The Dimensions of Variation in the Teaching of Data Structures

Raymond Lister
University of Technology, Sydney
Faculty of Information Technology
Sydney, NSW, 2007, Australia
+61 2 9514 1850
raymond@it.uts.edu.au

Ilona Box
University of Western Sydney
School of Computing and IT
Penrith, NSW, 1797, Australia
+61 2 4736 0715
ilona@cit.uws.edu.au

Briana Morrison
Southern Polytechnic State University
School of Computing & Software Engineering, Marietta, GA 30060
+770.528.4295
bmorriso@spsu.edu

Josh Tenenberg
University of Washington, Tacoma
Institute of Technology
1900 Commerce St., Tacoma, WA 98402-3100
+253.692.5800
jtenenbg@u.washington.edu

D. Suzanne Westbrook
University of Arizona
Department of Computer Science
Tucson, AZ 85721
+520.626.9196
sw@cs.arizona.edu

ABSTRACT

The current debate about the teaching of data structures is hampered because, as a community, we usually debate specifics about data structure implementations and libraries, when the real level of disagreement remains implicit – the intent behind our teaching. This paper presents a phenomenographic study of the intent of CS educators for teaching data structures in CS2. Based on interviews with Computer Science educators and analysis of CS literature, we identified five categories of intent: developing transferable thinking, improving students' programming skills, knowing "what's under the hood", knowledge of software libraries, and component thinking. The CS community needs to first debate at the level of these categories before moving to more specific issues. This study also serves as an example of how phenomenographic analysis can be used to inform debate on syllabus design in general.

Categories and Subject Descriptors

K.3.2 [**Computers and Education**]: Computer and Information Science Education – *computer science education, curriculum.* E.1 [**Data Structures**].

General Terms

Algorithms, Management, Design, Standardization.

Keywords: Data structures, CS2, introductory programming, phenomenography, STL, Java Collections Framework.

1. INTRODUCTION

In many computer science programs, students are now first taught an object-oriented programming language, such as Java or C++. Furthermore, these languages are now frequently being taught "objects first", which the ACM Computing Curricula 2001 [7] describes as an approach emphasizing "the principles of object-oriented programming and design from the very beginning. [The strategy] begins immediately with the notions of objects and inheritance...[and] then goes on to introduce more traditional control structures, but always in the context of an overarching focus on object-oriented design".

If CS1 students are being introduced to programming in the above way, what changes should now flow through to the rest of the programming curriculum? Specifically, should the teaching of data structures in CS2 change as a result of these changes in CS1? This issue has recently generated some discussion and disagreement. Tenenberg [10] has argued that the teaching of data structures should change, as "the existence of robust, standardized, generic frameworks of data structures and algorithms libraries such as STL gives rise to a new set of virtual machines". Westbrook [5] argues strongly that CS2 should not change, as "it is vitally important for students to see and implement the guts of these black boxes to develop both their programming skills ... and their design analysis skills".

How can there be such a difference of opinion on teaching something as basic to computer science as data structures? What influences the choices made about what to teach and how to teach it? In this paper, we explore the question of what are the variations in understanding that computer scientists have of the purpose of teaching data structures. We use a phenomenographic research approach to constitute the variation in perceptions. Our results are categories describing what underlies some course design choices. Based on interviews and text sources, we identify five categories representing various purposes of teaching data structures. The categories of description, the outcome space of the research, contribute to the CS community by providing a published taxonomy of purposes to inform debate about data structure course design.

1.1 Phenomenography

Phenomenography is a research approach that focuses on the qualitatively different ways people experience, understand, perceive, or conceptualize a phenomenon [8]. The underpinning philosophy is that there are a limited number of qualitative ways of experiencing phenomena. Phenomenographers usually collect their data by recording and transcribing interviews with a small number of interviewees. The transcripts are analyzed to identify one or more dimensions of variation; a dimension of variation is a set of categories somehow related, e.g. linearly or hierarchically, with a small number of categories in the set. Since phenomenographers wish to capture the variation in experiences, and not quantify the popularity of each experience (though this can be done in follow-up work), they can work with small numbers of interviewees.

