ABSTRACT
The constant comparative method was used to explicate the experience computing majors have with programming assignments in a CS1 course. Findings from a series of four interviews conducted with a purposeful sample of nine students revealed that the primary reflective experience with programming assignments was emotional. That is, the way students remembered and discussed their experiences with programming assignments was dominated by emotional experiences and reactions. We identified six stages whose dimensions capture the variation in students’ emotional experiences with programming assignments. This paper reports the beginnings of analysis geared at developing a theory of students’ programming assignment experience, rich in detail and grounded in student experience. When completed such a theory may lead to curricular supports, interventions, or tools, to help steer student experience away from the most harmful of emotional tolls.

Categories and Subject Descriptors
K.3.2 [Computer Science Education]: Introductory Programming.

General Terms
Human Factors.

Keywords
CS1, novice programmers, retention, emotion.

1. INTRODUCTION
The purpose of this study was to explore the experiences students had with programming assignments in an introductory computing course which used a media computation context. Students document that they spend a great deal of time doing required programming assignments in this course – at least as much as all other coursework combined (lecture, reading, etc.). We were interested in these experiences because of their potential impact on students’ views about computing and majoring in computing.

From a broader perspective, students’ experiences with programming assignments are somewhat unknown to computing educators – the instructor would rarely see students doing their homework. Here we provide a first look into the novice experience of programming, at least during assigned work.

The constant comparative method (Grounded Theory) [11] was used for discovering a theory or explanation grounded in actual reported phenomena. It is critical to note that the phenomenon that is being analyzed is not the actual (observed) student programming experience, but rather the perception of that experience reported by students shortly after a programming assignment was due. Our data comes from semi-structured interviews, repeated four times during the term, repeatedly modified based on students’ responses. Using the grounded theory process of performing data analysis to modify and refine observation (i.e., interviews), we discovered the emotional issues that dominated students’ perceptions and reflections on their programming experiences. In fact, students were often unable to describe processes or episodes (e.g., getting stuck, fixing a bug) except as triggered by an emotional connection.

The general research goal of this study was to explore how a group of computer science majors experience the process of doing programming assignments in a CS1 course taught with a media computation approach and using pair programming. This study offers a preliminary look at students’ emotional experiences when reflecting on a recent programming assignment. The results are the first step in developing a theory of students’ experiences that may guide instructors to steer students away from the most harmful emotional tolls.

2. RELATED WORK
The importance of the programming experience on issues of concern in introductory programming is unknown. However, a great deal of effort has been put into studies of student success in CS1 courses. This has spanned the range of attitudinal studies, to studies of the retention rate of context-based approaches (including media computation, games, and robotics). However, the majority of the previous CS education-related literature has focused on cognitive or behavioral aspects of the learning process whereas affective-related experiences have attracted less interest.

Eckerdal at al. [3] reports students’ emotional responses in learning problematic concepts (e.g., threshold concepts); they encounter strong emotions such as frustration and even depression. However, once the concept is understood, they feel excited, transformed, euphoric, and confident. Further, the authors suggest we recognize the relevance of students’ emotional responses to learning situations: “We need to consider how we can create a learning environment where the feelings that
programming is hard, magical and frightening are handled, and students move through them rather than give up.” [3, pp. 131]. In comparison, our work focused specifically on beginning students.

A potentially helpful model for the development of such a learning environment was developed by Kort et al. [6]. They suggest the learning process naturally involves a range of emotions that cycle from positive to negative affect. The model has three axes: 1) the emotional axis that symbolizes several emotional axes (e.g., anxiety-confidence, frustration-euphoria, dispirited-encouraged); 2) the learning axis, which symbolizes constructive learning-un-learning and; 3) the knowledge axis symbolizes evolving knowledge. This supports analysis of emotion as the learning process occurs. For example, a student might feel insightful at the start of the learning process, then confused as a result of encountering a difficulty; possibly transforming to frustration. Finally, after consolidating one’s own knowledge one may move on towards more positive emotions and the cycle starts again. We see similarities in the episodes students report in dealing with difficulties in programming.

Of more direct implication, [8] examined the connections between students’ emotions and their achievement in a CS1 course. Focusing on students’ observable affective states and behavior, they found feelings of confusion and boredom were associated with lower achievement. The preliminary study by Shaw [9] takes the Affective Events Theory as a starting point and suggests that emotions help explain job outcomes of senior-level undergraduate students engaged in a capstone system developer project. Our study aims at furthering the knowledge about CS majors’ emotional experiences at the very beginning of their studies.

