Drop and drag - Graphical User Interface programming to create enthusiasm in first year programming

Heiko E. R. Rudolph[*], Suresh Venkatachalaiah[+] and Roy Ferguson[*]

[*] School of Electrical and Computer Engineering, RMIT University
GPO Box 2476V, Melbourne, Victoria, Australia 3001
{heiko.rudolph/roy.ferguson}@rmit.edu.au

[+] Centre for Advanced Technology in Telecommunications,
School of Electrical and Computer Systems Engineering, RMIT University
Box 2476V, Melbourne, Victoria, AUSTRALIA, 3001
suresh@catt.rmit.edu.au

Abstract: Programming has been an integral part of computing courses at universities all over Australia especially in first year undergraduate engineering. Sound programming basics in first year are essential and the methods and impressions conveyed in first year are very important for future years. For this reason, we have sought to engage student interest by teaching students to create simple but functional Graphic User Interface (GUI) programs. There was some concern as to whether GUI programming was perhaps too difficult and complicated for an introductory programming course. To establish how well students would generally be able to adapt to GUI programming a project was proposed, which presented students with the choice of both traditional console based and GUI based programming exercises. Students were able to choose freely between both methods. This paper details the results and provides a survey of students' preferences

Keywords: C++ .NET Programming, First Year, GUI/Console Programming, Learning guides, Teaching and learning strategies

1. Introduction

In this paper we take an in-depth look at two very different ways of presenting a first year programming language. In our approach we try to achieve several goals, the foremost of which is to engage student enthusiasm and interest. It was hypothesized that enabling students to create small windows programs that executed in a standard windows operating system, would be an important factor in promoting enthusiasm and interest [6].

Teaching programming languages [1] and conducting surveys of learning methods. Teaching programming languages has been achieved with learning tools [1] and conducting surveys to improve outcomes. Some of the methods proposed in the literature [5][8] encourage students to work in teams to enhance motivation. There are several papers on teaching programming techniques. One of them proposed by Collins and Fung, [5] used a tool called "DChK" for checking errors in real-time.

"Proceedings of the 2005 ASEE/AaeE 4th Global Colloquium on Engineering Education

This tool also allows them to check spelling, misspellings and incorrect use of case. It also ensures that the code conventions used are in the course and are adhered to.

In our research, a task was to establish students' ability and preference for creating these small windows program using Graphics User Interface (GUI) programming. To this end students were exposed to both console programming and GUI programming and then asked to freely choose between the two methods. Several surveys were conducted to establish students' preferred programming environment. The surveys included comparison of choices and examination results. The comparisons extended over one semester with reference to the previous semester.

The paper aims to:

- ✓ Show directions, which engage and capture students' interest in programming at an early (first year) level.
- ✓ Highlight the strengths and weaknesses of two different programming environments.
- ✓ Give an insight into the considerations students face when asked to choose between traditional console programming and more the more familiar GUI environment.

GUI programming provides a different approach to programming from the traditional and popular console environment. Unlike console-based programming GUI programming was thought to be more interesting because of the familiar look and feel of programs using buttons and textboxes and the usual components of a windows based environment. For this project we opted for Visual Studio .NET that allowed us to create both console-based as well as GUI based programs. In this paper, the research component revolved around finding the interest levels of the students depending on which programming style they chose and which approach they found most appealing. Furthermore the research also attempted to evaluate the degree of success in teaching students GUI programming techniques. It has been well established that active participation from students and detailed feedback from them play an important role in teaching and learning [2][3].

In summary the project consisted of teaching students both console and GUI based programming and then surveying students' free and independent choices in completing laboratory assignments in either the console or the GUI style. Given a general trend away from programming in undergraduate engineering courses at our University the authors believe that student motivation regarding programming is an important factor in teaching introductory courses in particular. The literature highlights a number of areas requiring improvement in higher education teaching [7] such for example: the lack of presentation skills of lecturers and tutors as well as a lack of 'motivation and enthusiasm' for their subject.

The paper is organised as follows. Section 2 presents the rationale and motivation. Section 3 describes the relative merits and demerits. Section 4 describes the Research design. Section 5 presents the teaching methods used. Section 6 presents results and evaluation followed by conclusions.

