A Quantitative Study of GUI versus Text-based Object-Oriented Instruction

Per Andersen, Susan Mengel
Texas Tech University, per.andersen@ttu.edu, susan.mengel@ttu.edu

Abstract - Introductory object-oriented programming is considered difficult to teach and a number of different methodologies have been proposed to address this difficulty. One method, which is reported in this paper, involves the use of graphical user interfaces (GUI’s). The research question developed for this study is as follows: Does any statistical difference exist in test scores between a control group using text-based object-oriented programming (OOP) and a test group using GUI-based OOP? To answer this research question, quantitative data was collected through the use of common questions on the final CS2 exam for both groups. Data was collected over a period of 3 class offerings for five multiple-choice questions and one question requiring programming. No statistical difference between the groups was found for either the 5 multiple-choice questions or the open programming question. An analysis of the final CS2 course letter grade showed no significant difference between the test and control groups. Further, students who participated in the first year of the study were tracked through Data Structures, the course which follows CS2 in the curriculum sequence. A statistical analysis of the final letter grade for found no significant statistical difference between control and test groups.

Index Terms – Quantitative Study, Object-Oriented Programming

INTRODUCTION

Introductory object-oriented programming is considered difficult to teach as students new to programming are faced with the task of absorbing and processing new skills learned in CS1 that are typically imperative. Then, in CS2, students face the potentially difficult transition to a second programming paradigm, object-oriented programming. To avoid this problem of transition, the object-first strategy [1] has been proposed. Unfortunately in some approaches [2], educators have observed that students do not develop a sense of syntax and how to deal with error statements associated with syntax. In their discussion of the object-first strategy, Cooper et al [2] also point out that teaching objects-first requires students to learn many different concepts, ideas, and skills concurrently.

At Texas Tech University, the imperatives-first approach for CS1 is used and then objects are introduced in CS2. The experience with this approach reveals that students find it no less difficult to learn objects. A number of techniques have been proposed and tested to improve learning outcomes in object-oriented programming courses, including the use of GUI’s [1], sound, and games [3].

In the spring semesters of 2005, 2006, and 2007, multiple sections of CS2 based on object-oriented programming in C++ were offered to students. The possibility existed, therefore, to have one section of students taught using a text-based approach and another section using a GUI-based approach. For this study, the control groups consisted of students taught using a text-based, console input/output approach. The test groups consisted of students taught using the Microsoft Visual Studio .NET C++ GUI library and learned object-oriented programming in that context [4]. In 2005, 28 students participated in the control group and 18 in 2007. Due to scheduling issues, no 2006 control group was present. The test group consisted of 24 students in 2005, 25 in 2006, and 29 in 2007. The 2005 control and test group students were further tracked through a course in Data Structures in 2006. Given that today’s interfaces consume 48% of the code and 50% of the implementation time in computer application development for new applications [5], the incentive is obvious for teaching GUI development. However, is it effective at the freshman level, and is the effectiveness measurable in student performance?

The research question, therefore, can be stated as follows.

“Does the use of advanced programming tools, such as the Microsoft C++ Graphical User Interface (GUI) Development Libraries, in the teaching of introductory C++ Object-Oriented Programming, effect the students in a measurable way that is different than the students taught with the text-based, console approach used at Texas Tech University?”

The rest of the paper is presented as follows: related work, CS2 learning outcomes and objectives at Texas Tech University, a presentation of the test instrument, results, and conclusion.

RELATED WORK

The ACM Computing Curriculum 2001 lists four approaches to teaching introductory computer science: Programming-First, Breadth-First, Hardware-First, and Algorithms-First. In this study, the focus is on the programming first model,
which is traditionally the most widely implemented model for teaching introductory computer science courses [6]. Within this model, three approaches are found to teaching programming first. Approach one is the imperative first approach, which typically begins with programming fundamentals based on a procedural coding model followed by a second course introducing object-oriented programming. This model is followed in the Computer Science Department at Texas Tech University. Approach two is the object-first model which starts students with object-oriented programming from the beginning. Approach three is the functional programming model which teaches students programming utilizing languages such as Scheme and Haskell. The related work described here is for the imperative-first and object-first approaches.

