Undergraduate Conceptions of the Field of Computer Science

Michael Hewner
Rose–Hulman Institute of Technology
5500 Wabash Avenue
Terre Haute, IN
hewner@rose–hulman.edu

ABSTRACT

Students come to CS from a variety of backgrounds and with a variety of preconceptions. Some initially select CS with a very vague idea of the field they are majoring in. In this paper, I describe CS undergraduates' view of the field of Computer Science. The approach was qualitative and cognitive: I studied what students think CS is and how students reasoned about their courses and curriculum. Through the use of grounded theory in 37 qualitative interviews with students and student advisors, I extracted three different conceptions about CS found in undergraduate CS majors using Grounded Theory. Overall, students had reasonable views of CS at a high level but lacked specifics. Students had difficulty describing subfields of CS or anticipating the content of courses they selected.

Categories and Subject Descriptors

K.3.2 [**Computers and Education**]: Computer and Information Science Education — Curriculum

General Terms

Design, Documentation, Experimentation, Management

Keywords

Curriculum, Concentrations, Multi–disciplinary

1. INTRODUCTION

> "When I was younger, I called it programming, 'cause that's what I thought it was. When I first came to [college], my idea of what I might do was along the lines of: I'd make video games . . . But the actual term 'Computer Science', I had never heard until I declared my major as Computer Science, 'cause that was the closest thing to a programming word. And since then, it's only been the fact that this college calls it 'Computer Science' that makes me think of it. It seems like kind of an awkward term, honestly."
>
> —Freshman, Georgia Tech

Students come to CS from a variety of backgrounds and with a variety of preconceptions. Some (as in the quote above) select CS with a very vague idea of the field they are majoring in. Even if a student researches CS before they select a major, however, it is certainly reasonable to expect that their understanding of the field will develop as they progress through the undergraduate curriculum.

Although students' views of CS definitely develop, it is easy to imagine that students with an incomplete view of the field of CS might make poor educational decisions. For example, do most students in sophomore–level architecture courses understand why such a course is part of their curriculum? If not, does that make it more difficult for them to maintain motivation? If CS educators understand how students think about CS and what they expect, it is possible to design curriculum that can address students' views of CS.

In this paper, I describe CS undergraduates' view of the field of Computer Science. My goal was to answer two research questions:

1. What types of CS field conceptions exist in CS undergraduate students?
2. Do problematic view of CS cause educational problems?

My approach was qualitative and cognitive: I studied what students think CS is and how students reasoned about their courses and curriculum. Based on 37 interviews with students and student advisors, I extracted three main conceptions about CS found in undergraduate CS majors using Grounded Theory. Although these views do not reflect all viewpoints about CS, they do have implications for how educators design courses and curricula.

2. RELATED WORK

From the related work, we draw two main points:

1. From previous work in student views of CS, we infer that students entering the CS degree probably do not have a detailed understanding on the field [8, 3, 12]. Previous work suggests that undergraduate views of the field change over time, but specifics of what changes are unclear [2, 9].

The definitive version is available at: http://dx.doi.org/10.1145/2493394.2493414

2. From previous work in science education, know that students often have misconceptions about fields like science and improving their view is difficult [13]. Getting valid methods of student field–level ideas is also challenging; qualitative elicitation often needs to precede more structured instruments like surveys [1, 10].

The following two sections review these points in detail.

2.1 Conceptions of CS

Although little is known about the conceptions of CS undergraduates, there is a fair bit of research in pre–college students' interest in "computing careers". Very broadly, many students are interested in computing careers [7] but generally do not have concrete idea of what Computer Science means. Greening [8] asked high school students to complete the sentence "Computer Science is mostly about...": the majority said they didn't know or provided trivial answers like "computers". In a student of high school calculus students, Carter [3] found that 80% of students left blank the question "What is your impression of what Computer Science Majors learn? (leave blank if you have no idea)". Presumably, students who choose to major in CS may know more about the field than their peers but knowledge about CS among high school students definitely seems fairly low.

For college undergraduates, McGuffee [12] describes student responses to the question "What is Computer Science?" He reports that at the beginning of CS1, student conceptions are too broad, while at the beginning of CS2 students definitions are too narrowly focused on programming. Biggers et al. [2] compares attitudes about CS in seniors: some of whom left the CS major and some of whom stayed in the major to completion. Seniors who stayed in the major emphasized the broadness of CS as opposed to just programming. Similarly, Hewner and Guzdial [9] found that CS seniors essays about CS emphasized the breadth and excitement of the field.

