How Interactive Multimedia Authoring Transforms Object-Oriented Thinking

Timothy T. Yuen  
The University of Texas at San Antonio  
One UTSA Circle  
San Antonio, TX 78249  
timothy.yuen@utsa.edu

Min Liu  
The University of Texas at Austin  
1 University Station D5700  
Austin, TX 78712  
mliu@mail.utexas.edu

ABSTRACT
This paper discusses a cognitive study of how interactive multimedia authoring (IMA) affects novice computer science students’ conceptual understanding of object-oriented programming (OOP). In this study, interactive multimedia authoring refers to the construction of a role-playing game using a game template developed with Adobe Flash CS3 and ActionScript 2.0. Three cognitive processes of disequilibrium, exploration, and awareness were observed in this study, which aided the transformation of students’ understanding of OOP through interactive multimedia authoring.

Categories and Subject Descriptors
K.3.2 [Computers and Education]: Computer and Information Science Education – computer science education, literacy, self-assessment

General Terms
Human Factors

Keywords
Computer science education, object-oriented programming, Adobe Flash, multimedia authoring, games

1. INTRODUCTION
Object-oriented programming is one of the difficult concepts for students to understand in computer science [5, 6, 11]. OOP requires students to design solutions in an OO-way; that is, decomposing a problem into maintainable and interactive objects. Visual learning tools, such as Alice, Scratch, and LEGO, have long been used to present OO concepts to students in manageable, familiar, and entertaining contexts [3, 12, 15]. The visual elements provide concrete and real-life scenarios for the OO concepts that complement the textual nature of source code and abstractness of ideas. This study extended the concept of visualization by using interactive multimedia authoring as a way to facilitate and transform students’ conceptual understanding of OOP as in the case of cognitive tools [20]. The rationale was that the multimedia components would provide the metaphors in which students can see visual representations of their code, the authoring component provides students the opportunity to exercise their understanding, and the interaction with the game which allows students to experiment with their own implementations.

2. LEARNING THROUGH INTERACTIVE MULTIMEDIA AUTHORING
Constructivist learning regards learning as an active process of knowledge construction [1]. Thus, higher order thinking and problem solving are skills that are built up over time as learners gain more experiences. Learners’ mental models will be refined through multiple revisions and the adaption of new information [21]. Cognitive tools are designed to instigate change and reorganization of these models as a way to guide students towards a more expert way of thinking [20].

Past research in CS education research looked at visualization tools to aid in this knowledge construction. Such tools generally show how each line of code is executed, the flow of logic in the source code, or visualized behaviors of code [4, 10]. The purpose of interactive multimedia authoring in computing is to add different levels of representation through interactive and visual channels to scaffold OOP understanding and learning. Another goal is to provide students with a motivating environment. That is, immersing students in a multimedia environment may play on students’ personal experiences and interests in multimedia can be motivating [8]. The specific project of creating a game within an interactive multimedia authoring integrated development environment (IDE) can also play on the continual interests and experiences that younger people have with video games [13].

IMA follows the notion of constructionism in which knowledge construction works alongside the construction of an artifact [19]. In computer science, constructionism is a natural descriptor as students usually write programs as a way to exercise their understanding of a given topic such as a data structure or algorithm. IMA differs from visualization tools in that students are developing multimedia-rich and interactive application themselves. Multimedia is not just an after-thought but also something that students must be involved in creating so as to lead to the construction of better conceptual understanding [2, 18]. The multimedia output and interaction with the output is a direct effect of students’ coding and vice versa.

2.1 CSNüb
CSNüb is a template created in Adobe Flash that students use as a basis for creating a role-playing game (RPG). The default game is an underwater RPG named Operation SPLASH and features a
submarine moving around the ocean floor and encountering various life forms and objects. Students are provided base code, library of graphics, and basic documentation on the overall architecture of the game template. Students use, add, and customize the base code and graphics to create their own games.

3. METHODOLOGY
Since the concern centers on conceptual understandings and how they are formed and changed, a qualitative approach was taken to answer the following research question: How does creating a video game within an interactive multimedia authoring tool affect novices’ conceptual understanding of OOP?

