PC-based Development Environments and a Unix-centric Curriculum: Some Practical Issues

Evan Golub
Department of Computer Science
University of Maryland
College Park, Maryland 20742-3255 USA
<egolub@acm.org>

Abstract

As computers have become a more common household item, computer science students are able to work at home rather than in campus labs. At institutions that have Unix-centric resources, students are able to use these home computers to connect to campus machines remotely. However, some students want to use a PC-based development environment rather than the ones available under Unix. Do they gain an advantage? Are there problems that they will encounter when they bring their program into the Unix environment?

1. Introduction

At the University of Maryland, we teach the introductory computer sequence using C++. Our students use g++ or c++ in Unix. The test projects using automated scripts on this platform. However, many students have access to other compilers, typically integrated development environments (IDEs), for their home machines and often ask if they can use those, rather than connecting (usually via dialup lines) to the Unix machines. Several highly self-motivated students had already reported success in using IDEs at home to develop and test their projects before transferring them to the Unix platform and performing final tests. I set forth to observe two major issues involved in allowing officially students to use these tools in their projects. These issues were:

(1) Would students have difficulties when porting their projects to Unix?
(2) Would students scores on written exams, where they would have no access to the IDEs, be affected? Additionally, I planned to see where students could use the IDE during traditional classroom presentations.

To perform this experiment, I needed to arrange for software to be available, as well as the creation and presentation of new lab material to introduce the various aspects of this environment. Microsoft was approached with a proposal of what I intended to do, and awarded a gift of funding, software, and computers. The computer science department contributed space and resources as well. With these, I established a computer lab on campus. There, we could show students how to use Visual C++ and how they could work on assignments at the campus with lab assistants available. We also provided students with free copies of Visual C++ to use at home. We chose Microsoft and Visual C++ due to their previously expressed interest in educational issues and my familiarity with the IDE as one in which students had often requested permission to use.

2. Preparation

The class selected for the experiment was the local Computer Science II course. Within this course, we determined two basic groups for this experiment. The first would not use anything other than the standard Unix resources previously used in the course; the other would have the option of using Visual C++. For the group given the option of using Visual C++, all students in that group would have several recitations that met in the computer lab to introduce the compiler and environment. We selected the groups based upon the section for which they were registered. Students in my sections would be the experimental group while students in the other lecturer s sections would receive the same material as previous semester s students had. Within the experimental group, programming journals and a simple yes/no questionnaire were used to help determine which students used Visual C++ while working on their projects. We created a series of recitation labs to introduce the basic Visual C++ environment as well as the debugging environment and features.

3. Statistical Observations from Fall 1999

We used two different techniques during the Fall 1999 semester to determine which students in the experimental group used Visual C++: a checkbox on the second exam and journals for each of the projects. As with any self-reporting situation, there is a question of reliability. For example, since we gave the second exam right after project 3 was due, the results of the checkbox on the exam and the journal for the project should have been the same. However, the results differed, as shown in Figure 1. Figure 2 shows the averages on the second exam and third project, groups based on whether they reported using Visual C++.

	Reported using Visual C++	Did not report using Visual C++	Whole Class
Exam Checkbox	75	83	158
Project Journal	53	105	158

Figure 1

	Reported using Visual C++	Did not report using Visual C++	Whole Class
Exam 2 (divided according to exam checkbox)	78.86	79.27	79.08
Project 3 (divided according to exam checkbox)	82.77	77.83	80.18
Project 3 (divided according to journal entries)	85.94	77.82	80.18

Figure 2

The noticeable difference in the average of the project 3 grades for those reporting the use of Visual C++ between the two reporting methods shows the delicateness of this type of analysis. I felt that the checkbox on the second exam was the more accurate representation, but provide both distributions here for completeness. We might attribute this difference to better students being more likely to report which compiler they used in their journals as they were instructed to do.

In addition to the averages, we generated ANOVA results (shown in Figure 3) using the ANOVA analysis tool in Microsoft Excel 97 with alpha set to 0.05. The null hypothesis in this would be that the students results were similar regardless of their use of Visual C++ while working on projects.

	F	P	F-critical value
Exam 2 (divided according to exam checkbox)	0.033457	0.855092	3.898435
Project 3 (divided according to exam checkbox)	2.31646	0.130036	3.901761
Project 3 (divided according to journal entries)	6.502738	0.011735	3.901761

Figure 3

Reviewed Papers

4. Lab Recitations in Fall 1999 and Spring 2000

In our Computer Science II course, there are two recitations scheduled each week. Teaching-assistants introduce some compiler specifics along with practice problems and some supplemental material. For this experiment, we decided to have some of these recitations help in a computer lab, and to have the Visual C++ IDE introduced. The first lab focused on the details of creating a project using Visual C++, importing an existing project to Visual C++ and exporting a project from Visual C++ to the Unix environment. The second and third labs focused on using the debugger to trace through programs both to find errors and to observe how and when things such as constructors and destructors are called.

We had to make some changes in how to scheduled and run the labs. There were three sections of the course in the experimental group. Since the hands-on lab was scheduled during a recitation period, the three sections were rotated through over a week and a half period. In the Fall 1999 semester, I taught each of the lab meetings rather than the teaching assistants. We did this primarily to facilitate dynamically altering the content of the labs based upon the reactions of the students.

