ABSTRACT
In computer science, students could benefit from exposure to critical programming concepts from multiple perspectives. Peer review is one method to allow students to experience authentic uses of the concepts in a non-programming manner. In this work, we examine the use of the peer review process in early, object-oriented, computer science courses as a way to develop the reviewers’ knowledge of object-oriented programming concepts, specifically Abstraction, Decomposition, and Encapsulation.

To study these ideas, we used peer review exercises in two CS2 classes at local universities over the course of a semester. Using three groups (one reviewing their peers, one reviewing the instructor, and one completing small design or coding exercises), we measured the students’ conceptual understanding throughout the semester with concept maps and the reviews they completed. We found that reviewing helped students learn Decomposition, especially those reviewing the instructor’s programs. Overall, peer reviews are a valuable method for teaching Decomposition to CS2 students and can be used as an alternative way to learn object-oriented programming concepts.

Categories and Subject Descriptors
K.3.2 [Computers and Education]: Computer and Information Science Education – Computer science education.

General Terms
Measurement, Design, Experimentation, Human Factors

Keywords
Peer Review, CS Education, Learning, Object-oriented Concepts

1. INTRODUCTION
Teaching computer science, much like in other disciplines, we continually expose students to concepts in the discipline, to prepare them for professional activities and to teach them how to learn. One common way to do this is to have students create programs.

Students early in the computer science curriculum often define a successful program as one that one that compiles and runs. So, while they will be concerned about style guidelines and the requirements laid out in the specifications, if the program produces the correct output (i.e. the instructor’s output), they consider it correct. These working programs, however, may be poorly designed and poorly implemented.

This creates a need for ways to expose students to critical programming concepts from another perspective, one that rewards students for understanding appropriate uses of the concepts and reasons for applying them. In other words, we want to use a non-programming activity to explore the fundamental aspects of programming. Peer review is one solution to the problem. It has many known benefits [14], and is used in many other fields [4] but there is no consensus on how beneficial the review process is in helping students learn object-oriented programming concepts in early computer science courses.

Specifically, we are interested in how reviewing can improve novice students’ learning of the abstract, high-level concepts of Abstraction, Decomposition, and Encapsulation. These three were chosen because they are central to object-oriented programming [1] and good design. They are also meta-concepts that are likely to be new to students and misunderstood by them [11]. Increasing the students’ understanding of these concepts early in the curriculum gives them a better foundation for the rest of their classes and for their future careers.

Based on earlier work [9], we are exploring the implementation in terms of the type of review. In this context, the review’s type is either a peer review where students interact with each other or a training review where students practice on teacher provided materials. The decision to use a training review or a true peer review affects the social interaction between the students and may have effects on the learning of the students [9]. A better understanding of these issues is needed so that peer review may be effectively used in the computer science classroom.

2. RELATED RESEARCH
2.1 Peer Review
In the computer science education literature, there are a number of studies detailing the benefits of using peer review activities. There have also been a number of studies involving the creation of software applications to assist the peer review. Studies found in the literature have commonly evaluated how closely student reviews matched other students or instructor reviews [4, 12], basic student motivation and attitudes towards peer review [12, 13] and systems for performing reviews [14].

What is missing, in general, is a focus on the reviewer and learning. Many studies approach peer review from the view of the
administrator and provide programs to address those concerns. Others look at the feedback and what those being reviewed have to gain. How the reviewer learns and otherwise benefits from the act of reviewing is not typically central to the literature. Studies into motivation and improvements in class work do relate to the reviewer but do not necessarily show that it improved learning. Exploring this unaddressed issue helps round out the literature and understanding of peer review.

To complicate matters, research and experiences in other fields may not translate into the computer science classroom. Theories of peer review [8] and some of the research in attitudes towards peer review [13] indicate that how the review is done and with what materials, can influence the outcomes. A better understanding of how the reviewer benefits from the review in the computer science classroom is needed.

