WHICH ENVIRONMENT IS MORE SUITABLE FOR NOVICE PROGRAMMERS: EDITOR/COMMAND LINE/CONSOLE ENVIRONMENT VS. INTEGRATED DEVELOPMENT ENVIRONMENT?

by

EDWARD CLAUDELL DILLON, JR.

A THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science
in the Department of Computer Science
in the Graduate School of
The University of Alabama

TUSCALOOSA, ALABAMA

2009

ABSTRACT

When novice programmers begin programming, they face many problems due to the lack of programming experience. Integrated Development Environments are used as a way to help novices become more effective at learning to program. The question is whether or not such an environment is more effective when compared to a command line/console environment. Therefore, this study tried to address this question by performing interviews with students who were using these environments.

This study was composed of two groups of undergraduate students who were taking courses in Computer Science. Group one consisted of students who were involved in a course sequence beginning with the Microsoft Visual Studio IDE, then a command line environment for the last course in the sequence. The second group started programming with a command line environment. Interviews were conducted with both groups in order to gain information about these environments. The information retrieved showed that the Microsoft Visual Studio IDE is favored based on the students' responses to the questions. However, there was not enough significant differences amongst the results to say that an IDE in general is better than a command line environment. It was the intent that this information provided not only background information but also served as potential foundational evidence for determining which environment may be more suitable for novice programmers to use for programming. This information will also be used as a basis for further research and studies in this area.

DEDICATION

This thesis is dedicated to the people who have stood beside me during my pursuit of higher education. In particular, my parents, Edward and Linda Dillon, Sr., my sisters, Rosalind and Sheletha, my sweet heart, Michaela, and all of my friends who believed in my capabilities of completing this manuscript.

LIST OF ABBREVIATIONS AND SYMBOLS

α	Alpha value or type-1 error; it represents the probability that a statistical test will give a false positive error
df	Degrees of freedom
IDE	Integrated Development Environment
p	P-Value; it represents the actual alpha value that arises from a statistical test
(s)	One or more objects that may be available
t	Better known as a t-test; it represents a statistical hypothesis test that follows a plotted distribution if the null hypothesis is true
χ^2	Better known as a chi-square; it represents a distribution that occurs within the parameter of a population
~	Roughly equal to
%	Percent sign
<	Less than
#	Numerical symbol

ACKNOWLEDGEMENTS

I would like to thank my advisor and committee chair, Dr. Marcus Brown, for encouraging me to do research in this area. Also, I would like to thank one of my undergraduate professors, Dr. Dawn Wilkins, from my alma mater, the University of Mississippi, for introducing me to related research in this area concerning novice programmers and Integrated Development Environments. To those who were involved in funding me to work on my Masters through the Bridge to the Doctorate program, Dr. Viola Acoff and Dr. Louis Dale, thank you very much. I would also like to thank my committee members, Dr. Jeffrey Carver, Dr. Randy Smith, and Dr. Cecil Robinson for agreeing to be on the committee and also providing helpful comments and questions to improve my research. Finally, I would like to thank all of my fellow colleagues in my department as well as other departments who served as motivators and great friends during my endeavor.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

1. INTRODUCTION

In the field of Computer Science, first year students are expected to learn and master the skill of programming in order to be successful. However, lack of basic programming skills has intimidated many students and in many cases driven some of them away from Computer Science. In an attempt to improve confidence in these students, many Computer Science departments have adopted Integrated Development Environments (IDEs), instead of command line environments, because of their user-friendly design. The main question or hypothesis was whether *an IDE is actually a better learning environment compared to a command Line/console for novice programmers to use.* The intent of this study was to explore which environment was potentially better suited for introductory programming courses. This study was conducted by comparing an IDE, Microsoft Development Studio, used as a teaching environment to a Linux programmers' editor (vi/vim) and command line interaction. The goal was to draw preliminary conclusions about the relative advantages and disadvantages of IDEs and command line environments as introductory teaching environments.

