Identifying Important and Difficult Concepts in Introductory Computing Courses using a Delphi Process

Ken Goldman†, Paul Gross†, Cinda Heeren‡, Geoffrey Herman‡,
Lisa Kaczmarczyk∗, Michael C. Loui‡, and Craig Zilles‡

†kjg,grosspa@cse.wustl.edu, Washington University in St. Louis
‡c-heeren,glherman,loui,zilles@uiuc.edu, University of Illinois at Urbana-Champaign
∗lisak@ucsd.edu, University of California-San Diego

Abstract

A Delphi process is a structured multi-step process that uses a group of experts to achieve a consensus opinion. We present the results of three Delphi processes to identify topics that are important and difficult in each of three introductory computing subjects: discrete math, programming fundamentals, and logic design. The topic rankings can be used to guide both the coverage of student learning assessments (*i.e.*, concept inventories) and can be used by instructors to identify what topics merit emphasis.

I. Introduction

Developing tools for assessing student learning in computing is known to be a difficult task with a potentially large payoff [9]. Tool development faces the dual challenge of generality and reliability. If we can, however, develop learning assessment tools that are broadly applicable and enable educators to easily and reliably compare different instructional approaches, we can develop best practices for teaching computing. Furthermore, such assessment tools can motivate curricular improvements, as they permit educators to compare the effectiveness of their current approaches with these best practices.

The potential for good assessment tools is clear from the impact of the Force Concept Inventory (FCI), a multiple-choice test designed so that students must choose between the Newtonian conception of force and common misconceptions. In the last two decades, the teaching of introductory college physics has undergone a revolution that has been both motivated and guided by the FCI [10]. The FCI demonstrated that even students who had excelled on conventional examinations failed to answer the simple, conceptual questions on the FCI correctly. This failure exposed fundamental flaws in instruction. The results of administrations of the FCI to thousands of students led physics instructors to develop and adopt "interactive engagement" pedagogies [7]. Due to the impact of the FCI, "concept inventory" (CI) tests are being actively developed for a number of science and engineering fields (*e.g.*, [5]).

Unfortunately, there remains a notable lack of rigorous assessment tools in computing. As a result, there is currently no way to rigorously compare the impact on student learning of the broad range of creative practices developed by the computing education community.

We hope to help replicate the physics education revolution in computing education through the development of CIs for computing courses. We are currently working toward CIs in three introductory subjects: programming fundamentals (CS1), discrete mathematics, and logic design. We are following the same four-step process used by other developers of CIs [5].

1. Setting the Scope: A CI is typically administered as both a *pre-test* at the beginning of a course and a *post-test* at the end, to measure the "gain" resulting from instruction, on a select subgroup of representative topics. It is important to emphasize that a CI is not intended to be a comprehensive test of all significant course topics.

As a result, the scope of the test must be determined carefully to include an indicative subset of course topics that are important and that distinguish students who have a strong conceptual understanding from those who do not.

2. Identifying Misconceptions: Students should be interviewed to determine why they fail to understand key topics correctly. These interviews should identify students' misconceptions about these topics. Previous work suggests that only students can provide reliable information about their misconceptions [4].

3. Develop Questions: Using data from Step 2, CI developers construct multiple-choice questions whose incorrect answers correspond to students' common misconceptions.

4. Validation: The CI should be validated through trial administrations. In addition, the reliability of the CI should be analyzed through statistical methods.

In this paper, we report our progress on Step 1 above, for each of the three computing subjects we are focusing on. Because we seek to develop CIs that are widely applicable, we sought the opinions of a diverse group of experts using a Delphi process (described in Section II), an approach used to develop some previous CIs [6], [13].

We present our results in Section III. For each of our three subjects, our experts identified between 30 and 50 key topics. They rated the importance and difficulty of each topic for an introductory course on the subject, with consensus increasing (as demonstrated by decreasing standard deviations for almost all topics) throughout the multi-step Delphi process. From these results, we were able to identify roughly ten topics per subject that achieved consensus rankings for high importance and difficulty.

II. The Delphi Process

A Delphi process is a structured process for collecting information and reaching consensus in a group of experts [3]. The process recognizes that expert judgment is necessary to draw conclusions in the absence of full scientific knowledge. The method avoids relying on the opinion of a single expert or merely averaging the opinions of multiple experts. Instead, experts share observations (so that each can make an more informed decision) but in a structured way, to prevent a few panelists from having excessive influence as can occur in round-table discussions [11]. In addition, experts remain anonymous during the process, so that they are influenced by the logic of the arguments rather than the reputations of other experts.

For each of the three computing subjects, we used the Delphi process to identify key topics in introductory courses that are both important and difficult for students to learn. Specifically, we sought a set of key topics such that if a student fails to demonstrate a conceptual understanding of these topics, then we could be confident that the student had not mastered the course content. These key topics would define the scope of each CI.

Because the quality of the Delphi process depends on the experts, we selected three panels of experts who had not only taught the material frequently, but who had published textbooks or pedagogical articles on these subjects. Within each panel, we strove to achieve diversity in race, gender, geography, and type of institution (community college, four-year college, research university). Clayton [2] recommends a panel size of 15 to 30 experts. Our panel sizes were 21 for discrete math, 20 for programming fundamentals, and 20 for digital logic. Our Delphi process had four phases.

Phase 1. Concept Identification: We asked each expert to list 10 to 15 concepts that they considered both important and difficult in a first course in their subject. For each course, the experts' responses were coded independently by two or three of us, and we constructed topic lists to include all concepts identified by more than one expert. The reconciled lists of topics were used for subsequent phases.

Phase 2. Initial Rating: The experts were asked to rate each topic on a scale from 1 to 10 on each of three metrics: *importance, difficulty,* and *expected mastery* (the degree to which a student would be expected to master the topic

in an introductory course). The third metric was included because some concepts identified in Phase 1 might be introduced only superficially in a first course on the subject but would be treated in more depth in later courses; these concepts would probably be inappropriate to include in a concept inventory. We found that *expected mastery* was strongly correlated ($r \geq 0.81$) with *importance* for all three subjects. Thus we dropped the *expected mastery* metric from Phase 3 and Phase 4.

Phase 3. Negotiation: The averages and inner quartiles ranges (middle 50% of responses) of the Phase 2 ratings were calculated. We provided this information to the experts, and we asked them to rate each topic on the *importance* and *difficulty* metrics again. In addition, when experts chose a Phase 3 rating outside the Phase 2 inner quartiles range, they were asked to provide a convincing justification for why the Phase 2 range was not appropriate.

Phase 4. Final Rating: In Phase 4, we again asked the experts to rate the importance and difficulty of each topic, in light of the average ratings, inner quartiles ranges, and anonymized justifications from Phase 3. We used the ratings from Phase 4 to produce the final ratings.

III. Results

In this section, we report on the mechanics of the Delphi process, which were similar for each of the Delphi processes. Then, in Sections III-A, III-B, and III-C, we present the results of and our observations about the three Delphi processes we conducted: programming fundamentals, discrete math, and logic design, respectively. Finally, we conclude this section with a few concrete suggestions for use in future Delphi processes to identify the most important and difficult course concepts (Section III-D).

We found it rather straight-forward to reconcile the suggested topics into master lists. As an example, the topic suggestions *"Binary numbers, 2's complement", "Two's complement representation", "Unsigned vs. Signed numbers", "The relationship between representation (pattern) and meaning (value)",* and *"Signed 2's complement representation"* were synthesized into the topic

> "Number Representations: Understanding the relationship between representation (pattern) and meaning (value) (*e.g.*, two's complement, signed vs. unsigned, etc.),"

which we abbreviate here, for space reasons, as *Number Representations*. Notably absent from the assembled topic lists in all three subjects were any topics our experts ranked as both "non-important" and "non-difficult," as can be seen graphically for the programming fundamentals topics in Figure 1.

We found that the Delphi process was, in fact, useful for moving toward a consensus. The standard deviations of the responses decreased for almost all topics during each step of the Delphi process. Specifically, 62 out of 64 (programming fundamentals), 71 out of 74 (discrete math), and 90 out of 92 (logic design) standard deviations for the ratings decreased from Phase 2 to Phase 4. Typically, this consensus was achieved when the experts adjusted their rankings modestly. Most (88%) ratings changed by 2 points or less between step 2 and step 4 (no change: 32%, change by 1: 38%, 2: 19%, 3: 7%, 4: 3%, 5+: 2%).

From the importance and difficulty ratings, we computed a single metric with which to rank the topics. We ended computing an **overall ranking** using the distance for each topic from the maximum ratings (importance 10, difficulty 10), the L^2 norm, but found both the sum and product of the two metrics to provide an almost identical ranking. In the tables that follow, we highlight (using boldface) the top N topics by this ranking, selecting an N close to 10 such that there is a separation between the two groups of topics.

