Experience with GUI Programming in Lower Level Classes

Curt Hill

Mathematics Department

Valley City State University

Curt_Hill@mail.vcsu.nodak.edu

Abstract:

The Graphical User Interface is now the most familiar, yet many introductory programming classes ignore it. Windows programming has been an important part of CS1 at Valley City State University since the fall of 1997. This approach offers both opportunities and dangers to instructors and students. The success of teaching this interface in the introductory classes, often hinges on the use of a development environment that supports it well and is easy enough for students to use.

Introduction.

The Graphical User Interface (GUI) is extremely pervasive but its use in introductory computer science classes is not. The use of a new approach in any course must be justified by the benefits that such an approach will give to either the students or faculty.

There are many good reasons to be programming with the Graphical User Interface. The GUI is now the most common user interface, which in itself argues for its use in the introductory programming class. Students are quite familiar with this style of interface and tend to lose motivation when forced to generate batch mode programs or a console interface. The implication of using only the console interface is that programming is too difficult for them, which is not the message they should hear in the first class. Instead when they write with a graphical user interface they see themselves as accomplishing something useful which greatly increases their motivation. Many students will not become professional programmers, but almost all will receive some benefit if they can write simple programs to aid them in their chosen careers. They will not usually write such programs if they can only write them using a console-style interface. Potential employers will also be more likely to be impressed if the applicant has demonstrated programming using what is now the conventional interface. Those students who will join the craft need to be exposed to the types of sophisticated tools that they will use later. There are some additional advantages that will be considered later.

However, there are several good reasons **not** to use a GUI in the early programming classes. There is the quite reasonable concern that too much time will be spent on the graphical user interface and not enough on the traditional topics of the class. An easy to use programming environment is required for any mode of programming in the introductory classes. The console mode environments tend to be inexpensive or widespread, while the corresponding windows environments are usually proprietary products. Therefore they generally cost the student or the institution, a non-trivial consideration. The worst but perhaps most common reason is instructor anxiety. Language syntax and semantics have not changed that much since the early days, however the environment has. Learning the use of this paradigm and the needed tools puts additional stress on faculty who are sufficiently busy already.

Early experiences with the use of GUI

Valley City State University has been a laptop institution since the fall of 1996, making it the second such university. Each incoming freshman is issued a laptop computer, which is theirs to use for the duration of the term. The student receives the laptop after an initial one-hour presentation and is required to take a computer literacy class, which most take or pass out of in their first year. The laptops have a standard suite of programs, which in the last two years included Microsoft Windows 98, Microsoft Office, Netscape Navigator, Novell Netware and Novell Groupwise. Generally students entering the

programming classes are computer literate. This type of sophistication has made it difficult to stay with console-based programming.

The frustration of teaching console-style programming in a predominantly GUI environment became obvious to this instructor in the first year laptops were the norm. Therefore in the Spring semester of 1997 the programming topics course, a junior level class, was modified to teach not a new programming language but windows programming using C++. There were no tools available to ease the process. The cumbersome coding needed made the "Hello World" program several hundred lines. Since every windows program had certain parts in common: a minimum set of includes, a WinMain function, a function to register the window class, a function to initialize the window, a function to contain the message loop it was concluded that there must be several better ways of doing this. Therefore a program called WinShell was written and distributed to the students of this class that would jump started the process. It collected the input of several dialogs and then generated the shell of a Windows program. This generated program would have a menu, dialog boxes and event handlers for each of the menu entries, buttons or other controls, but no logic as to what to do in each event handler. When the generation was complete the student would arrange the items in the dialog box and add all the logic needed. Although this greatly eased the problem of generating a Windows program, it was not easy enough to use to throw into the mix of an introductory programming class.

Wolz, Weisgarber, Domen and McAuliffe [1] decry the minutia needed to master GUI systems and their objections are quite valid. These were exactly the types of problems that even the WinShell program failed to control. However, some of their objections are diminished by the introduction of better sets of tools. These tools make the programming much easier, but tend to make the environment somewhat more complicated. Generally the tradeoff is still advantageous.

It became obvious that better tools were indeed available. In the following term, Fall of 1997, Borland's CBuilder software was used in the introductory class. In that term GUI programming was introduced late in the semester. It has moved around some since that time and the current structure seems to have the right feel.

The structure of the introductory class.