1.1. Thesis Outline

A literature review is provided in Chapter 2. It discusses the issues novice programmers face while programming, the Integrated Development Environment and the editor/command line/console environment, and the characteristics of the Python and C++ language. Chapter 3 gives an outline of the study performed in which interviews and surveys were conducted with students who have used one or both programming environments. Chapter 4 shows the results of this study in the form of qualitative and quantitative data. Chapter 5 provides an analysis and discussion about the data extracted from these interviews. Chapter 6 gives potential threats to validity that are seen from this study. Chapter 7 lays out the future work for this study while Chapter 8 gives the conclusion.

2. LITERATURE REVIEW

This literature review provides a background composed of various studies and research related to this study. It consists of four topics: *Novice and their Issues with Programming, Integrated Development Environments, Editor/Command Line/Console Environments,* and *Python vs. C++*. The first topic talks about problems that novices typically face when learning how to program. This topic is broken down into six components: *Novice Behavior in Learning the Fundamentals of Programming, Mental Representation of Novices vs. Expert Programmers, Expected Programming Skills that Novices are to Develop, Identifying Common Programming Errors that Novices Tend to Make, Novice Behavior in Other Programming Languages,* and *Past Indications of Programming Errors made by Novices*. It is important to understand other possible issues that could be a factor for novices learning to program in addition to studying suitable learning environments that could be helpful for them.

The topic concerning Integrated Development Environments discusses their functionality as well as advantages. The Editor/Command Line/Console topic focuses on the behavior of the command line environment and its capabilities. The final topic, Python vs. C++, expounds on the characteristics of both languages. Since both languages are involved in the study, there is background information provided for both. These languages are also compared together in order to show relative advantages and disadvantages of each. This topic consists of three components: *Characteristics of Python, Characteristics of C++* and *Comparing Python and C++*.

2.1. Novices and their Issues with Programming

2.1.1. Ideal Problems that Novices Encounter when Programming

Many beginning computer science majors often struggle with learning the fundamentals of programming and problem solving techniques. A reason that they have a difficult time is because they do not understand the errors that they retrieve from the compiler. For example, some argue that the error messages are so “cryptic” to programmers that they have a difficult time just trying to figure out the errors that they made [1]. Another critical factor is not being able to comprehend a program. This is important for many reasons: it allows programmers to be more effective at completing other tasks in a program [3], it is very helpful for novices to be successful at debugging a program [4, 5, 6], and it is crucial for them to be able to extract necessary information from a snippet of code in order to make any intended modifications to a program [7].

2.1.2. Mental Representation of Novices vs Expert Programmers

In many cases, novice programmers are not thinking in the same way as expert programmers when dealing with code; their mental representation on how to approach a program differs. When Adelson compared novice and expert behavior, she concluded that novices tend to develop a mental representation that consists of concrete information, while experts develop a mental model consisting of functional information [8]. Wiedenbeck and Fix listed five abstract

characteristics of an expert's mental representation: being hierarchical and multi-layered in their thinking process, explicit mappings between different layers, recognition of basic programming patterns, being well connected internally, and being well grounded in the program text [3]. To explore these characteristics, these researchers performed a study where twenty novices and twenty expert programmers studied a 135 line Pascal program and were tested on their comprehension. There were eleven questions asked covering all five abstract characteristics. The experts scored higher on all five characteristics. This result suggests that developing these five abstract characteristics should aid a novice programmer's success in programming. The relevance between novices mastering these abstract characteristics for programming and the programming environments is based on which environment can be helpful in aiding these programmers with developing such characteristics.