2. Rationale and motivation for this research project

In Semester-2 of 2004, we introduced GUI programming into the curriculum of our first year computer systems engineering program partly replacing traditional console based programming. Among the questions posed were: How does the GUI program fit into our teaching program? How can we adapt such a style to a teaching style and what teaching methodology can we adapt to teach this style of programming? The project described here was known as "The GUI Project" or simply "the project". It allowed students the choice between traditional command-line programming or the use of a Graphics User Interface (GUI) style of programming. "GUI programming" in this context is defined to mean programming using visual form based programming, using 'Drag and Drop Form design'. Unlike older programmers who most likely experienced DOS and the command line environment, the current generation of students are likely to find themselves much more at home in the GUI (Windows) environment than previous generations. Furthermore the graphic windows environment is the current standard, as command-line programming once was in its day. Teaching GUI programming can be seen as simply teaching state of the art programming. It was postulated that students would enjoy writing programs that looked like the windows programs they use regularly, as much if not more so than console based programs. The simple GUI programs taught in this course used all the elements of graphics programming, such as buttons, windows, and menus. It was further hoped that students would be encouraged by experiencing early success in creating GUI programs that look and feel very similar to commercially produced software.

The project focused on teaching students the basics of creating simple graphics interface based programs with a minimum of theory. It was hoped that students would build on early successes. The focus of the project was not so much quantitative as qualitative and in this respect served as an exploratory exercise in comparing students' responses to GUI and command-line programming. The aims of the experimental project were to ascertain if:

(1) Students would generally be able to learn GUI style programming in the first year of University, as part of their initial introduction to computer programming. Reservations existed that the complexity of understanding required for GUI programming with its reliance on Object Oriented Programming theory and event driven program execution would be beyond the scope of most first year programming students.

(2) Students would find GUI programming more engaging than command-line programming since it produced executable programs that looked and behaved in the same way as the graphics windows operating system the programs were created on and which has become the standard computer interface for the majority of personal computer users in the world. Initially it was not expected that either method of presenting a programming language would be universally favoured, so the project served as a yardstick to explore the relative strengths of each method.

Above all the project sought to develop a spirit of enjoyment in programming by optionally asking students to enter a programming competition in creating a simple fun and easy to use computer programs that could be passed on to friends and non programmers. The last point may deserve more attention, in that peer approval and the '*wow factor*' in being able to produce programs that can be presented to peers, family and friends could also be a powerful motivator.

3. Relative merits of console VS GUI styles of programming

Each of the two styles of programming offers relative strengths and has corresponding weaknesses. A brief outline is given below:

	Advantages	Disadvantages
Console	• Complexity is minimised. Console programs are simple and easy to compile. Extraneous material is minimised. • Core programming concepts can be clearly highlighted by themselves to illustrate a point. • Core concepts do not change as quickly as complex GUI implementations.	• Unfamiliarity: most young computer users would be most familiar with GUI operating systems such as Windows. • Console programming is perceived as looking "boring" and not as likely to engage the interest of students early in their studies.
GUI	• GUI programs are 'state of the art' and the kind of programs that students are familiar with through general computer use. • GUI enables students to produce programs that are 'on par' with current technology i.e. programs that operate in a windows environment. • Students produce programs that can immediately run on the current computer platforms.	• Complexity: only a limited number of vendors produce systems that allow 'drag and drop' form design. • Event driven programming is a new concept that may present a hurdle. • Computers compiling GUI programs require more resources than those used in console based programming.

4. Research design

The fundamentals of programming using C++ as the language of choice were introduced using command-line programming within an Integrated Development Environment (IDE). The laboratory program formed the backbone of the project and consisted of 3 laboratory exercises that required students to create a simple arithmetic calculator program.

The first laboratory exercise required students to code a simple command line calculator and involved the basics of variable types, control loops, and command-line in/output.

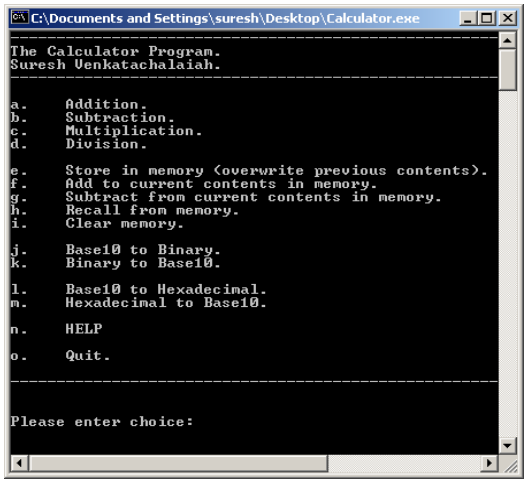


Figure 1. A screen-shot showing a console window (black) on a standard Windows XP Operating Environment. This is a popular and widely used method in lectures

The second laboratory exercise consisted of a GUI version of the same calculator as in laboratory one. No additional functionality in the program was asked for, so laboratory exercise essentially consisted of an exercise in learning to use the GUI drag and drop method, placing components visually and becoming familiar with the complex development environment. Up to this point the rationale was to expose all students to both methods of programming.