The introductory programming classes constitute a three-semester sequence in C++. The first of these is CSci 160, which is required or recommended for several majors. The rest of the sequence includes CSci 161 and CSci 242, and are taken mostly by those seeking a Computer Science minor. The first course, CSci 160, is where the students learn the bulk of their windows programming. The students of CSci 160 are required to buy the current version of Borland's CBuilder compiler and development environment, which is approximately fifty dollars at the bookstore. This becomes their property, can be installed

on other machines that they own or on future laptops and will be used in the subsequent classes.

The first course conventionally starts with console programs when dealing with the early parts of the language such as variables, expressions and simple input and output. Functions available from libraries are discussed and used early, but the students do not define any of their own until later in the course. Similarly, objects are used early but not defined by the students until early in the second course.

The importance of the CBuilder package to this class starts with console-style programs. The package generates parts of the program that are predictable, such as the includes, the main function header, and the return statement at the end of the main function. Hence the students only need to code the declarations, assignments and I/O statements, although the generated parts are visible and discussed in class. This has the tendency to constrain their thinking in a desirable way [2], so they will not have to deal with issues that they have not yet built a suitable conceptual framework.

The next C++ topic is about the decision statements and it is usually in this segment that the Windows programming paradigm is introduced. Typically this takes about two hours of classroom demonstration to accomplish. The first topic of Windows is the difference between the console model and the event model needed for a Windows program. This is usually readily accepted since the students are quite familiar with the new paradigm although from a user perspective. This discussion is seen as an advantage of this approach, since it does not take very long and exposes them to differing paradigms from a very practical viewpoint. The second topic is a demonstration of how a Windows program is constructed using CBuilder. It should be noted that the classroom has a large screen monitor that all the students may see and follow along on their own laptops. The typical program that is first written is similar to the last console program, so that the students may see the comparison in the code as well as the look and feel. This demonstration takes about twenty to fifty minutes.

The way a program is constructed in CBuilder is simple for students to accomplish. There is a palette of components such as buttons, labels, menus and edit fields. Each such component is a predefined class with properties, methods and events. The student drags the component from the palette to the window they are building and then sizes it by dragging. When a button needs an event handler, the student types in the name they want for the event handler or merely double clicks the component and a name is provided. At that point the method for that event handler is generated in a way similar to the generation of the main function. The properties may be modified simply at design time or at run-time. Most students have their laptops with them and have written their first Windows program during the demonstration.

In the first week of GUI programming the students are introduced to four basic components: the form or window, a button, static text or labels and the edit box. The students have used all of these before in applications, what they need to see is how to use them in programs. This is usually how to set the properties or use the event handlers.

After the initial demonstration of windows components, the course advances on two different fronts. The bulk of the class is on new language constructs, but new window components are also considered. In one sense they are unrelated, but they are used to reinforce each other. Shortly after the **`if`** statement is introduced the Message Box component is demonstrated in the context of error checking; the **`switch-case`** is used to handle a radio box component; and the multi-line edit box is used to hold the output of **`for`** loop.

The preferred lecture style of the course is to divide a 50 minute class period into two to four segments. Each segment deals with a different topic or activity. These segments are chosen to put some variety in the class time. One segment may be the lecture on a particular syntax issue, a demonstration of a program or program segment, or some group work such as the tracing of a program or the completion of a program fragment. McConnell [3] notes that student interest wanes after as little as 10 minutes and this changing of topic and tone seems to help in this regard.

The total time spent on the windows programming aspect of the class is hard to estimate since it is spread over the whole semester. Usually only one or two full class periods are entirely used on the topic, but the total amount of time used in class on the topic is about five to ten class periods of 50 minutes. This would be two to three weeks of a sixteen-week semester. Moreover, as mentioned earlier the windows components can be used, like many other applications, to emphasize topics that are more central to the course.

The students are more encouraged with their first windows program than a console program of equal difficulty. The Borland components contain familiar behaviors that make the program seem more sophisticated than the work put into it by the student. The testing is somewhat easier as well. A loop-less console program must be executed multiple times to test, while there is no such thing as a loop-less windows program. The loops are built into the system code so the student may click a button multiple times and observe the behavior.

There are clearly some negative consequences to this approach. One of these is using features of the language before they have been introduced. All of the Borland components are objects on the heap, thus the students need to use the −> pointer de-reference operator well before they are introduced to pointers or class definitions. The usual event handler has a pointer as a parameter, but that is less troublesome, since the system generates the method header. The development environment can be a problem as well. The environment is fairly complicated and students do not generally understand all the things that they can do from it, nor the many ways they can damage their work. In order to compile a typical windows program, five files are required: the two C++ files, a header file, a data form (which captures the shape and arrangement of the window and translates into the resource file), and a project file (which includes make instructions). These five are required to be emailed to the instructor for grading and sending the wrong set is a common occurrence. Another common problem is establishing the wrong event handler. Deleting this method requires a fix to the header file, which the students do not

usually see, let alone edit. These problems are annoying but not outweighing the advantages.