2.1.3. Expected Programming Skills that Novices are to Develop

It is nearly a universal requirement for first year computer science majors to develop the knowledge of problem solving so that each student will learn how to generate correct executable programs and maintain these skills by the completion of their introductory courses. A multi-national study of first-year CS students' programming skills showed that first year students needed to learn and utilize the following procedure while solving problems through computation: *abstract the problem from its description, generate sub-problems, transform sub-problems into sub-solutions, re-compose the sub-solutions into a working program,* and *evaluate and iterate*. Unfortunately, this expectation is not being met for the most part. To explain why this is so, the authors studied 217 first year students at several universities by creating three programming exercises for them to solve. Each exercise dealt with arithmetic expressions with varying

difficulties for each exercise based on execution, verification, validation, and style. The overall average score was 22.9 out of a possible 110 points, meaning that first year students still lack the needed programming skills at the completion of their introductory courses [2]. Therefore, there is a concern of whether first year students can master these problem solving skills and can a programming environment be helpful with this.

2.1.4. Identifying Common Programming Errors that Novices Tend to Make

There are generally two kinds of errors made while programming: syntax and semantic errors. Syntax errors are errors in a sequence of characters or tokens written in a particular programming language. These errors are typically made by the programmer who violates the grammar of the language and are detected by the compiler. Semantic errors (logic errors) are bugs that occur in a program that are not seen as errors by the compiler in many cases, but cause the program to run incorrectly based on poor logic.

Novice programmers are very likely to produce syntax errors in any programming language. In Java for example, some of the common syntax errors that are made by young programmers are [1, 9, 10]:

- undefined variable
- missing a semicolon
- misuse of curly braces, parenthesis, and/or quotations
- inappropriate file name
- failure to initialize a variable before trying to use it
- confusing the assignment operator = with the comparison operator = =
- invoking methods with improper arguments
- misusing the greater than or equal to and the less than or equal to operators (>= and <=), and
- using keywords as method names and/or variable names.

Here are some messages that a Java compiler would show on a syntax error [9]:

- cannot resolve symbol
- illegal start of expression
- class or interface expected
- incompatible types
- not a statement, and
- missing return.

Some semantic/logical errors made are [1]:

- invoking class method(s) on an object directly
- improper casting of a variable
- using a non-void method as a void method when a return value is required
- failure to use methods with parameters correctly, and
- class is considered abstract due to missing function(s).

In addition, there were resources that were used to generate a common list of errors made by novices in Java which came from viable sources such as computer science professors throughout the nation, and the Special Interest Group on Computer Science Education (SIGCSE), as well as teaching assistants, professors from the researchers' respective universities or academies, and actual observations done on the students' performance on assignments [1], [9], and [10]. However, one paper expounds on the discrepancies between what the instructors noticed as common errors and the actual errors being made by the students [9]. This paper shows a study performed at the United States Military Academy where all freshmen are required to take an introductory programming course. In this course, a Java IDE is used to store information about errors detected in compilation and each error is stored in a database. Comparing the identification of errors made by the faculty members to the errors detected from the experiment, the results indicated that the instructors were only 50% correct about what they saw as common

errors and the actual errors that were being made. Therefore, questions remain open about the discrepancy gap between what instructors see and the actual errors generated. Is this discrepancy common in classrooms? If so, can this gap be narrowed or removed entirely? Would eliminating the discrepancy gap improve the success for novice programmers in Java or any other language? Spohrer and Soloway said that what educators "knew" about novice programmers was primarily a "folklore," meaning what they saw as typical errors made by novices were based on their own experiences when they first started programming [12, 14]. These two authors stated "the more we know about what students know, the better we can teach them" [12]. This discrepancy could be greatly reduced if this philosophy is adopted by each instructor in introductory programming courses.

Many integrated development environments have many benefits to help the programmer from making the typical errors seen in this section. For example, some IDEs are able to underline errors that are being made while the programmer is typing. This can potentially help a user, especially a novice, to learn and understand those errors and what to do in the future to avoid them. Another benefit that IDEs offer to prevent common syntax errors is syntax highlighting where keywords, comments, methods, functions, etc. are highlighted in particular as a way for a programmer to avoid common errors such as *cannot resolve symbol*, *illegal start of expression*, or *class or interface expected*. Some IDEs are also able to provide popup windows with possible methods while the programmer is typing to help them figure out the appropriate method that is needed for their program without using invalid methods or *invoking methods with improper arguments*.