Interestingly, we found the ranking computed during Phase 2 of the Delphi process to quite accurately predict the top ranked topics in later phases. For example, in logic design, 10 of the top 11 ranked topics were the same for both Phase 2 and Phase 4. While the average importance and difficulty ratings changed significantly in some

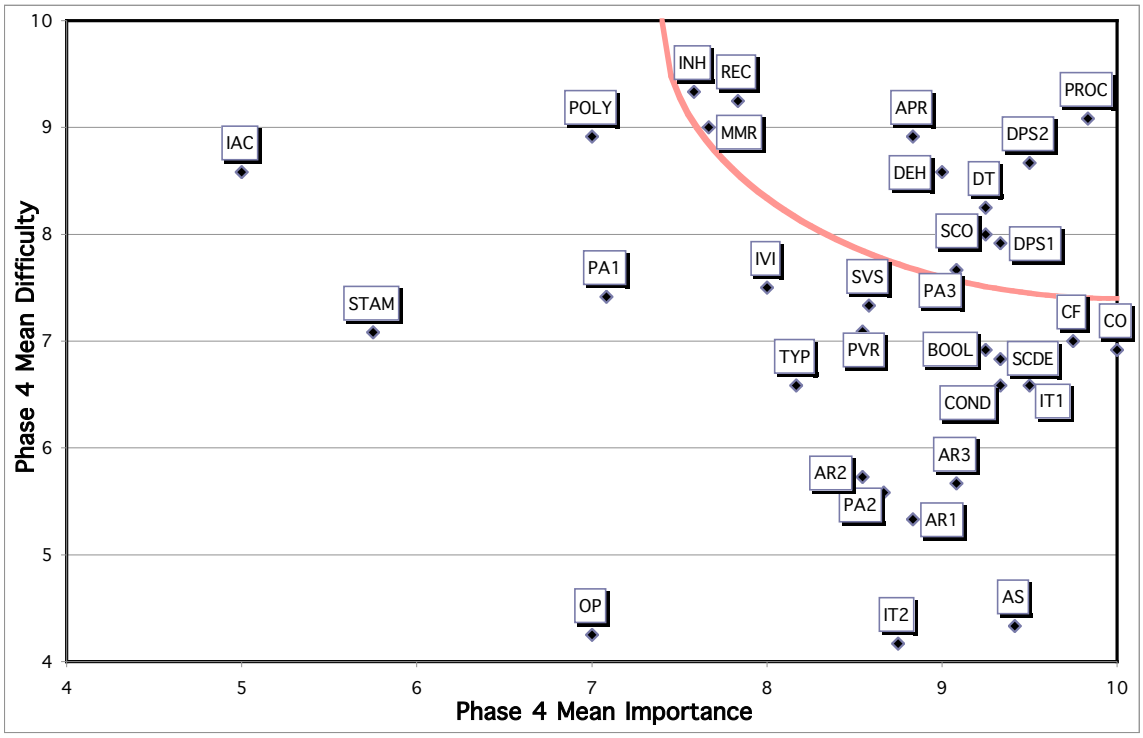

Fig. 1. Programming fundamentals topics plotted to show the selection of the highest ranked. *Topics were ranked by their distance from the point (10, 10); the quarter circle shows the separation between the top 11 topics and the rest.*

cases — 0.5 to 1.0 point shifts are not uncommon — these shifts occurred predominantly in the topics that made up the middle of the rankings.

A. Programming Fundamentals (CS1) Results

Finding consensus on the most important and difficult concepts for a CS1 course is inherently challenging given the diversity of approaches in languages (*e.g.*, Java, Python, Scheme), pedagogical paradigms (*e.g.*, objects-first, objects-late), and programming environments used. These factors influence the perceptions of topics that are important and difficult.

In soliciting volunteers, we contacted experts with a variety of backgrounds and experience, placing no explicit emphasis on a specific language, paradigm, or environment. In Phase 1, experts were asked to identify the languages they are teaching (and have taught with) and the development environments they use. The majority cited object-oriented languages, which is reflected in the strong representation of procedural and object-oriented topics in Figures 2 and 3.

Indicated inside the semicircle of Figure 1 and in grey in Figures 2 and 3 are the top 11 topics using the L^2 metric described in Section III. Despite the object-oriented influence, only one of these 11 topics (inheritance) is exclusively related to object-oriented paradigms or languages. This result implies that a programming fundamentals inventory based on the other 10 topics could be broadly applicable.

With the exception of inheritance, these topics were in the top 11 topics over the first three phases. From Phase 3 to Phase 4, *Syntax vs. Semantics (SVS)* fell out of the top to be replaced by *Inheritance (INH)*. The change in L^2 distance is very slight, from 2.81 to 3.01, but it is enough to move it outside the boundary. This change may relate to comments from experts arguing that the topic is not difficult:

> "Other than getting used to the terminology 'syntax'/'semantics,' this simply isn't that difficult of a

> concept, and I definately [sic] feel students master the basic concept that a syntactic statement has some underlying semantic meaning." *IMP = 10, DIFF = 2*
>
> "Students readily understand that there IS a difference between a textual code segment and its overarching purpose and operation. The true difficulty is for them to reconstruct the purpose." *IMP = 8, DIFF = 5*

With the exception of inheritance and memory models, the top topics dominantly relate to the design, implementation, and testing of procedures. This is not surprising given that universally across language and approaches, introductory computer science students design, implement, and test algorithms in some respect. Other topics tend to relate to either syntactic elements and their semantic meanings, and program/class design, which are more specific to language and approach choice.

Although we have identified the top 11 topics based on average expert ratings, a purpose of the Delphi process is to reach a *consensus* opinion among experts. This consensus is quantitatively described by the average rating and corresponding standard deviation. A smaller standard deviation implies a stronger consensus, while a larger standard deviation indicates expert disagreement.

The standard deviations from Phase 4, shown in Figures 2 and 3, do not all indicate a strong consensus. We can characterize topics with a weak consensus (those with standard deviation of 1.5 or greater) for importance into two types: *outlier* or *controversial*. Outlier topics (PA1, IT2, TYP, PVR, REC) are defined as those having a strong consensus with most experts but include one or two strongly dissenting rankings. For example, 14 of the 15 importance ratings for the *types* (TYP) concept were 8 or higher in Phase 3. The expert explanation given for the other rating in Phase 3 was:

> "There is too much emphasis on types in CS1." *IMP = 4, DIFF = 7*

In Phase 4, the same expert provided an importance rating of 4 while all other ratings were 8 or higher. Removing this rating, the standard deviation for the importance of the types concept in Phase 4 drops from 1.5 to 0.7, a significant change when considering if a consensus has been achieved. It appears that this response is an outlier from what is otherwise a strong consensus. Other dissenting explanations for outliers commonly cited language differences when an explanation was given. We will discuss language difference issues later in this section.

We use the term controversial to refer to topics that have clustering clustering around two ratings rather than a single rating, such as *inheritance (INH)* and *memory models(MMR)*. Inheritance, for example, in Phase 4 had ratings clustered around 6 and 9. This controversy can be seen in the following expert explanations from Phase 3:

> "Though the goal of good OO design is something that is quite difficult, in the context of CS1, we focus more on understanding more basic examples of inheritance." *IMP = 6, DIFF = 6*
>
> "Inheritance & polymorphism are the fundamental concepts of OO design that make it different from procedural programming." *IMP = 10, DIFF = 10*

One would expect the explanations from Phase 3 to have some effect on the Phase 4 results. Such an effect was not apparent in most cases. This is due in part to the types of explanations provided by experts for their ratings falling outside the range. In total, 98 out of 896 ratings were provided outside the middle range for importance and difficulty and 68 explanations were given for these ratings (65% explanation rate as 30 explanations were not given). Explanations could be classified into two groups: detailed and simple.

Detailed statements articulated the reasoning for why a rating was provided and argued for the viewpoint. About two thirds of the explanations given for importance and difficulty ratings were detailed. For example:

> *Control Flow (CF):* "I think 5-7 underestimates the difficulty of adopting a clear, consistent mental model across the entire set of control constructs students are exposed to in CS1. Even clearly modelling loops, conditionals, and function calls is challenging (and twines with concepts of scope, parameter-passing, etc.) that I would rate above 5-7. If we throw in tracing dynamic dispatch in an object-oriented language here as well, this is more like a 9." *IMP = 10, DIFF = 8*

> *Design and Problem Solving I (DPS1):* "I seem to have flipped to the opposite side of the range on this one. I find that really good design and decomposition takes a lot of practice and is not one which is mastered in the first semester (though we certainly try)." *IMP = 9, DIFF = 9*

Simple statements either stated flat opinions and did not provide reasoning for them or stated no opinion at all relating to the topic that was rated. This group also includes statements that reflected on the survey itself. For example:

> *Polymorphism (POLY):* "I think it is very important." *IMP = 9, DIFF = 10*

> *Parameters/Arguments II (PA2):* "I don't see this as difficult for the students." *IMP = 9, DIFF = 3*

> *Recursion (REC):* "I suppose I should look back and see if I'm claiming anything to be difficult. I seem to be regularly on low end of range." *IMP = 6, DIFF = 6*

There were an additional 8 explanations for ratings out of the expected mastery range and 8 comments given with ratings that were not outside any middle 50% range. For the latter comments, experts tended to criticize the grouping of concepts for a topic or criticize the structure of the survey. For example:

> *Abstraction/Pattern Recognition and Use (APR):* "WE [sic] do not like the combination of abstraction and pattern recognition used with this example. We teach abstraction as the use of functions to solve problems" *IMP = N/A, DIFF = N/A*

Across all these classifications, 15 comments related to language or curriculum differences as a factor. Some specific topics did not apply to all languages and approaches or their importance and difficulty were dependent on the chosen language (PA1, PA2, SCO, INH, POLY, PVR, AR1, AR3, IAC). In some topic descriptions examples were provided that referred to elements specific to some languages (SCO, IAC). For instance the *primitive vs. reference variables (PVR)* topic was not meaningful for Python, as indicated by the response:

> "We use Python. There are no value variables. All types (both primitive and user-defined) are identified with reference variables." *IMP = N/A, DIFF = N/A*

We emphasize that we did not actively choose to include a number of topics which may not be universal, but these topics and examples are reflection of our experts' opinions from Phase 1.