In computer science, Booth conducted the seminal phenomenographic work [3]. She identified three different ways that students understand recursion. Some students describe re[illegible]on as a syntactic construct, a template to be filled in. Some students describe recursion as a mechanism for implementing repetition, and some see it as self-referencing. More recently, Berglund [2] studied student understandings of network protocols, and Cope [6] studied understandings of the concept of an information system.

In his phenomenographical study, Trigwell [11] described several different conceptions that teachers bring to their teaching. At one extreme, teachers focus on the content of their course, seeing teaching as the act of transmitting knowledge and concepts to the student. At the other extreme, teachers focus upon the student, seeing teaching as the act of helping students to develop or change their own conceptions. Trigwell identified several positions between these two extremes.

In this paper, we report upon our own phenomenographical study, to investigate the qualitatively different intentions CS2 educators bring to their teaching of data structures.

2. METHOD

Our data came from two different types of sources. One type was te[illegible] from either papers concerning this issue [5, 10] or textbooks [[illegible]., 9, 12]. In the case of textbooks, we looked for an articulation of a position in the preface or introductory material.

The other type of data source was interviews, via electronic mail. We interviewed five people. We gave the interviewees the paper for the SIGCSE 2003 panel session on this issue of data structures [5]. We then asked them to describe how they teach CS2, and their justification for doing so. All the interviewees are academics, in three countries. Most currently teach CS2, while the remainder have a strong interest in the skills of students emerging from CS2, as those students then enter their own courses. Most but not all of the interviewees have a PhD in computer science. The interviewees were all colleagues of the authors but not necessarily at the same institution, and were approached to offer an opinion on this issue. One of those interviewees subsequently became a co-author on this paper.

The data from these two types of sources was then analyzed in the phenomenographic style. Our focus for analysis was on the intentions CS2 educators bring to their teaching of data structures. The analysis was an iterative process. We did not begin with the categories; we formed the categories from what we found in the data. Quotations from each source were placed into categories. The categories were revised. The placing of quotes and revision of categories was iterated until we reached a consensus of what were the categories. We only added a category when we could identify quotations in support of that category from at least two different sources. The outcome space is the qualitative description of and selected quotes supporting each category.

It is important to understand that a single interviewee (or other data source) is not assigned to a single category. People naturally have several intentions when teaching data structures, although some intentions are more important to them than others. Therefore, a data source may be quoted in more than one of the following categories.

3. RESULTS

From the data, we identified five categories of instructor rationale for the purpose of teaching data structures.

3.1 Developing Transferable Thinking

Here, data structures are a vehicle for developing thinking skills that are important and transferable beyond their immediate application to data structures:

- *I see several deep concepts which are essential in a tertiary education for an IT professional, and I think that learning how fancy data structures are implemented is an excellent way to grapple with these concepts. ... analysis of algorithms ... a model of memory and inter-memory references ... inheritance and polymorphism ...* (Interviewee03)
- *... awareness that the obvious or straightforward way to do things is often markedly inferior to clever ways that have been discovered by researchers. Data structure implementations provides an ideal vehicle for this.* (Interviewee03)
- *... they see that good data design can make algorithms simpler and more understandable. They of course see dangers also but that is what education is about - evaluation, THINKING, choice followed by better choice.* (Interviewee04)
- *The design of a data structure is like the solution to a riddle: the process of developing the answer is as important as the answer itself.* [1]

3.2 Improving Students' Programming Skills

Here, implementation of data structures is used for improving the programming skills of students, especially their dexterity with recursion and pointers:

- *... they were still required to implement a binary search tree. This was done to reinforce their knowledge and usage of recursion and pointers.* (Interviewee01)

- *The implementation had more to do with refining their programming skills than learning data structures.* (Interviewee01)
- *Other than programming in a functional language, I know no better way to get recursion into a students head than having them program operations on trees. ... there is real psychological impact on the student who has written three pages of incorrect non-recursive code, and then sees their friend with a two line recursive solution that works.* (Interviewee03)
- *... it is vitally important for students to see and implement the guts of these black boxes to develop both their programming skills by working with dynamic data structures and their design analysis skills by examining tradeoffs in implementations* [Westbrook in 5].
- *It is up to us as teachers to provide our students with chances to take things ... apart and just tinker with them* [Westbrook in 5].
- *... reading and using the code without having written something similar is like watching Olympic ping pong on TV. It sure looks easy, even somewhat repetitious; however, the level of precision is only experienced by trying to do the same.* (Interviewee05)