3. METHODOLOGY

In spring 2009, we interviewed 31 freshmen at a large, public research intensive university about their first year experience as a computing major, focusing on their first programming courses. However, we found that specific programming experience memory had faded. We were unable to get detailed stories about what led them to experience programming assignments in the way they did – only high-level reflections. In order to further explicate this experience, we chose to perform repeated interviews throughout students’ first programming course in fall 2009. The interviewer was not involved with the course design or delivery. The analysis was conducted jointly by interviewer and the instructor of the course. However, data was anonymized before analysis and the instructor did not know which of the students were interviewed. Additionally, the course was run mostly unmodified from a previous offering – preliminary notions from this study did not affect course delivery.

We chose interviews over other data collection methods (e.g., observation) for several reasons. Interviews allow students to express their thoughts using their own words; semi-structured interviews allow interviewees to bring up themes outside of the researcher’s preconceptions. Second, while behavior conveys some experiential aspects, observations do not reveal what students think and why they make decisions. Third, students often study at times difficult (and likely intrusive) to observe.

Repeated interviews were selected for several reasons. First, they track students’ experiences over the entire term. Second, repeated interviews were important methodologically as repetition allowed us to adjust our protocol based on experience in previous interviews. Third, during the term students became more comfortable, and shared more personal thoughts. Overall, repeated interviews were critical in eliciting rich, meaningful data.

3.1 Sample

At a university orientation meeting, a purposeful sample of 18 freshmen computing majors were invited to be interviewed about their computing history, thoughts about their major, and expectations concerning computing classes. Women and minority students were over-represented in the sample to support study of as great a variety of experiences as possible. If we had, instead, used random sampling there would have been significant risk of our results not reflecting experiences of under-representative groups.

Next, we selected nine students for term-long repeated interviews. We selected those who were comfortable speaking and maintained diversity in the data (e.g., diversity in gender, ethnicity, and prior computing experience). Out of those nine interviewees six were male, three female. Hispanic, Asian and Caucasian ethnicities were evenly represented in the data (three students each). Four of the interviewees did not have prior programming experience. The rest of the students had some experience (e.g. had taken a high-school computer science class or had been a part of a high-school robotics team) but no one had significant background in programming. The interviewees, at the beginning of the course, also varied in their confidence about keeping their chosen (computing) major. On the one hand, there were students who were very confident that they would graduate with a computing degree. On the other hand, there were students who had many interests and were planning to "give CS a go” to see if it would be something they would enjoy.

Our interviewee sampling criterion sought to represent a large variety of experiences to promote development of a theory applicable to a great range of computing learners, not to support quantitative representation of experiences. The instructor of the course was not openly involved in recruiting students (though did participate in analysis) and did not know who the interviewees were. Students were encouraged not to feel obligated to participate.

3.2 Data Collection

Face to face interviews with the nine students lasting 20-55 minutes were conducted 5 times during the 10-week term in weeks 3, 5, 7, 8 and 10. Because the week 3 interviews were mostly spent getting students comfortable with the format and process, we report on the more substantive interviews weeks from 5, 7, 8, and 10.

An initial semi-structured interview asking students to recall and reflect on the programming assignment just completed (usually within the past 1-3 days) was continually refined and modified until week 7. The last interview plan deviated from the previous ones by focusing more on the students’ overall course experience.

Using the constant comparison model (where episodes are compared with other episodes), each week the interviewer and the instructor reviewed the interview field notes and identified aspects of students’ perceptions and descriptions of their experience that were unexpected or particularly interesting. Through this process our initial protocol (seeking to explicate the process of working through a programming assignment) became more focused on asking students to identify and discuss episodes driven by “getting stuck” or “finding an error”. However, students struggled to explain the process of encountering difficulties. In general, they
were unclear on exactly what happened, in what order, and how they managed to get unstuck. However, a ubiquitous aspect to their reflections involved a focus on their emotional experiences while programming. This led us to refine our protocol so that, in weeks 7 and 8, students were guided in discussing programming experience episodes that led them to experience some emotional category. Using field note analysis of previous interviews, five sets of emotional categories were created. They were organized in boxes containing various in vivo terms; that is words from previous student interviews (Figure 1). These boxes became the skeleton of the refined interview protocol. Students did not always recall any episode that led to any emotion in every category. In the last interview, students were encouraged to be reflective and discuss any episode that they strongly remembered from any programming experience that term.

**Figure 1: Categories of student-described emotions**

This process of refining and re-targeting the primary phenomenon of interest (and the protocol to explore that phenomenon) is key to grounded theory methodology. Especially in an area as little studied as introductory students’ programming experiences, it was essential to let our target of study be guided by and grounded in the actual data – students’ reflections on their programming assignment. Through this process, we have more trust in the theory or proto-theory outlined, because it is less directed by the preconceptions of the researchers and much more directed by the actual perceptions of the students. While we initially sought to illuminate the process of programming assignments, we eventually focused our phenomena on emotional experiences while programming; what contexts and conditions led to them, what strategies students used, and what the consequences were.