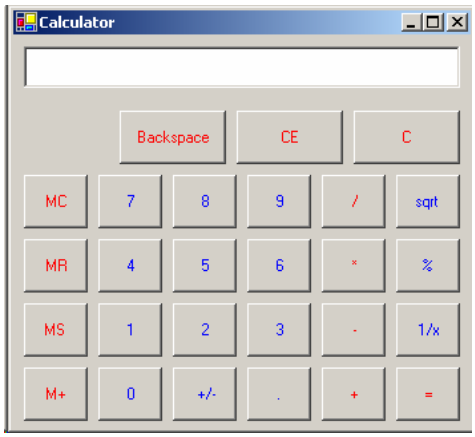


Figure. 2. A screen-shot showing a GUI window on a standard Windows XP Operating Environment.

In the third laboratory exercise students were asked to choose between programming styles using a command-line environment and a GUI environment. It was assumed that having used both methods students' choice would be largely influenced by the merits of each method in its own right.

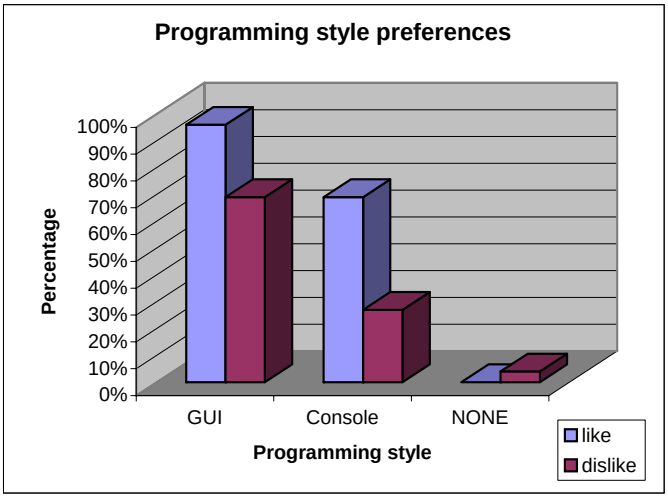


Figure3: Preference by programming style. For this exercise students were freely able to choose their style of programming.

5. Teaching support: LAB Guides

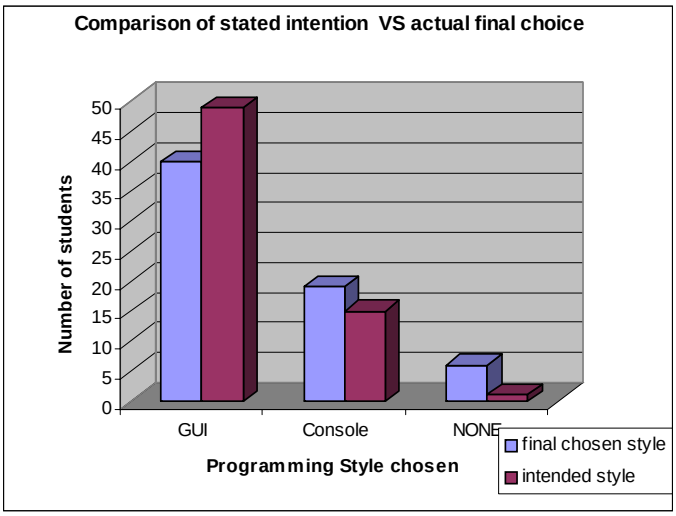
Based on literature [4] and previous experiences from teaching introductory programming it was clear that help in the form of laboratory guides for the students would be required. Therefore a series of laboratory guides was produced. The laboratory exercises developed were both console and GUI based and used a step-by-step approach.

All teaching material was put online and was based on the current textbook. Students found the laboratory guides useful in clarifying the requirements of the laboratory exercises. Specifically, the guides provided the missing details regarding practical GUI programming which was something that was perhaps missing a little from the lecture series and the prescribed textbook. The guides were also helpful by repeating material covered in lectures at a more detailed level. Student feedback indicated that the guides were an essential part of the program. Particularly useful was the step-by-step approach of the guides, tailored to the set laboratory exercises.