2.1.1 Motivation
One reason for using peer review in the classroom is that it is an active learning exercise that offers a different type of learning activity. This has the benefit of breaking the monotony of a class in a typical lecture format. Our use of the reviews as another way to explore programming concepts provides a different perspective to the students and can help challenge their preconceived notions of computer science.

2.1.2 Peer Review Theories and Frameworks
One way to look at the process is from a cognitive viewpoint. Bloom’s taxonomy [2] can be used for this. Peer review supports higher-level learning skills such as synthesis, analysis, and evaluation. Piaget’s [7] and Vygotsky’s [3] work provide a (social) constructivist approach to peer review. Peer review is an active process that engages the students in learning and can provide new constructivist approach to peer revision. Peer review is an active evaluation. Piaget’s [7] and Vygotsky’s [3] work provide a (social) constructivist approach to peer revision. Peer review is an active evaluation.

Definition: Abstraction—Using classes to represent logical, cohesive, and coherent ideas or items. Methods have a logical fit with the class’s abstraction and provide common actions for that class. Implementation may not be known but can still be used.

Definition: Decomposition—Dividing the functionality, data, and responsibilities of a program into manageable portions. This includes creating classes to perform a specific task and grouping them with association and aggregation to form more complex structures. This also involves the use of inheritance to separate out and take advantage of commonalities between classes.

Definition: Encapsulation—Hiding information and restricting access to portions of a class. While this involves setting the correct scope for methods and variables, it also concerns what can be accessed or changed, and when those actions can occur.

2.2 Object-oriented Programming Concepts
One of the reasons for using peer review is to improve student understanding of the discipline’s concepts. While there are many aspects of computer science we could enumerate and explore, we narrowed down the possibilities by examining those that are used early in an objects-first curriculum. We decided to focus on three broad concepts that are core to object-oriented programming: Abstraction, Decomposition, and Encapsulation.

Because we are specifically looking at these three concepts, it is important to clearly define them as we are going to use them. Our definitions are loosely based on work by Armstrong [1].

Definition: Abstraction—Using classes to represent logical, cohesive, and coherent ideas or items. Methods have a logical fit with the class’s abstraction and provide common actions for that class. Implementation may not be known but can still be used.

Definition: Decomposition—Dividing the functionality, data, and responsibilities of a program into manageable portions. This includes creating classes to perform a specific task and grouping them with association and aggregation to form more complex structures. This also involves the use of inheritance to separate out and take advantage of commonalities between classes.

Definition: Encapsulation—Hiding information and restricting access to portions of a class. While this involves setting the correct scope for methods and variables, it also concerns what can be accessed or changed, and when those actions can occur.

These are interesting concepts both because they are central to the discipline, because they require high-level thinking and advanced mental development [7], and because they can be directly expressed in designs and code.

It is also worth noting that students use these concepts in their projects whether they understand them or not. Part of the important of the concepts is that they are essential in building robust, elegant, and extensible program. On the other hand, understanding them is not required to build working programs. Research in this area has found that students can perform well in a class (as measured by grades, program completion, etc.) while carrying significant misunderstandings about the topics [11]. It is important these concepts are learned early and well.

Abstraction, Decomposition, and Encapsulation are critical concepts for the students to understand. Understanding them will help students understand the entire object-oriented programming process better. So, it would be greatly beneficial to the students to learn these ideas early. With peer review, there is the opportunity to give students more exposure to the concepts in authentic use. To use the process well, more information is needed to understand how the learning of the concepts is related and how the review process can be used to aid that learning.

3. METHOD
3.1 Participants
Participants were computer science students enrolled in selected sections of a CS2 course at two local universities. There were 64 students at one university and 70 from the other that participated in the study. The students’ class level varied from freshman to senior (with just a few seniors). Most were Computer Science majors (75%) but there were also several students considering majoring in CS. There were a total of 13 females in the study.
3.2 Design
For each class, the reviews were designed to fit into and support the educational goals of the class. In this case, reviews were a mixture of code and design reviews. We had each reviewer evaluate two programs or designs for each activity using a rubric to guide them. The same rubric was used for all of the exercise. We also had students create and update a concept map to help us gauge student learning over the length of the study.