All explanations were given by the participants in Phase 3, but it is not the case that all Phase 3 participants continued to Phase 4 of the programming fundamentals Delphi process. In fact, with 18 experts for Phase 2, 15 for Phase 3, and 12 for Phase 4, only 10 of these experts participated in these three phases. It is important to note the same 11 most important and difficult topics emerge even if the data is limited to the responses from 10 participants who completed the entire process. Additionally the maximum absolute difference in standard deviations for a particular topic is small (0.35) when comparing the ratings of all participants with ratings of those 10 participants.

We do not have feedback to account for the diminishing participation in the programming fundamentals Delphi process, but we speculate that Phase 4 starting near the beginning of the fall semester had an impact. We did not observe the same monotonically decreasing participation rates in either the discrete math or logic design Delphi processes, both of which completed earlier in the summer.

B. Discrete Math Results

Our Delphi results, shown in Figures 4 and 5, comport with the organization and coverage of core Discrete Structures concepts in the ACM Computing Curricula 2001 (CC2001) [14]. In general, the topics obtained in Phase 1 partitioned into the areas labeled "core topics DS1-DS6" in CC2001. In addition, our experts nominated topics related to algorithms. These algorithmic topics are absent from the Discrete Structures portion of CC2001 and from the subsequent SIGCSE Discrete Mathematics Report [8], but rather, they appeared as "core topics AL1 and AL5" in the algorithms portion of CC2001. Note that the Phase 1 responses were *suggested* important and difficult topics. The ratings in subsequent phases of the project clarified this part of the picture considerably.

Nine of the ten top ranked topics in Phase 4 (using the L^2 norm metric described in Section III) are included in the CC2001 Discrete Structures core. The tenth, "order of growth," is a CC 2001 core Algorithms topic, but appears to be frequently covered in discrete math courses — at least those taught for computing students — in preparation for other introductory computing courses.

The experts agreed on the importance and difficulty of reasoning skills in a discrete math course: they rated all four topics in the Proof Techniques category among the top ten. In particular, "Proof by induction" was rated as the most important of the 37 topics in Phases 2, 3, and 4; it was consistently rated among the most difficult topics. Two closely related topics "Recursive definitions" and "Recursive problem modeling" were also among the top 10 ranked topics. Among topics not selected as *both* important and difficult are the foundational topics in Logic: "Conditional statements" and "Boolean algebra." They *are* recognized among the very most important, however, because they are prerequisite to a deep understanding of proofs. The most difficult topics, "Algorithm correctness" in Phase 3 and "Countable and uncountable infinities" in Phase 4, were not deemed important enough to be included among the top 10.

Topics in the Algorithms and Discrete Probability categories had the least consensus (largest standard deviation) in their importance ratings. The inclusion of these topics depends on the context of the discrete math course in the local computer science curriculum: these topics may be assigned instead to other courses in the curriculum. All topics in Algorithms received high ratings on difficulty but low ratings on importance for a first course in discrete math.

From Phase 3 to Phase 4, the importance ratings of six topics decreased significantly: "Algorithm analysis," "Inclusion-exclusion," "Combinatorial proof," and all three topics in the Discrete Probability category. It appears that the panelists were persuaded by three kinds of comments provided during Phase 3: appropriateness for a first course, relevance to computer science, and curricular organization.

According to the panelists, some topics are generally important in discrete math, but a deep treatment may be inappropriate in a first course. For example, for the "Inclusion-exclusion" topic, one expert wrote,

> "A fair bit of computer science can be done without deep understanding of this concept. It is important enough that the first discrete math course should establish the idea, but much of its development and application can be left to more advanced courses."

Other important topics in discrete math may lack relevance to computer science, and they might be omitted from a course that supports a computer science curriculum. For example, for "Combinatorial proof," one expert wrote,

> "Not of much use in computer science, in my view."

The topic of "Algorithm analysis" is an interesting case. The following three comments capture the diversity of perspective among our experts:

> "This topic is central to computer science."
>
> "This is a vitally important topic for computer science students, but it isn't essential that "the first discrete math course" teach it. It can also be taught in an analysis of algorithms course that builds on the discrete math course."
>
> "I wouldn't necessarily put algorithms in the first course at all. I would consider proving correctness at least as important as determining the running time, however."

The decrease in deemed importance prompted by such comments, together with the fact that the closely related topic "Order of Growth" *is* considered to be among the most important and difficult, reflects the view of a first course in Discrete Math as foundational. Follow-on skills, particularly those related to programming, for example "Algorithm Analysis," can be taught in context in future courses.

Finally, local curricular decisions may determine whether a topic in algorithms or probability might be covered in a discrete math course. For instance, for topics in Discrete Probability, two experts wrote,

> "Less appropriate for a first course in discrete mathematics. This is more appropriate for an early probability course. Better to spend the time on other more appropriate and difficult topics."
>
> "It's not that probability is not important, it's just that how can you cover it as well as all the other important topics of discrete math? I think it's better served in a special course."

Though we are not surprised that these findings support the prescription of the CC2001 report for a course in Discrete Math, we observe that they also serve to fine tune that prescription. The topics, when ranked strictly by importance, suggest a course heavy in reasoning on one hand, and supporting specific skills and structures required in other courses on the other.

C. Logic Design Results

The data collected via the logic design Delphi process can be found in Figures 6, 7, and 8. For the most part, we found that importance rankings had higher standard deviations — largely attributable to differences in course coverage — than the difficulty rankings. A notable exception was the high standard deviation for the difficulty of "Number representations," where some faculty asserted that their students knew this material coming into the class, whereas others found their students having some trouble. This result is perhaps attributable to differences in student populations between institutions. Some expert comments concerning the "Number representations" concept demonstrate this variation in expectations:

> "Most students have already done this in early high school and it is almost completely a review of their general math class work."
>
> "Signed 2's complement is a concept that seems to be very difficult for some students to grasp. They naturally think signed magnitude. While they may be able to do the conversion relatively easily, their difficulty in understanding 2's complement arithmetic and particularly overflow indicates that they really don't easily grasp the relationship between the representation and meaning."
>
> "Students know this material long before they reach this course."

When a change in the mean occurred between Phases 3 and 4 (as identified below), we could attribute it to expert comments. Five types of comments seemed to have the most consistent and largest affects upon the means:

No longer important: The importance of some topics decreased as experts asserted that certain topics were no longer important due to advances in design tools and design techniques. Many of these topics were introduced

when the dominant means of building digital systems was composing small-scale integration (SSI) and medium-scale integration (MSI) chips. Topics specific to that design style (8, 9, 10, 14, 19, 25, 28) were argued to be no longer in the context of modern design tools (VLSI or FPGA). The comments experts made about topics 8, 14, and 28 are representative of these arguments.

> "Don't cares are largely a left-over from SSI days when each gate cost a lot. It's more important today to design correct and robust circuits, so it's better to emphasize always knowing exactly what the circuit will do, even in the presence of glitches, changed flip-flop values due to noise, etc. Don't cares should probably only be mentioned lightly, if at all." (Topic 8: Incompletely specified functions)

> "Nice to know, but rarely used in the tradition way (truth-table d's) because the HDLs [Hardware Description Languages] don't have a convenient means of specifying them, except in case statements. Now, if this question were tied in with that feature of HDLs, I would increase my 'importance' rating to 8-9." (Topic 8: Incompletely specified functions)

> "This topic should be banned from intro courses! It's a left-over from the days when we had a lab stocked with SSI/MSI components and needed to use them efficiently. It has no relevance today in the era of FPGAs and ASICs. Students barely understand the main uses of muxes and decoders; showing them tricky uses of those components just confuses them, and does not improve their understanding of those components or of logic design in general. It's like teaching a beginning drummer how to play drums with the sticks in his mouth rather than in his hands – cute trick, but rarely relevant." (Topic 14: Application of MSI)

> "This was nice material to know and use in practice before the advent of PLDs, but it's not very useful now." (Topic 14: Application of MSI)

> "A left-over from SSI/MSI days when we tried to minimize costly gates. Counters today are built with registers and logic. We need to break from past subjects when those subjects are not fundamental to the subject, but are instead tricks of the trade that are no longer relevant." (Topic 28: Race conditions)

Other topics (2, 17, 31) were more generically argued to be obsolete or of diminished importance in modern logic design and therefore also experienced decreases in importance.

