# 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 standardized tests of student learning (*i.e.*, concept inventories) and can be used by instructors to identify what topics merit emphasis.

**Categories and Subject Descriptors:** K.3.2 [Computer and Information Science Education]: Computer Science Education

**General Terms:** Human Factors.

**Keywords:** Curriculum, Concept Inventory, Delphi, Discrete Math, Programming Fundamentals, Logic Design

## 1. 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.*, [4]).

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 [4].

**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 [3].

**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 2), an approach used to develop some previous CIs [6, 13].

*SIGCSE'08,* March 12–15, 2008, Portland, Oregon, USA.
Copyright 2008 ACM 978-1-59593-947-0/08/0003 ...$5.00.

We present our results in Section 3. 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.

## 2. THE DELPHI PROCESS

A Delphi process is a structured process for collecting information and reaching consensus in a group of experts [2]. 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 [1] 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.

## 3. 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 3.1, 3.2, and 3.3, we present the results of and our observations about the three Delphi processes we conducted: programming fundamentals, discrete math, and logic design, respectively. Due to space constraints, we present only a summary of the complete results available in our technical report [5].

We found it rather straight-forward to reconcile the suggested topics into a 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 $N$ near 10 such that there is a clear 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

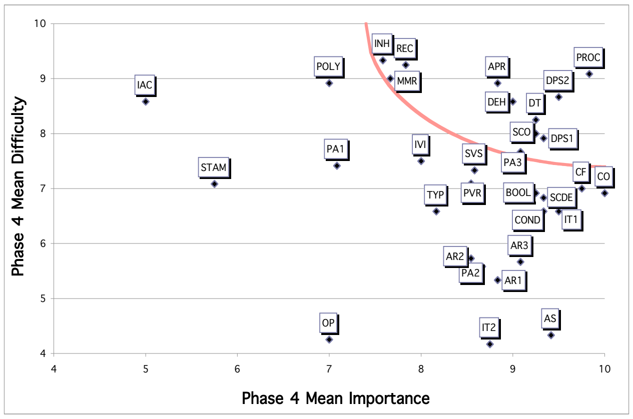

**Figure 1: Programming fundamentals topics plotted to show the selection of the highest ranked.** Topics were ranked by their distance from the point (10, 10); the quarter circle shows the separation between the top 11 topics and the rest.

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 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.

## 3.1 Programming Fundamentals (CS1) Results

Finding consensus on the most important and difficult concepts for a CS1 course is inherently challenging because of the diversity of approaches in languages (*e.g.*, Java, Python, Scheme), pedagogical paradigms (*e.g.*, objects-first, objects-late), and programming tools 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 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 Figure 2.

Indicated in bold in Figure 2 are the top 11 topics using the ranking described in Section 3. Despite the object-oriented influence, only one of these 11 topics 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.

Topics with a weak consensus (those with a large standard deviation, 1.5 or greater) for importance are of two types: outlier or controversial. Outlier topics (*e.g.*, TYP, PVR, REC, see Figure 2) have a strong consensus with most experts but include one or two strongly dissenting rankings. Language differences is a common reason given for dissent, when one is given.

Controversial topics such as *inheritance* (INH) and *memory models* (MMR) have agreement around two ratings rather than a single rating. This controversy can be seen in the following expert explanations for inheritance:

| ID | Topic | Importance | Difficulty |
|---|---|---|---|
| | Procedural Programming | | |
| PA1 | 1. Call by Ref. vs Call by Value | 7.0 (2.5) | 7.4 (1.2) |
| PA2 | 2. Formal vs. Actual Parameters | 8.6 (1.2) | 5.7 (1.7) |
| **PA3** | **3. Parameter scope, use in design** | **9.1 (0.9)** | **7.5 (1.0)** |
| **PROC** | **4. Procedure design** | **9.8 (0.4)** | **9.1 (0.8)** |
| CF | 5. Tracing Control Flow thru program execution | 9.8 (0.4) | 7.0 (0.6) |
| TYP | 6. Choosing/Reasoning about data types | 8.2 (1.5) | 6.6 (0.5) |
| BOOL | 7. Construct/evaluate Boolean expressions | 9.5 (0.7) | 7.1 (0.8) |
| COND | 8. Writing expressions for conditionals | 9.5 (0.5) | 6.7 (0.6) |
| SVS | 9. Syntax vs. Semantics | 8.6 (0.7) | 7.5 (0.5) |
| OP | 10. Operator Precedence | 7.1 (1.5) | 4.4 (0.5) |
| AS | 11. Assignment Statements | 9.5 (1.2) | 4.4 (0.5) |
| **SCO** | **12. Issues of Scope, local vs. global** | **9.4 (0.7)** | **8.0 (0.0)** |
| | Object Oriented Programming | | |
| CO | 13. Difference between Classes and Objects | 10.0 (0.0) | 6.9 (1.4) |
| SCDE | 14. Scope design (e.g., public vs private fields) | 9.4 (0.7) | 6.6 (0.5) |
| **INH** | **15. Inheritance** | **7.6 (1.7)** | **9.5 (0.5)** |
| POLY | 16. Polymorphism | 7.1 (1.4) | 8.9 (0.8) |
| STAM | 17. Static fields and methods | 5.7 (1.3) | 7.3 (0.6) |
| PVR | 18. Primitive vs Reference variables | 8.5 (2.4) | 7.0 (0.8) |
| | Algorithm Design | | |
| **APR** | **19. Abstraction/Pattern recognition and use** | **8.8 (0.4)** | **9.0 (0.4)** |
| IT1 | 20. Tracing nested loop execution correctly | 9.5 (0.5) | 6.6 (0.7) |
| IT2 | 21. Understanding loop variable scope | 8.7 (2.0) | 4.3 (0.9) |
| **REC** | **22. Recursion, tracing and designing** | **7.8 (2.4)** | **9.2 (0.9)** |
| AR1 | 23. Identifying off by one index errors | 8.9 (0.8) | 5.3 (0.5) |
| AR2 | 24. Reference to array vs array element | 8.4 (1.4) | 5.7 (0.7) |
| AR3 | 25. Declaring and manipulating arrays | 9.0 (1.4) | 5.5 (0.5) |
| **MMR** | **26. Memory model, references , pointers** | **7.5 (1.7)** | **8.9 (0.7)** |
| | Program Design | | |
| **DPS1** | **27. Functional decomposition, modularization** | **9.3 (0.6)** | **7.9 (0.8)** |
| **DPS2** | **28. Conceptualize problems, design solutions** | **9.5 (0.5)** | **8.5 (0.5)** |
| **DEH** | **29. Debugging, Exception Handling** | **9.0 (0.0)** | **8.6 (0.5)** |
| IVI | 30. Interface vs Implementation | 8.1 (0.8) | 7.5 (0.5) |
| IAC | 31. Desiging Interfaces, Abstract Classes | 5.0 (1.1) | 8.6 (0.7) |
| **DT** | **32. Designing Tests** | **9.3 (0.8)** | **8.4 (0.8)** |

**Figure 2: Programming Fundamentals (CS1) topics rated for importance and difficulty.** Data reported as: *average (standard deviation).*

> "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*

Many other comments related to language differences as a factor. For example, one expert noted that the *primitive vs. reference variables* (PVR) topic was not meaningful for Python, where "there are no value variables." Given such language and paradigm differences, we are discussing the merits of creating topic-specific sub-inventories, as complements to a "universal" CI.

## 3.2 Discrete Math Results

Our Delphi results, shown in Figure 3, 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 suggested 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].

After Phase 4, nine of the ten top ranked topics (using the $L^2$ norm described in Section 3) 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 and the third most difficult of the 37 topics. Two closely related topics "recursive definitions" and "recurrence relations" were also ranked among the top 10 ranked topics.

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.

## 3.3 Logic Design Results

The data collected via the logic design Delphi process can be found in Figure 4. 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.

When a significant change in the mean occurred between Phases 3 and 4 (as identified below), we could attribute it to expert comments from one of the following five classes:

**No longer important:** Some topics in many logic design courses were introduced when the dominant means of building digital systems was composing medium-scale integration (MSI) chips. Topics specific to that design style (8, 9, 10, 14, 17, 23) were argued as no longer important for targeting modern implementations (VLSI or FPGA).

**Not important in a first course:** The rated importance for topics 6, 10, 16, 22, 28, 36, 41-43, 45, 46 decreased as (in some cases multiple) experts argued that these topics were not appropriate for a first course, in spite of their overall importance.

**Important for future learning:** Two topics (11, 32) increased notably in importance when experts argued that they were important for developing thinking and in preparation for "the next level of digital logic design," respectively.