3.2.1 Concept maps
Before starting the study, the students were given a brief (10-15 minute) introduction to concept mapping. As an assignment, the students were then asked to draw a single concept map about their knowledge of Abstraction, Decomposition, and Encapsulation. They were also asked to update this map between the second and third study assignments and at the end of the semester.

Concept maps, as discussed in the literature, are a way to display an understanding of a body of knowledge graphically [5]. The creators of the maps select concepts that they think are part of a topic and connect those concepts according to their understanding of the relationships between them. This explicitly requires students to enumerate their knowledge and show how they think those ideas work together. This provides opportunities to find misconceptions.

There have been a number of studies that have looked at the changes in concept maps over time as an indication of changes in understanding over time [5]. Differences in what concepts were used and how they were linked show that the creator reevaluated the material in some way.

3.2.2 Groups
We used the same three groups in our experiment at both universities: a Peer review group, a Training review group, and a Control group. The peer review group reviewed the work of their classmates for each assignment. The training review group evaluated materials provided by the instructor. All of these resources were deliberately designed to have some strong and some weak points without being obviously a “good” or “bad” project. Both review groups were explicitly told who they were reviewing. The control group completed homework assignments, design or coding work, that were typical for the course. These exercises were designed to take about as long as the review tasks. Groups ranged in size from 15 to 26. For the complete experimental design details, see [10].

3.2.3 Study Activities
The sequence of activities in the study included a brief introduction to concept mapping and the creation of an initial concept map. This was followed by two review assignments. The original concept map was then updated and another two reviews were completed. Finally, the concept map was updated again near the end of the semester. Our work used a methodology similar to that found in Hay, Wells et al. [5]. The intervention was given as homework, spread over roughly ten weeks of classes. The timing of the assignments was worked into the routine of the class.

3.3 Data Analysis
Since concept maps can be used to display knowledge, this also makes them a good way to measure the students’ understanding of object-oriented concepts. As discussed earlier, we can measure change in that understanding by looking at the differences in what concepts were used and how they were linked.

To analyze the concept maps, we had to assign values to several aspects of the maps. These values were calculated for the portion of the maps addressing Abstraction, Decomposition, and Encapsulation. As one value, we used the Novak and Gowin score [6], which rewards connections between concepts. As our second value, we counted the concepts in each group. This rewards the use of concepts in the maps rather than the connections. Our third value, the standard concepts count, recorded the presence of those concepts that should be in all maps. These are terms or ideas that come from our definitions of Abstraction, Decomposition, and Encapsulation, from basic object-oriented theory (e.g. class), and from the literature [1]. Students who did not complete one of the concept maps were excluded from the analysis.

4. RESULTS
Once we had coded and measured the Novak score, the count of concepts, and the count of standard concepts for each concept, the values were analyzed using a MANOVA with repeated measures over time. The school and group the students belonged to were used as factors in the analysis. To look at effects at each university, we also applied a MANOVA with repeated measures over time to each school’s data. We used the scores from those students that completed all three maps. After the initial analysis, a post hoc analysis was completed on the significant factors. For most of our measures of the concept maps, there were significant changes over time and between schools.

4.1 Abstraction
Looking at the combined data for the concept of Abstraction, the mid- and post-map values for the Novak score and the number of concepts were significantly different from the pre-test (p<0.01, p<0.05 respectively). There were no differences found between the mid- and post-maps. We did not find any differences between the groups. There were significant differences between schools for the number of valid concepts (p<0.05) and the Novak score (p<0.05). The maps at Univ-2 were generally larger and with more connections but did not contain significantly more standard concepts. With the data separated by school, we saw the same basic patterns at both universities but, at Univ-1, they were not as large and not significant.

4.2 Decomposition
Decomposition provided us with a much different picture. In the combined data, school was a significant factor for all of the measures (p<0.05). As we have seen before, Univ-2 had higher scores than Univ-1. Viewing the schools separately, we found evidence of improvement over time at both universities.

