for academic group and $N = 10$ for industry group in Round 3 and $N = 9$ for academic group and $N = 11$ for industry group in Round 4.

The industry experts also fell short of the agreement threshold in their rankings but achieved statistical significance in their rankings for 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 reproduced for Round 4. The coefficient of concordance values for each category were included and explained in the subsequent survey so the participants 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. 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 surveys. Round 4 rankings for competencies by both groups are presented in Table 1. The academic group made only slight changes to their rankings for competencies from Round 3. The situation was similar for the industry group's rankings, though to a reduced extent. Most items experienced a decrease in IQR, again pointing to less variation in competency rankings.

The Round 4 results for programming languages are shown in Table 2. The academic group changed little in their rankings from Round 3 to Round 4. Java remained the top language, (median rank = 1.0, IQR = 2.0), followed by C++ (median rank = 2.0, IQR = 1.5). The industry group had a few more noteworthy changes in their rankings of programming languages. 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 (median rank = 11.0) but experienced a sizable increase in variability (IQR = 9.0) among its rankings.

The final round rankings for assessments by each group are shown in Table 3. Again, the academic group exhibited little difference in their ranked lists. Lab exercises were deemed the most important assessment by the group (median rank = 1.0, IQR = 1.5), followed by smaller programming activities (median rank = 2.0, IQR = 1.0). 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.

Table 3: Rounds 3 & 4 Median Rankings of Assessments for Introductory Computer Science

Assessment	Round 3				Round 4			
	Academic Group		Industry Group		Academic Group		Industry Group	
	Median	IQR	Median	IQR	Median	IQR	Median	IQR
Case Studies	9.0	4.5	6.5	2.3	8.0	2.5	7.0	1.0
Code Reviews	6.0	3.0	4.5	2.0	6.0	3.0	5.0	4.0
Concept Questions	5.0	2.5	6.0	5.8	4.0	3.5	5.0	3.0
Final Exams	7.0	3.0	8.0	3.0	8.0	3.5	9.0	3.0
Threaded Discussions	10.0	1.5	9.0	4.3	10.0	1.0	11.0	3.0
Interviews with Professionals	10.0	1.5	8.5	4.0	11.0	3.0	9.0	4.0
Lab Exercises	2.0	2.0	4.0	6.8	1.0	1.5	3.0	2.0
Quizzes	6.0	3.5	8.5	7.5	6.0	2.5	8.0	8.0
Small Program Activities	2.0	1.5	3.0	2.3	2.0	1.0	1.0	4.0
Team Program Assignments	4.0	3.5	6.0	5.5	3.0	2.5	6.0	4.0
Term Projects	6.0	4.5	3.0	8.3	6.0	3.0	4.0	5.0

Note. $N = 9$ for academic group and $N = 10$ for industry group for Round 3 and $N = 9$ for academic group, and $N = 11$ for industry group for Round 4.

Kendall's coefficient of concordance (W) tests were again conducted. Consensus 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 those from industry. Kendall's W values again showed statistically significant ranked lists 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 conformity in their rankings but this time achieved statistical significance in their lists 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 and the lists lacked statistical significance ($W_{IL} = 0.12$, $r_{IL} = 0.02$, $p = 0.25$).

CONCLUSIONS

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. Professionals in academia and industry can provide invaluable input on the content, and though their interests are varied, there can be similarity on recommended course components such as competencies, programming languages, and assessments. See Table 4 for a list of the competencies and Table 5 for the assessments suggested by the experts in this study.

The experts recommended a CS course that provides students with a focus on programming and SE process along with training in professional soft skills, such as problem solving, critical thinking, and teamwork. These same attributes were identified by the National Association of Colleges and Employers (2016) as being most important for career readiness. Those designing curriculum for CS and related fields should focus on helping students to develop these abilities. These experts also recommended that assessments be based on the opportunity to learn by doing; in the form of smaller and team programming activities, lab exercises, 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 types of assessments are very much in line with the recommendations of Crawley, Malmqvist, Östlund, Brodeur, and Edström (2014). Interestingly, the assessments recommended by these experts seemingly point more to an introductory course in SE, other than one in CS.

Table 4: Top Recommended Competencies for Introductory Computer Science by Both Groups (Unranked)

Competency
Demonstrate problem solving
Demonstrate critical thinking and reasoning
Write functioning procedural programs employing programming fundamentals
Describe process and practices in Software Engineering
Demonstrate teamwork and interpersonal group skills
Write functioning 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 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 (Enbody, Punch, & McCullen, 2009). Though visual programming languages like Alice, Greenfoot, and Scratch were not known to many of the participants in this study, an increasing number of experts in the literature recommend they should continue to be considered to introduce concepts in programming before transitioning to a language like Java or Python (Daly, 2011; JTFCC, 2013; Malan & Leitner, 2007).