3.3 Knowing "What's Under the Hood"

This category acknowledges a place for teaching the libraries. However, this category is reductionist. The assumption is that students must understand the parts from which the libraries are constructed if they are to use those libraries effectively:

- *Several of the ADTs introduced in this book are directly supported by the Java 2 collection classes. Programmers will naturally prefer to use these collection classes rather than design and implement their own. Nevertheless, choosing the right collection classes for each application requires an understanding of the properties of different ADTs and their alternative implementations. This book aims to give readers the necessary understanding.* [12]
- *A graduate should be convinced that fancy technology is understandable, and adjustable; they should feel that they can be masters of the magic that the Wizard hides behind the curtain. ... There are many powerful tools that IT students use; at least one should be dissected and reduced to parts that the student can imagine producing for themselves. Of the many choices for a technology whose internals can be uncovered, the collection class library is much more accessible than the compiler, operating system, DBMS, or graphics package.* (Interviewee03)
- *... they were still required to implement a binary search tree. This was done to... gain an appreciation for the STL implementations of red-black trees for sets and maps... Students began to consider the multiple options available and how to make choices between implementations.* (Interviewee01)
- *I believe your students will have questions about the STL. ... For example, what is a template? What's in a container? ... How does the list 'thing' work?* (Interviewee04)
- *Carpenters don't start their apprenticeship on a roof, they begin by learning about joints, weight bearing, angles and forming structure. Similarly, we don't start our students building compilers and editors (i.e. their tools) we start them on iteration, functions and parameter passing.* (Interviewee04)

3.4 Knowledge of Software Libraries

This category gives a central role to teaching data structure libraries. Of all the views, it is the most utilitarian, seeing data structures as a set of tools used for solving problems:

- *... many career paths will never lead the graduate to read or write code which implements the operations of a binary search tree, B-tree, hash table, heap-structured priority queue, etc. So for these structures, it's enough to know how to read and write code that uses them, based on their presence in good collection libraries...* (Interviewee03)
- *... because we used the STL, I was able to cover many more data structures than I normally would have.* (Interviewee01)
- *We have such wonderful resources at hand, we need to build upon the work that has already been done and allow our students to solve more difficult applications using the tools we have provided them. Electrical engineering students are not asked to design or implement existing capacitors; nor are construction students asked to design and implement hammers.* (Interviewee01)

3.5 Component Thinking

This category emphasizes the importance of students learning component engineering principles, such as black-box interaction, and code reuse.

- *Software Engineering is moving away from emphasis on the creation of code, toward emphasis on components and code reuse.* [Lister in 5]
- *...the advent of object-oriented methods and the emergence of object-oriented design patterns has lead to a profound change in the pedagogy of data structures and algorithms... the proper use of object-oriented techniques requires a fundamental change in the way the programs are designed and implemented...The primary goal of this book is to promote object-oriented design using Java and to illustrate the use of the emerging object-oriented design patterns... In particular,... singleton, container, enumeration, adapter and visitor.*[9]
- *By using the STL, the students could see how they didn't have to implement a class to gain the behavior of a data structure; they could use existing data structures with a collection of algorithms and gain the same behavior without additional work.* (Interviewee01)

- *... Software engineering teaches abstraction, reuse, and information hiding.* (Interviewee05)
- *Since novice student designs often have leaky interfaces, with I/O details and other assumptions about the particular domain of application creeping into what should be generic algorithms and data structures, the use of the framework thus encourages better design by enforcing a cleaner separation of responsibilities.* [10]
- *Students of the next generation will be programming virtual machines at much higher levels of abstraction ... The full use and extension of standardized data structures frameworks ... places a stronger emphasis on abstraction and design.* [10]

4. DISCUSSION - PHENOMENOGRAPHY

Before identifying the two dimensions of variation we see in our categories, we first need to discuss some issues relating to the methodology of phenomenography and the categories themselves.