Overall, the existing research suggests that student views of the field of CS are initially not very sophisticated [12, 8, 3]. Studies on CS seniors [2, 9] suggest that views do change over the course of an undergraduate career and may have an effect on student retention. But how views change and what different kinds of views exist in undergraduate CS majors past the first few courses is still an open question.

2.2 Studies of Field Conceptions

One area where field conceptions have been studied in detail is science education. Science educators often want students to learn "epistemology of science" — how research is conducted, how scientific theories are made, etc. Most science classes don't focus on explicitly teaching epistemology except in passing, but many educators believe that understanding the overall scientific endeavor is essential to student understanding. In general, the results of student epistemology of science research suggests that students have a variety of different specific misconceptions about the field of science and that attempts to educate students often have little effect [13].

Eliciting student epistemologies of science is actually a very similar problem to eliciting student conceptions of CS. Similar to CS, eliciting conceptions is difficult because there is no single expert viewpoint of science [13]. Most of the early work focused on quantitative measures of science, but criticism that researcher–designed quantitative instruments were not truly measuring students' viewpoints has motivated more qualitative approaches [10]. Aikenhead and Ryan, for example, argue that attempting to measure student epistemologies of science without initial qualitative elicitation makes it impossible to determine what quantitative survey results actually mean [1]. In their work designing an instrument for assessing student epistemologies of science, they found that semi–structured interviews provided the most unambiguous measure of student views, but that doing interviews was very time intensive.

Epistemology of science research suggest that students can lean specifics about a field like science while still maintaining significant misconceptions about the field as a whole. The research also suggests that qualitative understanding of student viewpoints needs to precede quantitative evaluation if the results are to have validity.

3. METHOD

My study was an open–ended qualitative interview study designed to understand what conceptions exist and how they affect student educational decisions. The primary data for this study came from interviews with undergraduate CS majors. I interviewed 33 students about their view of CS. The interviews were between 45 and 60 minutes. Student interviews were supplemented with a written survey (see section 3.3) as well as interviews with four student advisors.

3.1 Sampling and Recruitment

Recruitment was done through presentations in CS classes. Students were asked to volunteer and offered a gift certificate to compensate them for participating. To select students to interview, I used the grounded theory practice of theoretical sampling [4]. In theoretical sampling, a researcher begins with an initial population to interview and then selects future candidates based on what would further help elaborate the developing theory. This allows the researcher to discover factors that seem to have an effect on interview responses and pursue them. However, this method does not attempt to make the sample statistically representative.

I selected students to interview in order to get a range of academic success, gender, and ethnicity. Students were recruited from CS programs at three different schools:

- *Georgia Tech* is a competitive engineering school with a curriculum that allows a great degree of student choice in CS courses.
- *Duke University* is a competitive liberal arts school with a more proscribed CS curriculum but greater focus on multi–degree programs.
- *Spelman College* is a traditionally African–American Woman's college. Students interviewed usually had not taken CS course prior to coming to Spelman.

Students were interviews at all stages of their undergraduate careers, with particular focus on sophomores and juniors.

3.2 Interview Method

Initial interviews used fairly focused questions, including questions like "how would you define CS". As I gained practice it was clear that kind of questioning was not productive. Students were very wary of defining the field. Students also did not have knowledge about subfields of CS (e.g. architecture, programming languages) – even at a very high level.

As a result, the interview process developed significantly throughout the study (in accordance with a grounded theory approach).

In later interviews, questions began with asking students about their own experiences in the major. I asked students to describe their courses and courses they were interested in the future. Usually this would naturally segue into a discussion of CS as a whole and students felt more comfortable asserting their options. I would also ask questions designed to test the boudaries of student understanding of the field, for example:

1. Whether they considered a particular course they mentioned was completely CS, or a mix of CS and some other field
2. For an example of a "really Computer Sciencey" job
3. If they considered someone doing a particular job to be "doing Computer Science"
4. For what they expected to learn in a particular (future) CS course, or what they were intended to learn in a previous or current CS course.