The laboratory guides acted as building blocks for successive sessions. Of necessity much of the information in the learning guides could not be adequately provided in the lecture setting.

6. Results and evaluation

Indicators: Student choices of programming style and perceived motivation are the chief indicators in this study. In particular we looked at stated intentions before choosing a programming style and compared this with actual programming styles used. The usual laboratory submissions as well as focus groups and on line surveys were used to establish these results. The results in figure 3 suggest that the students were attracted more towards the GUI instead of the console programming style prior to the final laboratory exercise. It should be noted that the availability of GUI programming as an alternative was flagged at the beginning of the course, and students were aware of the choice they would be asked to make.



Figures 4: Student choices programming style

Slightly fewer students (15%) than had indicated this in their original intention, decided to implement the GUI style in the final laboratory exercise (shown in figure 4). From survey feedback this was thought to be mainly due to a lack of coverage of the GUI style in lectures and the greater complexity of the GUI itself.
Overall number of student's who preferred GUI and who chose GUI in the final analysis is approximately double that of those who preferred and later chose the console style.

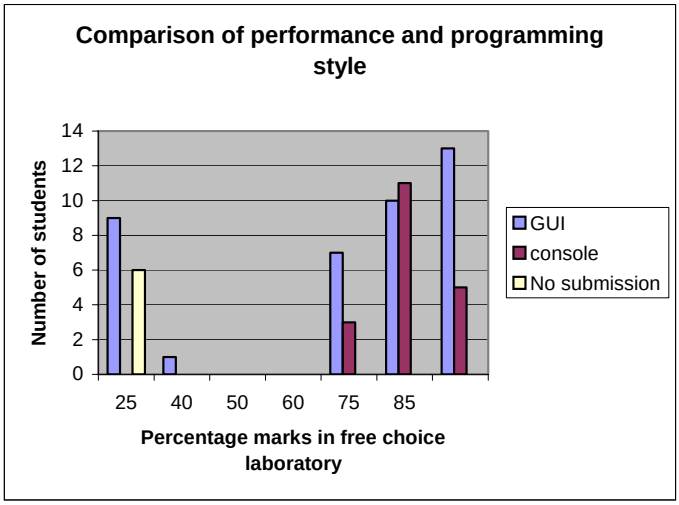


Figure 5: Performance by programming style. For this exercise students were freely able to choose their style of programming.

Grades obtained versus chosen programming style: In figure 5, we see that in the higher graded population (>85%) the GUI style was chosen by approximately double the number of students. It could be said that the high performance students

chose the GUI style as a challenge and generally fared quite well. In the 75-85% grade range the preferences for programming style were approximately equal whereas in the 60-75% range again more approximately double the number of students chose GUI.

It is interesting to note that the 75-85% group chose GUI and Console in virtually equal proportions. It might be speculated that this group simply chose the most direct and lowest risk route to fulfilling their laboratory requirements. We do not know how many of the students in each group had prior programming experience and this would be worthwhile investigating in the future. Interesting was that amongst those students who scored very low, all attempted GUI with none attempting the console style. This group was also fairly homogenous in age and came from the engineering stream. It appeared that students appreciated the bold step taken in introducing GUI programming.

Comparison of courses with and without GUI programming

It was of interest to find out if the introduction of GUI programming may have affected the performance of students when compared to essentially the same course but without the GUI components. Both from formal laboratory and examination results, through surveys and focus groups, it was apparent that the GUI component did not present a hurdle to first year students undertaking introductory programming. Figure 6 illustrates the pass/fail ratios compared to a semester when no GUI programming was taught. This is all the more significant considering that the lecture program did not particularly stress GUI programming, instead restricting itself to a number of short demonstrations and referring students to the laboratory guides. Students did comment in focus groups that the lecture program should have put more emphasis on the GUI aspects of programming.

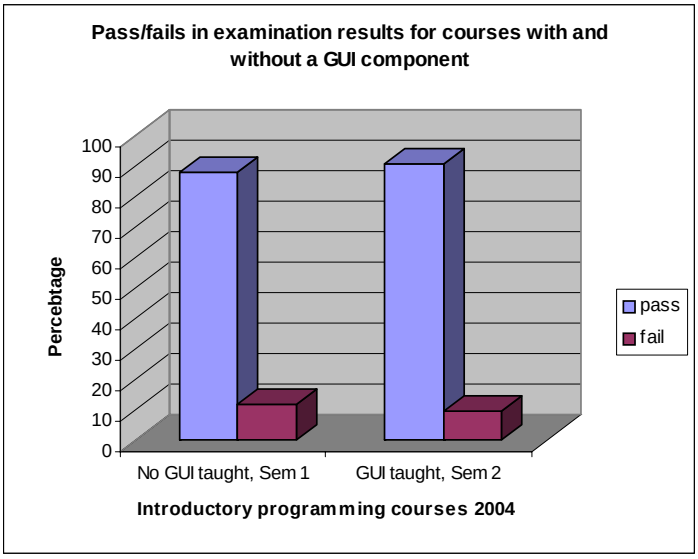


Figure 6: Pass/fails in examination results for courses with and without a GUI component.