TABLE 5: Top Recommended Assessments for Introductory Computer Science by Both Groups (Unranked)

Assessment
Smaller programming assignments
Lab exercises
Concept questions
Term projects
Code reviews
Team programming assignments

A suggestion for additional 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. Finally, most academic programs have industry advisory groups that are excellent resources to provide this level of detail and for recommendations aimed at continuous improvement.

***Simon G. Sultana**, Ph.D. is an Associate Professor in the Department of Computer Science and Mathematics at Fresno Pacific University, California.*

***Philip A. Reed, Ph.D.** is a Professor in the Department of STEM Education and Professional Studies at Old Dominion University, Norfolk, VA, and is a member of the Beta Chi Chapter of Epsilon Pi Tau.*

REFERENCES

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

Anderson, G., Ferro, D., & Hilton, R. (2011). *Connecting with computer science* (2nd ed.). Boston, MA: Course Technology.

Ben Arfa Rabai, L., Cohen, B., & Mili, A. (2015). Programming language use in US academia and industry. *Informatics in Education, 14*(2), 143-160.

Boone, H. N., Jr., & Boone, D. A. (2012). Analyzing Likert data. *Journal of Extension, 50*(2), 1-5. Retrieved from http://www.joe.org/joe/2012april/tt2.php

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.

Crawley, E. F., Malmqvist, J., Östlund, S., Brodeur, D. R., Edström, K. (2014). *Rethinking engineering education: The CDIO approach* (2nd ed.). New York: Springer.

Dale, N., & Lewis, J. (2016). *Computer science illuminated*. Burlington, MA: Jones & Bartlett Learning.

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

Enbody, R. J., Punch, W. F., McCullen, M. (2009). Python CS1 as preparation for C++ CS2. ACM SIGCSE Bulletin, 41(1), 116-120.

Field, A. (2009). *Discovering statistics using SPSS* (3rd ed.). Thousand Oaks, CA: Sage Publications Inc.

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.

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.

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

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.

Hambrusch, 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.

Hays, D. G., & Singh, A. A. (2012). *Qualitative inquiry in clinical and educational settings*. New York: Guilford.

Joint Task Force on Computing Curricula [JTFCC]. (2001). *Computing curricula 2001*. New York: Author. Retrieved from http://www.acm.org/sigcse/cc2001

Joint Task Force on Computing Curricula [JTFCC]. (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

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.

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.

Kinnersley, B. (n.d.). Collected information on about 2500 computer languages, past and present. Retrieved from http://people.ku.edu/~nkinners/LangList/Extras/langlist.htm

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

Lunt, B., Ekstrom, J., Lawson, E. A., Kamali, R., Miller, J., Gorka, 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.

Malan, D. J., & Leitner, H. H. (2007). Scratch for budding computer scientists. *ACM SIGCSE Bulletin, 39*(1), 223-227.

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.

National Association of Colleges and Employers. (2016). April 2016: Job outlook 2016 spring update. Retrieved from https://www.odu.edu/content/dam/odu/offices/cmc/docs/ nace/2016-spring-update.pdf

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

O'Grady, S. (2013, April 4). Academia and programming language preferences [Blog post]. Retrieved from http://redmonk.com/sogrady/2013/04/04/academia-and-programming-languages/

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.

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. (2015). The RedMonk programming language rankings: June 2015. Retrieved from https://redmonk.com/sogrady/category/programming-languages/

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* (pp. 34-55). Bristol, UK: Jessica Kingsley Publishers, Ltd.

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

Schneider, G. M., & Gersting, J. (2015). *Invitation to computer science.* Boston, MA: Cengage Learning.

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.

Shein, E. (2015). Python for beginners. *Communications of the ACM, 58*(3), 19-21.

Sitlington, H. (2015). Using the Delphi technique to support curriculum development. *Education + Training, 57*(3), 306-321.

Stefik, A., & Gellenbeck, E. (2011). Empirical studies on programming language stimuli. *Software Quality Journal*, 19(1), 65-99.

Sultana, S. (2016). *Defining the competencies, programming languages, and assessments for an introductory computer science course* (Doctoral dissertation). Retrieved from http://digitalcommons.odu.edu/stemps_etds/10/

Surakka, S. (2007). What subjects and skills are important for software developers? *Communications of the ACM, 50*(1), 73-78.

U.S. News & World Report. (2015). University Directory. Retrieved from http://www.usnewsuniversitydirectory.com/

Wilhelm, W. J. (2001). Alchemy of the oracle: The Delphi technique. *Delta Pi Epsilon Journal*, 43(1), 6-26.

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,* 701-708.

Zaiontz, C. (2013). Real statistic using Excel. Retrieved from http://www.real-statistics.com/ kendalls-w/