
AC 2011-660: PHENOMENOGRAPHY AS A TOOL FOR INVESTIGATING UNDERSTANDING OF COMPUTING CONCEPTS

Gregory Bucks, Ohio Northern University

Gregory Bucks recently graduated with his Ph.D. from the school of Engineering Education at Purdue University. He received his BSEE from the Pennsylvania State University and his MSECE from Purdue University. While at Purdue, he has been heavily involved with the EPICS program as well as working with the First-Year Engineering program. He is currently a visiting assistant professor in the Electrical and Computer Engineering and Computer Science department at Ohio Northern University, where he is teaching introductory circuits and a variety of introductory programming courses.

William C. Oakes, Purdue University, West Lafayette

William Oakes is the Director of the EPICS Program at Purdue University, one of the founding faculty members of the School of Engineering Education and a courtesy faculty member in Mechanical Engineering and Curriculum and Instruction in the College of Education. He is an fellow of the ASEE and NSPE. He was the first engineer to win the Campus Compact Thomas Ehrlich Faculty Award for Service-Learning. He was a co-recipient of the 2005 National Academy of Engineering's Bernard Gordon Prize for Innovation in Engineering and Technology Education for his work in EPICS.

Phenomenography as a Tool for Investigating Understanding of Computing Concepts

Abstract

Computing has become a foundational subject across the engineering disciplines and offers significant opportunities both in practice and from an educational perspective. Maximizing this potential requires deep understanding of how students learn and apply computing concepts. There has been a great deal of work exploring understanding in computing education, focused primarily on what constitutes knowledge in computing and the processes engaged to utilize this knowledge in solving computing problems. There is also a sizable body of work exploring the misconceptions held by novices in computing education. However, little work has been done exploring the types of conceptions that computing students hold for the fundamental computing concepts apart from identifying misconceptions. Uncovering the different types of conceptions held by students independent of specific computing languages or environments is essential to understanding how students learn computing concepts and ultimately to develop better pedagogical and assessment techniques.

Phenomenography is a research methodology uniquely designed to uncover the different conceptions held by individuals about a given concept because the main tenet of phenomenography is that any phenomenon can be understood or experienced in a limited number of qualitatively different ways. Thus, the goal of phenomenography is to uncover those different ways of understanding. In recent years, phenomenography has begun to be used to explore the way that students experience the act of learning to program, both from a procedural and object-oriented perspective. However, it has not been used to explore the understanding held by individuals of specific concepts in computing. This paper describes how phenomenography was employed to explore the fundamental computing concepts of conditional and repetition structures. In addition, a discussion will be presented on how the results of this study, along with follow-on studies employing this methodology exploring additional fundamental programming concepts, can lay the groundwork for the development of language and computing environment independent assessment instruments. These instruments are needed for valid assessment and comparison of the pedagogical variations inherent in using the variety of programming languages, environments, and paradigms available today.

Introduction

Computers are an integral part of the engineering landscape. They are used to model potential solutions, collect and analyze data, and create new parts through computer aided design packages and computer controlled machinery. They are increasingly becoming integral parts of the products of design, from sneakers that track the distance traveled to smart building materials that can report on the stresses and strains they are experiencing. Because of this, computing skills have been identified as one of the attributes that future engineers will be required to possess¹. Due to this increasing prevalence of computing technologies in both the design and

implementation of engineering solutions, many first-year engineering curricula include either a course devoted entirely to computing concepts, or incorporate those concepts into other introductory courses.

Despite this increasing need for engineers with strong computer skills, little opportunity is given for engineering students to develop those skills. Learning to program is widely seen as a very difficult task for many students². It requires a student not only to develop an understanding of specific concepts, but also a way of thinking. In addition, in many learning environments, students are forced to learn a new tool, in the form of the programming environment being used, along with these concepts and patterns of thinking³. Because of this, many students will not develop a sufficient level of proficiency in programming, even after progressing through the traditional two or three course introductory programming sequence^{4,5}. This is a significant problem, especially in the engineering disciplines, where many students will be required to use some form of programming during either their academic and/or professional career, but very few receive more than one or two semesters' worth of instruction due to fact that most engineering curricula are already overloaded with disciplinary coursework. This experience generally occurs during the first year of study, often in the first semester, and is only touched on again if it is required for a specific course. Thus, an important area of research for engineering education is to design ways to help students learn programming and computer skills more efficiently to better prepare students in the time allotted in the curriculum.

One possible reason for this discrepancy between the learning outcomes desired by instructors and student performance is that the instructional methods used as well as the very nature of the material do not match well with the learning styles of most engineering students. There have been numerous studies, predominately using the Felder-Silverman learning styles model⁶, that have looked at the learning styles preferences of engineering students⁷⁻⁹, and those preferences are consistent across populations¹⁰. What these studies have found is that engineering students tend to be more visual and sensing in their learning styles preferences. However, since most programming languages taught in introductory courses are text-based, there is a mismatch between what is being taught and how many students prefer to learn. It has been shown that interpretation of the written word, while presented in a visual manner, is processed in the same way as spoken words¹¹. This means that students who prefer to learn in a visual manner may have difficulty assimilating programming content generally conveyed in a verbal context. Interpreting written words favors individuals who prefer to learn intuitively rather than from a sensing perspective, again putting engineering students at a disadvantage.

Many students also lack appropriate models on which to build conceptions of the important programming structures and concepts they are required to learn¹². In conjunction with this, many text-based languages use syntax that incorporates many English terms, causing students to resort to using models previously developed for the natural language use of these terms. However, this poses a significant problem for some terms because the model for how the word is used in natural language differs from how it is used in a programming context.

For example, in natural language, the term "while" has a slightly different meaning than it does in programming usage. In natural language, "while" implies that as soon as whatever the condition tied to the statement is no longer satisfied, the activity will cease. In a programming

context, the conditional statement associated with the “while” is only checked once during an iteration. This can cause students issues if they believe that as soon as their condition is met, the loop will exit¹³.

In order to address these issues, many instructors and researchers have begun to develop different tools to help students visualize what is occurring within the computer as the code executes¹⁴. These tools have become significantly more robust and interactive as both computing and graphics technologies have developed, and have been shown to have a positive influence on student learning and engagement with computing material^{15, 16}. Programming languages themselves have also become more graphical. Languages such as Alice, LabVIEW, and MATLAB Simulink utilize the graphics power of modern computers to allow users to program using graphical representations instead of the more traditional text-based code.