4.1 Phenomenography

In considering the above results, we need to keep in mind two mistakes that can arise from a misunderstanding of phenomenography. First, phenomenography is a qualitative method of research, not quantitative. Hence we draw no conclusions about the broad popularity of any of the above categories within the computer science community. To make such conclusions would require significantly more data and a different research approach. The aim of phenomenographic research is to capture diversity. Second, the categories do not represent a single position adopted by one or more individuals. Typically, if an individual is shown the categories generated from phenomenographic research, they will identify with more than one position. There may be some positions to which they identify very strongly, and some positions to which they do not identify at all, but it is rare for a person to identify with only one category.

Together, the papers, textbooks, and interviewees provided eleven separate sources of data, which is a relatively low number of sources for a phenomenographic study, but not unusually low. Phenomenographers often continue to collect data until they believe they have reached "saturation". That is, they collect data and analyze it concurrently, ceasing to collect data when they have several consecutive interviews that do not lead to the identification of new categories. From our eleven sources, we do not claim to have reached saturation.

Interviewing more subjects may add more categories, but is unlikely to invalidate the categories we have identified in this paper. Suppose we interviewed another subject who articulated the following position: *in high school, I was taught Euclidean geometry, which I loved, and I think the implementation of data structures, particularly recursive data structures, has some of that same simple beauty. An appreciation of such beauty is an essential part of a liberal arts education.* Such an interview might lead us to add a category, "Aesthetic Appreciation", but it does not invalidate the existing categories. (One of our interviewees did describe the low level data structures conventionally taught as being "cute", which may be evidence for such an extra category.)

Phenomenographers do not necessarily identify a unique set of categories from the same data. For example, if Trigwell [11] examined our sources, he may find evidence for the same categories he identified in his study of approaches to teaching, which we discussed in the introduction of this paper. The categories identified in any study are to some extent dependent on the intent of the phenomenographer. Our intent was to illuminate the debate on the teaching of data structures, and we chose our categories accordingly.

If phenomenographers do not necessarily identify a unique set of categories from the same data, is phenomenographic work therefore not repeatable? (And therefore not science?) Phenomenographic work is repeatable in the following sense. If two people were given some categories, and some quotes from data, those people would usually place the quotes into the same categories. The readers can determine for themselves whether they would place most of the above quotes into the same categories as those into which the authors have placed them.

4.2 Data Structure Categories

In choosing our categories, we wanted to focus on the teaching of data structures as teachers conceived it should be done in an ideal world. One of our interviewees raised the issue of legacy code. That is, he justified his very traditional approach to teaching data structures by the argument that there is already a great deal of code "out there" which does directly implement linked lists, trees, etc. Programmers will be required to maintain such code for many years to come. We don't disagree with that argument, but we regard it as separate to our concern. By focusing on how data structures should be taught in an ideal world, we sought to identify any possible long-term trends in the teaching of data structures. If the way data structures is taught is ripe for change, then the existence of legacy code will complicate and slow the change, but legacy code will not stop such a change.

For some of the authors, the most unexpected insight to emerge from this study was the separation of "Knowledge of Software Libraries" from "Component Thinking". Until this study, some of the authors had not made this distinction. We believe that this is a common error in the general debate. People opposed to the early teaching of these software libraries may see such teaching as merely being unprincipled instruction for an application program interface, and thus almost an exercise in rote learning. The "Component Thinking" category is a much more principled position. Its proponents see these software libraries as requiring a different way of thinking about program design.

Because the distinction is not usually made between "Knowledge of Software Libraries" and "Component Thinking", the difference between proponents of "Component Thinking" and "Transferable Thinking Skills" is exaggerated during debate. Both of these positions focus on the cultivation of student thinking. The positions merely differ on whether component based thinking needs its own design methodology or whether the design issues of lower level data structures transfer to this higher level of abstraction. A greater focus on that issue might lead to a more constructive debate within our community.

4.3 The Dimensions of Variation

From four of our five categories, we identify two dimensions of variation, as shown in Table 1.

Table 1. The dimensions of variation

	Computer Science vs.	Object Engineering
Abstract	Developing Transferable Thinking	Component Thinking
vs. Concrete	Improving Students' Programming Skills	Knowledge of Software Libraries

In one of the dimensions, the variation is in the degree of abstraction. The categories "Improving Students' Programming Skills" and "Knowledge of Software Libraries" both emphasize implementation skills, whereas the categories "Developing Transferable Thinking" and "Component Thinking" both emphasize the design process.