3.3 Checks to Ensure Validity

When attempting to understand student conceptions, there is a risk of misinterpretation and bias. This is a common problem in qualitative research; even when participants and researchers act in good faith, it is difficult to understand when backgrounds and assumptions are different. There are a variety of techniques to mitigate this risk [11]. I used two: triangulation from multiple data sources and member checking.

For triangulation, I used a written survey instrument with concrete questions about CS, handed to students after the in person interview. In practice, this survey turned out to be not very useful: as stated before, students tend to be very wary of questions that ask them to define CS. As a result, students' written responses rarely conveyed much that could be compared against their interviews.

With three students, I also used member checking: providing the student with my analysis of their conception, and asking for feedback. In one case, I contacted the research participant after the initial interview and reinterviewed them with the my interpretation of their viewpoint. In two others, I presented my analysis after the regular interview process concluded. The students agreed with my analysis (even after careful probing to attempt to mitigate power differential that makes agreement suspect [4]).

3.4 Grounded Theory Analysis

The theory of student conceptions presented here was developed based on A grounded theory is based off careful line–by–line analysis of interview transcripts. My process was based off the approach outlined by Charmaz [4]:

1. First the researcher develops initial codes that describe what is being expressed in each line of the data.
2. Second, the researcher goes back through the body of research accumulated and selects 'focused' codes that explain larger segments of the data.
3. Third, the focused codes are abstracted into categories in a tentative theory that is then checked against other parts of the data to test its explanatory power. There are several techniques to help the researcher attempt to develop the categories in this larger theory including axial coding [6], theoretical coding [4], and situational maps [5].

For example, consider the quote below:

> "The way I see computer science is using a computer or — I guess not necessarily even a computer, but using an algorithm . . . Really, if you have an algorithm, you can just do it by hand or build a machine . . . [Computers] aren't necessarily central to carrying out an algorithm."
>
> —Junior, Duke University

One of the things I coded about this quote was the student's explicit differentiation between computer science and physical computers. The initial coding was abstracted into the code "you don't need a computer for CS" which brought together several different examples of students explicitly differentiating computer science (as the study of algorithms) from programming/computers. This definitely was different from several other students who explicitly explained CS as the study of programming. Eventually, this code and several others were abstracted into the superordinate category of the "theory–view" conception of CS. I then looked for examples of this code and insured that it occured in interviews that had other characteristics of the theory–view, testing that the overall category of theory–view made sense and was consistent with all the interview data.

4. THREE CONCEPTIONS OF CS

4.1 Theory–View: CS as Mathematical Study of Algorithms

> " . . . theory definitely is important - very, very important to Computer Science in sort of understanding the more theoretical aspects like what people are working on, like what are the constraints, where are the known problems, that sort of thing . . . So computer science, I feel like is much more actually theoretical and programming is just another skill essentially."
>
> —Junior, Duke University

The theory–view of CS focuses on the theoretical and mathematical aspects of CS as the most essential part of CS. 'Theory' as students used the word encompassed more than just the contents of a theory course: it includes the abstract portion of most CS courses (e.g. data structures, LL and LR grammars), but algorithms are frequently mentioned as the central idea. In this view, programming is a useful offshoot of the mathematics of CS, but it's clear that it is an application and not the core. Students with this conception would frequently emphasize that Computer Science exists beyond actual physical computers: they were the only group to mention that CS exists in puzzle games or algorithms humans execute in everyday life.

For theory–view students, CS was an academic discipline. All of them agreed it was possible to do programming without doing Computer Science. Some suggested that probably every professional programmer encountered hard problems and therefore was doing CS, while for others the programmer had to be working on a hard problem that required intricate algorithms.

It's important to note that this view of CS did not coincide with an interest in doing theoretically–oriented CS work. All of them agreed that theoretical CS was important to know, but none of these students were interested in pursuing theoretical CS as a career. Just like students with other conceptions of CS, they expressed frustration that proofs were difficult and not something they could see themselves doing long–term.