> "It's nice to know and easy to test, but students and engineers today use calculators to do this when it's really required. It may be a shame, but I think it's inevitable that they will continue to do so. And the calculators are more accurate, when it really counts." (Topic 2: Conversion between number systems)

> "Modern design does not require this distinction, nor does understanding how digital circuits form the basis for computers. It's a left-over from the 70s/80s. Best to show how gates can form a flip-flop, show how flip-flops can form a register, and then just use registers. Details like latches versus flip-flops can be covered in an advanced course if desired, but there are more important items to cover in an introductory course (*e.g.*, RTL)." (Topic 17: Latches vs. flip-flops)

> "ASMs were a great thing in their day, but it is far more important today to teach HDLs and corresponding design styles that result in well structured and readable code." (Topic 31: ASM charts)

Not important in a first course: The rated importance for topics 6, 10, 12, 16, 17, 22, 23, 27, 31, 33, 41-43, 45, 46 decreased as (in some cases multiple) experts argued that these topics are not appropriate for a first course, in spite of their importance. Experts typically mentioned three reasons for why a topic is not important in a first course on Digital Logic Design:

1. complete understanding of a given topic is not necessary,

> "I rank the importance a bit lower, since it is possible to follow the methods without fulling understanding the concepts, particularly for duals. In an introductory course, the theory for this topic may be less critical." (Topic 6: Complementation and duality)

> "While concepts of adder design are important, the specifics of CLA implementation are not necessary in an introductory course." (Topic 16: Carry lookahead adder)

2. a given topic is not essential in a first course and other topics should be taught instead, and

> "Advanced topic, not suitable for an intro course. Better to move on to RTL design so that students leave an intro course with a clearer understanding of the power and relevance of digital design." (Topic 27: Race conditions)

> "Advanced topic, handled automatically by tools. Better to focus on design and test in intro course, leave technology-specific details to more advanced courses." (Topic 46: Clock distribution)

3. teaching multiple methodologies for a given topic can be confusing.

> "An intro course should show one way of doing things first, and not immediately confuse students with multiple ways." (Topic 22: Mealy vs. Moore)

> "I believe an intro course can be a solid course and not even discuss mealy vs. moore. It's a nice topic but should be skipped if the students are not solid with one type (whichever is done first)." (Topic 22: Mealy vs. Moore)

One comment bridged both the second and third types of comments.

> "For an introductory course, it's sometimes best to show one way of doing things, and move on to the next subject, so students get the big picture. If a course spends too much time showing 2 or 3 different ways of doing each thing, the course can't get to RTL design, which is really more relevant today than logic-level manipulation. I completely understand the need to have a solid base, so we shouldn't rush through the lower-level material. But at some point we have to say enough is enough." (Topic 10: Minimal POS)

Important for future learning: Two subjects (11, 32) increased notably in importance when experts argued that the subjects were important for future learning.

> "2-level design is often the real easy part of the course, which can be done pretty much by rote: there are good programs available. Multilevel synthesis introduces the student to the type of thinking they need to develop - which I view as critical for a college level course.," (Topic 11: Multilevel synthesis)

> "This topic is very important in order to get to the next level of digital logic design, *i.e.*, to design microprocessor circuits." (Topic 32: Converting algorithms to register-transfer statements and datapaths)

Asserted to be hard: Consensus difficulty levels increased for a number of topics (6, 7, 11, 20) when experts asserted that the topics were subtle or challenging for students.

> "I scored [Multilevel synthesis] as difficulty 10 on my original submission - I have brought it down to 9. It is very difficult because most students are not comfortable with multiple levels of application of the distributive law in any algebra." (Topic 11: Multilevel synthesis)

> "[State transitions is] similar to recursion in programming, current state and next state is difficult to understand because it is time dependent." (Topic 20: State transitions)

Solvable by rote: Experts argued that topics 12 and 23-25 were not difficult, as a rote method for solving them could be taught. The following comment is representative.

> "Starting from a sequential circuit, it is a ROTE method to derive the NS table or state diagram. This requires no understanding." (Topic 24: Analyzing sequential circuit behavior)

A number of experts argued against the inclusion of many of the topics under the **Design Skills and Tools** and **Digital Electronics** on the grounds that teaching these topics in an introductory course depends on an overall curriculum decision. Experts argued that most of these topics (35-36,39-43,46) belong in a laboratory focused class where technology-specific information is important or in a laboratory class where a hardware design language is taught. These topics are not essential for students to understand in a "theory" based class.

Comments from **Design Skills and Tools** mostly argued that the importance of some of these topic were strictly tied to whether HDLs should be taught in a first course.

> "I think the importance of this topic depends on the audience for the course. If students will be taking additional courses and using the tools, then introducing them is important. If the students are taking only this digital logic course (as an out-of-major elective, for example) then the time spent on CAD tools can probably be used more productively on other topics." (Topic 35: Using CAD tools)

> "While experience with CAD tools is important, a first course in digital logic is probably better off emphasizing the first principles. We are also pressed for time to cover all the 'basics'." (Topic 35: Using CAD tools)

> "The importance of this depends on whether or not CAD tools are incorporated into the course." (Topic 39: Debugging, troubleshooting, and designing simulations)

Comments from **Digital Electronics** were mostly concerned that technologically specific topics were unnecessary in an introductory course.

> "An elementary course could avoid discussion of active hi/low - unless there is a lab component." (Topic 40: Active high vs. active low)

> "Deserves some mention, but this is really technology specific detail more relevant in an advanced course." (Topic 41: Fan-in, fan-out)

> "Not a critical topic if the course is focused more on design than lab work. We want students to understand the issues (probably 1-2 homework problems) but not to typically have to worry about those details. Our course is 'theory' and does not have a hardware lab." (Topic 46: Clock distribution)

Furthermore, many concepts are very important in the larger scope of digital logic design as a discipline, but may not be as important for students to learn in an introductory course. Expert comments on these concepts reflected the tension between wanting students to understand these concepts well and questioning the importance of teaching these topics in a first course. Comments from Topic 19 (Asynchronous flip-flop inputs) demonstrates this tension.

> "This topic is essentially a relic of chip-based design styles and not necessarily relevant to all introductory courses."

> "Should be mentioned, but not stressed, in an intro course, which instead should probably just use synchronous inputs in all designs. Whether to use asynchronous or synchronous resets/sets is not agreed upon in the design community – why confuse the poor intro student?"

> "As a digital designer, if you don't understand how these work, and especially the related timing considerations (*e.g.*, recovery time specs) you're dead."

D. Reflections on the Delphi process

For future applications of the Delphi process, we have the following suggestions:

1) Explicitly require that experts provide responses for their ratings outside the middle 50% range in Phase 3. Our implementation of the survey put experts on the honor system to provide explanations where appropriate. Such a system would remind experts who may have forgotten or not noticed that their rating required a response.
2) Provide a clearer explanation, and possibly examples, of the types of explanations that should be provided for a rating outside the middle 50% range. We only asked experts to explain their opinion, and in some cases this did not produce the results we had hoped. Simple explanations are unlikely to be convincing and do not help us achieve a deeper understanding for how an expert chose a rating.
3) Have a mechanism for both accepting critique of the survey design and topic explanations, as well as adapting to it. Experts communicated survey suggestions through the comments in Phase 3 and in email to us. Comments provided in Phase 3 were received too far into the survey to have any effect, and through email it was difficult to determine how and if changes should be made to the survey design. It may be useful to let experts comment on the survey itself and make revisions before beginning Phase 2.
4) Provide instructions for how experts are to deal with language and curriculum differences or design the survey around these differences. For many topics, some experts were asked to rate the importance and difficulty of a concept that had no role in their experience. It is a difficult task to require of the expert that imagine a pedagogical setting where the topic would be of interest and then rate its importance and difficulty. This is done to some degree by allowing experts to choose "not applicable" (N/A) for some topics, but it may be the case that the important and difficult concepts according to their experience are not represented. This issue is more challenging, and striving for the maximum diversity of experts can not be stressed enough, but it may also require the survey to have sections built around specific languages or approaches.

IV. Conclusions and Implications

We believe that a revolution in the way that computing is taught will not occur until educators can clearly see the concrete benefits in student learning that new pedagogies offer. To build learning assessment tools that are sufficiently general to apply to the broad range of curricula and institutions in which computing is taught, it is necessary to identify a representative set of topics for each course that are both undeniably important and sufficiently difficult that the impact of pedagogical improvement can be measured. This paper documents an effort to identify such a set of topics through a Delphi process, where a consensus is drawn from a collection of experts through a structured, multi-step process. From this process, we identified roughly ten topics for each of programming fundamentals (CS1), discrete math, and logic design. These results provide guidance of where we (and others) should focus efforts for developing learning assessments and can also be used by educators as guidance on where to focus instructional effort.

While the consensus importance ratings may be taken at face value (*i.e.*, faculty are unlikely to use an assessment tool that focuses on topics they deem as unimportant), the difficulty ratings should be taken with a grain of salt. If nothing else can be learned from the force concept inventory, it showed that many teachers have an incomplete (at best) understanding of student learning. As such, in the next step of our concept inventory development, we plan to validate the difficulty ratings asserted by our experts through student interviews and, in doing so, wholly expect that some topics that our experts ranked as easy will, in fact, be rife with student misconceptions. As part of this work, we hope to contribute to the literature of computing misconceptions where it exists (programming fundamentals, *e.g.* [1], [12]) and develop one where there is little prior work (discrete math, logic design).