More interesting than the differences between the schools are the differences between the groups over time. We looked at the individual groups over time and compared the groups at each point in time. The control group did not significantly change over time but the other groups did. For the Peer group, the mid-map Novak value was higher than the pre-map’s (p<0.05). The post-map’s Novak value is slightly lower than the mid-map’s and is not significantly greater than the pre-map’s. The changes in the Training group were more dramatic. Each score is greater than the last (p<0.001). As can be seen from Figure 1, the Training group is definitely diverging from the others. This is a good indication that the Training group understood the concepts faster than the other
two. It should be noted that the Peer group also appears to be understanding the concepts better than the Control group, but it is fairly slight.

Figure 1: Decomposition Novak Score by Group (All)

We did not see this difference in groups when analyzing the data from each university. We only saw effects over time. However, looking at the data, we see a very similar effect. For Univ-2, both the Peer and Training groups have similar growth. It seems likely that both groups are experiencing benefits from the reviews but it also appears that the Training group is benefiting somewhat more.

We also compared the values for each map across groups. There were no significant differences at any of the three times. This seemed somewhat at odds with the changes found within the groups. After examining the data, it became apparent that there was a high level of variance within each group. So, while there are definitely changes in two of the groups, it is not visible when looking at the data only in terms of groups. We found similar interactions and results with the standard concepts value.

4.3 Encapsulation

When we looked at Encapsulation, we saw results over time similar to those with Abstraction but slightly more pronounced. Time was a significant factor for all the measures. Aside from the valid concepts, the mid- and post-map values were both significantly greater than the pre-map values (p<0.005) but were not different from each other. With the valid concepts score, there was also a significant difference between the mid- and post-maps (p<0.01). We did not find significant differences between schools except with the standard concepts (p<0.05). Univ-2’s scores are still higher than Univ-1’s overall, but with a smaller gap than seen earlier.

4.4 Comparison between Concepts

In the previous section, we looked at how the concepts changed over time. In this section, we compared the three concepts. This was done to contrast the students’ relative understanding of Abstraction, Decomposition, and Encapsulation. Previous work [9] has found differences in how students reviewed these concepts. That was an indication that the students may understand them at different levels or be learning them at different rates. Here, we are exploring that idea further to better define what those differences are. This will help us determine if peer review is equally useful for all the concepts or if it is better for some than others. It also gives us additional insight into the students’ understanding of the object-oriented concepts.

To do this, we compared the Abstraction, Decomposition, and Encapsulation Novak scores across each test. We used a MANOVA with repeated measures by concept for the pre-, mid-, and post-maps. As before, the school and group the students belonged to were used as factors in the analysis. The analyses were run on the entire data set and on the data from the individual schools. After the initial analysis, a post hoc analysis was completed on the significant factors.

First, we examined the Novak values. For the pre-map, we found interactions between the concept and the school (p<0.01). At Univ-1, there were differences between Encapsulation and Decomposition (p<0.001). From the results for each school, it appears that Univ-1 came into the class with a weaker understanding of Decomposition and Univ-2 entered with a more balanced knowledge of the concepts.

Examining the mid- and post-maps, we just saw the differences by school and by concept. All the scores from the universities were significantly different (p<0.05) with Univ-2 having the higher values. The more interesting finding was with the concepts. The Novak score for Encapsulation was significantly higher than either the Abstraction or Decomposition scores (p<0.05). Viewing the data by school, these differences are reflected at Univ-1. They were not significant at Univ-2, however. The data shows the same general trend as found at Univ-1, but it is much more modest. It appears that students find Encapsulation easier to understand and explain than the other two concepts. The change in the concepts’ Novak scores over time supports this claim.

5. CONCLUSIONS

From its uses in other situations and in other disciplines [14], we had expected to see bigger gains from those reviewing (peer or training reviews) as they were exposed to more and more varied examples of the concepts in actual use. When looking at the concept maps, we, again, found no sign of an effect from the different groups except with Decomposition.