3.1 Sample and Participants
Twelve participants (n=12) were recruited from a CS1 and CS2 class at a large university in central Texas in 2007. At this particular university, students may begin with either the CS1 or CS2 course depending on their prior computing experience. In both cases, OOP was not taught until the last third or last half of the semester, respectively. All participants were self-selected. They were all undergraduate males, ranging from ages 18 to 23 with a mean age of 19.25. With the exception of one student who was a Computer Engineering major, all participants were CS majors or had intended to major in CS. Two CS majors were also Math majors.

When asked to rate their programming knowledge on a scale of 0 (needs improvement) to 3 (excellent), participants reported an average score of 1.583. On the same scale, participants reported an average score of 1.667 when asked to rate their knowledge of OOP. Over half of the students reported a score of 1. Only two students reported experience with Adobe Flash.

3.2 Methods
Clinical interviews were used to solicit the participants’ cognitive processes and conceptual understanding of OOP. Clinical interviews are generally task-based environments in which the researcher consistently probes students’ thinking through questioning and perhaps altering the parameters of the tasks depending on each student’s level of development [7]. Process tracing methods were used to structure the researcher interaction with the participants. Process tracing methods involve the observation of behaviors, think-aloud protocols, and retrospective reports [9]. Thus, all behaviors committed by participants, such as compiling the game, opening files, and looking at sample code were recorded into a behavioral log.

3.3 Tasks
Participants were given three tasks to complete during the clinical interviews. Tasks involved implementing a specific feature of the game as well as a concept of OOP. Task 1 asked participants to give hit points (HP), attack points (AP), and defense points (DP) to the submarine and the squid characters. These are standard properties associated with characters in RPGs. It required participants to use variables inherited from a parent class. Tasks 2 and 3 asked participants to implement the interaction between the submarine and an energy barrel and a rock, respectively. In each of these tasks, participants had to write new classes, which should extend from existing classes in the CSNüb template as well as use inherited properties and methods.

These tasks required participants to exercise their understanding of OO systems, class design, inheritance, encapsulation, and object interaction. There was no time limit.

4. ANALYSIS
A microanalysis of the data was conducted through an open coding procedure [17, 22]. Each fragment of the behavioral logs was coded to describe the action and/or theme. Additional data also included field notes, transcripts of the participants’ interviews during the tasks, the source code, and the game for the purposes of triangulation [14].

4.1 Cognitive Processes and Factors
Open coding of the data yielded three categories related to how the participants knew and applied their knowledge of OOP. These categories are the cognitive processes of disequilibrium, exploration, and awareness.

4.1.1 Disequilibrium
Disequilibrium is a state of cognitive conflict in which a student’s mental model cannot adapt new information received [21]. Disequilibrium occurred when the visual and textual feedback from testing their games was inconsistent with participants’ expectations. Though compile time errors also provided conflicts, it is important to emphasize that participants gained most from not just observing the visual feedback, but interacting with their games.

There were two examples of problems that were common among the participants: the never-ending game problem and the _rotation problem, both of which placed them in a state of disequilibrium. Each problem can be tied to a discrepancy in the participants’ understanding of OOP.

Redundant code was a label given to the ongoing trend in which participants would declare variables or methods that were already inherited from a parent class. Task 1 asked participants to assign specific values for hit points, attack points, and defense points to the submarine and squid characters. Seven participants went to those respective classes and declared new variables and/or methods (e.g., private var HP:Number). When viewing the Submarine and Squid classes, there is no mention of hit points, attack points, or defense points. However, both classes descend from the Character class, which is where they are defined. This represents the problems of narrow perspective or awareness of the entire OO architecture.

Though no compile time error occurred from redundant code, problems did arise during run-time. The most prominent problem that stemmed from redundant code was when the game would not end. In the background, there is a listener that continues to check on the hit points of the submarine (the main character in the game). Should the hit points ever reach the value of zero, the game will end with a lose state, which can be seen on the Game Over screen. Similarly, the listener also checks to see how many enemy life forms (e.g., a squid) and items (e.g., an energy barrel) are remaining. When all of those objects are gone and the submarine has 1 or more hit points remaining, then the game will end with a win state. The feedback from the Game Over scene or lack thereof presented the participants with the conflict.

The _rotation problem stems from Task 2, which asked the participants to program the interaction between the submarine (main character) and a rock. The submarine should hit the rock and turn around 180°. Participants were aware that there was a _rotation property in the Submarine class. So, they simply set the value to 180 (e.g., this._rotation = 180). _rotation is a top-level property of Flash’s MovieClip class, from which all of CSNüb’s
classes. However, participants were unaware that Flash’s implementation of an object’s _rotation is a fixed value. That is, a value of 180 will turn the object facing down no matter its current orientation. Eight participants experienced the _rotation problem since they were unaware of its behavior.