There were approximately 60 students registered for each section of this course, and each lab was approximately two-thirds full. This is slightly higher than the average attendance for labs in general for this course, but it is impossible to say if it was due to increased interest, or the fact that they knew their instructor would be present. At the end of each lab, 10 to 15% of the students would usually stay to ask additional questions about the IDE. Based on student comments, I developed the impression that they were mostly happy to have the IDE presented to them, and that they were. Also, students were happy that they were not being mandated to use the IDE. Additionally, the graphical debugging environment was vastly preferred over text-based ones such as *gdb*. In previous terms when I attempted to do detailed debugging recitations, student opinions were very negative and consequently, typically limited discussion to how to use *gdb* to find out what line you core dumped on. However, students appeared more receptive to graphical debugging. Some also verbally acknowledged that they felt that they understood certain issues in parameter passing and class composition better after seeing things in motion via the debugging environment.

In the Spring 2000 semester, the initial plan was to have the teaching-assistants take over running the hands-on labs. For the first lab, I met with the teaching-assistants in advance and presented the lab material to them in the same manner as presented to the previous classes. Additionally, students made comments regarding the important points of each exercise in the lesson. We told each teaching-assistant to practice going through the material and to be prepared to present it during the scheduled lab time.

Each teaching-assistant taught their lab session, and I attended to observe. In all three cases, the labs did not go well. In the case of one lab, the teaching-assistant clearly did not review the material sufficiently and was not able to present the material correctly. It was the case that all three teaching-assistants did not spent any time thinking about what they were presenting, nor had they taken note of any of the comments made to them when they were shown the material. At this point, I had to decide either to intensify the training or to remove the teaching-assistants from the equation. I decided that I would return to presenting the labs. There were three factors to this decision. The first was the high rate of turnover in teaching assistants for this course. However, the decision was more strongly motivated by the issue that scheduling all of the recitations to cycle using one or two lab facilities would produce logistical problems. The final factor was that I was interested in a process that other institutions could easily apply. Both teaching-assistant training as well as lab room usage were things that I felt could hinder this. The result was that for the remaining hands-on labs, I did the presentations and began considering other ways in which I could present the material.

In the end, a workbook [3] seemed to be a good way to organize the material. I divided the material that I had presented in the labs into individual exercises for the workbook.

Additionally, to help encapsulate the individual points, I subdivided some of the material in the workbook to make it more amenable to a variety of learners. Examples that students had expressed favor towards, and had asked many what-if questions regarding were lengthened and explained in detail. In the Fall 2000 semester, rather than holding the hands-on labs, students were given the option of purchasing a copy of the workbook. Since this was a newly written work, I did not do a hard sell on it. Rather, I encouraged students that were more self-motivating to purchase it and asked them to contact me with anything that they felt was confusing in the workbook. While only about 10% of the enrolled students in my sections of the class purchased the workbook, several came by to complement it, and there was no negative comment. This was hoped for, as an excellent undergraduate student who had been in one of the experimental sections the previous year reviewed the original draft and was able to identify poorly worded or unclear sections.

5. Project Submissions on the Unix Machines

One of the major motivations of this experiment was to see whether there would be problems with students developing projects on a different platform than the one on which the projects would be tested. Consequently, it is important to discuss the information given to students and the results. Students were given directions regarding two aspects of working within Visual C++ specific to this issue.

First, we showed students how to turn off the Microsoft extensions to C++ by going to the project-settings dialog and checking the box to disable them. It was anticipated that with this option (specifically /Za) set, any programs that compiled and worked under Visual C++ would do so under g++ on the Unix machines as well. Second, we showed students how to achieve input and output redirection using debug mode and setting the program arguments (which are essentially the equivalent of command-line arguments). We also showed them how to use *windiff* to compare their output files to sample ones which are posted with sample input files. In this way, students would be able to test their projects in the same manner than they were accustomed to on the Unix machines, which is also the same way in which the projects are tested for grading.

Over the course of the Fall 1999 and Spring 2001 semesters, I found that very few students (fewer than 10) reported having problems on the Unix machines with a program that had worked correctly under Visual C++. In these cases, the problem always turned out to be that the program was reading memory that had been allocated but had not been initialized by the student s code. We saw an example of this several times in string comparison functions. This occurred when students wrote code that went beyond the final position of the string, into an un-initialized portion of the array. The problem appeared to be that Visual C++ created executables that would blank out memory while g++ would not. This in fact did turn out to be the case due to the default compiler settings in Visual C++ including the /GZ option which initializes all local variables not explicitly initialized by the program to 0xCC [6].

The effect this had upon the student code examples for string comparison was that if longer strings had been used in previous function calls, bits of those values would sometimes still be in the memory locations. A question that this raised was whether it would be best to remove this compiler switch from the list. Opinions on this vary. On the one hand, removing it would allow dirty memory to exist, as g++ allows. However, there is no reason to think that we would manage the memory in the same way, so between platforms, it would probably be different dirty memory. The result would be that a program might still work on one platform, but not the other.