The college has an electronic portfolio requirement. Each student is to amass evidence of their education as they advance to their degree. This portfolio is usually organized as a presentation or web site on a CD ROM and then is available to supplement their résumé in job applications. Courses that are important in major or minor programs often have a portfolio project, which should demonstrate accomplishments towards their careers. Usually the later classes have the better projects, since the skills learned in later classes better match the skills desired by employers. However, the fall of 2001 CSci 160 class had a portfolio project that was a useful program with several desirable characteristics.

The project was to produce a windows program that calculated the payment and amortization table for a loan. The interface used most of the commonly seen controls such as labels, buttons, data entry boxes, multi-line memos and menus. The code also demonstrated several common techniques: verification of inputs, a non-trivial calculations and multiple computations within a loop. This was not the size of a term project, most of the students took only about a week on it, but in a single program it shows many of the things that they were to learn in the first programming course. Screen shots of this project are shown in the appendix.

A brief literature review.

This paper reflects one instructor's attempt to keep an introductory course current and relevant as the computing environment changes. There is no claim of being the first to try such techniques. Rather there have been several papers published that show the progress on this frontier. Several of these will be now mentioned and briefly discussed.

There are several who have used in C++ in their classes, with a suitable development environment. Woodworth and Dann [4] describe teaching CS1 with Borland's C++Builder. They cite the market opportunities for GUI programs and programming. Mutchler and Laxer [5] document using Microsoft Visual C++ in their CS 1 class at Rose-Hulman. They sought to do so to generate enthusiasm in their students by generating graphic and multimedia applications. Szuecs [6] used Borland's Object Windows Library but in a successor of CS 2. The intent was to expose students to creating contemporary user interfaces.

Java is a more recent language with a sophisticated event handler model and better platform independence. Bruce, Danyluk and Murtagh [7] used Java in their version of CS1. They argue that this is current paradigm of programming so should be taught initially to students. Wolz and Kaufman [8] attempt to simplify the usual Java I/O with a toolkit in their CS1.

Rasala [9] promotes the idea of a toolkit in the first course. He cites toolkits written in C++ and Java at various institutions. The Borland Visual Component Library or Object Window Language and the Microsoft Foundation Classes are all toolkits but intended for professionals, while these are for students.

Conclusions.

The key to teaching GUI programming in the first programming class is a development system and framework that makes it easy enough for a student to use without devoting a substantial portion of the class time. If that condition is satisfied then the extra work is worth it and all the problems are manageable. Several such systems do exist for various programming languages.

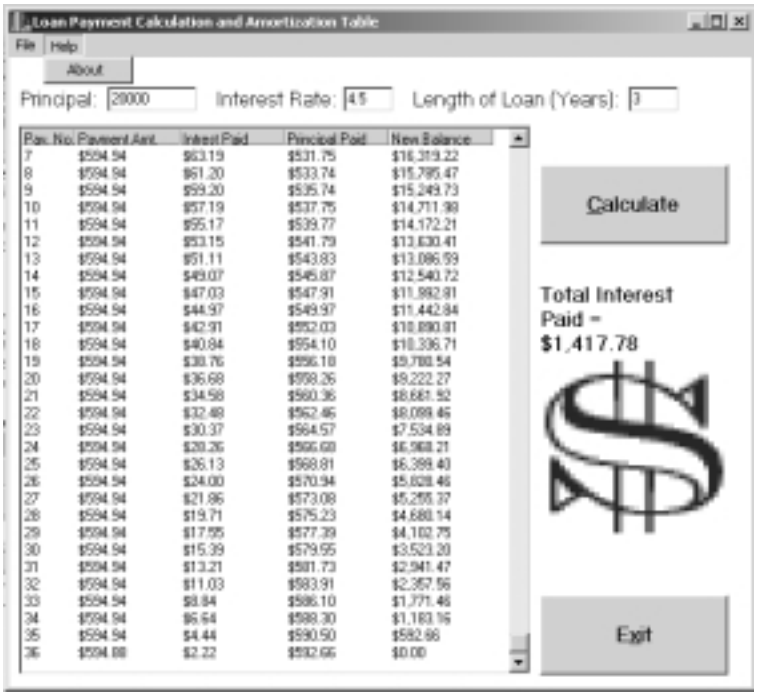
References.