4.2 Programming–View: CS as Programming–Centric but Including Supporting Subfields

> "I'd say that computer science is a study; is a discipline, and that programming is how it takes form; how it's actually represented in the world. So I'd say programming is probably the end goal behind computer science, but - I mean, it's like comparing the study of automobiles to building a car. Well, you can contribute to the study of automobiles without ever actually building a car; many people do. But yes, we study it so we can build them."
>
> —Freshman, Georgia Tech

In programming–view, programming is central activity of Computer Science but it is supplemented by several subfields that do not directly involve programming. In this case, programming encompasses topics like data structures and the implementation of algorithms (for example, doing a project in a graphics class). Students acknowledged that a computer scientist must understand ideas like Big O, incomputablity, the structure of a processor, etc. However, the central activity of Computer Science is definitely programming: that is, someone proving something about an algorithm is doing a less Computer Science oriented activity than implementing that same algorithm.

Professonal programmers were seen as the exemplars of Computer Science in programming–view; CS was not primarily an academic discipline. Unlike the theory–view students, programming–view students did not emphasize that some programming is not CS. Students in this group valued expertise with particular technologies and often were interested in problems that had concrete technical aspects (e.g. an e–commerce solution with a database).

Programming–view students varied in how 'supplemental' the non–programming subfields of Computer Science appeared to be. On one extreme, the non–programming subfields of Computer Science are clearly fields in their own right with practitioners (e.g. in the analogy quoted at the beginning of the section, the "many people" who contribute without building a car). At the other extreme, there was definitely some *content* that Computer Scientists needed to know beyond programming, but someone who worked exclusively on the supplemental aspects was seen as on the edge of the discipline (e.g. someone who works on proofs about programs might be more of a mathematician than a Computer Scientist).

Some students in this group exhibited potentially problematic conceptions about CS. They readily agreed that non–programming aspects of their courses were useful to them, but they often had difficulty articulating good reasons why. Programming–view students argued that courses like discrete math were supposed to teach good mental habits or logical thinking skills. When asked about computer architecture courses, programming–view students could readily say that understanding the hardware could promote efficient programs, but had much greater difficulty thinking of an example why. They also frequently incorrectly anticipated the content of future courses.

4.3 Broad View: CS as Having Many Different Subfields

> "You can basically almost work anywhere. You can work for these corporate business, Microsoft, Google. You can work for the government, CIA, FBI. You can work as a computer analysis; you can work for the police department . . . You can build programs for them. You could work in their database and organize their files. You can analyze various things . . . "
>
> —Freshman, Spelman College

The third view of CS was as a very broad category that was interdisciplinary and included many distinctive (and equally important) subfields. Students with this viewpoint almost universally emphasized that Computer Science was much more than just programming and that a degree in Computer Science had a wide variety of applications.

Broad–view students struggled to articulate a division between using Computer Science and simply using a computer. They often gave examples of how programming could be used in interesting ways. They also often wanted to make clear that programming was not the only possible thing Computer Science could provide but it was difficult for them to come up with concrete examples. Occasionally students would veer into potentially problematic conceptions by strongly emphasizing fields in CS (like logic) that were not part of their school's curriculum. However, it is important to note that no one in this view subscribed to the simple notion that everything that involves using computers involves Computer Science.

Broad–view students included researchers, professional programmers, and others in their view of the field. "Researchers" in this case were generally academics working on some specific application of Computer Science. CS Theory tended to have a limited role and on occasion was ignored entirely. Topics like data structures and algorithms were considered important regardless of which area of CS one wished to pursue. Broad–view students were also likely to mention ethics and communication skills as things needed by all Computer Science majors.

Broad–view students often had goals outside of a traditional computer programmer role. They often described initially viewing Computer Science as about programming, but then discovering a wider view. Not all had negative experiences with programming, but that was common.

On occasion, broad–view students incorrectly identified the contents of future courses (e.g. saying that Operating Systems was about Linux distributions). This was more pro-

nounced in areas they had less interest in, like operating systems and architecture courses. Even though they generally brought up a greater variety of subfields of computer science than other students (especially interdisciplinary ones), they generally did not have detailed knowledge about them.

4.4 Students Attempting to Combine the Views

> "In my view at least, they do a lot of research in sort of — well, they spearhead a lot of those really cutting edge fields like [muffled] computing or sort of a security encryption, algorithms or like just algorithms in general maybe with different applications and things like that. But for me like computer science, like they're much more sort of into the research aspects and pushing the technologies on the theoretical fronts and sort of the experimental stages.
>
> Whereas I would call - I guess call myself a software engineer where I use these technologies and I like to learn about these systems, these new technologies being developed and learn how I can actually combine and build a system that can support a service and application."
>
> —Junior, Duke University

Most of the students interviewed fit fairly unambiguously into one viewpoint or another. There are a few that lie on the border between one viewpoint and another. Here's a student who has reconciled The Theory View and the Programming View by contrasting Computer Science (theory) with a field of "software engineering". Note that by the words "software enginneering" the student appeared to be thinking about programming – not the academic subfield of CS called software engineering. Insofar as the term "computer science" is being applied to the theory side, one might say this is a theory—view but the view seems more nuanced than that.