With Decomposition, however, we found effects between the groups. Both of the reviewing groups showed signs of improvement over those that did not review and the students who were reviewing the instructor-provided material proved to have the most growth. Although the nature of the growth was somewhat different at each school, the effects are plain. Reviewing program designs and code can help students learn and understand the concept of Decomposition at this level. For the other concepts we were measuring, Abstraction and Encapsulation, we did not find any noticeable differences (significant or otherwise) that would lead us to believe that the reviews affected them in the same, perhaps smaller, way. With those two concepts, the students that reviewed fared about as well as those who were doing additional design or coding exercises.

Exploring the differences between the students’ understanding and learning of these concepts helped us shed some light on our findings. Looking at the concept maps and the rubric data, we started to see specific distinctions between the concepts. Based on its changes over time and on its comparison to the other concepts, students seemed to have the best understanding of Encapsulation and seemed to be learning it well. Their understanding grew over time and appeared to be stronger than that of the other two concepts. On the other hand, the students’ knowledge of Abstraction seemed to be progressing the least. The data suggests that the students do
not have a strong grasp on what it means to create classes that form a clear representation of a concept or real world object. When it comes to Decomposition, there were signs that students were actively trying to learn it.

Among other things, this suggests a partial ordering to the concepts. Teaching Encapsulation early and then devoting more time to Decomposition might be beneficial to the students as they seemed to have picked up Encapsulation quickly. The ideas behind Abstraction may need to be introduced early, but may not be understood until later classes. From the data, it is not clear how Abstraction relates to the other concepts. This ordering provides us with more information about to how focus our efforts.

To understand why this review process is effective for only one of the three concepts, we considered the nature of the ideas they portray. Abstraction and Encapsulation can both be applied very locally, for instance, to a single class [1]. Either the class represents a coherent real world object or ideas or it does not. The same is true of Encapsulation. Visibility is specific to a class as are the ways in which the class’s data is accessed and modified. Both of these concepts are also relatively easily mapped to language constructs. Abstraction can be thought of as the collection of variables and methods in a class. Basic Encapsulation can be achieved through a small number of key words. Students can (and should) understand each of these two concepts on a broader scale so that they can see how simple, well defined classes simplify the program and improve reuse as they work together and how the correct protection of a class’s inner workings reduces errors and unneeded dependences with other classes. But, as they are learning these concepts, they do not have to look at those wider perspectives.

Decomposition is different. By its very nature, it involves a bigger picture. Student must understand how classes can work together to build something more than the individual parts and they must see how problems can be divided into smaller and smaller pieces until they are solvable. It cannot be done at the class level and it does not map as nicely to language constructs. Yes, there are plenty of features that make Decomposition easier to achieve but there is nothing that helps a novice designer decide how to divide the functionality. The same computations could be done as part of a method, as its own method, as part of another class, in a completely separate class, and so forth. There are multiple possible approaches that are supported by the language that may or may not be appropriate for any given situation. Decomposition requires the student to identify the pieces of a problem and to consider how they are connected.

So, why are reviews helpful for this concept? Reviews provide students with someone else’s interpretation of the big picture for a problem. They can get experience working at this higher level by analyzing and comparing the design choices with their own [3]. As they try to understand why a solution is the way it is, they can start to develop their knowledge of good and bad design choices.

This proposed explanation also helps us understand why Training reviews (instructor-provided material) appear to be more effective than Peer reviews. The instructor’s designs and code can provide a richer experience and a wider selection of design decisions than are likely to be found in a student’s work at this level in the curriculum. It can give students practice with ideas and techniques they have not been taught nor have thought of yet. The difference between the student’s level of understanding and the instructor’s (even in work with poor decisions deliberately inserted, as we did in the study) offers an opportunity for learning.

In closing, we found that the use of reviews can be a valuable tool for helping students learn OO Decomposition.

6. REFERENCES