For many years programming languages have been taught using the basic input/output features of the languages, usually text only. Many programming language textbooks begin by introducing a language with a “Hello World” example outputted to a console interface. A realization occurred that this approach is less adequate in holding the attention of students and motivating them. In his paper titled “Why Isn’t My Professor Using Graphics in the Freshman Programming Course” [7], Park commented that traditional teaching approaches “are often so bland as to induce narcolepsy in many students”. Students as a result perform poorly and lose interest in computer science. Wicentowski and Newhall [8] also support Park’s position. They state “that projects that are fun and challenging, that seem real” motivate students to spend the time necessary to learn programming skills.

Many computer science departments have utilized various programming tools in an attempt to address the needs of today’s computer science students. Today’s student is far different in terms of their exposure to technology than a student of 30 years ago. Students today are exposed to computing technologies from a very early age. Thirty years ago, students would typically encounter their first programmable computing situation as university freshmen. With the advent of technologies such as the Internet and high performance gaming systems, today’s student has inclusion of graphics and animation into introductory programming courses. Examples of these programming tools include Alice [2], BlueJ [17], and JAVA Power Tools [1]. All of these tools support a graphical view of objects. Alice utilizes a drag and drop integrated development environment that allows students to build 3D animation programs. Students can write programs utilizing Alice’s built-in objects or write their own objects. The results of a study of CS1 students show that CS students at high risk for dropping out who took the course based on Alice had a mean GPA of 2.8 while those that did not had a mean GPA of 1.3. The study did find that one of the weaknesses of this approach is that students did not develop a sense of syntax since statements and expressions were dragged and dropped into a coding window.

The motivating factors driving the development of these programming courses are improvements in retention and student performance. Very little actual quantitative data has been found in the literature on the benefits of this approach for C++, the language Texas Tech Computer Science majors are taught. Qualitative data collection is the typical approach used in assessing these courses. An exception is the paper by Moskal et al [18] where the authors provide some quantitative data and report GPA improvements in students that are taught programming with Alice, as well as improvements in retention. The qualitative results, based on student feedback, as reported by Franceschi [16] on the use of animation in teaching CS1 are more typical of the assessment of these approaches.

One of problems of concern in utilizing GUI-based tools for teaching objects is a weakening of basic programming skills. This problem was reported by Leska and Rabung [3]. They found that in their development of a gaming approach to student projects for CS1, some topics normally taught in CS1, such as file I/O and exception handling, were not covered, but left for CS2.

Soloway et al, in their article “Teaching the Nintendo Generation Programming” [9], described the traditional approach to teaching programming as out-dated. They go on to state that this out-dated approach becomes a contributing factor to the large dropout and failure rates (15-30%) in CS1 courses. They created a pilot programming class developed around Squeak, a Smalltalk based multimedia-programming language. Their use of Squeak was an attempt to engage students with a learning environment that better reflects the kind of things that got them excited about Computer Science in the first place. Alphonce and Ventura [10] share the view of Soloway et al and are teaching object-oriented programming with JAVA, through the use of a JAVA graphics library package called NGP. NGP is a small, but elegant graphics library for Java. They point out that “Teaching object-oriented programming is hard. Keeping the attention of CS1 students is perhaps even harder.” They felt that using NGP with Java would help with teaching and capturing the attention of the students.

The object-first approach has to some extent driven the development of programming tools in support of the inclusion of graphics and animation into introductory programming courses. Examples of these programming tools include Alice [2], BlueJ [17], and JAVA Power Tools [1]. One of problems of concern in utilizing GUI-based tools for teaching objects is a weakening of basic programming skills. This problem was reported by Leska and Rabung [3]. They found that in their development of a gaming approach to student projects for CS1, some topics normally taught in CS1, such as file I/O and exception handling, were not covered, but left for CS2.
The literature is generally very positive about the use of tools such as graphics, animation, and GUIs for teaching objects. If programming courses can be taught with GUI tools without losing content, the capacity to hold the interest of students can be improved. The result should be an increased level of understanding of object-oriented programming. The quality of students and their success in later courses should improve and retention levels should increase, which would benefit the students and the institute.

If the inclusion, in introductory courses, of GUI development tools has some measurable benefit in teaching C++ object-oriented programming, then this study could provide important insights. For example, this study could show instructors how to proceed with this approach and guide them in the content they should include in future sections of CS2 and perhaps other courses as well.

**OVERVIEW OF CS2 LEARNING OUTCOMES AND OBJECTIVES**

A summary of learning outcomes for CS2 at Texas Tech University is given in the following list.