When talking to juniors and seniors, one definitely gets the view that students' viewpoints about Computer Science are still evolving. This was borne out in the interviews with graduated students; they usually discussed their view as continuing to change past their graduation. The three viewpoints are attractive to students: they provide a single coherent explanation and that makes students want to subscribe to one or another. However even experts don't agree on a simple definition of Computer Science, so any simple view is inevitably going to have some contradictions. As students become more sophisticated, it is reasonable to expect them to combine views and allow for differing opinions about CS. This process seems to be just starting for most undergraduates; for most of the students in this study, a single view provided a sufficient explanation of CS.

4.5 Potential Problems With the Three Main Conceptions

One of the goals of this research was to identify potential student issues with CS. If a student has expectations about CS, and these expectations are in conflict with the curriculum of their school, there is a potential for difficulties.

Based on interviews, students' conceptions were accurate at a high level. No students considered CS to be about application use or as just IT work, for example. No students felt that the content of CS was overall useless to them, and that they needed to learn real skills independently. All the main conceptions are at least reasonable views of CS.

But student views of CS also had some potential problems. This section will highlight two:

1. Students knew few specifics about the contents of future courses.
2. Students did not often understand the role of theory in CS.

4.5.1 Lack of Specifics About Future Courses

> "Like I was signing up for fall classes. Okay, do I want to take processer design or operating systems class? And, to be honest, that stuff looks very similar to me from my shoes, right. I don't know anything about either one, so how am I supposed to distinguish them?
>
> So is there anything I wish like I'd been told? Well, yeah. I wish people would say like - I mean it's sort of impossible to tell you about it until you're actually in it and doing it . . . they don't sit you down and say, okay, look at this screen of assembly code. That's what you're gonna do if you go into platforms. Or look at this screen of Python code. That's what you're gonna be doing if you're in artificial intelligence, right?"
> —Junior, Georgia Tech

Almost anytime I asked students to speculate about the content of a future course, students would explain they really did not know much about what the course entailed. This was true of required courses, elective courses they were looking forward to taking, or even courses they had signed up for in the next semester. It might not be fair to call this a potential problem because students had such a explicit assumption that entering a course with no concrete expectations was normal. That does not always mean that students could not speculate correctly. When I asked students to speculate and predict the content of their future courses, some of them could do it with fair degree of accuracy. The main point is that students weren't familiar with the specifics of their future courses and didn't normally think about them. This is consistent with vague nature of main conceptions: students were not aware of the subfields covered in their later courses, and so those subfields did not form part of their descriptions of the field of CS.

This particular issue challenged some of assumptions which began this research. Initially, questioning focused on cataloging concrete potentially problematic conceptions in specific areas (e.g. architecture, compilers, etc.). When it became clear that undergraduates do not reason about CS that specifically, the focus of the interview process changed away from detailed questions about particular areas.

4.5.2 Role of Theory

The place of theory in CS was definitely an area of student contention. For some students, it was central part of Computer Science, while for others it seemed almost a minor detail for analyzing an algorithm's speed and a few other obscure details. Many students disagreed with particular theoretical topics; they felt that theory was not useful to them. I did not consider this a problematic point of view:

experts and educators frequently disagree about what theoretical topics are useful in a particular course. However, it was also common for students to overemphasize the coding aspect of CS, especially when the theory was a mathematical idea they found difficult to learn:

> "'Cause when I was working on the project I didn't have any idea what I was doing for the calculus part until I took one of my friend to just like tell me, "These are the formulas you need to do." And then once I knew all the formulas I could just code them - like it wasn't a problem to code them. It was just I didn't know any of the formulas 'cause I don't really enjoy Calc 3 [for CS Majors]. So I feel like it's - like it should be subdivided.
>
> Like math majors should be able to know all that stuff if they need to but CS majors - that's not their priority. We don't need to know the calculus part. Like we can, I guess, talk to other people that are specialized in that. Like our specialty is creating code."
>
> —Sophomore, Georgia Tech