The other dimension of variation is "Computer Science" versus "Object Engineering". The category "Developing Transferable Thinking" relates to the Turing Machine as a universal computational device, while the category "Improving Students' Programming Skills" relates to the realization of the Turing M[illegible]hine in the von Neumann architecture. On the other hand, '[illegible]ect Engineering" is not about building universal computational devices, but instead devices that are well suited to specific purposes.

The fifth category "Knowing What's Under the Hood" transcends the Computer Science vs. Object Engineering dialectic, but it is a position more concrete than abstract.

5. CONCLUSION

The categories we have identified can be used to inform debate about course design. We believe that current debate about the teaching of data structures is hampered because, as a community, we usually debate whether a particular data structure should be taught, when the real level of disagreement remains implicit – the intent behind the teaching of that data structure, as revealed in the above categories. Our hope is that educators will use the categories we have identified to make explicit the real differences of opinion.

Beyond data structures, this paper demonstrates how phenomenography can be used as a tool for syllabus design in general. It can be used to define various positions, before debating [illegible] pros and cons of the positions. People who may not normally j[illegible] a debate on syllabus design can be encouraged to articulate their position in a non-confrontational environment. We found the effort of analyzing our data led to a suspension of judgment. The effort of finding a category for each quote leads to a concentration on understanding what the author of the quote means, not on a judgment of the validity of author's argument. Because judgment was suspended, we were more susceptible to persuasion. By the time we finished the analysis, we saw merit in all the categories, not just the categories to which we had subscribed beforehand. Indeed, we found categories of which we were not even aware prior to this study. Beginning with a phenomenographic study may therefore lead to a more inclusive and comprehensive approach to syllabus design in general.

6. ACKNOWLEDGMENTS

The authors thank the interviewees for donating their time. They also thank Shirley Booth for her advice in the early stages of this project, and Anders Berglund for a very helpful discussion on the dimensions of variation. The authors first met and began discussing this topic during the *Bootstrapping* Project which was funded by the National Science Foundation (No. DUE-0122560). 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 Science Foundation

7. REFERENCES

[1] Bailey, D. Java Structures. McGraw-Hill, 2003 (2nd ed.)

[2] Berglund, A. (2002) "On the Understanding of Computer Network Protocols" http://user.it.uu.se/~andersb/lic/

[3] Booth, S. (1997) "*On Phenomenography, Learning and Teaching*" Higher Education Research & Development, Vol. 16, No. 2, pp. 135-158. The entire issue is devoted to phenomenography.

[4] Budd, T. Data Structures in C++ using the Standard Template Library. Addison Wesley, 1998.

[5] Collins, W, Lister, R, Tenenberg, J, Westbrook, S. *The Role of Framework Libraries in CS2* SIGCSE'03, February 19-23, 2003, Reno, Nevada, 403-404.

[6] Cope, C. (2002) "*Seeking Meaning: The Educationally Critical Aspect of Learning About Information Systems*", Proceedings of the Informing Science + IT Education Conference, Cork, Ireland. http://proceedings.elicohen.net/

[7] Joint Task Force on Computing Curricula. Computing Curricula 2001 Computer Science. Journal of Educational Resources in Computing (JERIC), 1 (3es), Fall 2001.

[8] Marton, F. Phenomenography-a research approach to investigating different understandings of reality, Journal of Thought, vol. 21, pp. 28-49, 1986.

[9] Preiss, B. Data Structures and Algorithms with Object-Oriented Design Patterns in Java. Wiley, 1999. http://www.brpreiss.com/books/opus5/public/front.pdf

[10] Tenenberg, A *Framework Approach to Teaching Data Structures* SIGCSE'03, February 19-23, 2003, Reno, Nevada, 210-214.

[11] Trigwell, K. "*Phenomenography: Discernment and Variation*"

http://www.learning.ox.ac.uk/iaul/Phenom_ISL_paper.pdf

[12] Watt, D, Brown, D. Java Collections: An Introduction to Abstract Data Types, Data Structures, and Algorithms. Wiley, 2001. http://www.dcs.gla.ac.uk/~daw/books/JC/preface.html