However, despite the fact that programming knowledge has been a focus of research for the past three to four decades, very few instruments have been published to assess an individual’s level of understanding that do not depend on the individual producing code in a specific language. Currently, most assessment occurs by requiring students to write code in a specific language¹⁷. This significantly limits the ability to compare the effects of different languages and pedagogical techniques on student learning, such as algorithm visualizations which may be tied directly to a specific language, because the assessment is tied directly to the language taught and assessed. The development of language-independent instruments that focus on the understanding of fundamental concepts are essential for research exploring these new tools and languages. Research into the different ways individuals understand the various concepts associated with computing can lay the foundation for the development of tools to assess conceptual understanding independent of a given language, and ultimately lead to developments in instructional methods.

This paper discusses an approach that can lay the groundwork for the development of such instruments. The approach is to use a series of phenomenographic studies of the ways students understand different fundamental programming concepts to create a theoretical framework. This foundation can be used to create an instrument, similar to the concept inventory type assessments that have received much attention in recent years. This instrument could be developed to focus on the different ways a student understands a given concept rather than on his or her ability to create code in a specific language. In the following sections, a brief introduction to phenomenography, a description of the study exploring student understanding of conditional and repetition structures, and the results of the study are presented. The final section discusses how these results can be utilized in the development of an assessment instrument as well as what still needs to be done in order to reach the final destination.

Phenomenography as a Research Methodology

Phenomenography is an educational research method developed in the early 1980’s by a research group in the Department of Education at the University of Gothenburg in Sweden¹⁸. It arose out of work exploring the ways that students experienced learning, approached their studies, and the level of understanding and retention they achieved¹⁹. It has since become a widely used educational research methodology, primarily in Europe and Australia²⁰.

The main tenet of phenomenography is that any given phenomenon can be understood in a limited number of qualitatively different ways²¹. Thus, the aim of phenomenography is to try to uncover what those different ways of understanding are. Phenomenography differs itself from other research approaches in that with phenomenography, the focus is on the interaction between the population and the phenomenon. Therefore, the researcher must attempt to bracket off any preexisting relationship between him or herself and these other two additional components.

This variation in experience and understanding is uncovered through in-depth interviews, which are analyzed and sorted at the transcript level²², as opposed to other interview analysis techniques, such as thematic analysis, which focus more at the statement level. Interviews are conducted in such a way as to try to draw out the understanding or experience that an individual has about a given phenomenon. This often involves the use of general, open-ended questions in conjunction with clarifying and follow-up questions to probe more deeply into the understanding of the individual²³.

The result of a phenomenographic study is what is referred to as an outcomes space²⁴. The outcomes space consists of a number of categories of description, which describe the different ways that the phenomenon under investigation was understood or experienced. The outcomes space then depicts the relationship between these categories, generally forming some type of hierarchical structure. The hierarchy does not necessarily represent a transition from a worse state to a better state, but instead a transition from a less complete understanding to a more complete understanding. An additional finding is what is referred to as “themes of expanding awareness,” which are themes that appear throughout the categories of description, but in different ways²⁰.

Phenomenography has been used to study a wide variety of topics, including reading comprehension¹⁸, physics^{25, 26}, programming^{19, 24, 27}, and design^{23, 28-30}. Of particular significance to this study are those above dealing with the use of phenomenography in the field of computer science. Booth was one of the first to apply this methodology to the field of computer science during her dissertation work. She did a phenomenographic study exploring the different ways that first-year engineering students in their first programming class understood the idea of recursion³¹. The study by Bruce, Buckingham et. al.²⁷ explored the ways in which novice programmers learn to program, and Stamouli and Huggard²⁴ explored how students learn object-oriented programming.

However, none of the previously mentioned studies exploring computing from a phenomenographic perspective utilized the methodology to explore specific concepts within the field, instead choosing to investigate students’ experiences with programming as a whole. Phenomenography has been used effectively to explore specific concepts in other fields, such as physics and design. While physics is not a great match for the types of abstract concepts inherent in computing education, the fact that phenomenography has been used to explore the types of understanding students hold about different aspects of design, many of which are abstract and not directly accessible, bodes well for the use of phenomenography in exploring computing concepts. One of the purposes of this study was to explore the viability of using phenomenography to explore specific concepts in computing education.

Study

The main purpose of this study was to uncover the different ways that individuals understand different programming concepts, specifically the concepts of conditional and repetition structures. Based on the goals of this project, the following two research questions were posed:

- 1) What are the qualitatively different ways that the conditional and repetition structures found in most programming languages are understood?
- 2) What are the ways that first-year engineering students understand these concepts?

To answer these questions, a phenomenographic approach was chosen. One of the essential elements when designing a phenomenographic study is the selection of the participants. Since the purpose of phenomenography is to uncover the qualitatively different ways that a specific phenomenon is understood, the participants should be selected to obtain the maximum possible diversity²⁰. This implies the use of a purposeful sampling method.

To obtain the maximum possible variation, participants were selected from all academic levels (first-year through senior) as well as faculty involved in introductory programming courses. The reason for including faculty in the sample is to gain an expert perspective and to insure that the entire spectrum, from novice to expert, is captured. The students come from a variety of majors, but were screened to insure that they had a recent programming experience to be included in the study.

To answer the second question related to first-year engineering students, the students from within this population were oversampled relative to the other populations to ensure an accurate representation. The students in this group were selected from the major first-year engineering course at Purdue University, which is required for all engineering majors. These students were selected because of the focus on understanding held by engineering students and the varied levels and types of experiences these students have had with programming. The breakdown of participants is shown below in Table 1.

Table 1: Participant Breakdown

| Participant Group | Number of Participants |
|---------------------------------|------------------------|
| Engineering/CS Professors | 5 |
| Upper Level Students (Jr./Sr.) | 7 |
| Middle Level Students (Soph.) | 2 |
| First-year Engineering Students | 22 |

As stated above, the main data used in phenomenographic studies are interview transcripts³². These interviews are generally in-depth discussions aimed at uncovering the interviewee's way of understanding or experience of the given phenomenon. They may take the form of an open-ended discussion with very few predetermined questions or be of a more structured design. However, the key element is the use of follow-up questions and other methods for probing the understanding of the interviewee.

For this study, a semi-structured interview protocol was employed in order to obtain a measure of consistency among the interviews, with each interview lasting approximately 60 minutes. The

protocol was designed to allow the participants to talk about their previous experiences with programming in a general sense in order to prime them for the reflection needed to answer the in-depth questions about their understanding²³. This is a key component in conducting a phenomenographic research study, as it is only through this process of reflection that the participant is able to fully articulate their experience or understanding, especially if the phenomenon under investigation is a specific event or experience. However, reflection is a skill that must be developed³³, and thus the protocol must be design in such a way as to promote this type of activity.