6. The IDE in the Classroom

In a previous semester, I had attempted to use live demonstrations using g++ while remotely connected to the Unix machines. The results were disappointing. The students raised two concerns. First, they expressed that it was distracting to see switching between different telnet windows. Second, just as they had not liked using *gdb*, they did not like trying to watch a *gdb* session. However, it would still be beneficial to use intuition along with examples from the literature about engaging the students in general [7] in lower-level classes [5]. We could also use examples for using animations to teach the analysis of algorithms [2] and operating systems algorithms [4] indicated that this type of demonstration. We should note that other literature [8] indicates that this might not be the case. During these experimental semesters, the immediate goal was not to determine whether there were measurable benefits to this type of in-class demonstration. Rather, the goal was to see whether using an IDE such as Visual C++ could address both of the issues raised by students during the previous attempt.

I found that we could minimize or eliminate the primary issue of distraction and disassociation created by switching between multiple windows by using an IDE. The environment is specifically designed to integrate the editor, file manipulation, compilation and debugger into a single workspace. It was quick and easy to switch between different files as well as to compile and run the programs. When a compiler error was raised, it was easy to view both the error as well as the location in the code that it was raised on screen.

For the same reasons the graphical debugger appeared to be preferred by students when working on programs, it appeared to have been received well in class presentation. Students seemed very comfortable asking to place breakpoints on lines of code, or for code to be moved and altered. We can probably attribute this to the fact that we can insert breakpoints via direct manipulation of the code window. Additionally, the Visual C++ IDE allows for various visualizations of the relationship between classes. As an example, we can see the inheritance-based relationships between a class and the rest of the classes in a project. We can do this by right clicking on the class name under *ClassView* and

selecting to see derived classes or base classes relative to that class.

Again, this paper makes no claim as to there being measurable benefits from this approach. In fact, a comparison between the results of students in my sections and those who were not shows no statistical difference. However, it was good to see that students appeared more receptive to the more visually attractive environment than they had been to the text-based one.

7. Conclusions

From the results, it appears safe to conclude that students would not become dependant upon the tools that an IDE such as Visual C++ provides if they begin using it at this level. The exam and project results show that there is no significant effect of students using the software. The higher project scores among those who used Visual C++ were not significant, so it is not possible at this point to make any conclusions in that regard.

Although the recitation labs appeared to be successful, we feel that this method of introducing students to the compiler would not scale well. This is partially due to the experiences with the teaching assistants and partially due to the physical lab resources that would be required. Consequently, the lessons learned in those labs resulted in the creation of a workbook to introduce those concepts. The workbook addresses the means to set the compiler to be ANSI-compliant as well as how to redirect input from text files and output to text files.

There were some issues with projects working in Visual C++ but not on the Unix machines. However, in all of the cases observed, the problem was a subtle programming error that would have occurred and needed to be fixed even if the student had been doing all of their development on the Unix machines. The difference was that they would have detected the error at an earlier point. However, it should be noted that since the program was known to be mostly sound, the students were able to quickly identify where the problem was, and correct that problem.

We also observed that students appeared receptive to the use of the compiler during class. This was especially true during the portions of the class when we discussed inheritance and many what-if questions were raised and could be answered empirically using Visual C++.

Acknowledgements

Special thanks to the people at Microsoft and the University of Maryland with whom I have worked. These include (alphabetically): Craig Cumberland, Larry Davis, John Gannon, Laura Goyer, Dylan Greene, Kurt Messersmith, Susanne Peterson, Jandelyn Plane, Robert Rodgers, and John Spencer.

References

[1] Naps, T., et. al., An overview of visualization: its use and design. Report of the Working Group on Visualization. *Proceedings of the Conference on Integrating Technology into Computer Science Education* (1996), 192-200.

[2] Goodrich, M., Tamassia, R., Teaching the Analysis of Algorithms with Visual Proofs. *Proceedings of the 29th SIGCSE technical symposium on Computer Science Education* (1998), 207-211.

[3] Golub, E., *A Visual C++ Workbook*. 2000.

[4] Hartley, S., Animating Operating Systems Algorithms with XTANGO. *Proceedings of the 25th SIGCSE technical symposium on Computer Science Education* (1994), 344-348.

[5] Lewandowski, G., Computer Science Through the Eyes of Dead Monkeys: Learning Styles and Interaction is CS I. *Proceedings of the 29th SIGCSE technical symposium on Computer Science Education* (1998), 312-316.

[6] Microsoft, /GZ (Catch Release-Build Errors in Debug Build). <http://msdn.microsoft.com/library/devprods/vs6/visualc/vccore/vcre-fgz(catchrelease-builderrorsindebugbuild).htm>, 2000.

[7] Rodger, S., An Interactive Lecture Approach to Teaching Computer Science. *Proceedings of the 26th SIGCSE technical symposium on Computer Science Education* (1995), 278-282.

[8] Stasko, J., Badre, A., Lewis, C., Do Algorithm Animations Assist Learning? An Empirical Study and Analysis. *Conference proceedings on Human factors in computing systems* (1993), 61-66.