### 3.3 Data Analysis

Interviews were professionally transcribed for analysis. NVivo (a software package supporting qualitative analysis) was used for full coding, after initial coding by hand. Grounded theory is an appropriate analysis methodology for studying a phenomenon. Strauss and Corbin [11] propose the following framework to elicit detail about a phenomenon:

- **A** Causal conditions -> **(B)** Phenomenon ->  
- **(C)** Context -> **(D)** Intervening Conditions ->  
- **(E)** Action/Interaction Strategies -> **(F)** Consequences

**Figure 2: Phenomenon elicitation framework**

As stated by Strauss and Corbin [11, pp. 251]: “The purpose of grounded theory is to specify the conditions that give rise to specific sets of action/interaction pertaining to a phenomenon and the resulting consequences.” Here, we seek to specify and describe in a deep, detailed way the phenomenon of the student experience with doing a programming assignment -- through the lens of the 6-part framework described above -- with specific attention to emotional consequences. In general, grounded theory analysis should always begin with open coding and in some cases axial and then selective coding follows.

Open coding analysis initially identified in vivo words describing students’ emotions, with analysis proceeding one subject at a time. The goal of this coding was to “capture the theme or episode in the text” [12]. Open code examples include: concern about academic success, disbelief after facing multiple difficulties, frustration in trying to decide what to do, and rapid emotional change. We began with both authors jointly reading and coding two purposefully selected interviews (week 7:S1, S8). At this time, a rather large amount of codes were developed and, in conjunction with the interviewer’s observer-knowledge of other interviews and subjects, we proceeded to a first round of coding for categories – the end result of the open coding process. Categories developed from grouping related in vivo codes and naturally reflected common programming processes. As each interview often described multiple mini-episodes, the basic structure of the stages was quickly clear (Figure 3).

**Figure 3: Emotional stages of experiencing programming assignments**

Axial coding is a next step involving fleshing out dimensional properties related to categories, which reflect the range and variation of codes in those categories. This gives substance and depth to category definition. By comparing in vivo codes within a category repeatedly and discussing them in the context of the entire interview corpus, we defined category dimensions reflecting students’ experiences. For example, in the encountering difficulties category students’ frustration and confusion had various extents to which these emotions affected students, had varying degrees in the frequency students reported experiencing these emotions, and were affected based on the percieved ownership of the problem.

After open and axial coding the two interviews, all interviews were openly and axially coded in an intertwined process by the interviewer. While perhaps counter-intuitive, this is a natural process and is described by Strauss and Corbin [11, pp. 98]: “Though open and axial coding are distinct analytic procedures, when the researcher is actually engaged in analysis he or she alternates between the two modes.”

Next, the authors collaboratively reviewed the categories and the range of emotions in each and elected to continue with selective coding on a category-by-category level. Selective coding is the process of developing a “descriptive narrative about the central phenomenon of the study” [11, pp. 116]. In this work, we will treat each category as a phenomenon in itself and develop
begin by looking at each in vivo code, identifying the episode’s emotional consequences. We work backwards filling in as many of the four remaining components of [11]’s framework for eliciting detail about a phenomenon (causal condition, context, intervening conditions, and action/interaction strategies). Through review and discussion of these episodes, we developed representative descriptive stories capturing the dimensional ranges observed.

3.4 Threats to Validity
Including the instructor of the course in the team conducting the analysis may compose a threat to the validity of the research. However, we did take measures to reduce the impact of this threat: the instructor was not involved in recruiting students for interviews or conducting the interviews. The other researcher, who did the interviews, was not involved with organizing or delivering the course. The interviewer anonymized the data before analyzing it with the instructor. No changes were made to the delivery of the course as a response of the preliminary results. In addition, the results emerged as a result of discussions between two researchers (the instructor and the interviewer) diminishing the possibility that one researcher could have seen only what she wanted to see in the data. Each preliminary category had to be supported by the data and make sense to both analyzers. Additionally, theoretical sensitivity is an important quality the researcher should bring to bear in any grounded theory study. Theoretical sensitivity refers here to the researchers’ “awareness of the subtleties of meaning of data” [11, pp. 41]. Therefore, having the instructor, who knows the context and the content of the course well, involved with the analysis lends strength to analysis.

To help the reader in judging the validity of this research we have provided our reasoning for using a grounded theory approach and detailed description of the data collection and analysis procedures. We acknowledge this type of qualitative research has limitations – e.g., due small sample size, the generalizability of our results is limited. However, we believe the in-depth description of students’ experiences provide instructors and researchers novel insight into students’ programming worlds that is not readily observable in a classroom setting. In [7] we reflect more on our motivation and experience in using a grounded theory approach.