A portion of the interview required a talk-aloud protocol³⁴, where the participant worked through a simple problem and discussed the thought processes employed as the participant developed a solution. Talk-aloud protocols have been used extensively in exploring the ways that engineers design³⁵⁻³⁷. The technique has also been used within phenomenographic studies²⁶. During the talk-aloud portion of the interview protocol, the participants were prompted to discuss their thoughts and reasoning as they attempted to solve the problem, which is described below. This activity was followed up by questions concerning why the participant employed certain strategies and discussing any issues that arose while solving the problem. By having the participants talk through how they would solve the problems, their understanding of how they use the different concepts under investigation was probed more directly.

The problem was designed to be moderately difficult, but not so difficult that it would not be solvable in the timeframe of the interview. The solution required the use of both a conditional and repetition structure to allow the exploration of the use of these concepts by the participants. The exit conditions for the repetition structure had elements that could indicate the use of a For loop or a While loop so that the reasoning behind choosing one over the other could be explored as well. The aspect of the problem requiring the conditional structure employed conditions that were related to one another, allowing the participants the opportunity to select from several different conditional structures (If statements, nested If statements, Switch statements) when solving the problem. The problem statement presented during the interview is shown in Table 2.

Table 2: Problem Statement

| |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Everyone has played the game where one person thinks of a number in a given range and someone else tries to guess it. Your job is to create a program that will create a random integer number within a given range and allow the user of your program to guess the number. The program should tell the user whether his/her guess is too high or too low, and stop when the user has guessed the number correctly or has reached the maximum allotment of five guesses. |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

It is very important that the interview protocol be piloted prior to collecting the final data³⁸. The initial protocol was piloted with two students, one a junior in interdisciplinary engineering with moderate programming experience and one a senior in computer engineering with significant programming experience. After these initial pilot interviews were transcribed and analyzed, several modifications were made to the protocol resulting in the final version used during the study.

Normally, data analysis consists of reviewing the transcripts of the interviews and, at the transcript level, sorting them into categories based on the similarities and differences between

them^{22, 32}. Since there were two different concepts being explored in each of the interviews, the transcripts were broken up into two pieces, one focusing on the conditional structures and one focusing on the repetition structures. Each piece consisted of any portions of the main interview during which the participant discussed the concept along with the solution to the problem. It was then these two separated sections of the transcripts that were analyzed and sorted into categories.

This process required a number of iterations and feedback before the final categories and outcomes spaces were decided upon, as is generally the case in phenomenographic studies²³. During this process, versions of the categories of description and outcomes spaces were generated and tested against the data. Changes were made based on discussions with fellow researchers and members of the primary researcher's dissertation committee. While others contributed to the final development of the categories and outcomes spaces through this iterative feedback process, only the primary researcher read all of the data and was primarily responsible for the generation of the results.

Results

This section discusses the type of data that was collected during the study and how this type of data resulted in the development of the categories of description and outcomes spaces for the two concepts. The results of the analyses for the conditional structures and repetition structures are briefly described, with the results for the ways of understanding conditional structures presented first followed by a description of the ways of understanding repetition structures. More detailed explanations of these results can be found at the following references^{39, 40}. However, the main focus of this section is on how the phenomenographic methodology helped to illuminate the ways of understanding held by the participants, which is presented following the description of the outcomes spaces and categories of description.

Conditional Structures:

The analysis of the interview transcripts resulted in the development of five categories of description, which outline the five qualitatively different ways that conditional structures were understood by the participants. There are two components that describe how an individual understands conditional structures. The first component is a shift from an understanding that is very limited and focused on the development of the conditional statement to a focus on how the conditional structure affects the execution of the program. Along with this increasing awareness of the effect on the program is an increasing ability to describe conditional structures in one's own words and less of a need to rely on the descriptions used in instructional settings.

The second major component on which the categories vary is the sophistication of the conditional structures used in a problem solving situation. Sophistication refers to the ability to recognize when conditions are interrelated and the complexity and robustness of the solution created when working with such conditions in a problem solving situation. The five categories are described in Table 1.

Table 1: Categories of Description³⁹

| Categories of Description (Understanding of Conditional Structures is...) | Summary |
|---------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Category 1: Limited | The understanding of conditional structures in this category is described as limited in that there is a limited ability to describe the functionality of conditional structures, often confusing them with other structures, and a limited ability to apply them in a problem solving situation. |
| Category 2: Recounted | The understanding of conditional structures in this category is described as recounted in that there is an ability to recite the definitions and functionalities of conditional structures often provided during introductory programming courses and an ability to apply them in very specific situations. |
| Category 3: Syntax Focused | The understanding of conditional structures in this category is described as syntax focused in that the main focus of descriptions is on the construction of the conditional statement and the structure as a whole along with an ability to apply these structures in moderately complex situations. |
| Category 4: Execution Focused | The understanding of conditional structures in this category is described as structure focused in that the main focus of descriptions is on the affect the structure has on the execution of the program along with an ability to apply these structures in moderately complex situations. |
| Category 5: Adaptive | The understanding of conditional structures in this category is described as adaptive in that the main focus of descriptions is on the affect the structure has on the execution of the program along with an ability to apply these structure in highly complex situations. |

The resulting outcomes space is depicted in Figure 1. Through the analysis of the data, it was found that the understanding of conditional structures discussed by the participants formed a hierarchical, two dimensional structure, where each subsequent category builds upon the understanding represented in the previous. Each category was developed from the experiences of a subset of the participants and each participant contributed to only a single category. The two dimensions present in the diagram are the two components discussed previously on which the understanding of conditional structure varies. Category one is not considered to be inclusive with the other categories in terms of the understanding professed because there is a significant difference in the level of understanding between category one and category two, and is signified by the differing color assigned to category one. While it is not contiguous with the other categories, it is a part of the hierarchy.