V. Acknowledgments

We thank the experts that participated in the Delphi processes including: (Digital Logic) Gaetano Borriello, Donna Brown, Enoch Hwang, A. Scottedward Hodel, Joseph Hughes, Dean Johnson, Eric Johnson, Charles Kime, Randy Katz, David Livingston, Afsaneh Minaie, Kevin Nickels, Mohamed Rafiquzzaman, Frank Vahid, Ramachandran Vaidyanathan, Zvonko Vranesic, John Wakerly, Nancy Warter, and Sally Wood. (Discrete Math) Doug Baldwin, David Bunde, Mike Clancy, Doug Ensley, Elana Epstein, Andy Felt, Judith Gersting, David Hemmendinger, David Hunter, Richard Johnsonbaugh, David Luginbuhl, Bill Marion, Shai Simonson, Bob Sloan, Leen-Kiat Soh, Allen Tucker, and Doug West. (Programming Fundamentals) Chutima Boonthum, Joseph Chase, Michael Clancy, Steve Cooper, Timothy DeClue, John Demel, Peter DePasquale, Ernie Ferguson, Tony Gaddis, Michael Goldwasser, Cay Horstmann, Deborah Hwang, John Lusth, Sara Miner More, Kris Powers, Kathryn Sanders, and Steve Wolfman.

This work was supported by the National Science Foundation under grants DUE-0618589, DUE-0618598, DUE-618266, and CAREER CCR-03047260. The opinions, findings, and conclusions do not necessarily reflect the views of the National Science Foundation or the authors' institutions.

References

[1] M. Clancy. *Computer Science Education Research (S. Fincher and M. Petre, editors)*, chapter Misconceptions and Attitudes that Interfere with Learning to Program. Taylor and Francis Group, London, 2004.

[2] M. J. Clayton. Delphi: A Technique to Harness Expert Opinion for Critical Decision-Making Task in Education. *Educational Psychology*, 17:373–386, 1997.

[3] N. Dalkey and O. Helmer. An experimental application of the delphi method to the use of experts. *Management Science*, 9:458–467, 1963.

[4] D. Evans. Personal communication, January 2006.

[5] D. Evans et al. Progress on Concept Inventory Assessment Tools. In *the Thirty-Third ASEE/IEEE Frontiers in Education*, Nov 2003.

[6] G. L. Gray, D. Evans, P. Cornwell, F. Costanzo, and B. Self. Toward a Nationwide Dynamics Concept Inventory Assessment Test. In *American Society of Engineering Education, Annual Conference*, June 2003.

[7] R. Hake. Interactive-engagement vs traditional methods: A six-thousand-student survey of mechanics test data for introductory physics courses. *Am. J. Physics*, 66, 1998.

[8] B. Marion and D. Baldwin. Sigcse commitee report: On the implementation of a discrete mathematics course. *Inroads: ACM SIGCSE Bulletin*, 39(2):109–126, 2007.

[9] M. McCracken, V. Almstrum, D. Diaz, M. Guzdial, D. Hagan, Y. B.-D. Kolikant, C. Laxer, L. Thomas, I. Utting, and T. Wilusz. A multi-national, multi-institutional study of assessment of programming skills of first-year cs students. In *ITiCSE-Working Group Reports (WGR)*, pages 125–180, 2001.

[10] J. P. Mestre. Facts and myths about pedagogies of engagement in science learning. *Peer Review*, 7(2):24–27, Winter 2005.

[11] J. Pill. The delphi method: substance, context, a critique and an annotated bibliography. *Socio-Economic Planning Sciences*, 5(1):57–71, 1971.

[12] E. Soloway and J. C. Spohrer. *Studying the Novice Programmer*. Lawrence Erlbaum Associates, Inc., Mahwah, NJ, USA, 1988.

[13] R. Streveler, B. M. Olds, R. L. Miller, and M. A. Nelson. Using a Delphi study to identify the most difficult concepts for students to master in thermal and transport science. In *American Society of Engineering Education, Annual Conference*, June 2003.

[14] The Computer Society of the Institute for Electrical and Electronic Engineers and Association for Computing Machinery. Computing Curricula 2001, Computer Science Volume. http://www.sigcse.org/cc2001/index.html.

ID	Topic	Importance				Difficulty			
	Procedural Programming Concepts	**Phase 2**	**Phase 3**	**Phase 4**	**SD4-SD2**	**Phase 2**	**Phase 3**	**Phase 4**	**SD4-SD2**
PA1	**Parameters/Arguments I:** *Understanding the difference between "Call by Reference" and "Call by Value" semantics*	7.5 (2.3)	7.5 (2.1)	7.1 (2.4)	0.1	7.1 (1.9)	7.4 (1.6)	7.4 (1.2)	-0.8
PA2	**Parameters/Arguments II:** *Understanding the difference between "Formal Parameters" and "Actual Parameters"*	8.3 (1.9)	8.9 (0.9)	8.7 (1.2)	-0.8	6.0 (1.9)	6.1 (1.7)	5.6 (1.7)	-0.2
PA3	**Parameters/Arguments III:** *Understanding the scope of parameters, correctly using parameters in procedure design*	8.9 (1.0)	9.2 (0.9)	9.1 (0.9)	-0.1	7.0 (1.3)	7.7 (1.0)	7.7 (1.1)	-0.2
PROC	**Procedures/Functions/Methods:** *(e.g., designing and declaring procedures, choosing parameters and return values, properly invoking procedures)*	9.5 (0.7)	9.7 (0.6)	9.8 (0.4)	-0.3	8.3 (1.7)	8.9 (0.9)	9.1 (0.8)	-0.9
CF	**Control Flow:** *Correctly tracing code through a given model of execution*	9.6 (0.6)	9.7 (0.5)	9.8 (0.5)	-0.2	6.0 (1.5)	6.3 (1.0)	7.0 (0.6)	-0.9
TYP	**Types:** *(e.g., choosing appropriate types for data, reasoning about primitive and object types, understanding type implications in expressions (e.g., integer division rather than floating point))*	8.1 (1.6)	8.3 (1.3)	8.2 (1.5)	-0.1	6.1 (1.7)	6.6 (1.1)	6.6 (0.5)	-1.2
BOOL	**Boolean Logic:** *(e.g., constructing and evaluating boolean expressions, using them appropriately in the design of conditionals and return expressions)*	8.7 (1.4)	9.0 (1.3)	9.3 (1.0)	-0.4	6.8 (2.1)	6.9 (1.1)	6.9 (1.0)	-1.1
COND	**Conditionals:** *(e.g., writing correct expressions for conditions, tracing execution through nested conditional structures correctly)*	9.2 (0.9)	9.5 (0.6)	9.3 (0.9)	0.0	6.0 (1.6)	6.3 (1.2)	6.6 (0.8)	-0.8
SVS	**Syntax vs. Semantics:** *Understanding the difference between a textual code segment and its overarching purpose and operation*	7.7 (2.2)	8.9 (0.8)	8.6 (0.7)	-1.5	7.1 (1.8)	7.4 (1.9)	7.3 (0.9)	-0.9
OP	**Operator Precedence:** *(e.g., writing and evaluating expressions using multiple operators)*	6.5 (2.8)	6.7 (1.8)	7.0 (1.5)	-1.3	4.7 (1.7)	4.3 (1.1)	4.3 (0.6)	-1.0
AS	**Assignement Statements:** *(e.g., interpreting the assignment operator not as the comparison operator, assigning values from the right hand side of the operator to the left hand side of the operator, understanding the difference between assignment and a mathematical statement of equality)*	9.2 (1.6)	9.8 (0.6)	9.4 (1.2)	-0.4	5.1 (2.2)	4.7 (1.0)	4.3 (0.5)	-1.7
SCO	**Scope:** *(e.g., understanding the difference between local and global variables and knowing when to choose which type, knowing declaration must occur before usage, masking, implicit targets (e.g., this operator in Java))*	8.2 (2.0)	8.9 (1.5)	9.3 (0.8)	-1.3	7.3 (1.8)	7.5 (1.6)	8.0 (0.0)	-1.8
	Object Oriented Concepts								
CO	**Classes and Objects:** *Understanding the separation between definition and instantiation*	9.1 (1.1)	9.7 (0.6)	10.0 (0.0)	-1.1	6.3 (2.4)	6.4 (1.8)	6.9 (1.3)	-1.1
SCDE	**Scope Design:** *(e.g., understanding difference in scope between fields and local variables, appropriately using visibility properties of fields and methods, encapsulation)*	9.0 (1.5)	9.5 (0.7)	9.3 (0.7)	-0.9	6.4 (1.6)	6.9 (1.5)	6.8 (0.8)	-0.8
INH	**Inheritance:** *(e.g., understanding the purpose of extensible design and can use it)*	6.9 (2.8)	6.9 (2.5)	7.6 (1.6)	-1.2	8.1 (1.9)	8.8 (1.2)	9.3 (0.9)	-1.0
POLY	**Polymorphism:** *(e.g. understanding and using method dispatch capabilities, knowing how to use general types for extensibility)*	6.3 (3.0)	6.9 (2.1)	7.0 (1.3)	-1.6	8.3 (2.3)	8.9 (1.6)	8.9 (0.8)	-1.5
STAM	**Static Variables and Methods:** *(e.g., understanding and using methods and variables of a class which are not invoked on or accessed by an instance of the class)*	5.3 (2.6)	5.3 (1.8)	5.8 (1.3)	-1.3	7.3 (2.2)	7.2 (0.9)	7.1 (0.9)	-1.3
PVR	**Primitive and Reference Type Variables:** *Understanding the difference between variables which hold data and variables which hold memory references/pointers*	8.6 (2.2)	8.5 (2.1)	8.5 (2.3)	0.0	7.2 (1.8)	7.2 (1.5)	7.1 (0.8)	-1.0

Fig. 2. Programming fundamentals (CS1) topics rated for importance and difficulty (part 1).