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

**Solvable by rote:** An expert argued that topics 12 and 24 were not difficult, as a rote method for solving them could be taught.

| Topic | Importance | Difficulty |
|---|---|---|
| Logic | | |
| 1. Conditional statements | 9.5(0.6) | 5.3(1.1) |
| 2. Boolean algebra, prop logic | 8.8(0.4) | 5.7(0.7) |
| **3. English into predicate logic** | **8.6(1.1)** | **7.3(0.9)** |
| **4. Quantifiers and predicate logic** | **8.3(0.8)** | **7.3(0.8)** |
| Proof Techniques | | |
| **5. Recognize valid/invalid proofs** | **8.5(1.1)** | **8.1(1.1)** |
| **6. Direct proof** | **8.8(1.0)** | **7.7(1.0)** |
| **7. Proof by contradiction** | **8.5(1.1)** | **8.2(0.9)** |
| **8. Proof by induction** | **9.7(0.5)** | **8.5(0.8)** |
| Sets | | |
| 9. Set specification and operations | 8.2(0.9) | 3.9(1.1) |
| 10. Proof of set equality | 7.1(0.6) | 5.5(1.4) |
| 11. Countable/uncountable infinities | 4.7(1.8) | 9.0(0.8) |
| Relations | | |
| 12. Definition | 7.4(1.1) | 4.6(0.7) |
| 13. Properties | 7.5(1.1) | 5.7(0.9) |
| 14. Equivalence relations | 7.9(1.2) | 6.0(1.3) |
| 15. Equivalence classes | 8.1(0.7) | 6.6(1.1) |
| Functions | | |
| 16. Composition and inverse | 8.2(0.9) | 5.1(1.0) |
| 17. Injections and surjections | 7.5(1.2) | 6.4(1.1) |
| **18. Order of growth** | **8.1(2.0)** | **8.3(1.1)** |
| Recursion | | |
| **19. Recursive definitions** | **8.7(1.3)** | **8.0(1.1)** |
| 20. Recursive algorithms | 6.3(2.7) | 8.1(0.8) |
| **21. Recurrence relations** | **8.0(1.4)** | **7.7(1.0)** |
| 22. Solving recurrences | 6.4(2.0) | 6.7(1.2) |
| Algorithms | | |
| 23. Algorithm correctness | 4.5(2.6) | 8.6(1.0) |
| 24. Algorithm analysis | 5.5(2.2) | 7.5(1.1) |
| 25. Decidability, Halting Problem | 3.3(2.3) | 8.6(1.1) |
| Counting | | |
| **26. Basic enumeration techniques** | **9.1(0.9)** | **6.4(1.4)** |
| **27. Permutations/combinations** | **8.7(1.0)** | **7.1(1.3)** |
| 28. Inclusion-exclusion | 6.4(1.6) | 6.1(1.9) |
| 29. Combinatorial identities | 6.4(1.4) | 6.5(1.4) |
| 30. Pigeonhole principle | 5.9(1.9) | 6.1(1.7) |
| 31. Combinatorial proof | 5.0(1.9) | 7.9(1.3) |
| Discrete Probability | | |
| 32. Conjunction, complement | 5.9(2.5) | 5.6(1.4) |
| 33. Conditional probability | 4.9(1.9) | 7.5(0.7) |
| 34. Expected value | 5.6(2.4) | 6.9(0.9) |
| Other Topics | | |
| 35. Mod, quotient, remainder | 8.0(2.1) | 3.1(1.6) |
| 36. Fundamental graph definitions | 8.7(0.5) | 3.5(1.1) |
| 37. Modeling by graphs and trees | 7.7(1.7) | 6.6(0.9) |

**Figure 3: Discrete math topics rated for importance and difficulty.** Data reported as: *average (standard deviation).*