4. FINDINGS
From students’ emotional experiences, a six-phase process emerged defining the programming assignment experience. Perhaps comfortably, these stages should not be surprising to CS1 educators. However, these stages appeared as we analyzed students’ emotional experiences – and they did not cover every part of the programming process. For example, we have no “working on the assignment” stage – just “working along” has little emotional impact. Finally, through selective coding, we found an over-arching category of emotional experiences. These were not tied to any one stage, but were students’ reflections or descriptions of the process as a whole. The codes in this category all reflected auxiliary emotional experiences; emotions not directly tied to the actual programming assignment. As one would expect, both positive and negative emotional consequences are observed in each category, with negative experiences being more commonly reported.

In this paper we report on the four categories where we saw the most interesting results - shown in solid boxes in Figure 3. In future work, we’ll continue by exploring the remaining three categories and making a closer analysis of the interaction of the categories as they define the emotional programming experience as a whole.

4.1 Getting Started
Getting started refers to the time when the student engages with the programming assignment. This includes the times when students start working with the assignment for the first time as well as times when students re-engage with the assignment after taking a break. In elapsed time, this phase may be very short including activities, such as, reading the assignment, doing preparatory work, and discussing with a partner. The experiences of it’s Greek to me, OK, what now?, and self-efficacy assessment characterize the first phase of the process of doing programming assignment.

4.1.1 The “It’s Greek to me” experience
Some students faced their first challenge while reading the programming assignment directions and realizing that they do not understand the written guidelines. This is a separate phenomenon from not knowing where to start or how to do the assignment. Not understanding the directions meant that the process of doing the assignment was stymied right at the outset. The degree of not understanding varied from being totally lost “I really didn’t have an idea of what it was saying” [I5-H] to mentions of having a hard time understanding the written directions. Not understanding the directions led to not knowing what to do and this led to the feelings of confusion and puzzlement. The degree of those feelings varied from being totally lost to just being confused. Some students identified the densely written directions that contained a lot of details as a cause not understanding the instructions. Thus the reason for not understanding was perceived to relate to an outside cause.

Okay. I was puzzled a lot when I was reading the instructions because it’s so dense. I kind of wish that they were simpler in format, and then maybe to the side a little bit more detailed. But when the detail comes at you full screen, I think it’s just too hard to understand it the first time. I was confused with some of the content sometimes in terms of what I’m supposed to do … [15-B]

The strategies students mentioned to get a better understanding included reading the assignment again and discussing it with a programming partner.

4.1.2 The “OK, what now?” experience
The “OK, what now?” experience relates to the moment right after reading the directions. It is characterized by students’ perception of their ability or knowledge to do the assignment. This experience had two distinctive variations – not knowing how or where to start and not knowing how to do the assignment. The former experience was characterized by the feeling of really not knowing at all where to start and left students puzzled, confused and “froze up”.

I froze up in a like – it’s kind of the same as confused – after reading it, like okay, what do I do now? So starting each piece was always the difficult part because where to start was challenging. [15-A]

I’d really have to say the most difficult part of both of the codes was figuring out – just getting in your mind, “What am I doing
and how is this going to look in Java?” So like just for – like edge detection I was thinking, “Edge – how would you even begin to do edge detection,” but of course the book had the answer. So I was like, “Oh okay.” [I2-J]

The latter case refers to the situation where the student understands where to start but how, exactly, to do the rest of the assignment is still somewhat unclear. In this situation students used strategies, such as, looking for examples in the textbook, doing research online and starting writing the code down even though one did not exactly know what to do.

Once we started writing it down that eventually became like, “Oh this is what we’re doing. Okay this is making sense now.” Before when you’re just looking at a blank sheet you’re like, “I – where do I start? I don’t know.” But once the thing gets – it starts to get built it’s easier. [I2-J]

The two embodiments of the OK, what now experience differ in regard to how lost students felt. Not knowing how to start was characterized as being totally lost whereas not knowing how to do the assignment was characterized more positively by having some strategies to employ. “It wasn’t like extremely confusing. It was okay.” [I5-E]

4.1.3 The self-efficacy assessment experience

Reading the assignment directions resulted in students reflecting on their abilities to do the assignment – in two different ways. On the one hand, a student who was totally lost and had no idea of what the assignment was about or how to proceed assessed his ability to do the assignment as low: “What I am doing? What is this code? What is going on?” [I2-J]. This reaction resulted in concern about academic success “I am failing this homework so bad right now” [I2-J].

On the other hand, in some cases students read the assignment directions with delight because they noticed familiarities. Familiarity reinforced the student’s positive assessment of his/her ability to do the assignment. Students often labeled as “easy” assignments, which recalled previous tasks already done. The assignment was characterized more positively by having some strategies to employ. “It wasn’t like extremely confusing. It was okay.” [I5-E].

Wow. This is actually – sounds a lot easier than the other things that I’ve had to do before … [I2-J]

4.2 Encountering Difficulties

When a student very first encounters a problem in a programming assignment, they experience some very strong emotions – and this emotional response precedes and is separate from their experience in dealing with that problem. Unsurprisingly, the majority of emotional consequences of encountering a problem are negative – though this can range from confusion and puzzlement to much more anguished frustration, anger, and sense of “oh no, not again”.