There were also students who's conception of CS simply did not include any aspect of theory or mathematical components of CS, even when pressed:

> "Programming I think is a lot of math, as well as when you're first starting off. When you're learning binary, when you're learning about memory and RAM, and that type of thing. I think that is definitely where the math comes in ... If you're trying to calculate something. If you're trying to build a program that is going to give you the sine, or cosine or a tangent, or the sine or cosign of something anywhere, you would definitely have to know what that is."
>
> —Senior, Spelman College

This definitely seems to be something that the student has forgotten or not did not understand, not simply an argument that theory was not useful. It's clear from the curriculum standards that these topics were part of the student's curriculum.

Students difficulty with recognizing theory as an aspect of CS is interesting, because almost all students included the idea of algorithms as some part of their definition of CS. It definitely seems possible for students to view the idea of algorithms as central in CS, while leaving out or questioning the mathematical analysis of algorithms. For all students, the idea of taking some problem and devising a new algorithm to solve it was an important CS activity. But for many students, devising a solution to an algorithmic problem was simply a skill separate from formal mathematics.

4.6 Change of Conception

This section discusses how students believed their views of CS had changed over time. Because this is based on student reflection rather than longitudinal interviews, it's important to treat this data with caution. That said, student reflections on how their views changed provides insight into what changes seemed significant to students in retrospect, even if it is not a completely accurate view of the entire process of conception change.

4.6.1 CS is Not Just Programming

> "I mean, since I was little I just saw, when I think of computer science I thought of my dad all the time and all he did, he was a coder, a developer. So I just like imagine him when someone says computer science. Oh dad, what does he do? Code, that's what he does ... after coming to college my idea of what computer science actually changed ... I think it changed pretty quickly. Like joining the different organizations I saw people always talking about different [specializations] they're taking ..."
>
> —Sophomore, Georgia Tech

Many students talked about how initially they viewed CS as just programming but that view changed either in high school or early in college. Many students had computers science classes in high school that they described as basically programming. Even before students had taken a high–school CS class, they had somehow heard it was programming. Students described expecting CS to be learning additional programming constructs, or languages, or specific applications (e.g. how to build webpages) when they initially enrolled in CS. In addition to talking about non–programming subjects or careers with CS, student also emphasized that this programming–only view did not realize the importance of algorithms which they later came to appreciate. Many students remarked that CS seemed more interesting after it became clear it was not just programming (even students who enjoyed programming).

Students often contrasted their current views with thinking of CS an earlier view of CS as "just programming", but only one student actually asserted that CS was just programming in the interview. This change seemed to begin quite early in the curriculum — just about the time students were introduced to data structures. Some views that I classified as programming–view often were very close to just–programming in that students could not think of examples of CS activities outside of programming. Students have heard that CS is not just programming, but it seems like their view of the non–programming aspects of CS evolve over time.

4.7 CS Deeper Than Expected

> "I guess [the intro course is] a really easy class ... And I was like, 'Oh, I get this programming stuff.' And I got to [data structures] , and I was like, 'Whoa, I don't get this.' And so that was - I realized then that it was a little more complex. And I got it at the end of the class, but I was kind of more apprehensive about taking any more classes after that. And then after that I took [architecture] then I was even more apprehensive."
>
> —Senior, Georgia Tech

Similar to discovering that CS was not just programming, many students remarked that a lot more went into CS than initially anticipated. This viewpoint change also occurred early in the curriculum — anywhere from the first intro course to computer architecture. This was seen a less positive change by students: students were often attracted to

CS because it seemed easy and they performed better than their peers. Students experiencing this change often began to wonder if they had the made the right decision to major in CS.

The exact topic matter that was deeper than expected varied. Some students talked about designing code as being more challenging than they anticipated. Others were surprised about learning details of hardware. In all cases, it seems that the students conceptually found the new material interesting, but it was also more challenging than they first imagined.