1. Wolz, Ursula, Scott Weisgarber, Daniel Domen and Michael McAuliffe (1996). Teaching introductory programming in the multi-deia world. Proceeding of SIGCSE Integrating Technology into Computer Science Education Technical Symposium on Computer Science Education, June 1996 at Barcelona, Spain, pp. 57 - 59.
2. Buck, Duane and David J. Stucki (2000). Design Early Considered Harmful: Graduated Exposure to Complexity and Structure Based on Levels of Cognitive Development. SIGCSE Technical Symposium on Computer Science Education, March 8 - 12, 2000 at Austin, Texas, pp. 75 - 79.
3. McConnell, Jeffrey J. (1996). Active learning and its use in Computer Science. Proceeding of SIGCSE Integrating Technology into Computer Science Education Technical Symposium on Computer Science Education, June 1996 at Barcelona, Spain, pp. 52 - 54.
4. Woodworth, Pan and Dann, Wanda, 1999. Integrating Console and Event-Driven Models in CS1. *SIGCSE Bulletin*, March 1999, pp. 132-135.
5. Mutchler, David and Cary Laxer (1996). Using multimedia and GUI programming in CS 1. Proceeding of SIGCSE Integrating Technology into Computer Science Education, June 1996 at Barcelona, Spain, pp. 63 - 65.
6. Szuecs, Laszlo (1996). Creating Windows Applications Using Borland's OWL Classes. *SIGCSE Bulletin*, February 1996, pp. 145-149.
7. Bruce, Kim B., Andrea P. Danyluk and Thomas P. Murtagh (2001). Event-driven Programming is Simple Enough for CS1. Proceedings of SIGCSE Integrating Technology into Computer Science Education, June 2001 at Canterbury, UK, pp. 1 - 4.

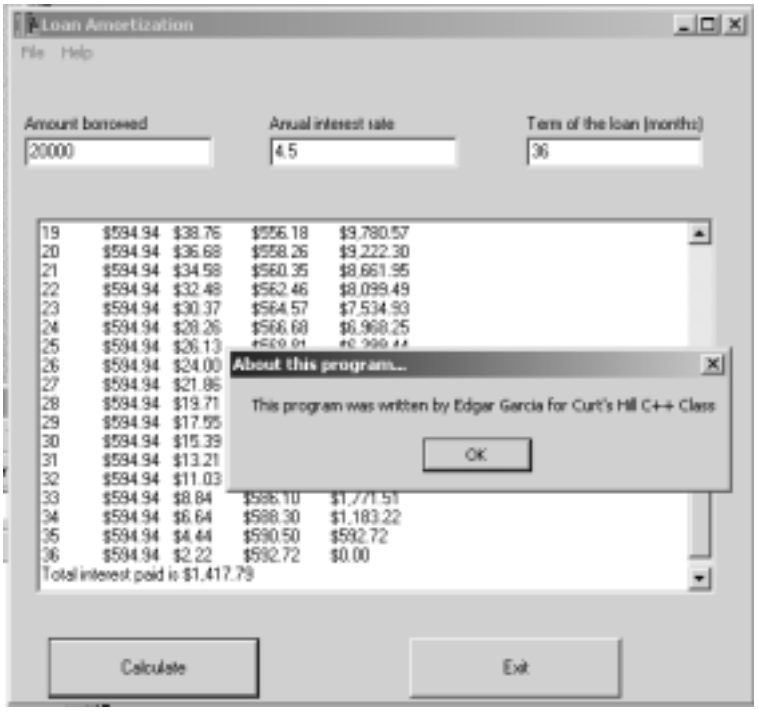
8. Wolz, Ursula and Elliot Koffman (2000). A Java package for Novice Interactive and Graphics Programming. Proceeding of SIGCSE Integrating Technology into Computer Science Education Technical Symposium on Computer Science Education, June 27 - July 1, 1999 at Cracow, Poland, pp. 139 - 142.
9. Rasala, Richard (2000). Toolkits in First Year Computer Science: A Pedagogical Imperative. SIGCSE Technical Symposium on Computer Science Education, March 8 - 12, 2000 at Austin, Texas, pp. 185 - 191.

Appendix.

The following graphic shows the window of the loan program written by Brandon Flowers after the calculation button was clicked.



The following portrays the loan program of Edgar Garcia after the Help About menu item had been clicked.



I would like to acknowledge and thank both Brandon and Edgar for allowing the use of their programs in this paper.