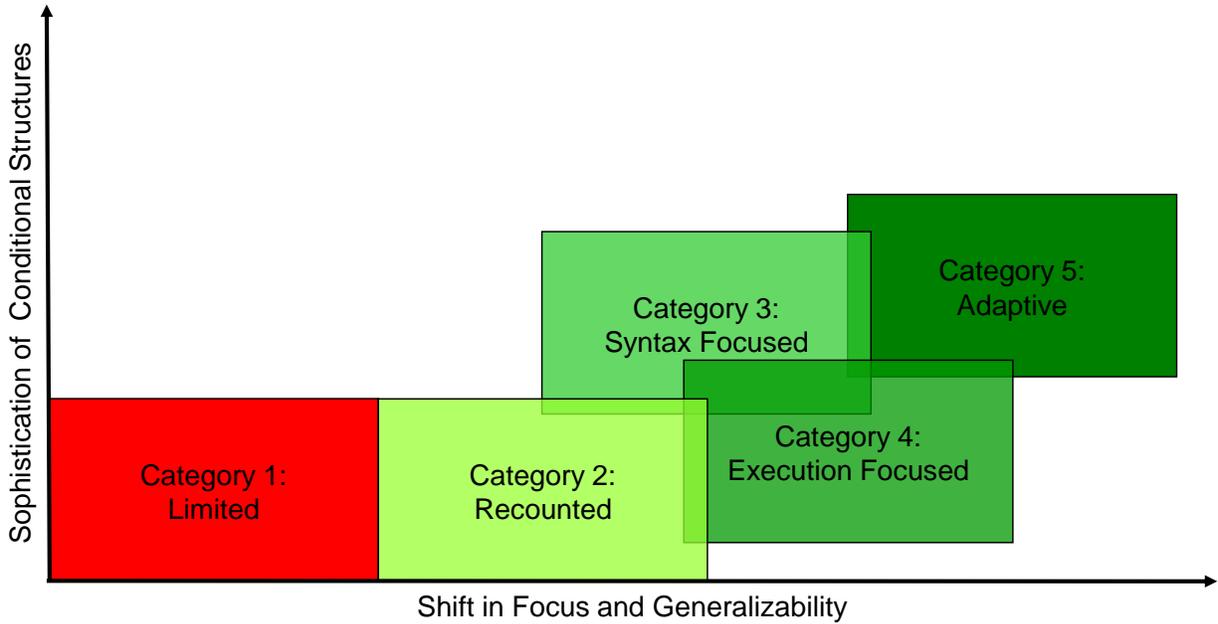


Figure 1: Outcomes Space³⁹

Repetition Structures:

The analysis of the interview transcripts resulted in the development of six categories of description for the ways of understanding repetition structures. There are two components that describe how an individual understands repetition structures: the understanding of the relationship between the two main repetition structures (For and While loops) and the ability to identify the need for a repetition structure in a problem-solving situation, select an appropriate structure, apply it to the given situation, and justify its use. The six categories are described in Table 4.

Table 4: Categories of Description⁴⁰

| Categories of Description (Understanding of Repetition Structures is...) | Summary |
|-----------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Category 1: Limited | Understanding of repetition structures in this category is described as limited in that there is an inability to describe the functionalities and differences between the two main repetitions structures as well as a significantly limited ability to apply them in a problem solving situation. |
| Category 2: Recounted | Understanding of repetition structures in this category is described as recounted in that there is an ability to recite the definitions, functionalities, and differences of the two main repetition structures presented in most introductory programming courses and an ability to apply them in a small set of very specific situations. |
| Category 3: Internalized | Understanding of repetition structures in this category is described as internalized in that there is an ability to describe the functionalities and differences between the two main repetition structures, the beginnings of an awareness of the relationship between the two which transcends what is typically covered in an introductory course as well as an ability to apply them in a small set of specific situations. |
| Category 4: Related | Understanding of repetition structures in this category is described as related in that there is an ability to define a concrete relationship between the two repetition structures, namely that For loops are seen as specific versions of While loops, as well as an ability to apply them in a variety of situations. |
| Category 5: Comprehensive | Understanding of repetition structures in this category is described as comprehensive in that the two repetition structures are seen as two instances of the same concept, which allows for them to be applied in a wide variety of situations. |
| Category 6: Adaptive | Understanding of repetition structures in this category is described as adaptive in that the two repetition structures are seen as two instances of the same concept, and this understanding of the two structures allows for them to be substituted within a problem solving situation. |

The resulting outcomes space is depicted in Figure 2. Through the analysis of the data, it was found that the understanding of repetition structures discussed by the participants formed a hierarchical, two dimensional structure, where each subsequent category builds upon the understanding represented in the previous category. The two dimensions present in the diagram are the two components discussed previously on which the understanding of repetitions structure varies. Category one is not inclusive with the rest of the categories and is represented in the outcomes space by a different color. Additionally, category two acts as a transition between the limited understanding in category one and the beginning of the portion of the space depicting the evolution of the understanding once a relationship between the two main repetition structures is seen starting in category three.

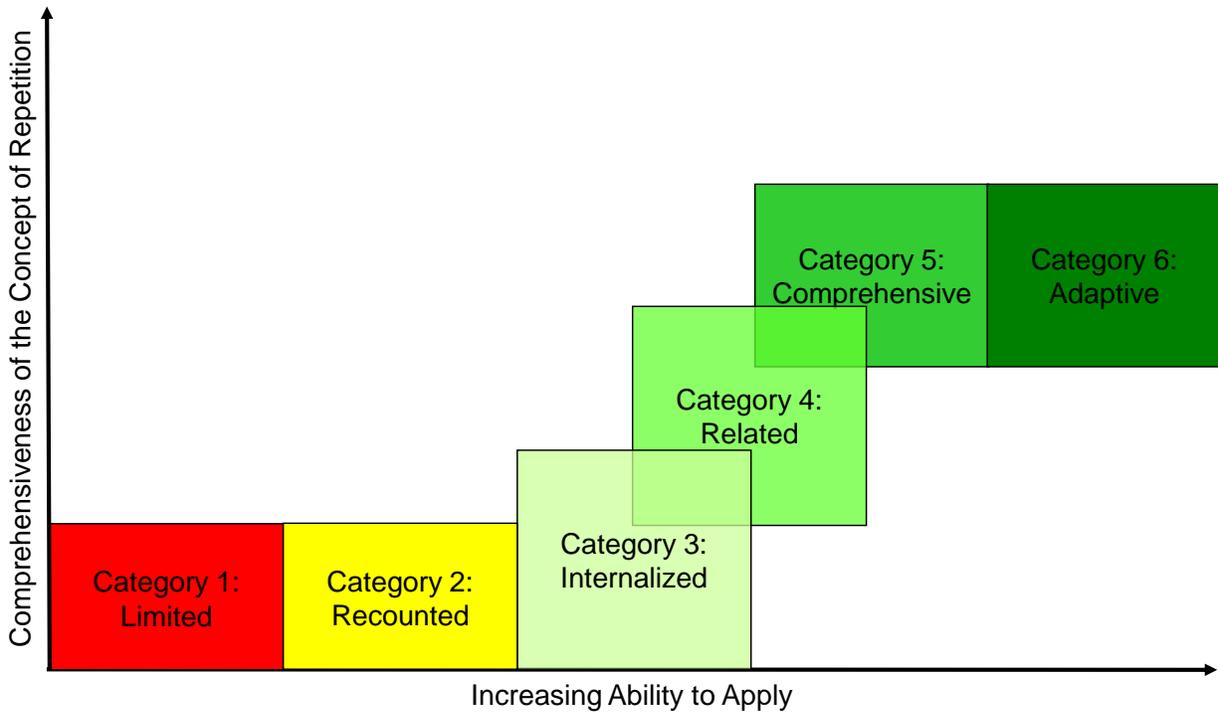


Figure 2: Outcomes Space⁴⁰

Phenomenography as a Tool for Investigating Understanding in Programming:

As was stated above, two pieces of data were collected for each participant, both through the interview protocol. The first portion of the interview consisted mainly of probing questions attempting to elicit statements illuminating the individual’s understanding of the given concepts. The second portion of the interview consisted of the talk-aloud protocol requiring the interviewee to produce a solution to the problem presented in the section outlining the study. Both of these pieces of data were used in order to develop the categories of description and outcomes spaces that describe the different ways each participant understood the concepts.

In order to probe the participants’ understanding during the first part of the interview, open-ended questions were used that allowed the participants the freedom to respond naturally. Following the participants’ initial responses, probing questions were used to capture their overall understanding. These questions tended to require the participants to describe previous experiences they had using the concepts followed by questions asking them to talk about the concept in different contexts, from what they feel the most important thing to know about the concept is to comparing the concept to other ideas.

The following are quotes that exemplify the types of responses received during this portion of the interview. This first quote is from one of the participants whose responses contributed to the least comprehensive of the categories for the repetition structures, where the understanding was defined by an inability to describe the two main types of repetition structures.

I believe um a for loop is if, I'm trying to remember um, wow, cause I think isn't one used for if you have many inputs and you wanna go continuously like without stopping, I believe, then I think one of 'em, the other one, when you put the inputs in 'em, I believe it s-, I don't know if it stops after each input gets taken? Hmm, not too sure again. It's been a little while.

In comparison, the following quote is from someone whose responses contributed to the second category, where the way of understanding was defined by an ability to describe the two different types of repetition structures, but did not see any connection between the two.

A for loop, it works for only, like, certain amount, like choices, like tries. And like for while loop, it goes like again and again and again until it meets the requirements, like, for stop.

But when this person was asked if there was any relationship between these two loops, the following response was given.

Besides the fact that they're loops, no.

As can be seen from these two sets of quotes, there is a distinct difference in the way the interviewee is able to talk about the concept, which in this case is repetition structures. To further illuminate how these quotes describe differences in understanding, the following is a quote from the third category of description for repetition structures, where the way of understanding is defined by a continued ability to describe the functionality of the two repetition structures but also the beginnings of an understanding of the connection between the two repetition structures is expressed.

Um, you have to have a basic understanding that the main difference is you want to run x times versus an unknown amount of times is the main difference in the for and the while loop. If it weren't for the difference of you want it to run x times or an unknown amount of times, they're practically identical. The way they're set up is slightly different, but all their characteristics are basically the same.

This quote shows the same ability to describe how the two different repetition structures function, though the description is a bit more complete in this quote. However, the type of understanding displayed is more comprehensive in that this person was able to talk about a connection between the two structures, even if that relationship is not described in detail.

The data from the talk-aloud protocol produced similar results. However, for this portion of the interview, not only were the interview transcripts collected, but the drawings, flowcharts, and pseudocode the participants used to describe their solutions were also collected, which added another dimension to the analysis.

The data from the interview transcripts paralleled the types of understanding expressed during the initial interview questions. For instance, the following quote comes from someone who contributed to the first category of description for repetition structures. This quote demonstrates

a general understanding of how to solve the problem, but a continued inability to demonstrate understanding of the two repetition structures.

And for these two, I would put them in a loop so that, because I have 5 guesses, the number of times the loop runs will be 5... I would put [the loop] around [the conditional check for high/low] ... The number should stay the same, right? So we wouldn't put [the random number generation] in the loop. But we want to use it to put in the number 5 times because you're giving them 5 chances, so I would put it in a loop for this entire thing, but this wouldn't be in the loop because you would obviously get the answer correct. But I don't know what loop I would put it in. Like, I couldn't tell you that.

This second quote comes from a participant who contributed to the development of the second category of description for repetition structures. As before, this person is able to talk about the functionality of the two different types of repetition structures, but has difficulty connecting the two concepts, as demonstrated by her inability to find a way to merge the two exit conditions.

I don't know. That was the thing that I was thinking. 'Cause like for loop doesn't have a stop button in there, right? I don't remember how to make it stop. Do I use while loop for that? It has to be 5 guesses. Oh, I don't know...the while loop, it's easier to make it stop when like someone guesses it right. But I don't know how to set the 5 guesses. I kinda don't remember it. Does while loop have like something that counts like how many tries they did? I don't think so. I don't remember.

Finally, this last quote comes from someone who contributed to the development of the third category of description for repetition structures, which was differentiated from category two by the beginnings of an ability to see a connection between the two types of repetition structures. This is evidenced in the following quote as the participant was able to talk about solutions to the problem using both types of repetition structures.

I mean, you can use while loop for it, too. In that case, you'd have to use 2 logics: If the user input and the computer-generated number is correct, then it should stop the program. And the other condition would be when the number of incrementations that is given in the while loop equals 5, then it should stop the program. So you can use while loop for it, but, I mean, I'd recommend for loop.

Three examples of the solutions written down by participants are shown below. These three solutions are all for participants who contributed to the fifth category of description for repetition structures. As can be seen these solutions, which are all correct solutions for the problem, are very different in terms of their form. The first depiction utilizes a pseudocode similar in many ways to MATLAB or C++. The second solution outlines the solution as a bulleted list identifying the steps required to solve the problem. The last solution is produced using a graphical programming language called LabVIEW, which is produced by National Instruments. All three of these solutions, in conjunction with the interview transcript, serve to provide another glimpse of how the participants understand the concepts under investigation, particularly how they are able to utilize their knowledge to develop the solution.

This technique can be used to investigate understanding gained through different programming paradigms, as evidenced by the depictions created by the participants. Some of these students learned programming primarily using traditional text-based languages while others first learned programming using LabVIEW. As long as the researcher analyzing the results is familiar with the different languages, the different depictions can be compared.