4.8 Learning About Subfields of CS

> "Like before when I thought of robotics, it was kind of like two different classifications. You had either the robotics like industrial robots, which was just an arm doing some kind of task, moving something, or you had a humanoid robot, ... [Through class] I saw that there are very broad fields of robots ... there's robots that hop on one leg, robots that hop on two legs. We saw robots shaped like snakes that wiggle around and can climb up poles ... Just things like that I was like I had no idea we were even trying to do that much less that you could."
>
> —Junior, Georgia Tech

This last change of conception was in many ways the most interesting because it seemed to represent an elaboration of the main conceptions identified above. Among all the three main types of conceptions, students generally had a very vague understanding with regard to particular subfields of CS. Students from a broad viewpoint might mention that you could build robots in Computer Science but (maybe beyond one example) they could not elaborate on robotics or any area in particular. But, for a few students, a recent change occurred that encouraged them to deeply look into an area of CS. As a result of this research, they had significantly greater detailed knowledge which significantly expanded their idea of what was possible in Computer Science. In most the cases I interviewed, these students were seniors who had just started looking into this new subfield. They did not have details on the connections between their field and other areas of CS. This suggests that there are potentially more elaborated viewpoints of CS for some students after graduation.

5. DISCUSSION

5.1 Commonalities Between The Three Main Viewpoints

Although the three viewpoints are different, there are a few key ideas that are common to all conceptions. These ideas are worth highlighting because they are things an educator can probably assume most of his students agree with (at least in courses for sophomores or later):

1. *Programming is an important skill.* Students of all groups expected to do programming in their courses. Even when they personally did not enjoy it, or when they did not feel it was the "core" of Computer Science, they still considered it a major part of a CS education.

2. *Programming is not all of CS.* Even programming–view students acknowledged the importance of other skills. Although students sometimes misidentified the purpose of learning specific non–programming skills, all seemed convinced that other topics could be useful to them. This is not to say students agreed with everything they were taught: students complained about useless content in certain courses.

3. *Algorithms are essential to CS.* Students in every category mentioned the ideas of algorithms as essential to CS. For all groups, the idea of someone sitting down and coming up with an algorithm to solve a challenging problem was maybe the "most" CS–like activity possible.

4. *No detailed knowledge of subfields of CS.* Undergraduates tended to reason about CS in fairly broad strokes. Parts of CS that corresponded well with the students' outside knowledge might get mentioned (e.g. networking, databases, and robotics for example) but other less obvious areas would tend to get lumped together. For example, students would talk about the "low–level" parts of CS which seemed to contain (approximately) architecture, operating systems, and compliers (and sometimes building hardware). Even when students were pursuing a particular field in CS, they were just beginning to do research and did not yet understand the different subfields of a larger area like graphics.

5.2 Dealing with Vague Student Expectations for Classes

Students exhibited an accurate but high level view of Computer Science. They generally took an exploratory attitude toward class, and did not know specifics about the goals of the classes they selected. They usually did not have specific long–term educational goals about CS.

From an instructor's perspective, this confirms the initution that instructors must motivate the content they present. It seems logical to think that any student who registers for a particular elective must have some purpose in mind, but based on my interviews students generally take classes without any concrete goals for the class. Instead, students explore courses to see if they find the topic interesting.

The three perspectives provide evidence that student expectations for CS are diverse. The good news is that the is little evidence for students with very problematic conceptions of CS (e.g. CS as application use, CS as learning a particular language, etc.). Students of all conceptions expect CS classes to have some non–programming and some programming topics. However for courses that emphasize a particular aspect of CS (e.g. theory courses, HCI courses), it may be a good idea to very clearly set expectations upfront. For example, students with a programming–view might like to know that a HCI course will focus entirely on eliciting user stories and doing mockups; programming–view students might not expect that in a course.

Because students don't expect specific content from their courses, student critiques that course content is 'useless' may be more about frustration than the usefulness of the content. Students would criticize the content of their courses when they had a bad learning experience. Similarly, instructors occasionally complain that students have negative preconceptions about their course content. Students probably do

not have good reasons to complain about the content of their course, but neither are instructors right that students have negative preconceptions.

5.3 Are Student Conceptions of CS a Problem?

I began this research based on the concern that inaccurate student conceptions of the field of CS would cause educational problems. The literature suggests that students do not have a detailed conception of CS when they begin the major [12]. Given this, it seemed reasonable to ask if students knew enough about CS to make good decisions about persisting/leaving the major or choosing specific courses or specializations.