4.2.1 The hit by lightning experience

A pervasive and striking commonality in students’ experience in encountering a problem is the sense that this experience comes completely out of the blue, occurs in the context of a state of student confidence about their program (“I’m sure it’s right”), and leaves them with little idea about what is wrong or what to do next. In this sense, these students have an experience somewhat like someone hit by lightning when just pleasantly walking down the street.

I mean, you think it should be working [I4-J]. We weren’t getting any compiler errors [I2-D]. We know – we think the code is right. [I4-J].

Why is this not working? What is going on? [I4-J] I was puzzled and confused…we didn’t know where they [the errors] came from. [I5-G] We were really confused…we couldn’t figure out what the problem was. And then we asked tutors and they couldn’t figure it out either, so we were just wondering what was going on, basically [I4-G].

Students seem to be “struck” by errors just when everything seems to be going well. There is a definite sense of being taken back and almost betrayed by the error.

The causal conditions triggering these events varied, but seemed unrelated to the emotional consequence. Their reports ranged from a general “it didn’t work”, to “the program didn’t do anything”, to more specific descriptions comparing incorrect and correct behavior. However, many of the reports were general in flavor with little detail about what was wrong. While it may seem this is an artifact of the post-problem interview (e.g. that students just don’t remember), we believe it to actually be more likely to reflect the complete “struck by lightning” experience in that, even now, after completing the assignment, the student still has very little descriptively to say about the cause of the problem.

The consequences that resulted after encountering the error seems to reflect the struck by lightning experience as well. After “being hit”, students are dazed, with little sense of what to do next. They don’t know what was wrong, where the errors came from, or what to do next. These experiences left students puzzled, confused, frustrated, overwhelmed, and annoyed.

Additionally, later in the term, an additional effect emerges in some students where they are resigned to this “hit” – “oh no, not again” – where they have yet again been sure that things were now right, and then, no, that does not work again. In the most positive cases, this results in both a sense of tired resignation and in some reluctance to face this problem again. However, in many cases this sense of “not again” leads to anguish and feelings of being beaten down.

Okay, this should work. Compile. And then the compiler says “no” and we’re like, Oh no, not again. [I5-J] Not this. [I4-F] Oh, come on. It’s not doing this to us again, is it? Oh man. Give us a break here. [I2-J]

4.2.2 The self-efficacy experience

Another, distinctly different, experience with encountering a problem involves reflection on one’s abilities. Here students remember, “we felt bad about [when] we made mistakes” [I4-F]. Often these reflections of personal failures, while stemming from the encounter with the problem, extend to a reflection on the entire experience with the problem:

Occasionally, there would be moments where I did feel stupid because it was such an obvious answer and me and my partner just kept going around in circles [I5-I].

Self-efficacy was also measured in the perception of the ease with which others in the class seemed to “get it”.

4.2.3 The rapid change experience

In an elaboration of the struck by lightning experience, students were affected by the rapidity of emotional change that encountering an error caused.
So then it quickly changed to like the four stages of grief and we – again denial. [I2-J]

Although the phenomenon of encountering a difficulty does tie very closely into dealing with difficulties, it was clear that the first encounter with a problem was a distinct emotional experience for students. However, the emotional consequences and the nature of the problem they encountered lead directly into the causal conditions for dealing with a difficulty.

### 4.3 Dealing with Difficulties

Students reflected on the process of dealing with a difficulty by describing what they actually did and the emotional consequences of those actions in the context of their programming process. These experiences fell into two broad categories determined by whether students used feedback or simply kept trudging along with seeming blindness and little hope. We label these the feedback guided experience and hamster wheel experience.

However, other experiences when dealing with difficulties were not directly related to the programming experience. These included experiences involving self-efficacy, course-specific issues, and other computing factors.

#### 4.3.1 The feedback guided experience

A distinct manner of dealing with difficulties was for students to use some form of feedback in guiding their actions. While this sounds promising, the dimensional range of the use of feedback varied widely as described in Table 1.

<table>
<thead>
<tr>
<th>Binary Information</th>
<th>Low Information</th>
<th>High Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>It works/It doesn’t work (can refer to compiles or runs)</td>
<td>I noticed that something I changed altered the output, but I don’t know what the problem is</td>
<td>I know what the problem is, and what I need to do, but I might struggle to actually do it</td>
</tr>
</tbody>
</table>

**High Information:** We got a lot of “out of bounds” errors. And trying to figure out how exactly we were getting those – we knew generally how we were getting them, but we were trying to figure out how much, and so that was kinda confusing. And we thought we had all the right numbers, and then it’s be wrong and we’d be puzzled and look at it and try some other stuff and eventually got it working. [I3-D]

In high information experiences, the student seems to know at the problem level what the issue is. This information drives a quite specific experimentation process; but even then, things don’t always work out right away leaving students puzzled or confused.