Both the issues of the never-ending game and _rotation were discovered when playing the games. When in a state of disequilibrium, a learner tries to resolve the situation by making sense of the information. Piaget refers to the processes of assimilation and accommodation; that is, changing the incoming information to conform to one’s mental model or changing one’s mental model to accept the new information [21]. This paper offers a different take on Piaget’s ideas by presenting the cognitive processes of exploration and awareness as methods of adaptation.

4.1.2 Exploration
Exploration was largely the initial method by which participants tried to resolve their moments of disequilibrium during the _rotation and never-ending game problems. Exploration was first noted when participants would start opening other files to see what could help them resolve the problems. The interesting part was participants would do this in a methodical manner. That is, they would open other files that were in relationship to the current file they are working with. For example, if participants were working with the Submarine class and ran into a problem, they saw that it extended the Character class. When they could not find their solution in Character, they would see that it extended the Object class and opened that file. One participant referred to this as “going up the chain.” Participants also traced out the interactive relationship between classes. In Task 2, one participant noted that the DisplayPanel object somehow received information from Submarine object and looked in the DisplayPanel to see how it was getting the hit points, attack points, and defense points. He noted, “At this point, I’m trying to look what’s in the display panel. Because at this point I don’t see a display for health. And possibly it would tell me where the health values are set.”

Table 1 is a tally of the number of times a participant looked up another file with the explicit purposes of exploring OO relationships between classes. These counts do not include random exploration of files.

<table>
<thead>
<tr>
<th>Task</th>
<th>Min # of Lookups</th>
<th>Max # of Lookups</th>
<th>Mean # of Lookups</th>
<th>Std. Dev.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>5</td>
<td>2.417</td>
<td>2.065</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>20</td>
<td>7.833</td>
<td>5.997</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>17</td>
<td>7.083</td>
<td>4.999</td>
</tr>
</tbody>
</table>

An increase in Task 2 and Task 3 can be explained by the fact that they required participants to design two new classes (Rock and EnergyBarrel) that must extend from one of the base class (Object) for the purposes of keeping the OO nature of CSNübü. More importantly, the game engine worked under the assumption that all game objects are descended from the top-level Object in CSNübü. Task 2 was also the first time that two objects would interact with each other.

Another explanation for the increase in Task 2 over Task 1 is that this is the first task in which participants were asked to implement an interaction between two objects: the submarine and a rock. The role of the game engine is to detect the collision of the two objects and delegate the reaction effects to the submarine. One participant remarked: “I assume there’s going to be multiple obstacles in this game and I don’t know exactly [what] an obstacle should do yet. So, if I look at the comments and the methods provided in the parent class, I’ll know what a rock is supposed to do.” This reaction showed the participant trying to understand the behavior of the two classes involved and how to possibly understand the interaction between them.

In Task 1, one participant said, “If I’m going to set the following values, I’m going to find the class of it” while another said “I gotta go find the information for the submarine and both of the squids.” These two participants’ statements illustrated their initial understanding that the Submarine and Squid classes were actually part of a larger, interconnected system. It should be noted that not all participants showed this initial understanding, which could also explain why there were fewer look-ups in Task 1 compared to the other three tasks. Thus, exploration is a result of a participant’s awareness of the OO system.

4.1.3 Awareness
Awareness is the level at which participants understood the OO architecture of the CSNübü template. The previous discussion exploration showed that participants explored files in a methodical manner with respect to tracing out the hierarchical and interactive relationships between classes. How a participant explored was a reflection of his level of awareness of the OO system. Awareness was best demonstrated in Task 1 when participants wrote redundant code. As evident from the redundant code issue, many of the participants experienced narrowed vision of each class. That is, they did not consider that the Submarine class could have properties and behaviors that are inherited from other classes.

For example, one participant in Task 1 said the variables to represent the HP, AP, and DP “should be at the very top” of the Submarine class. He did not see this nor did he notice that Submarine extends Character, which resulted in the declaration of redundant variables. Another participant noted the variables “should be in the constructor, but there are no instance variables. Do I create those variables?” That is, he was used to seeing variables initialized within the constructor, but participants were provided with an empty constructor. Another participant in the same mindset said, “I’ll probably have to define them myself...and then initialize the constructor.” He also contemplated writing setter and getter functions for them as well within the Submarine class. However, its parent class Character already had the variables and the methods. High awareness was demonstrated by the participants that attempted to gain the bigger picture through exploration before writing code. For example, one participant wanted to know how the two objects worked together before any implementation: “If I knew how it was going to use the HP object...how the game is actually going to determine how HP is implemented, it may be useful.”