Based on this research, student conceptions of the field are in some ways better and in some ways worse than anticipated. None of the students interviewed had very problematic conceptions of CS (e.g. CS is IT work). Most students, however, had a very high level view of CS and that did not include details of sub–fields of CS or concrete plans. In particular, students did not generally have concrete expectations for the content of their courses. For courses students had already taken, students did not often understand why specific content was included in the curriculum.

However, given the difficulty of teaching field based conceptions [13] student conceptions of the field of CS do not seem to be a major problem. Although students (like experts) have diverse views about the field of CS, they know enough to make reasonable educational decisions.

6. CONCLUSION

The main contribution of this work was to develop a theory of student conceptions of the field of CS. Students had three main views:

1. *Theory–View: CS as Mathematical Study of Algorithms.* Students who held this view thought of CS as a primarily theoretical and mathematical discipline. The design of conceptually difficult algorithms was most central to CS, as were other mathematical ideas like Big O and NP–Completeness. Programming was viewed as useful but peripheral to CS, and students emphasized that CS could exist without any physical computer.

2. *Programming–View: CS as Programming–Centric but Including Supporting Subfields.* Students who held this view considered CS to be mainly about programming, but emphasized that other subfields were also necessary to do good programming. Writing programs to solve large and technically challenging problems was the central activity. Students with this view varied on the importance of non–programming subfields.

3. *Broad–View: CS as Having Many Different Subfields.* Students who held this view thought of CS as mix of many different computer–related subfields. Theory, Robotics, Programming, and (often many) others are all equally important parts of a broad CS 'umbrella'. In this view, comparatively little knowledge was considered 'essential' to a Computer Scientist; students emphasized the differences between subfields and the freedom to pursue different paths.

All three of these viewpoints were a high–level view of CS. Students did not know many details about subdisciplines of CS or the content of future courses.

Overall, student views of the field of CS are not as problematical as educators may fear. Students expect to learn both theoretical concepts and programming in their courses. Students do lack details on the specifics of their courses, and that is something that educators need to keep in mind as they design their materials. As long as educators motivate their content carefully and understand that there is a diversity of student opinion about the "true" nature of the field, it should be possible to design courses that are meaningful to students with every kind of CS conception.

7. ACKNOWLEDGMENTS

This research supported in part by a grant from the National Science Foundation BPC Program #0634629.

8. REFERENCES

[1] G. S. Aikenhead and A. G. Ryan. The development of a new instrument: "views on science–technology–society" (vosts). *Science Education*, 76(5):477–491, 1992.

[2] M. Biggers, A. Brauer, and T. Yilmaz. Student perceptions of computer science. In *Proceedings of SIGCSE 2008*, pages 402–406, Portland, OR, USA, 2008. ACM.

[3] L. Carter. Why students with an apparent aptitude for computer science don't choose to major in computer science. In *Proceedings of SIGCSE 2006*, pages 27–31, Houston, Texas, USA, 2006. ACM.

[4] K. Charmaz. *Constructing Grounded Theory*. Sage Publications Ltd, 1 edition, Jan. 2006.

[5] A. Clarke. *Situational Analysis: Grounded Theory After the Postmodern Turn*. Sage Publications, Thousand Oaks, Calif, 2005.

[6] J. Corbin and A. C. Strauss. *Basics of Qualitative Research*. Sage Publications, Inc, 3rd edition, 2008.

[7] W. E. Foundation and ACM. New image for computing: Report on market research. http://www.acm.org/membership/NIC.pdf, 2009.

[8] T. Greening. Computer science: through the eyes of potential students. In *Proceedings of the 3rd Australasian conference on Computer science education*, pages 145–154, The University of Queensland, Australia, 1998. ACM.

[9] M. Hewner and M. Guzdial. Attitudes about computing in postsecondary graduates. In *Proceeding of ICER 2008*, pages 71–78, Sydney, Australia, 2008. ACM.

[10] N. G. Lederman. Students' and teacher's conceptions of the nature of science: A review of the research. *Journal of Research in Science Teaching*, 29(4):331–359, 1992.

[11] Y. S. Lincoln and E. G. Guba. *Naturalistic inquiry*. SAGE, 1985.

[12] J. W. McGuffee. Defining computer science. *SIGCSE Bull.*, 32(2):74–76, 2000.

[13] W. A. Sandoval. Understanding students' practical epistemologies and their influence on learning through inquiry. *Science Education*, 89(4):634–656, 2005.