2.1.5. Novice Behavior in Other Programming Languages

McIver [14] performed a study of 26 first year students who had never done any programming and were about to take an introductory computer science course. These students were placed into two separate groups where one group learned to program in LOGO and the other group programmed in GRAIL. Both groups were given the same exercises to perform in their respective languages. Even though both groups exhibited different kinds of errors while programming, the error tendencies for the group using GRAIL were much lower than the group using LOGO. The average number of syntax errors for LOGO was 31.08 while GRAIL only had an average of 13.62 *($p < 0.01$)*. The means for logic errors were much closer but still showed that the group using LOGO made more errors (17.77) than GRAIL (9.54, $p < 0.01$). McIver noted that three students who programmed in GRAIL went on to solve advanced problems beyond the course syllabus while there were no LOGO students who decided to do the same. This study addresses the notion of whether or not certain languages are better than others for teaching introductory programming classes, which has also been a subject of ongoing research, debate, and study. Similarly, the study at hand involves two different languages: C++ and Python.

2.1.6. Past Indications of Programming Errors made by Novices

This problem of identifying common errors by novice programmers has been researched for some time now. For example, in 1976 a study was done at Auburn University by Chabert and Higginbotham investigating errors made by novices using assembly language [11]. In this study, two classes were studied to see the kinds of errors that would be made while programming. Generally both courses were taken by students who had no previous experience with assembly languages. The kinds of errors that were found were undefined symbols, addressability errors,

invalid delimiters, invalid syntax, incorrect specifications of register or mask field, invalid op codes, undefined operation codes, invalid displacement, and end of data on input [11]. Even though there is a significant amount of difference between assembly language and currently used languages like Java, some of the errors encountered by novices then are identical to the errors seen now.

Another early study of novices' errors or misconceptions comes from Bayman and Mayer in 1983 at the University of California-Santa Barbara [13]. This study allowed 30 undergraduate students who had no experience with programming to learn the BASIC language on an Apple II computer. After the completion of the course, the students were given a test of nine statements. Their task was to write out the steps that the computer would perform for each statement. Overall, the students had trouble with: input, flow of control, assignments, and output [13].

2.2. Integrated Development Environments

Since programming is a critical expectation in the field of Computer Science [2], it is necessary to study effective ways to help novice programmers improve their skills through finding more suitable programming environments. There are several specialized environments known as Integrated Development Environments (IDEs) that have been created to aid programmers in developing code. A typical IDE contains a text editor and a built-in compiler or interpreter. More advanced IDEs may have a built-in debugger [22]. Many IDEs help reduce the errors generated by programmers through text editors that analyze the syntax of the code as it is being written. These environments are also helpful in formatting the code to look neat automatically. When compiling or executing a program through the use of an IDE, the user may click on a “compile” or “run” button that is typically provided as a menu option rather than using a command line interface [23]. In many cases, an integrated development environment makes programming much easier for users. Typically, integrated development environments come in two different categories: pedagogical or professional.

Pedagogical IDEs are integrated development environments built specifically for beginner programmers to help them learn to program. These kinds of environments are simple because they lack many features. For example, Dr. Java, a pedagogical Java IDE, has restrictions on its behavior in order to keep its interface very simple. This tool does not offer plug-in architecture, the ability to complete code for the programmer, access for navigating, or program refactoring options [23]. The main reason for such IDEs being this simple is so that these

environments can appeal to the novice programmer. Typically, pedagogical IDEs are considered "half-strength" because they are intended to be used for smaller projects [24]. These environments generally attempt to reduce the underlying complexities of programming from inexperienced programmers by its great ease of use. In addition, pedagogical IDEs are created to meet the specific requirements seen in introductory programming courses. For example, BlueJ and Dr. Java, pedagogical IDEs for Java, are much smaller than professional Java IDEs such as Eclipse and NetBeans.