Five participants began Task 1 having a higher-level understanding of OOP. When they were unable to find the variables for the hit points, attack points, and defense points in the Submarine class, they knew to explore the parent class. Here is yet another example of how awareness affects exploration. It should be noted that two of these participants were unsuccessful in completing Task 1: they found the variables in the parent class,
but copied and pasted the variable declarations into the Submarine class which resulted in redundant code.

There was a transition between a narrow awareness to a larger sense of the OO system while participants worked through the tasks in this study. Of the seven participants that showed signs of low awareness, only three were able to resolve the redundant code problem through exploration of files when they encountered the never-ending game problem in subsequent tasks. It gave some participants cause to explore other files.

After an unsuccessful try at Task 2 with redundant code, a participant finally noticed that Submarine extended Character from the class header. He began to laugh when he saw that the points variables he declared in Submarine and Squid were already declared in their parent class Character, “It seems just what I did. [laughs] I could have been using that for a more efficient solution.” He corrected his code to take advantage of the inherited variables after this discovery. Awareness of how objects are related or interact together was changed as participants explored and explicitly traced out such relationships in the code. As in the redundant code issue, it can take cognitive disequilibrium to bring attention to students’ inadequacies with regard to awareness.

5. DISCUSSION

Regular programming (e.g., in Java or C++) generally results in numerous occasions in which students experience disequilibrium: bugs in the code, run-time errors, etc. Though these occur in IMA as well, the visual components of the game provide students with multiple representations of the problem: verbal cues from the debugger and source code and the visual components of the game. We want to continually engage the students in moments of disequilibrium—not to further frustrate—but to motivate metacognition and change in students’ mental models. Multimedia learning theories contend that feedback is good in that multiple sensory channels are engaged when providing a different representation of the idea [16].

CSNüüb is described as a game template and not a game engine. This purposed description is to draw attention to its simplicity whereby students who enter states of disequilibrium can easily sift through the source code and have a general overview of the OO architecture. The data showed that exploration of the source code is the method by which students tried to understand the OO design of the template, especially when they were in the state of disequilibrium.

Such exploration leads to a greater awareness of the OO system and how objects were related and worked together to create the final game. As students designed more complex objects, they saw the results through playing the game. For the students that entered the activity with a very novice-level or narrow view of programming, they started from a bottom-up approach. The game template put the student in a situation where there was enough support for them to understand the system from a top-down approach.

Interactive multimedia authoring provides cognitive scaffolding on multiple levels. First, the existing source code gives students models. Second, the feedback from the game gives a visual and interactive element in which they can see their code in action. As students construct their better games (e.g., more effective OO design, adding game features, etc.), this should represent a more complex understanding of the OO nature of the game template. This supports the constructionist perspective of learning in that the artifact one creates is a representation of the mental model [19].

6. CONCLUSIONS

The three cognitive processes are not separate entities. In fact, they work together in assisting students with achieving a higher level of thinking and problem solving (See Figure 1). Exploration helps resolve disequilibrium by giving students more information to understand the problem space. Conversely, disequilibrium triggers the exploration process, which was done in a methodical manner according to the OO relationship. The scope of awareness with respect to the OO design of the game template was a major factor in how students explored the classes and other resources. Low awareness resulted in random exploration of files and classes. High awareness resulted in the methodical exploration of classes. When students explored, they found various forms of cognitive scaffolding: through code modeling and multimedia-based feedback from playing the game.

![Figure 1. Relationship between disequilibrium, exploration, and awareness during interactive multimedia authoring](image)

The model above illustrates how interactive multimedia authoring can be used to challenge students’ thinking of OOP. With further study using a wider range of students, in terms of gender and aptitude at the CS0/CS1/CS2 levels, we can construct a more refined model of how interactive multimedia authoring affects conceptual understanding of OOP or any other topic requiring complex or abstract thinking. It shows a process in which we, as instructional designers, can engage students and positively affect their conceptual understanding, higher-order thinking, and problem solving skills.

7. REFERENCES