- Analyze a problem and develop an algorithmic solution
- Compare and evaluate alternative designs
- Master C++ class and template syntax and semantics
- Trace the execution of recursive functions
- Design recursive algorithms
- Apply the fundamentals of C++ class design
- Implement a hierarchical set of classes
- Use dynamically allocated objects
- Code and apply simple data structures: lists, stacks, queues
- Apply the components of object oriented design: encapsulation, inheritance and polymorphism
- Use the Standard Template Library containers and algorithms

In spring 2005 a common textbook was used for the control and test groups, but the teaching plans between the two instructors were not closely coordinated. In spring 2007, the control and test group instructors coordinated their teaching schedule and the material covered out of a common textbook [21]. Students in the control group were allowed to use any of the compilers and IDEs available in the teaching labs which included Microsoft Visual Studio .NET, Bloodshed C++, and GNU gcc. Before students in the test group could begin to develop GUI-based code, they needed to become familiar with the resources provided by Microsoft associated with developing GUI code in Visual Studio .NET. Students were guided in using these resources, such as MSDN and msdn.microsoft.com, until they could find answers to their questions on using the interface tool on their own.

As the development environments were available for download, students could program on the lab computers or on their own computers. They worked independently on their assignments.

The CS2 course covered the following material:

- Classes, object instantiation, member/accessor/utility functions, public/protected/private declarations, templates, scope
- Separating implementation from the interface
- C++ Compilers and how they work
- Libraries
- Standard headers
- Storage classes, auto (default) register, static, extern
- Call stack / activation records
- Function overloading and templates
- Recursion
- Arrays and vectors, multidimensional arrays
- Constructors, copy constructor, destructors
- Default arguments, member-wise assignment
- Composition - Has-a, Inheritance - Is-a, Friends
- This pointer
- Static class members
- Overloading operators, stream insertion and extraction operators, binary and unary operators
- Polymorphism, virtual/pure virtual functions
- Abstract and concrete classes
- Linked Lists
- File I/O

In addition, the students in the GUI group were taught managed classes in C++ .NET.

- Forms
- Controls (button, textbox)
- Events

**TEST INSTRUMENT DESIGN**

The study involved two types of assessment, objective assessment and performance assessment [19]. The objective assessment was administered through a series of common multiple choice questions on the final exam for control and test groups. The performance assessment is administered through a single common programming question on the final exam.

The common exam questions were designed to test basic knowledge and learning outcomes for CS2. Four of the multiple choice questions involved program traces. The fifth multiple choice question tested the students' knowledge of prototypes for templates. The first question tested the students basic skills at tracing while a single loop where the condition was altered within the loop. The goal of this question was to identify students lacking basic programming skills. Although students should not be able to enroll in CS2
without having first obtained these basic skills in CS1, some students may have just passed CS1, but still be weak programmers. A basic knowledge of linked lists is one of the learning outcomes for CS2; therefore, questions 2 and 3 were linked list traces similar to [20]. Question 4 tested students on their basic understanding of objects and methods. Students traced through source code that utilized a string class to manipulate data with supplied methods. Questions 5 and 6 are listed below. Question 5 tested students’ understanding of templates.

5) Consider this prototype for a template function:

```cpp
template <class Item>
void foo(Item x);
```

Which is the right way to call the `foo` function with an integer argument `i`?

A. `foo<int>( i );`
B. `foo<Item>( i );`
C. `foo(<int> i );`
D. `foo(<Item> i );`
E. `<int>foo(i);`

Question 6 tested students’ ability to put into practice, in a limited way, the skills they had developed in CS2. Two key concepts in this question are recognizing the need to allocate storage for the width and height values since width and height are defined as pointers, and recognizing the need as well to de-allocate this storage in the destructor.

6) The following `CRectangle` class is missing its constructor, destructor, width and height and area methods, please write them out so that the code in main() works.

```cpp
class CRectangle {
    int *width, *height;
    public:
        CRectangle (int,int);
        ~CRectangle ();
        int get_width(void);
        int get_height(void);
        int area (void);
    }

    int main () {
        CRectangle rect(3,4);
        cout << "Rectangle " <<
        rect.get_width() << " by " <<
        rect.get_height();
        cout << " has area: " << rect.area() << endl;
        return 0;
    }
```

RESULTS

The multiple choice questions provide an unbiased result while the open question is used to see how well the students can complete a class structure and instantiate it correctly. The confidence interval used for all statistical analysis is 95%.