Professional IDEs on the other hand are rich in features, but their complexity may be too much for a novice programmer to handle. They are considered "full-strength" because of their many features [24]. Despite their complex behavior, these IDEs offer programmers the opportunity to write code more quickly and with greater quality. An example of an IDE with such available traits is Eclipse. This Java IDE is productive as well as efficient for the development of Java programs. Some of the benefits that Eclipse provides to the programmer are [25]:

- syntax highlighting for keywords
- code auto-completion for variables and predefined methods
- code assistant that gives method hints
- packages importing
- wizards to eliminate manual repeated typing for classes, methods, constructors, etc.
- a package-class hierarchy view and a class fields and methods outline view (makes it easier for the programmer to navigate without dealing with implementation details)
- javadoc documents
- user preference on indentation, color, fonts, general project/class comments, etc.

In some cases, professional IDEs can enable plug-ins from other sources that will cause a modification to these environments in order to make them more suitable to the novice user.

Pedagogical systems like Dr. Java and Gild have either been plugged into or compared to the Eclipse environment in order to cater to novice programmers [26, 23].

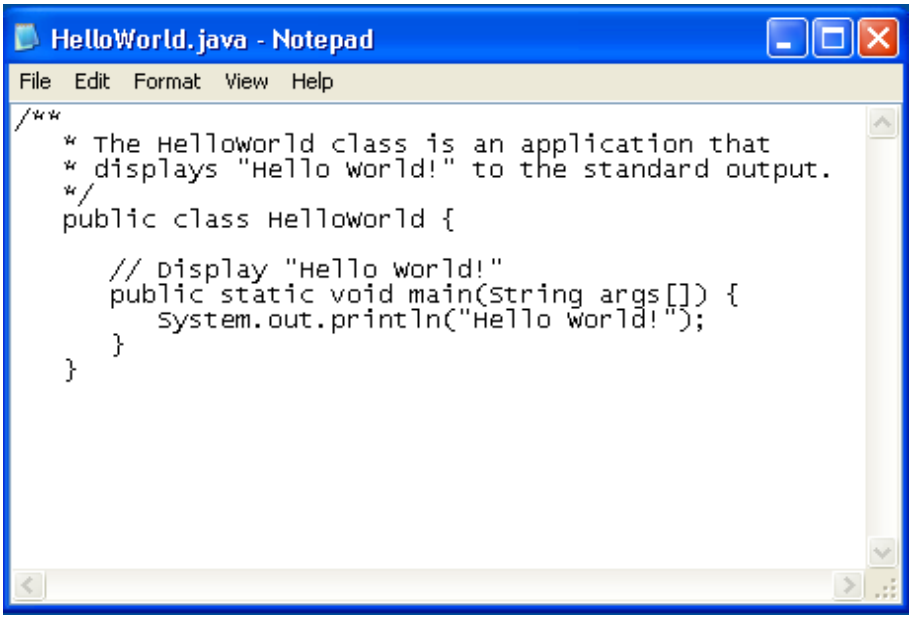
2.3. Editor/Command Line/Console Environments

Besides using an IDE, an alternative environment is the command line. A key difference in using such an environment versus an IDE is that the programmer may have to use an independent editor to write a program then compile and execute the program from the command line. For example, if a programmer is programming in Java on the Windows platform, Notepad and other editors like UltraEdit [28] are available for writing applications and the command prompt is used as the command line to compile and execute the application. On Linux and Unix operating systems, the command line is considered more of a shell or console that behaves as an interpreter because it retrieves commands from the user in order to perform specific tasks [27]. This behavior reveals another difference in the command line/console which is that it does not provide graphical operations or a Windows-Icon-Mouse-Pointer (WIMP) interface [27] for the programmer like IDEs do. This style of programming forces a programmer to develop an understanding of using the command line by providing inputs in order to achieve a desired output.