In the three examples below, all used some form of nested conditional structure to handle the three possible outcomes of the comparison between the random number and the guessed number, which shows an awareness of the relationship between the three outcomes. The participants also talked about this as they were describing their solution. All three were able to handle to multiple exit conditions for the required repetition structure, namely allowing for five guesses but stopping immediately if the user guesses the number correctly. The ways that this problem was handled in the solutions depicted below takes on different forms, but all three correctly handle this situation. All three were also able to describe how their solution addressed this aspect of the problem during their explanations. This shows an awareness of how to select, adapt, and implement a repetition structure to meet the needs of the given problem.

```

x = randn(1) * r
#guesses = 0
while #guesses < 5
  for guesses = [0, 1, 2, 3, 4]
    y = %input %d
    if = x
      print "YES, CORRECT"
      guess = 1000;
    else
      if y > x
        too high
      else
        too low
      guess = guess + 1
    end
  end
end

```

Figure 3: Participant Solution 1

- ① Programmer defines the range in which the random numbers will be generated. Random numbers generated through MATLAB function.
- ② User is allowed to enter ~~guess~~ up to 5 guesses. ✓
- ③ Based on what is entered, the program will output feedback as of high or low;
- ④ Looping structure to allow up to 5 guesses. Reasonably a for loop.
- ⑤ To determine high or low, use a selection structure, like "if". So if ($\# \text{ entered} > \text{randint}$)
 $\text{print}(' \text{Your guess is high}')$
- ⑥ After 5 guesses, print to screen the correct answer,

Figure 4: Participant Solution 2

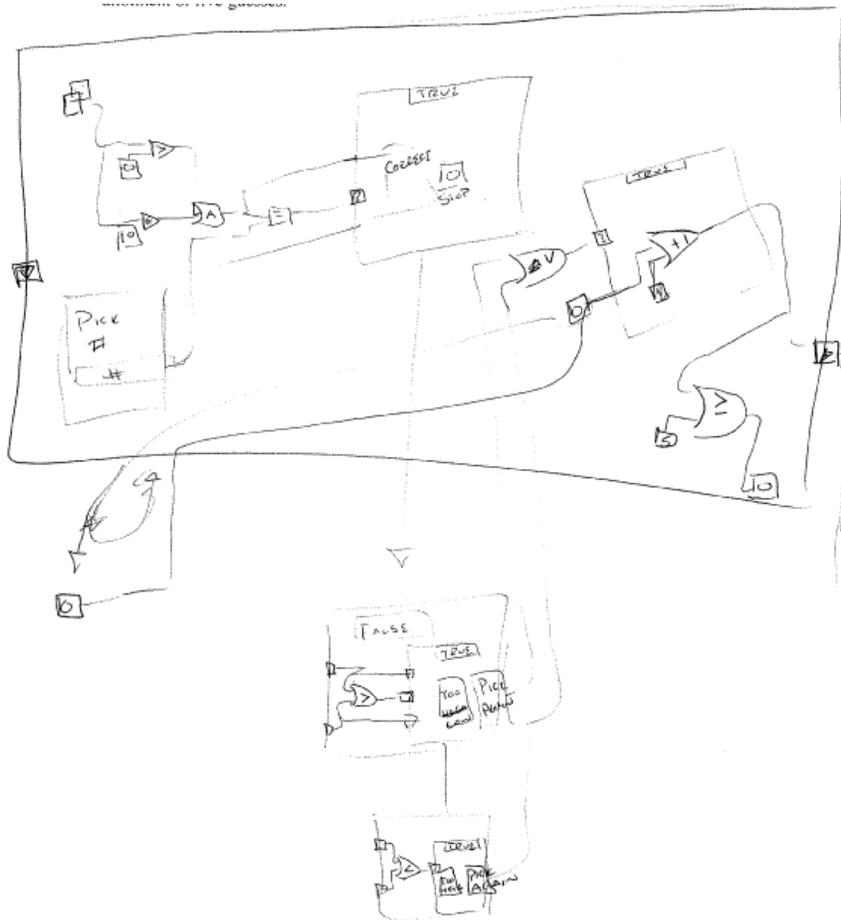


Figure 5: Participant Solution 3

Future Directions and Implications for Assessment

As was discussed in the introduction, the primary motivation for undertaking this investigation was to lay the groundwork for the development of language-independent assessment instruments. This goal is motivated by an increasing need for engineering students to have strong foundations in computing as well as the increasing number of pedagogical techniques employed in the introductory computing courses with no standardized way to assess their effectiveness. Therefore, the development of instruments that can allow different languages and different techniques to be investigated and compared is of utmost importance.

One avenue to explore in the development of a language-independent instrument is a concept inventory-like assessment that covers the gamut of fundamental concepts covered in introductory computing classes. Concept inventories have gained in popularity in recent years as a framework for creating assessments of this type. Concept inventories are multiple choice assessments with one correct answer and several distractors, which are related to the common misconceptions held by students⁴¹.

Several attempts have been made to develop concept inventories for programming fundamentals. One of the first is the Computer Engineering Concept Inventory (CPECI)⁴². This concept inventory is designed to cover three main areas of computer engineering: programming fundamentals, digital logic, and computer architecture and organization. The programming fundamentals portion consists of 26 questions designed to cover the six fundamental programming topics outlined in the Computer Curricula 2001 report⁴³. Unfortunately, the CPECI is targeted at a level of understanding beyond what is traditionally covered in the introductory CS1 course, which is the level that most engineering students progress to in the introductory programming course sequence. There is also an ongoing endeavor at the University of Illinois at Urbana-Champaign to develop a concept inventory focused on programming fundamentals⁴⁴, but it is only in the preliminary stages of development.

Concept inventories are excellent tools to determine whether a student has developed a correct understanding of a given concept. However, as the results of the previously discussed study show, there can also be varying degrees of correctness in terms of an individual's understanding, which is equally important in developing a strong knowledge of a given subject. It is proposed that a concept inventory-like assessment can be developed based off of phenomenographic studies focusing on the various introductory computing concepts that would allow for these various levels of understanding to be captured and compared across populations and based on different pedagogical settings.

With a traditional concept inventory, the possible answers are tied to the correct conception and common misconceptions held by students, which are generally determined based on the experiences of those developing the instrument and on prior studies. The proposed instrument would be structured in much the same way, with one slight difference: the possible options for the multiple choice questions dealing with a given topic would be tied back to the categories of description generated from the phenomenographic study, which would allow the individual's overall way of understanding to be assessed. This can give a more complete picture of how an individual is dealing with the content rather than simply having the correct conception.