The multiple choice questions were given a score of 1 for a correct answer and 0 for an incorrect answer for a total of 5. The analysis results are shown in Table I. An analysis using an independent t-test comparison of the two groups, using each student’s mean score of the five multiple choice questions, reveals no statistically significant difference between the two groups ($t(49)=-.526, p=.601$).

<table>
<thead>
<tr>
<th>Group</th>
<th>N</th>
<th>Mean</th>
<th>Std Dev</th>
<th>Std Err Mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>Control</td>
<td>27</td>
<td>3.9259</td>
<td>1.23805</td>
<td>.23826</td>
</tr>
<tr>
<td>Test</td>
<td>24</td>
<td>4.0833</td>
<td>.82970</td>
<td>.16936</td>
</tr>
</tbody>
</table>

Table II is the mean score out of 20 for the two 2005 groups for the open question using a standard scoring methodology and an independent blind grader.

<table>
<thead>
<tr>
<th>Group</th>
<th>N</th>
<th>Mean</th>
<th>Std Dev</th>
<th>Std Err Mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>Control</td>
<td>27</td>
<td>14.3704</td>
<td>4.64218</td>
<td>.89339</td>
</tr>
<tr>
<td>GUI</td>
<td>24</td>
<td>14.2083</td>
<td>3.12047</td>
<td>.63696</td>
</tr>
</tbody>
</table>

The guidelines for grading the open question are as follows. The correct construction of each class method is worth 2 points for each method, for a total of 10 points. The key for the constructor is the student recognizing the need to allocate storage for the width and height values since width and height are defined as pointers; worth 2 points and worth 2 points for assigning their values correctly. For the destructor, the student needs to recognize that the storage allocated for width and height must be released (2 points). The correct use of the pointers for calculating area is worth 2 points. Returning the values of width and height is given 1 point. Various miscellaneous errors cost 1 point, such as not calculating the area correctly. The total points are 20.

An analysis using an independent t-test comparison of means using each student’s question score on the open question revealed no statistically significant difference between the two groups ($t(45.779) =.148, p=.883$).

The statistical analysis was repeated for student data collected for the 2007 control and test groups. An analysis using an independent t-test comparison of means on the multiple choice questions scores reveals no statistically significant difference between the two groups ($t(38)=-.231, p=.818$). Again multiple choice questions were given a
score of 1 for a correct answer and 0 for an incorrect answer for a total of 5. See Table III.

### Table III

<table>
<thead>
<tr>
<th>Group</th>
<th>N</th>
<th>Mean</th>
<th>Std. Deviation</th>
<th>Std. Error Mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>Control</td>
<td>13</td>
<td>4.0769</td>
<td>1.03775</td>
<td>.28782</td>
</tr>
<tr>
<td>Test</td>
<td>27</td>
<td>4.0000</td>
<td>.96077</td>
<td>.18490</td>
</tr>
</tbody>
</table>

Again for 2007, an analysis using an independent t-test comparison of means of the open question scores reveals no statistically significant difference between the two groups ($t(13.7)=.075, p=.941$). See Table IV.

### Table IV

<table>
<thead>
<tr>
<th>Group</th>
<th>N</th>
<th>Mean</th>
<th>Std Dev</th>
<th>Std Err Mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>Control</td>
<td>13</td>
<td>14.3077</td>
<td>6.51133</td>
<td>1.80592</td>
</tr>
<tr>
<td>Test</td>
<td>27</td>
<td>14.1667</td>
<td>2.49230</td>
<td>.47964</td>
</tr>
</tbody>
</table>

A factor not included in the above statistics is the number of students withdrawing from the course before the end of the semester. In 2005, five control group students and five test group students withdrew. In 2007, 4 control group students and two test group students withdrew.

For students in the 2005 spring semester, their progress through the next course in the sequence, Data Structures, was also examined. See Table V. No statistically significant difference existed in the final average GPA for the course ($t(24)=0.0, p=1.0$). Student GPA was assigned numerical values as follows 0 for F, 1 for D, 2 for C, 3 for B and 4 for A.