To provide an example of how to program using the command line environment, let's say that a programmer is writing a simple Java program called *HelloWorld* using the Windows platform. The programmer uses Notepad to write the application and the Command Prompt to compile and eventually execute the program to achieve the output. In order to compile the HelloWorld application, the programmer will have to type the command: *javac HelloWorld.java*

and to execute: *java HelloWorld*. Figures 1 and 2 illustrate how programming differs between the command line environment and an IDE:

HelloWorld Application



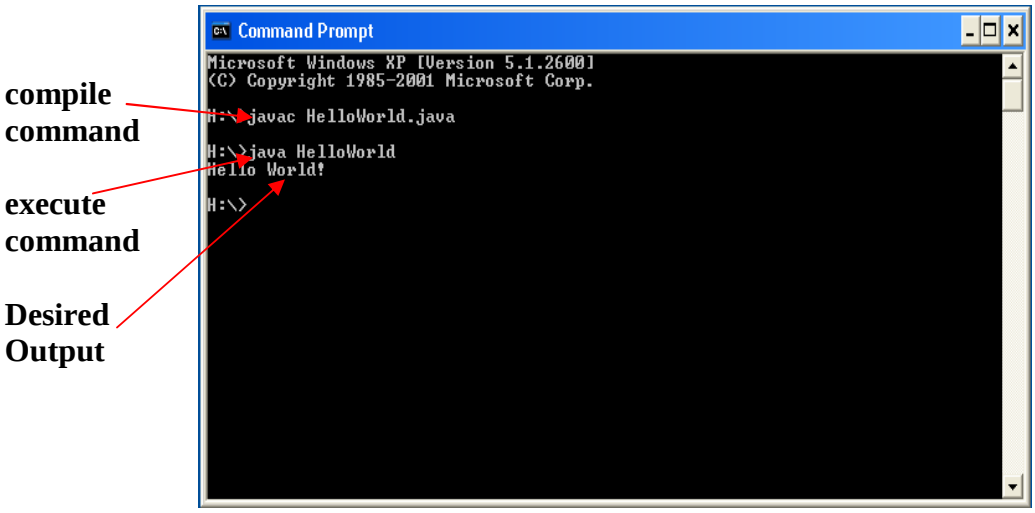


Figure 1: Command Line Environment (using Windows platform)