ID	Topic	Importance				Difficulty			
	Algorithmic Design Concepts								
APR	**Abstraction/Pattern Recognition and Use:** *(e.g., translating the structure of a solution to the solution of another similar problem)*	8.2 (2.1)	8.9 (0.9)	8.8 (0.4)	-1.7	8.4 (1.7)	9.0 (1.0)	8.9 (0.5)	-1.2
IT1	**Iteration/Loops I:** *Tracing the execution of nested loops correctly*	8.8 (1.4)	9.3 (0.9)	9.5 (0.5)	-0.9	6.1 (1.9)	6.5 (1.1)	6.6 (0.7)	-1.2
IT2	**Iteration/Loops II:** *Understanding that loop variables can be used in expressions that occur in the body of a loop*	8.4 (2.1)	8.7 (1.5)	8.8 (1.9)	-0.2	5.1 (2.2)	4.2 (1.0)	4.2 (0.9)	-1.2
REC	**Recursion:** *(e.g., tracing execution of recursive procedures, can identify recursive patterns and translate into recursive structures)*	7.0 (2.9)	7.2 (2.3)	7.8 (2.2)	-0.6	8.3 (1.7)	8.9 (1.5)	9.3 (0.9)	-0.8
AR1	**Arrays I:** *Identifying and handling off-by-one errors when using in loop structures*	8.6 (1.6)	8.9 (0.9)	8.8 (0.8)	-0.8	5.6 (2.0)	5.7 (0.9)	5.3 (0.5)	-1.5
AR2	**Arrays II:** *Understanding the difference between a reference to an array and an element of an array*	8.3 (2.3)	9.1 (1.0)	8.5 (1.4)	-0.8	5.8 (2.0)	5.6 (1.4)	5.7 (0.6)	-1.4
AR3	**Arrays III:** *Understanding the declaration of an array and correctly manipulating arrays*	9.1 (1.3)	9.2 (0.9)	9.1 (1.4)	0.1	5.6 (2.3)	5.7 (1.1)	5.7 (0.7)	-1.6
MMR	**Memory Model/References/Pointers:** *(e.g., understanding the connection between high-level language concepts and the underlying memory model, visualizing memory references, correct use of reference parameters, indirection, and manipulation of pointer-based data structures)*	8.2 (2.0)	7.9 (2.3)	7.7 (1.8)	-0.2	8.3 (1.5)	8.5 (1.4)	9.0 (0.7)	-0.8
	Program Design Concepts								
DPS1	**Design and Problem Solving I:** *Understands and uses functional decomposition and modularization: solutions are not one long procedure*	9.2 (1.0)	9.5 (0.7)	9.3 (0.7)	-0.4	7.2 (1.6)	7.3 (1.3)	7.9 (0.8)	-0.8
DPS2	**Design and Problem Solving II:** *Ability to identify characteristics of a problem and formulate a solution design*	9.0 (1.4)	9.6 (0.5)	9.5 (0.5)	-0.8	7.7 (1.6)	8.3 (0.7)	8.7 (0.7)	-1.0
DEH	**Debugging/Exception Handling***: (e.g., developing and using practices for finding code errors)*	8.8 (1.0)	8.9 (0.7)	9.0 (0.0)	-1.0	7.6 (1.8)	8.2 (1.1)	8.6 (0.5)	-1.2
IVI	**Interface vs. Implementation:** *(e.g., understanding the difference between the design of a type and the design of its implementation)*	7.3 (1.7)	7.6 (1.1)	8.0 (0.9)	-0.9	7.4 (2.5)	7.4 (1.1)	7.5 (0.5)	-2.0
IAC	**Interfaces and Abstract Classes:** *(e.g., understanding general types in design, designing extensible systems, ability to design around such abstract types)*	5.7 (3.1)	5.2 (1.5)	5.0 (1.0)	-2.1	8.4 (1.5)	8.5 (0.8)	8.6 (0.7)	-0.8
DT	**Designing Tests:** *(e.g., ability to design tests that effectively cover a specification)*	7.6 (2.6)	8.5 (1.6)	9.3 (0.8)	-1.8	7.6 (2.2)	7.9 (1.5)	8.3 (0.9)	-1.3

Fig. 3. Programming fundamentals (CS1) topics rated for importance and difficulty (part 2).

Topic	Importance				Difficulty			
Logic	**Phase 2**	**Phase 3**	**Phase 4**	**SD4-SD2**	**Phase 2**	**Phase 3**	**Phase 4**	**SD4-SD2**
1. **Conditional statements**: *Understanding the if-then statement, converse, inverse, contrapositive, and biconditional, and vacuous truth (with empty antecedent); negating an implication.*	9.2(1.2)	9.6(0.5)	9.5(0.6)	-0.6	5.5(1.8)	5.2(1.2)	5.3(1.1)	-0.7
2. **Boolean algebra and propositional logic**: *In particular, logical equivalence and the application of distributivity and DeMorgan's Laws.*	8.6(1.2)	8.8(0.6)	8.8(0.4)	-0.8	5.0(2.1)	5.2(1.4)	5.7(0.7)	-1.4
3. **Translation of English statements into predicate logic**: *For example, understanding that a universal quantifier commonly modifies an implication, and an existential quantifier modifies a conjunction.*	**7.8(2.3)**	**8.5(0.8)**	**8.6(1.1)**	**-1.2**	**7.4(1.5)**	**7.1(1.2)**	**7.3(0.9)**	**-0.6**
4. **Quantifiers and predicate logic**: *Including nested quantifiers and negating a statement with quantifiers.*	**7.9(1.5)**	**7.9(0.9)**	**8.3(0.8)**	**-0.7**	**7.2(1.5)**	**6.9(1.0)**	**7.3(0.8)**	**-0.7**
Proof Techniques								
5. Recognize valid and invalid proofs	**8.3(1.9)**	**8.6(0.9)**	**8.5(1.1)**	**-0.8**	**8.1(2.0)**	**8.5(0.5)**	**8.1(1.1)**	**-0.9**
6. Direct proof	**8.9(1.7)**	**8.9(1.4)**	**8.8(1.0)**	**-0.7**	**7.4(2.2)**	**7.6(1.6)**	**7.7(1.0)**	**-1.2**
7. Proof by contradiction	**8.6(2.0)**	**8.9(1.3)**	**8.5(1.1)**	**-0.9**	**8.2(1.8)**	**8.3(1.3)**	**8.2(0.9)**	**-0.9**
8. Proof by induction: *Including induction on integers, strings, trees, etc.*	**9.3(1.9)**	**9.6(0.6)**	**9.7(0.5)**	**-1.4**	**8.1(2.2)**	**8.8(1.5)**	**8.5(0.8)**	**-1.4**
Sets								
9. **Set specification and operations**: *Includes notation, Cartesian product, sets of sets, and power set*	8.6(1.4)	8.5(1.2)	8.2(0.9)	-0.5	4.4(1.8)	4.0(1.1)	3.9(1.1)	-0.7
10. **Proof of set equality**: *In particular, by showing a pair of inclusions.*	7.2(1.8)	7.2(0.8)	7.1(0.6)	-1.2	5.9(1.7)	5.8(1.0)	5.5(1.4)	-0.3
11. **Countable and uncountable infinities**: *Including proofs, and in particular, the diagonalization argument for proving a set is uncountable.*	5.1(2.7)	4.4(1.7)	4.7(1.8)	-0.9	8.7(1.1)	8.8(0.8)	9.0(0.8)	-0.3
Relations								
12. **Definition**: *Understanding of a relation as a subset of a Cartesian Product, and also as a way of modeling paired relationships between items from sets.*	8.1(1.6)	8.1(1.4)	7.4(1.1)	-0.5	4.8(1.9)	4.2(1.0)	4.6(0.7)	-1.2
13. **Properties**: *Reflexivity, symmetry, anti-symmetry, transitivity and the proofs of each.*	7.8(1.6)	7.9(0.8)	7.5(1.1)	-0.5	6.0(1.9)	6.4(1.3)	5.7(0.9)	-1.0
14. **Equivalence relations**: *In particular, proof that a given relation is an equivalence relation.*	7.6(1.9)	7.6(0.6)	7.9(1.2)	-0.7	6.3(1.7)	6.2(1.0)	6.0(1.3)	-0.4
15. **Equivalence classes**: *Specifying and understanding equivalence classes.*	8.2(1.6)	8.2(0.7)	8.1(0.7)	-0.9	6.2(1.7)	6.4(1.0)	6.6(1.1)	-0.6
Functions								
16. **Composition and inverse functions and relations**	8.5(1.3)	8.5(0.9)	8.2(0.9)	-0.4	6.0(2.1)	5.3(1.0)	5.1(1.0)	-1.1
17. **Injections and surjections**: *In particular, proof that a function is (or is not) one-to-one or onto.*	8.1(1.5)	8.1(1.0)	7.5(1.2)	-0.3	6.6(1.5)	6.7(0.9)	6.4(1.1)	-0.4
18. **Order of growth**: *Including Big-O, little-o, theta, and omega, and definitions and proofs thereof.*	**9.0(1.5)**	**8.9(1.9)**	**8.1(2.0)**	**0.5**	**8.2(1.3)**	**8.5(0.8)**	**8.3(1.1)**	**-0.2**

Fig. 4. Discrete math topics rated for importance and difficulty (part 1).