**Low Information:** We run it and we look what happened and it’s not what we want. We don’t immediately see what’s wrong. We look around at various parts of the code. You know, maybe change a couple of things and it’s still not running right, but every time we change something we’ll get a different outcome -- hopefully. So that’ll help lead [us] to figure out which actual line is wrong... I was really confused for [a time]. [I2-D]

The contrast between low and high information lies in the “knowledge” that the student has about the error. In the low information situation, all that is known is that there is an error, and perhaps some basic information on where it occurs. This experience relies on pure experimental observation to link a code change (or changes in this case) to desirable (or not) changes in output. In another scenario, the student was trying to “flip a line” (reverse an array) and it’s not clear that low-level experimentation leads to full comprehension even when it leads to success.

**Binary Information:** So we were just sitting and thinking. And then we tried something and it didn’t work. We went back and we sat there and we thought and we tried something and it didn’t work. So I mean at that point, it was a little more annoyed rather than frustrated, because it was like we had no idea which part would make it work and we were just trying to wrap our heads around [it]. [I4-J]

Here students deal with difficulties using the most basic of feedback possible – whether the program runs correctly (or compiles) or not. Changes might be made between efforts, but those changes do not appear to be guided in any way from the previous effort.

#### 4.3.2 The hamster wheel experience

In contrast to those students acknowledging in some way the use of feedback in guiding their efforts (however minimally), others’ emotional experiences were characterized by a focus on the repeated efforts at fixes and the simple repetition of efforts with no reflection or recognition of their purpose. These experiences often led to notably negative emotional reactions.

I guess I experience some sort of despair because if you go through it again and you go through it, you just feel like you’re stuck. [I4-E] You try it once, you try to fix [it] and it turns out like, “Not again.” It’s like “not again”. So then you have to go back and fix it ... I was sick of this code not working and whatever we were trying wasn’t working. [I2-J]

#### 4.3.3 The self-efficacy experience

Sometimes students’ reflections on their dealing with difficulties revealed that they used the difficulties they encountered to self-assess their possible success in the course, or their “ability” to be able to deal with this difficulty.

“I have an idea what to do right now, but trying to get it to work is not working out for me right now”. It just felt like more annoyance and frustration... than absolute confusion. [I4-J] I felt kind of stupid when I didn’t put x--, but it wasn’t like last time. I didn’t say “Oh, my god, I’m so stupid.” I was like, “Oh, okay, that’s how you do it.” [I4-B] [I was] frustrated and tired and a little bit apprehensive ‘cause what if something like this was on the mid-term, how would we figure it out? [I2-F]

Fairly uncommonly, this self-efficacy assessment had a positive impact.

I always knew that there was like some way of solving this that I just wasn’t seeing at the time, so I just – I wasn’t really that stressed ‘cause I knew she wouldn’t hand us an impossible assignment. [I5-J]

#### 4.3.4 The course-specific experience

Other experiences stemmed from specific factors related to the programming assignment as part of a “course” – with the limitations and deadlines associated.

[I was] scared that we weren’t going to finish it or figure it out, on Monday night, especially when it was 11:00pm, and we couldn’t figure it out, and we didn’t have a working [beginning
portion of the assignment] yet. So that was a little scary because it was due the next day. [I3-G]

4.3.5 The other computing factors experience
Unrelated computing problems that are not intended to be part of the programming experience also impact students.

I was frustrated when... I had to use another program to make it [the input] longer. I wouldn’t have known to do that. So that really frustrated me. [I5-B] Just the lack of knowledge of these new functions on Linux. [I5-B]

4.4 Overarching Emotional Experiences
While our chosen protocol often focused students on specific recent programming assignment experiences, at times students stopped to reflect more generally on their overall experiences. These situations contribute to our understanding of students’ general overarching reflection on the programming experience. While not directly episodic, they fall in categories similar to ones seen in our previous stages including the self-efficacy assessment experience and the life outside the assignment experience.

4.4.1 The self-efficacy assessment experience
Students often reflect generally on their experience in relation to their peers. The emotional consequences of these reflections are often negative, but at times can actually be positively reinforcing. These experiences affect students’ perception both of how capable they are in relation to other students and also of their abilities related to what is expected in the course. Some students compared their own performance to other students (including their programming partner) and concluded that they are less capable of doing the assignment. This left them feeling inadequate or stupid.