### Table V

<table>
<thead>
<tr>
<th>Group</th>
<th>N</th>
<th>Mean</th>
<th>Std Dev</th>
<th>Std Err Mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>Control</td>
<td>12</td>
<td>2.5000</td>
<td>1.24316</td>
<td>.35887</td>
</tr>
<tr>
<td>Test</td>
<td>14</td>
<td>2.5000</td>
<td>1.01905</td>
<td>.27235</td>
</tr>
</tbody>
</table>

Although no data was collected for spring 2006 for the control group, data was collected for the test group. Therefore, the year to year results on the exam can be examined as shown in Table VI which reveals no significant difference (ANOVA $F_{(2,70)}=2.683, p=.075$).

With the open question for the test group during the 3 years, see Table VII, some differences existed, specifically between the spring 2005 and spring 2006 test groups (ANOVA $F_{(2,70)}=4.156, p=.02$).

In Table VIII, the mean GPA for the three test groups are presented and no significant difference was found. (ANOVA $F_{(2,73)}=.292, p=.677$).

### Table VI

<table>
<thead>
<tr>
<th>N</th>
<th>Mean</th>
<th>Std Dev</th>
<th>Std Err Mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>2005</td>
<td>24</td>
<td>4.1250</td>
<td>.85019</td>
</tr>
<tr>
<td>2006</td>
<td>22</td>
<td>4.5455</td>
<td>.67098</td>
</tr>
<tr>
<td>2007</td>
<td>27</td>
<td>4.0000</td>
<td>.96077</td>
</tr>
<tr>
<td>Total</td>
<td>73</td>
<td>4.2055</td>
<td>.86537</td>
</tr>
</tbody>
</table>

### Table VII

<table>
<thead>
<tr>
<th>N</th>
<th>Mean</th>
<th>Std Dev</th>
<th>Std Err Mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>2005</td>
<td>24</td>
<td>12.9167</td>
<td>4.11695</td>
</tr>
<tr>
<td>2006</td>
<td>22</td>
<td>15.5227</td>
<td>2.23861</td>
</tr>
<tr>
<td>2007</td>
<td>27</td>
<td>14.1667</td>
<td>2.49230</td>
</tr>
<tr>
<td>Total</td>
<td>73</td>
<td>14.1644</td>
<td>3.19402</td>
</tr>
</tbody>
</table>

### Table VIII

<table>
<thead>
<tr>
<th>N</th>
<th>Mean</th>
<th>Std Dev</th>
<th>Std Err Mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>2005</td>
<td>26</td>
<td>2.8462</td>
<td>1.22286</td>
</tr>
<tr>
<td>2006</td>
<td>23</td>
<td>3.0870</td>
<td>1.08347</td>
</tr>
<tr>
<td>2007</td>
<td>27</td>
<td>2.8519</td>
<td>.90739</td>
</tr>
<tr>
<td>Total</td>
<td>76</td>
<td>2.9211</td>
<td>1.06787</td>
</tr>
</tbody>
</table>

### Conclusions

The results show that using GUI’s as a tool for teaching C++ object-oriented programming does not negatively impact the students. On the other hand, it does not significantly improve the students’ performance. Students in the control and test groups who took Data Structures in 2006 did so in the same section with no statistical difference in their final course grades. Data structures was offered using a text-based approach. Students, however, were not discouraged from completing assignments using a GUI approach and one student did so. Although no difference was found statistically between the control and test groups on average, more students enrolled in the GUI-based sections of CS2 than the text-based sections. Drawing any conclusions from this fact is difficult other than to say that students do not appear to be avoiding the GUI sections even though that approach may take a little more effort.

### Acknowledgements

This study has been approved by the, Texas Tech University Institutional Review Board for the Protection of Human Subjects Committee.
REFERENCES


AUTHOR INFORMATION

Per Andersen, (B.Eng’80-M’97) received the Ph.D. degree in computer science from Texas Tech University, Lubbock, Texas, in 2002. He joined the Computer Science Department at Texas Tech University, Lubbock, in 2002 as an Assistant Professor. Dr. Andersen’s research interests include computer science education, distributed computing, human computer interaction and computer-based learning of health care issues for low-literacy patients.

Susan Mengel, (BS’82-MS’84) received the Ph.D. degree in computer science from Texas A&M University, College Station, in 1990. She joined the Computer Science Department at Texas Tech University, Lubbock, in 1996 as an Assistant Professor. Currently she is an Associate Professor and Graduate Advisor. She serves as the Associate Editor of Computing for the IEEE Transactions on Education and is Vice-President of the Texas Tech Faculty Senate. Her research interests include data mining and engineering education.