Topic	Importance				Difficulty			
Recursion	**Phase 2**	**Phase 3**	**Phase 4**	**SD4-SD2**	**Phase 2**	**Phase 3**	**Phase 4**	**SD4-SD2**
19. **Recursive definitions**: *For example, of sets, strings, trees, and functions.*	**8.7(1.9)**	**8.8(1.1)**	**8.7(1.3)**	**-0.6**	**7.8(1.9)**	**8.2(1.1)**	**8.0(1.1)**	**-0.8**
20. **Recursive algorithms**: *Quick sort, Merge sort, Towers of Hanoi, etc.*	7.8(3.0)	6.5(2.8)	6.3(2.7)	-0.3	7.9(1.9)	8.3(1.4)	8.1(0.8)	-1.1
21. **Recurrence relations**: *In particular, specifying an appropriate recurrence relation from an algorithm or problem.*	**8.4(1.5)**	**8.2(1.1)**	**8.0(1.4)**	**-0.1**	**8.2(2.0)**	**8.3(0.9)**	**7.7(1.0)**	**-1.0**
22. **Solving recurrences**	7.4(1.9)	7.2(1.5)	6.4(2.0)	0.1	7.3(2.6)	7.4(2.5)	6.7(1.2)	-1.4
Algorithms								
23. **Algorithm correctness**: *Including loop invariants and proof of correctness by induction.*	6.4(3.1)	4.5(2.6)	4.5(2.6)	-0.5	8.5(1.5)	8.9(0.9)	8.6(1.0)	-0.5
24. **Algorithm analysis**: *Including those containing nested loops.*	7.6(3.0)	7.1(3.1)	5.5(2.2)	-0.8	7.2(1.4)	7.9(0.4)	7.5(1.1)	-0.3
25. **Decidability and Halting Problem**	4.6(3.4)	3.3(1.9)	3.3(2.3)	-1.1	8.3(1.7)	8.6(1.1)	8.6(1.1)	-0.6
Counting								
26. **Basic enumeration techniques**: *Including rule of sum and rule of product and an understanding of when each is appropriate.*	**8.9(1.3)**	**9.1(0.9)**	**9.1(0.9)**	**-0.4**	**6.4(2.1)**	**6.4(1.5)**	**6.4(1.4)**	**-0.7**
27. **Permutations and combinations**	**9.1(1.1)**	**9.1(0.9)**	**8.7(1.0)**	**-0.1**	**6.9(2.1)**	**6.9(1.3)**	**7.1(1.3)**	**-0.8**
28. **Inclusion-exclusion**: *Including the ability to diagnose overlapping cases and insufficiency.*	7.4(2.2)	7.6(1.5)	6.4(1.6)	-0.6	7.0(2.3)	6.8(1.6)	6.1(1.9)	-0.4
29. **Combinatorial identities**: *Including the Binomial Theorem.*	7.1(1.7)	7.3(1.0)	6.4(1.4)	-0.3	6.5(2.2)	6.8(1.4)	6.5(1.4)	-0.8
30. **Pigeonhole principle**	7.5(1.9)	5.4(2.0)	5.9(1.9)	0.0	5.8(2.9)	6.1(2.2)	6.1(1.7)	-1.2
31. **Combinatorial proof**: Also called bijective counting argument.	6.4(2.7)	6.1(2.2)	5.0(1.9)	-0.8	7.9(2.0)	8.4(0.5)	7.9(1.3)	-0.7
Discrete Probability								
32. **Probability of the complement of an event**	7.4(2.5)	6.9(2.2)	5.9(2.5)	0.0	5.6(2.4)	5.4(0.7)	5.6(1.4)	-1.0
33. **Conditional probability**	6.5(2.7)	6.1(2.2)	4.9(1.9)	-0.8	7.4(1.9)	7.7(1.1)	7.5(0.7)	-1.2
34. **Expected value**	6.8(3.2)	6.6(2.8)	5.6(2.4)	-0.8	6.7(2.0)	6.8(1.2)	6.9(0.9)	-1.1
Other Topics (Number Theory and Graph Theory)								
35. **Mod operator, quotient, and remainder**	7.3(2.9)	7.8(2.2)	8.0(2.1)	-0.8	3.6(2.1)	2.9(0.9)	3.1(1.6)	-0.5
36. **Fundamental graph definitions**: Including edge, vertex, degree, directed/undirected, etc.	8.8(1.4)	8.8(0.9)	8.7(0.5)	-0.9	3.6(1.7)	3.4(0.9)	3.5(1.1)	-0.6
37. **Modeling by graphs and trees**	8.5(1.5)	8.0(1.8)	7.7(1.7)	0.2	6.8(1.5)	7.1(0.8)	6.6(0.9)	-0.6

Fig. 5. Discrete math topics rated for importance and difficulty (part 2).

Topic	Importance				Difficulty			
Number Representations	**Phase 2**	**Phase 3**	**Phase 4**	**SD4-SD2**	**Phase 2**	**Phase 3**	**Phase 4**	**SD4-SD2**
1. **Number representations**: *Understanding the relationship between representation (pattern) and meaning (value) (e.g., two's complement, signed vs. unsigned, etc.).*	8.6(1.9)	8.4(1.9)	8.7(1.0)	-0.9	4.3(2.0)	4.1(1.5)	4.1(1.4)	-0.6
2. **Conversion between number systems**: *e.g., Converting from binary or hexadecimal to decimal.*	7.7(2.5)	7.8(2.2)	7.5(1.2)	-1.3	3.8(1.8)	3.5(1.9)	3.1(0.9)	-1.0
3. **Binary arithmetic**: *Includes topics such as binary addition, binary subtraction (particularly borrow bits), not including optimized circuits (e.g., carry-lookahead).*	8.2(1.7)	8.2(1.9)	8.3(1.0)	-0.7	5.3(1.9)	4.6(1.3)	4.4(1.0)	-1.0
4. **Overflow**	7.9(1.9)	8.1(1.3)	7.8(1.4)	-0.6	5.6(2.3)	5.5(1.8)	4.7(1.1)	-1.3
Combinational Logic								
5. **Boolean algebra manipulation**: *Use of boolean algebra to minimize boolean functions or perform boolean proof.*	6.6(2.1)	7.1(1.2)	7.1(1.1)	-1.0	7.1(1.6)	7.2(1.1)	7.3(0.9)	-0.8
6. **Complementation and duality**: *Understanding the concepts of complements and duals and not just methods for computing complements and dual.*	6.7(2.4)	7.1(1.5)	6.6(1.3)	-1.0	6.4(1.7)	6.3(1.1)	6.6(0.7)	-0.8
7. **Converting verbal specifications to boolean expressions**	**9.1(1.5)**	**9.6(0.5)**	**9.5(0.5)**	**-0.9**	**6.8(1.2)**	**7.1(1.1)**	**7.5(1.0)**	**-0.3**
8. **Incompletely specified functions (don't cares)**	8.2(1.5)	8.2(1.7)	7.4(1.5)	-0.1	5.5(1.6)	5.6(0.8)	5.6(0.8)	-0.7
9. **Finding minimal sum-of-product expressions using Karnaugh maps**	7.6(1.9)	7.6(1.4)	7.2(1.4)	-0.4	5.6(1.1)	5.3(0.6)	5.1(0.5)	-0.5
10. **Finding minimal product-of-sums expressions using Karnaugh maps**	6.2(2.6)	6.5(2.3)	5.7(2.0)	-0.6	6.3(1.1)	6.4(0.7)	6.0(1.0)	-0.1
11. **Multilevel synthesis**: *Designing combinational circuits with more than two levels.*	**7.2(2.1)**	**7.0(1.3)**	**7.3(1.3)**	**-0.8**	**8.0(1.3)**	**7.7(0.7)**	**7.9(0.8)**	**-0.5**
12. **Hazards in combinational circuits**: *Static and dynamic hazards.*	5.6(2.2)	5.6(1.7)	4.7(1.0)	-0.5	7.1(1.5)	7.4(1.1)	6.6(1.7)	0.1
13. **Functionality of multiplexers, decoders and other MSI components**: *Excludes building larger MSI components from smaller MSI components.*	**9.0(1.4)**	**9.7(0.5)**	**9.6(0.4)**	**-0.6**	**5.7(1.2)**	**5.7(0.8)**	**5.9(0.9)**	**-0.4**
14. **Application of multiplexers, decoders and other MSI components in implementing Boolean functions**: *Excludes building larger MSI components from smaller MSI components.*	7.2(2.8)	7.9(2.3)	6.6(2.2)	-0.4	6.1(1.3)	5.8(0.9)	5.9(1.1)	-0.2
15. **Hierarchical design**	**9.1(1.2)**	**9.4(0.7)**	**9.5(0.6)**	**-0.6**	**6.8(1.2)**	**6.6(0.7)**	**6.6(0.9)**	**-0.3**
16. **Carry lookahead adder**	6.3(1.7)	6.4(1.5)	5.7(1.5)	0.0	7.6(1.1)	7.4(0.5)	7.3(0.6)	-0.6
Sequential Logic								
17. **Difference between latches and flip-flops**: *Understanding the difference in functionality between unclocked latches and clocked flip-flops.*	8.3(2.4)	8.8(1.8)	8.3(1.6)	-0.5	6.1(1.3)	6.4(0.5)	6.3(0.6)	-0.7
18. **Edge-triggered and pulse-triggered flip-flops**: *Understanding the difference between different clocking schemes.*	7.1(2.1)	7.8(1.7)	6.9(1.8)	0.0	6.7(1.4)	6.7(0.6)	6.6(1.0)	-0.6
19. **Asynchronous flip-flop inputs**: *Direct set/clear.*	7.6(1.8)	7.4(1.7)	7.3(1.5)	-0.3	5.6(2.0)	6.2(1.0)	6.4(0.8)	-1.1
20. **State transitions**: *Understanding the difference between the current state and the next state, and how the current state transits to the next state.*	**9.7(0.5)**	**9.7(0.5)**	**9.8(0.4)**	**-0.1**	**7.3(1.4)**	**7.4(0.8)**	**7.6(0.6)**	**-0.7**