I did feel inadequate in some 'cause I would look across the hallway and people that knew what they were doing really fast finished it in like half an hour tops. And would take it into class and it would be incredible and I’d feel a little inadequate like, “Am I supposed to be like them and able to do it really, really quick, or am I far back and inadequate?” [I5-F]

Specifically, because students were pair programming, the partner-peer relationship triggered additional self-efficacy concerns. While students sometimes self-assessed their abilities compared to their partner, additionally, the partner relationship often led to frank, not-so-tactful assessment by the partner. Being in a partner programming relationship that was unsupportive, or downright hostile, affected students’ perception of their own abilities.

Well, usually I feel those when my partner makes me feel like stupid, I guess. Because I usually try to contribute, but when his response is more like, "Oh, no, you're wrong. It's obvious that it's wrong." Or like something very like – I mean, I guess because he's taking a programming before, so he has like a background on it, and this is my first one. So like to him it's obvious, but to me it's like, "Sorry." [I3-A]

In other cases positive effects of students’ self-efficacy assessment were seen. The positive assessment concerning own abilities often left students feeling relaxed and confident. There were several contributing factors to positive self-assessment including previous knowledge due to high-school programming experience, studying the textbook before starting the assignment, or noticing similarities to previous programming assignments.

I guess I was really relaxed while writing this. I mean, there wasn’t much to it that I didn’t already know. Because the reverse method is in the book, that wasn’t too much of a surprise. [I4-E]

Success in relation to having a clear understanding of what to do and how to get the assignment done or already having a large part of the assignment working contributed to the relaxed feelings.

We really got it to work, so we just put it together and sent it off. [I4-H] We knew what we were doing, I mean, I can imagine if we came in and we had no idea what was going on and we were really confused, we wouldn’t be as relaxed after that ... we were getting a lot of stuff right, which helps with being relaxed. So yeah, just being successful at the programming helps. [I3-D]

Again, partner-peers impacted self-efficacy assessments in positive ways. Partners provided reassurance by lifting each other up and thus decreasing stress, contributing to increased energy, and reassuring in times of difficulty.

Well, she really just reminds me of the things that I actually do contribute to the group. And so, I mean I do make silly mistakes, but so does she. And she reminds me of that. I know that, it’s just that I need to be reassured. [Laughs] And then she tries to lift me up... and then so I am all excited when things go right, and then as things are going right, I mean she’s still tired... And then I’m high-energy here, and so I’m like, “Debbie, let’s do this,” and she’s like, “No.” [Laughs] [I3-B]

4.4.2 The life outside this assignment experience
Another emotional consequence resulted from students’ concern about life outside the programming assignment. The pressure to keep up with other studies and manage overloaded schedules caused students to be stressed, tired, and apprehensive. Students found it challenging to find time to work with their partner, limiting their time available to work on the assignment – thus adding extra stress. Experiences specifically regarding partner effects will be explored in future work.

[I was] stressed because we have to finish this on time, and we can’t meet on Monday, Tuesday or Saturday and Sunday. So we had to do it on Friday. But she has to go somewhere or I have to go somewhere, this kind of thing. [I5-B]

4.4.3 The helping others experience
Another positive emotional experience was related to when students would help others struggling with the assignment. They did this when they felt confident enough in their own understanding (usually after finishing). These helping experiences left them feeling accomplished, proud, and happy.

I felt accomplished and happy [because] on top of finishing mine, I helped two other friends and a random person... explaining to [them] helped me understand it even more. Yeah, it just increased the feeling of being proud and happy and accomplished. [I5-A]

5. DISCUSSION
It is clear that students’ emotional experiences in distinct phases of working on a programming assignment are rich, impactful experiences. These likely affect students’ formation of their views regarding computing and of themselves as successful computing majors. Although experiences in various phases have unique properties, common themes prompt further discussion. This paper reports the first three parts of the results relating to the students’ experiences. Even though the partial results do not form a coherent theory yet, they give a glimpse of some important aspects of the emerging theory.
5.1 Auxiliary Emotional Load

Cognitive load theory says that learning is impeded the more cognitive issues one is required to handle simultaneously. In instructional design, one should attend to this load, and recognize when unnecessary items are adding to cognitive load. Here, many of the emotional experiences students have with programming assignments add to their emotional load. A common theme of auxiliary emotional load emerged. These loads occurred in two areas: self-efficacy loads and outside factors. This may be important to pay attention to, because they represent emotional tolls beyond what is “strictly necessary” in a programming experience. Instructors or curriculum may work to alleviate this.

5.1.1 Self-Efficacy Assessments for Better or Worse

In every stage of the programming assignment experience analyzed in this paper, we find a common impact on emotions from students’ self-assessment. Whether this is an assessment of their ability to do the work before or during the programming process, this experience was pervasive and seems likely to have an impact on both the ability and the persistence of students in completing the assignment. Verifying this proposed connection was beyond the focus of this study, but is an interesting avenue for future work.