## 4. 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

| Topic | Importance | Difficulty |
|---|---|---|
| **Number Representations** | | |
| 1. Number Representations | 8.6(1.0) | 4.3(1.4) |
| 2. Number System Conversions | 7.6(1.1) | 3.1(0.8) |
| 3. Binary Arithmetic | 8.4(1.0) | 4.4(1.0) |
| 4. Overflow | 7.9(1.3) | 4.8(1.1) |
| **Combinational Logic** | | |
| 5. Boolean Algebra Manipulation | 7.0(1.1) | 7.3(0.8) |
| 6. Complementation and Duality | 6.4(1.4) | 6.6(0.9) |
| **7. Verbal Spec. to Boolean Expr.** | **9.5(0.6)** | **7.4(1.0)** |
| 8. Incompletely Specified F'ns | 7.5(1.3) | 5.4(0.9) |
| 9. Finding Minimal SOP | 7.1(1.5) | 5.1(0.6) |
| 10. Finding Minimal POS | 5.7(1.9) | 5.9(1.0) |
| **11. Multilevel Synthesis** | **7.3(1.3)** | **7.9(0.8)** |
| 12. Hazards | 4.8(1.7) | 6.9(1.6) |
| 13. Functionality of MSI | 9.6(0.8) | 5.9(0.8) |
| 14. Application of MSI | 6.4(2.4) | 6.1(1.1) |
| **15. Hierarchical Design** | **9.5(0.6)** | **6.6(0.9)** |
| 16. Carry Lookahead Adder | 5.8(1.7) | 7.3(0.6) |
| **Sequential Logic** | | |
| 17. Latch/Flip-Flop Difference | 8.1(1.9) | 6.4(0.6) |
| 18. Edge-Triggered Flip-Flops | 6.8(2.1) | 6.7(0.9) |
| 19. Asynch. Flip-Flop Inputs | 7.3(1.5) | 6.3(0.9) |
| **20. State Transitions** | **9.8(0.4)** | **7.5(0.7)** |
| **21. Verb. Spec. to State Diagram** | **9.8(0.4)** | **8.3(0.7)** |
| 22. Mealy/Moore Difference | 6.8(1.7) | 6.1(0.9) |
| 23. State Machine Minimization | 5.1(1.8) | 6.9(1.0) |
| 24. Analyzing Sequential Circuit | 8.5(1.5) | 5.7(1.5) |
| 25. Excitation Tab. to Seq. Circuit | 7.9(2.1) | 6.1(1.1) |
| **26. Seq. Circuit and State Diagram Correspondence** | **8.9(0.9)** | **6.6(0.7)** |
| **27. Relate Timing Diag/State Mach.** | **9.4(0.7)** | **8.2(0.9)** |
| 28. Race Conditions in Seq. Circuits | 4.9(2.0) | 8.4(0.6) |
| 29. Counters | 6.6(1.8) | 6.3(1.1) |
| 30. Shift Registers | 7.8(1.0) | 5.9(1.2) |
| 31. Algorithmic State Mach. Charts | 6.3(2.0) | 6.6(0.8) |
| **32. Converting Algorithms to RTL** | **8.5(1.0)** | **8.0(0.6)** |
| **33. Designing Datapath Control** | **8.3(1.8)** | **7.8(0.8)** |
| 34. Memory Organization | 6.8(1.6) | 6.3(1.0) |
| **Design Skills and Tools** | | |
| **35. Using CAD Tools** | **8.4(1.6)** | **6.9(1.0)** |
| 36. HDL vs. Programming Lang. | 7.2(2.4) | 6.9(1.1) |
| 37. Programmable Logic | 7.4(1.6) | 6.1(0.8) |
| **38. Modular Design** | **8.9(1.6)** | **7.2(0.6)** |
| **39. Debug, Test, Troubleshoot** | **8.4(2.2)** | **8.6(0.6)** |
| **Digital Electronics** | | |
| 40. Active High vs. Active Low | 7.0(1.9) | 5.7(0.9) |
| 41. Fan-in, Fan-out | 6.7(1.6) | 4.8(1.2) |
| 42. High-Impedance Outputs | 6.8(1.8) | 5.7(0.8) |
| 43. DC and AC loading | 5.8(2.2) | 6.5(0.8) |
| 44. Propagation Delay | 7.7(1.2) | 5.8(1.0) |
| 45. Setup and Hold Time | 7.4(1.9) | 6.5(0.8) |
| 46. Clock Distribution | 6.6(2.0) | 6.1(1.1) |

**Figure 4: Logic design topics rated for importance and difficulty.** Data reported as: *average (standard deviation).*

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 so doing, 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.* [12]) and develop one where there is little prior work (discrete math, logic design).

## 5. ACKNOWLEDGMENTS

We thank the experts that participated in the Delphi processes. 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.

## 6. REFERENCES

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

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

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

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

[5] K. J. Goldman et al. Identifying Important and Difficult Concepts in Introductory Computing Courses using a Delphi Process. Technical Report UIUCDCS-R-2007-2917, University of Illinois Computer Science Department, November 2007.

[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.