Fig. 6. Logic design topics rated for importance and difficulty (part 1).

Topic	Importance				Difficulty			
Sequential Logic, cont.	**Phase 2**	**Phase 3**	**Phase 4**	**SD4-SD2**	**Phase 2**	**Phase 3**	**Phase 4**	**SD4-SD2**
21. **Converting verbal specifications to state diagrams/tables**	**9.4(0.6)**	**9.6(0.6)**	**9.8(0.4)**	**-0.2**	**8.2(1.4)**	**8.3(0.8)**	**8.3(0.6)**	**-0.7**
22. **Difference between Mealy and Moore machines**: *Differences in characteristics, behavior, and timing.*	7.4(1.8)	7.6(1.4)	6.9(1.6)	-0.1	6.3(1.7)	6.4(0.6)	6.0(0.9)	-0.7
23. **State machine minimization**: *Particularly recognizing when two or more states are equivalent.*	5.8(2.0)	5.8(1.6)	5.1(1.6)	-0.2	6.9(1.8)	7.1(1.1)	6.7(1.0)	-0.8
24. **Analyzing sequential circuit behavior**	8.3(2.1)	8.7(2.1)	8.4(1.6)	-0.6	6.8(1.8)	6.7(1.4)	5.9(1.3)	-0.3
25. **Synthesizing sequential circuits from excitation tables**	7.7(2.7)	8.4(1.9)	8.1(2.1)	-0.6	6.7(1.6)	6.5(1.1)	6.1(1.3)	-0.5
26. **Understanding how a sequential circuit corresponds to a state diagram**: *Recognizing the equivalence of a sequential circuit and a state diagram (regardless of implementation).*	**8.4(2.4)**	**9.2(1.0)**	**8.9(1.0)**	**-1.5**	**6.8(2.1)**	**6.9(1.0)**	**6.6(0.6)**	**-1.4**
27. **Relating timing diagrams to state machines, circuits**	**8.9(1.2)**	**9.5(0.5)**	**9.5(0.6)**	**-0.5**	**8.2(1.0)**	**8.1(1.3)**	**8.2(0.8)**	**-0.1**
28. **Race conditions in sequential circuits**: *Knowing how to identify race conditions and how to avoid race conditions when designing a sequential circuit.*	6.0(2.8)	6.3(1.9)	5.1(1.8)	-0.8	8.5(1.0)	8.5(0.5)	8.4(0.6)	-0.4
29. **Designing/using synchronous/asynchronous counters**: *Designing counters with arbitrary count sequences using arbitrary flip-flops.*	6.8(2.2)	7.1(1.6)	6.5(1.6)	-0.4	6.3(1.2)	6.6(0.9)	6.4(0.7)	-0.1
30. **Designing/using shift registers**	7.6(2.1)	8.1(1.0)	7.9(0.9)	-1.1	5.9(1.6)	6.2(0.9)	6.0(0.9)	-0.4
31. **Algorithmic state machine (ASM) charts**: *Separation of datapath and control path in ASM charts.*	6.5(2.8)	6.4(1.9)	5.9(1.7)	-0.8	6.6(1.8)	6.7(0.8)	6.4(0.8)	-1.0
32. **Converting algorithms to register-transfer statements and datapaths**	**7.8(2.3)**	**8.1(1.0)**	**8.4(1.0)**	**-1.3**	**7.8(1.4)**	**7.7(0.7)**	**7.9(0.6)**	**-0.8**
33. **Designing control for datapaths**	**8.1(1.8)**	**8.6(0.9)**	**8.5(1.8)**	**0.0**	**7.6(1.5)**	**7.5(0.7)**	**7.7(0.8)**	**-0.8**
34. **Memory organization**: *Organizing RAM cells to form memory systems (decoding schemes, etc.).*	6.8(1.3)	7.4(1.1)	7.1(0.9)	0.2	6.0(1.5)	6.1(0.9)	6.1(0.9)	-0.5
Design Skills and Tools								
35. **Using CAD tools**	**8.2(2.4)**	**8.9(1.6)**	**8.7(1.4)**	**-0.9**	**7.0(1.6)**	**7.2(1.1)**	**7.0(0.9)**	**-0.6**
36. **Verilog/VHDL vs. programming languages**: *Understanding the differences between hardware description languages and standard programming languages.*	7.4(2.8)	8.3(2.1)	7.6(2.1)	-0.4	6.9(1.8)	6.9(1.2)	7.0(1.1)	-0.7
37. **Programmable logic**: *Understanding the structure/use of programmable logic such as PLAs and FPGAs.*	7.9(2.5)	8.2(1.7)	7.8(1.1)	-0.9	6.5(1.7)	6.4(0.8)	6.2(0.9)	-0.9
38. **Modular design**: *Building and testing circuits as a compilation of smaller components.*	**8.1(2.1)**	**9.2(1.0)**	**8.8(1.8)**	**-0.5**	**6.7(1.5)**	**7.1(1.0)**	**7.1(0.5)**	**-0.9**
39. **Debugging, troubleshooting and designing simulations**: *General debugging skills with a focus on designing rigorous test inputs/simulations for circuits.*	**8.0(2.7)**	**8.8(1.9)**	**8.5(2.0)**	**-0.5**	**8.3(1.3)**	**8.6(0.8)**	**8.8(0.4)**	**-0.7**

Fig. 7. Logic design topics rated for importance and difficulty (part 2).

Topic	Importance				Difficulty			
Digital Electronics	**Phase 2**	**Phase 3**	**Phase 4**	**SD4-SD2**	**Phase 2**	**Phase 3**	**Phase 4**	**SD4-SD2**
40. **Active high vs. active low**	7.4(2.4)	7.6(1.6)	6.9(1.6)	-0.5	5.3(2.2)	5.2(1.2)	5.4(0.8)	-1.2
41. **Fan-in, fan-out**	6.7(2.6)	7.2(1.8)	6.7(1.2)	-1.0	4.7(2.3)	4.6(1.3)	4.9(0.9)	-1.2
42. **High-impedance outputs: tri-state, open-collector, totem-pole**	7.1(2.7)	7.9(1.8)	6.8(1.6)	-0.9	6.1(2.1)	6.0(1.0)	5.5(0.7)	-1.3
43. **DC and AC loading, noise margins**	5.6(2.6)	6.3(1.7)	5.8(1.8)	-0.4	6.8(1.4)	6.6(0.8)	6.2(0.7)	-0.6
44. **Propagation delay, rise/fall time**	7.1(2.1)	7.7(1.1)	7.5(1.1)	-0.9	5.9(2.1)	6.0(1.5)	5.8(1.0)	-1.2
45. **Setup and hold time, metastability**	7.4(2.6)	8.1(1.5)	7.5(1.6)	-0.8	6.6(2.1)	6.6(1.2)	6.5(0.8)	-1.3
46. **Clock distribution, clock skew**	7.1(2.5)	7.7(1.7)	6.6(1.9)	-0.5	6.6(2.0)	6.4(1.3)	6.1(1.2)	-0.9

Fig. 8. Logic design topics rated for importance and difficulty (part 3).