Carol Dweck [2] shows that focusing students on the “performance” aspect of a task (e.g., this test will tell us how smart you are) can reduce performance. In contrast, focusing students on the task at hand as a “learning experience” that, while possibly difficult, can grow their intelligence, increases performance. It seems challenging to get students to approach programming assignments with a growth mindset. One approach reported by Cutts [1], uses standard homework feedback with a specific statement to remind students that it takes time to learn programming and that making mistakes is a valuable way to learn. Our evidence that student performance assessment occurs ubiquitously in the programming experience lends weight to the importance of continued work to address student growth mindsets – perhaps especially in developing software tools to support students in growth mindset processes within the programming environment as recommended by [10].

In contrast, students with positive self-efficacy assessments not only avoided negative emotional experiences, but gave enthusiastic reports of how this assessment made them feel better confident, and reassured. Clearly there is much benefit to be gained from getting students to, in some way, focus on what they got right and what they do understand.

5.1.2 Impact of Outside Factors

Another form of auxiliary emotional load is seen across the programming experience. These emotions stem from factors outside the specific programming assignment including course issues (deadlines, etc), other life issues (time management, students’ other obligations), and other “computing” problems not meant to have been part of the challenge of the programming assignment. It was clear that while students were able to see outside sources as the cause of these experiences, they were still a part of their reflection of the programming experience. Some of these loads seem challenging to remove, but options may exist (e.g., rolling deadlines with late penalties).

5.2 Really Lost, or Just Lost?

Students struggle when programming – but exactly how bad is it? There are those students who are completely unaware of how to start or what to do. They don’t know “how to read” – assignments or gather information from compiling or running code. They don’t know what to attend to, and, in the case of dealing with errors, they don’t seem to understand that taking in information from the process should guide them. This differs significantly from those who think they know what to do; though they may struggle or even be wrong in what they think they know. Though we discern this difference from analysis of students’ emotional experiences, Garner [4] also report this difference in studying the kinds of questions introductory students ask in programming labs. Their problem 2 (can’t understand the task) and problem 3 (stuck on program design) are reflected in our it’s Greek to me and OK, what now? experiences with getting started.

From an emotional standpoint, these two states would seem likely to have very different effects on students. Additionally, in order to effectively help students make progress, it is important to first identify which state they are in. Anecdotally, many faculty and staff interacting with a student start from the viewpoint that students are in the second, more knowing state. Doing so may lead to “assistance” that is meaningless to students.

5.3 The Mark of the Utter Novice

Perhaps one of the most striking and surprising categories of experience was the hit by lightning experience when encountering a difficulty. Not only was this reported in a variety of occasions, but also the stories were particularly poignant.

As an educator, it is hard to imagine anyone wanting to continue programming if an unexpected error leads to this sort of experience. We found some subjects reporting less of this as the term went on. Perhaps this is the mark of the true novice. Being completely “struck down” with no idea what happened or why, completely out of the blue, is not an experience anyone “who programs” has (any more). Experienced programmers still experience errors, even stubborn or unexpected ones, but the reaction is not such an incredible emotional hit. Additionally, these errors are somewhat expected; programmers know they have skills to work their way out of these issues.

5.4 Future Work

A descriptive theory of students’ experience with programming assignments would be valuable to guide development of structures which address negative experiences and improve the process overall. However, building a valid theory of that magnitude will require a much larger effort. Through our use of reflective interviews, we found limitations on the kind of information we could obtain – students’ weren’t able to report even basic details on process. We encourage future work to look at a more robust view of the programming experience, but through the lens of grounded theory. We specifically recommend repeated sessions with a single student to increase descriptive data value.

Even in just understanding the emotional experience of programming, in this paper we have limited analysis to four of the seven categories found. Although these were chosen for the depth of their interesting results, we expect that, in future work, looking at all the stages together, we will find additional interesting connections and overall analysis which should yield further findings and recommendations for supporting students’ experiences with programming assignments.
6. CONCLUSIONS
In this paper we report a partial theory explicating students’ emotional experiences in doing introductory programming assignments. Developed from grounded theory analysis of a set of four longitudinal interviews with nine students, we see that students experience distinct emotional episodes relating to getting started with their assignment, encountering difficulties, and dealing with difficulties. Additionally, overarching emotional experiences were observed. Ubiquitously in these various stages of experience were issues of auxiliary emotional load (emotions not directly tied to the programming process). Common amongst these were emotional consequences from assessing one’s ability either compared to others or with regards to being able to succeed academically. We also find reported two forms of being lost – one where one can’t even understand where one is and the other where one doesn’t know what to do. Finally, we describe the true novice experience of encountering a difficulty as similar to being hit by lightning – an unexpected, shocking experience which leaves one dazed and confused.

7. ACKNOWLEDGMENTS
This work is supported by the National Science Foundation under NSF CNS-0933635. We thank the students who so generously talked to us and to Dov Zazkis for his help on this project.

8. REFERENCES