7. Conclusion

Given the choice of GUI and traditional console programming students did not clearly prefer one method to the other. Those students who were high achievers tended towards the GUI style, while those who were not as confident usually opted in the reverse proportion for the console.
Student's performance in both laboratory and examinations showed no significant difference between courses that did or did not have a GUI component.

In summary the authors would like to say that: Introducing GUI programming in first year is a realistic and feasible option. We have found the GUI component motivated students and could be taught in parallel with console programming without causing confusion.

We have learnt that GUI should be supported through additional guidance and learning material. The provision of a discussion board used during the course was particularly useful for GUI programming, because is allowed for a lively interchange amongst students.

8. Acknowledgements

The authors wish to thank the RMIT Academic development group for their support of the Action Research in Teaching and Learning (ARTL) grant, which made this project possible. We are particularly thankful to Warren Nageswaran, Rosemary Chang and Peter Muir for their support and encouragement. We would also to thank PJ Radcliffe, Andrew Jennings, David Jones and the "WEBRATS" team for their constructive criticism throughout the project.

9. References

1. Toll, D. G., Barr, R. J., 1998. Ma computer-aided learning system for the design of foundations advances in engineering software. , Volume 29,Issues 7-9, 637{643. 11
2. Nkambou. R, Frasson. C and Gauthier. G, A new approach to ITS-curriculum and course authoring: the authoring environment, Computers & Education, Volume 31, Issue 1, August 1998, Pages 105-130.
3. David Crowe and Hossein Zand, Computers and undergraduate mathematics: I: setting the scene, Computers & Education, Volume 35, Issue 2, 1 September 2000, Pages 95-121.
4. J. Geller and R. Dios, A low-tech, hands-on approach to teaching sorting algorithms to working students, Computers & Education, Volume 31, Issue 1, August 1998, Pages 89-103.
5. Trevor D. Collins and Pat Fung, A visual programming approach for teaching cognitive modelling, Computers & Education, Volume 39, Issue 1, August 2002, Pages 1-18.
6. Brenda Cheang, Andy Kurnia, Andrew Lim and Wee-Chong Oon, On automated grading of programming assignments in an academic institution Computers & Education, Volume 41, Issue 2, September 2003, Pages 121-131.
7. John Milliken and L. Philip Barnes, Teaching and technology in higher education: student perceptions and personal reflections, Computers & Education, Volume 39, Issue 3, November 2002, Pages223-235.
8. Ricardo Rosas, Miguel Nussbaum, Patricio Cumsille, Vladimir Marianov, Mnica Correa, Patricia Flores, Valeska Grau, Francisca Lagos, Ximena Lpez, Vernica Lpez et al., Beyond Nintendo: design and assessment of educational video games for first and second grade student,Computers & Education, Volume 40, Issue 1, January 2003, Pages 71-94

Biographical Information

HEIKO RUDOLPH is a lecturer at RMIT University in the School of Electrical and Computer Engineering.. He holds a Dip. Ed. from Monash University and a Bachelor of Engineering from RMIT, completing his PhD at The University of Melbourne in 1996 in the field of Biomedical Engineering (Pain Control). His research interests include Biomedical Engineering (pain relief), software, vision to sound mapping and novel ways to encourage learning communities

SURESH VENKATACHALAIAH is presently a PhD candidate at CATT Centre (Centre for Advanced Technology in Telecommunications, RMIT University. He has his B.Engg. in Instrumentation & Electronics from Bangalore University, India. He completed his master's degree in Computer Systems Engineering with focus on network related subjects. He has also lectured and supervised students of first year Engineering. His research interests include the areas of communications and software development.

ROY FERGUSON is Director of Teaching and Learning in the School of Electrical and Computer Engineering, RMIT University. He received a B.Eng (Hons) and a M.Eng.Sc both from The University of Western Australia. His research interests include Intelligent Systems in Engineering, and Learning Technology.