For instance, a question related to understanding of repetition structures may take the following form:

Question: When you are creating a program which requires that a set of actions be repeated, but you do not know how many times, which answer below best represents your response?

- a) I only know how to use one type of loop, so I will use that and make it work.
- b) I don't know which loop I would use, so I would try to use one and, if that didn't work, try the other.
- c) It sounds like a situation where I would use one of the two types of loops, so that is the one I would use.
- d) I can use either type of loop to develop a solution to the problem, so I would pick whichever type I felt like using.
- e) I can use either type of loop to develop a solution to the problem, but one type is specifically designed to handle situations like this, so that is the one I would choose.

This question is an illustration of what could be included in an actual assessment, which has been formulated based on the categories of description presented. One of the attributes that differentiates the different categories of description for repetition structures is the ability to select an appropriate structure for a given situation. Another is the ability to see the relationship between the two different loop types (for and while) and how they are each different instances of the same idea. Traversing the answers from a) to e) corresponds to progressing through the categories of description from the least comprehensive to more comprehensive ways of understanding. People who contributed to the development of the least comprehensive category (category 1) were only able to talk about one type of loop. This corresponds to answer a). People who contributed to the middle categories (categories 2, 3, and 4) were able to talk about both types of loop, but their understanding of the relationship between the two loop types was limited, corresponding to answers b) and c). People who contributed to the development of the most comprehensive categories (categories 5 and 6) were able to use each loop type interchangeably. What separated category 6 from category 5 was that those people contributing to category 6 were able to more appropriately justify their selections of loop type for different problems. This can be seen in the difference between answers d) and e).

However, in order to develop such an instrument, much more work needs to be done. Additional phenomenographic studies must be developed to probe the other fundamental computing topics, such as variables, arrays, mathematical operations, logical operations, and user-defined functions. Once the categories of descriptions for these different topics have been developed, the long and arduous process of developing the instrument can begin. This will involve the development of questions, the seeking of peer review to validate the questions, then repeated testing, analysis, and redesign of the instrument to determine its validity and reliability.

Limitations of Phenomenography:

Despite the numerous advantages to employing a phenomenographic approach in exploring the understanding held by individuals, there are a number of limitations that must be discussed. The

primary limitation of phenomenography is the amount of time required to use such an approach. The study described in this paper required 3 major iterations for the development of the conditional structures categories and outcomes space and 5 major iterations for the development related to repetition structures. Each iteration requires the transcripts to be reread and tested against the current version of the categories, and usually involves several minor iterations as the next version of the categories is developed. This requires the researcher to spend the time to become intimately familiar with each of the transcripts. When each transcript is on the order to 25-30 pages, this equates to a significant time commitment.

Phenomenography also requires that the interviewer(s) be adept at asking follow up questions of the participant. In most cases, employing a phenomenographic approach limits the ability to use a fully structured interview protocol because the need to probe the participants' understanding. This requires the use of follow-up questions specifically tailored to the responses provided by the participant. All of the potential questions the interviewer may need to ask cannot be captured in a structured interview protocol, requiring the interviewer to adapt to each response and ask appropriate questions. This is another reason for piloting the interview protocol prior to collecting the data for the study, as it provides an opportunity for the interviewer to become accustomed to thinking of questions in the spur of the moment to probe the understanding of the participant more deeply.

Related to this is the need for the participant to reflect on their own experiences and understanding of the given concept during the interview process. Reflection is an important aspect of any professional's practice, but is often difficult to develop³³. Because of this emphasis on reflection, the interview protocol must be carefully developed in such a way as to aid the participants in the process of reflecting on their understandings and experiences, especially if the target population includes younger individuals or individuals not used to reflective activities.

Another potential limitation of phenomenography is that it only captures the participants' understanding or experience at a specific point in time. If the study was to be conducted again, with the same participants, at a different time, the responses of the participants might be completely different, as their responses are shaped by their own experiences with the given phenomenon. What should not change significantly are the categories and outcomes spaces developed, as these are created from the combined responses of all the participants. In order to address this issue, it is important to include as much diversity and quantity among the participants to reach a stable description of the ways in which the participants understand or experience the given phenomenon.

Despite these potential limitations, phenomenography is a promising tool for investigating conceptual understanding. With proper techniques, and enough time, all of the limitations of the methodology can be managed.

Conclusion

This paper has presented a new approach to exploring conceptual understanding in computing education through the use of the phenomenographic research methodology. Phenomenography has been shown, through the results in the described study, to effectively elicit different ways of

understanding of conditional structures and repetition structures among undergraduate engineering students. Phenomenography has also proven to be effective at exploring the understanding held by the participants regardless of their background and the programming paradigms they were familiar with. Through the analysis of the interview transcripts and the products of the talk-aloud problem solving activity, five different ways of understanding for conditional structures and six different ways of understanding for repetition structures were uncovered.

These findings have an exciting implication for the development of assessment instruments in the field of computing. The categories of description can provide the theoretical framework to develop concept inventory-like assessments, where each response to a given multiple choice question is tied to a different category, rather than to common misconceptions. The development of such an instrument holds tremendous possibilities for assessing the wide variety of programming environments and pedagogical techniques currently available and in use in introductory programming courses today.

Acknowledgements

This work was made possible by a grant from the National Instruments Foundation. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Instruments.

Bibliography

1. National Academy of Engineering, *The engineer of 2020 : visions of engineering in the new century*. 2004, Washington, DC: National Academies Press. xv, 101 p.
2. Robins, A., J. Rountree, and N. Rountree, *Learning and teaching programming: a review and discussion*. Computer Science Education, 2003. **13**(2): p. 137-172.
3. Mayer, R.E., *Cognitive aspects of learning and using a programming language*, in *Interfacing Thought*, J.M. Carrol, Editor. 1987, The MIT Press: Cambridge, MA. p. 61-79.
4. McCracken, M., et al., *A multi-national, multi-institutional study of assessment of programming skills of first-year CS students*. ACM SIGCSE Bulletin, 2001. **33**(4): p. 125-180.
5. Thomas, L., et al., *Learning styles and performance in the introductory programming sequence*. ACM SIGCSE Bulletin, 2002. **34**(1): p. 33-37.
6. Felder, R.M. and L.K. Silverman, *Learning and teaching styles in engineering education*. Engineering Education, 1988. **78**(7): p. 674-681.
7. Felder, R.M. and J. Spurlin, *Applications, reliability and validity of the index of learning styles*. International Journal of Engineering Education, 2005. **21**(1): p. 103-12.
8. Rosati, P.A. *The learning preferences of engineering students from two perspectives*. in *ASEE/IEEE Frontiers in Education*. 1998. Tempe, Arizona.
9. Litzinger, T.A., et al., *A psychometric study of the index of learning styles*. Journal of Engineering Education, 2007. **96**(4): p. 309-319.
10. Zualkernan, I.A., J. Allert, and G.Z. Qadah, *Learning styles of computer programming students: a middle eastern and American comparison*. IEEE Transactions on Education, 2006. **49**(4): p. 443-50.
11. Felder, R.M., *Learning and teaching styles in foreign and second language education*. Foreign Language Annals, 1995. **28**(1): p. 21-31.
12. Ma, L., et al., *Investigating the viability of mental models held by novice programmers*. ACM SIGCSE Bulletin, 2007. **39**(1): p. 499-503.

13. Bonar, J. and E. Soloway. *Uncovering principles of novice programming*. in *Proceedings of the 10th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*. 1983. Austin, Texas: ACM.
14. Myers, B.A., *Visual programming, programming by example, and program visualization: a taxonomy* ACM SIGCHI Bulletin, 1986. **17**(4): p. 59-66.
15. Naps, T.L., et al., *Exploring the role of visualization and engagement in computer science education*. ACM SIGCSE Bulletin, 2003. **35**(2): p. 131-152.
16. Hundhausen, C.D., S.A. Douglas, and J.T. Stasko, *A meta-study of algorithm visualization effectiveness*. Journal of Visual Languages & Computing, 2002. **13**(3): p. 259-290.
17. Carter, J., et al. *How shall we assess this?* in *Innovation and Technology in Computer Science Education*. 2003. Thessaloniki, Greece.
18. Marton, F., *Phenomenography - describing conceptions of the world around us*. Instructional Science, 1981. **10**: p. 177-200.
19. Booth, S., *On phenomenography, learning and teaching*. Higher Education Research and Development, 1997. **16**(2): p. p. 135-158.
20. Akerlind, G.S., *Variation and commonality in phenomenographic research methods*. Higher Education Research and Development, 2005. **24**(4): p. 321-334.
21. Marton, F., *Phenomenography: exploring different conceptions of reality*, in *Qualitative approaches to evaluation in education: the silent scientific revolution*, D.M. Fetterman, Editor. 1988, Praeger Publishers: New York. p. p. 176-205.
22. Dortins, E., *Reflections on phenomenographic process: interview, transcription and analysis*, in *Annual International Conference of the Higher Education Research and Development Society of Australasia*. 2002: Perth, Western Australia.
23. Daly, S. *The design landscape: a phenomenographic study of design experiences*. in *2009 ASEE Annual Conference and Exposition*. 2009. Austin, TX.
24. Stamouli, I. and M. Huggard. *Phenomenography as a tool for understanding our students*. in *International Symposium for Engineering Education*. 2007. Dublin City University, Ireland.
25. Dall'Alba, G., et al., *Assessing understanding: a phenomenographic approach*. Research in Science Education, 1989. **19**: p. 57-66.
26. Marton, F., *Phenomenography - a research approach to investigating different understandings of reality*. Journal of Thought, 1986. **21**: p. 28-49.
27. Bruce, C., et al., *Ways of experiencing the act of learning to program: a phenomenographic study of introductory programming students at university*. Journal of Information Technology Education, 2004. **3**: p. 143-160.
28. Mann, L., G. Dall'Alba, and D. Radcliffe. *Using phenomenography to investigate different ways of experiencing sustainable design*. in *ASEE Annual Conference and Exposition*. 2007. Honolulu, HI.
29. Reid, A. and I. Solomonides, *Design students' experience of engagement and creativity*. Art, Design and Communication in Higher Education, 2007. **6**(1): p. 27-39.
30. Zoltowski, C., *Students' ways of experiencing human-centered design*, in *Engineering Education*. 2010, Purdue University: West Lafayette, Indiana. p. 202.
31. Booth, S., *Learning to program: a phenomenographic perspective*. 1992, Acta Universitatis Gothoburgensis: Goteborg.
32. Akerlind, G.S., J. Bowden, and P. Green, *Learning to do phenomenography: a reflective discussion*, in *Doing developmental phenomenography*, J. Bowden and P. Green, Editors. 2005, RMIT University Press: Melbourne.
33. Schön, D., *The reflective practitioner*. 1983, London, UK: Temple-Smith.
34. Ericsson, K.A. and H.A. Simon, *Protocol analysis: verbal reports as data*. 1984, Cambridge, MA: The MIT Press.
35. Atman, C.J., et al., *A comparison of freshman and senior engineering design processes*. Design Studies, 1999. **20**(2): p. 131-152.
36. Atman, C.J. and J. Turns, *Studying engineering design learning: four verbal protocol analysis studies*, in *Design Learning and Knowing*, M. McCracken, W. Newstetter, and C. Eastman, Editors. 2001, Lawrence Erlbaum: New Jersey.
37. Restrepo, J. and H. Christiaans, *Problem structuring and information access in design*. The Journal of Design Research, 2004. **4**(2).
38. Bowden, J.A. and P. Green, eds. *Doing Developmental Phenomenography*. Qualitative Research Methods, ed. J. Bowden. 2005, RMIT University Press: Melbourne, Australia. 183.

39. Bucks, G. and W. Oakes, *Part I: ways of understanding conditional structures in computer programming languages*. Submitted to the Journal of Engineering Education, 2011.
40. Bucks, G. and W. Oakes, *Part II: ways of understanding repetition structures in computer programming languages*. Submitted to the Journal of Engineering Education, 2011.
41. Hestenes, D., M. Wells, and G. Swackhamer, *Force Concept Inventory*. The Physics Teacher, 1992. **30**: p. 141-158.
42. Michel, H., et al. *Computer Engineering Concept Inventory*. 2006 [cited July 27, 2007]; Available from: <http://www.foundationcoalition.org/home/keycomponents/concept/computer.html>.
43. The Joint Task Force on Computing Curricula, *Computing curricula 2001*. Journal of Educational Resources in Computing, 2001. **1**(3): p. 240.
44. Goldman, K., et al., *Identifying important and difficult concepts in introductory computing courses using a delphi process*, in *Proceedings of the 39th SIGCSE technical symposium on Computer science education*. 2008, ACM: Portland, OR, USA.