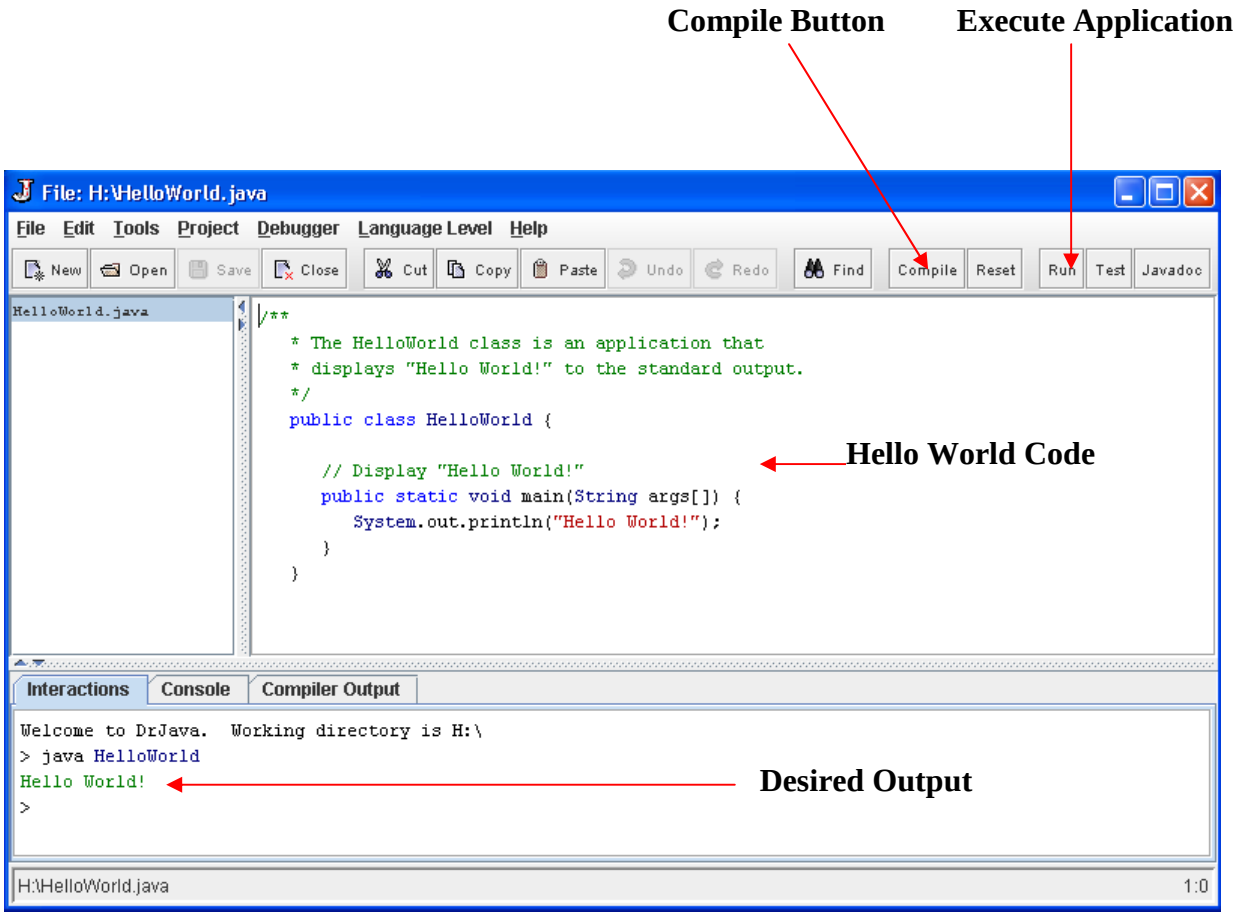


Figure 2: Hello World Application using an Integrated Development Environment (JCreator)

There are cases where the command line/console environment can be more useful than the IDE. For example, the command line/console environment is very useful when the platform being used has no way of displaying a graphical user interface. It is also beneficial for those applications where an advanced graphical user interface is not necessary. When dealing with web-related software, the command line/console environment is also great to use because many of the back ends to the web software are a collection of command line utilities [29].

2.4. Python vs C++

2.4.1. Characteristics of Python

The language of Python was developed by Guido van Rossum at the National Institute for Mathematics and Computer Science in the Netherlands during the late 1980s. This language in many cases is used as a scripting language and is similar to other scripting languages such as Perl, Tcl, and Ruby. The characteristics that make Python different from such scripting languages are [20]:

- clear and readable syntax
- object orientation that is intuitive
- the procedural code is naturally expressed
- full modularity
- has exception-based error handling
- high level and dynamic data types
- extensive standard libraries
- extensions and modules that can be written in C, C++, and Java
- can be embedded in applications in the form of a scripting language

General benefits that Python has to offer to users are its simplicity and safe usage. It can be utilized with languages that are object-oriented, and can be fun to use. Python's simple format makes it easier for users to learn. Python is also safe to use because it provides a garbage collection to prevent memory leaks or dangling pointers thus making it unlikely for the user to get a logical segmentation error. Even though Python is mainly a scripting language, it can also take on the role of being object-oriented by allowing typical object-oriented coding schemes

[18]. In addition, Python offers an interactive interpreter. This interpreter can allow students to execute (interpret) fragments of a program. This helps students be more effective in pointing out errors or possible bugs that are in the program because it will execute until an error is detected. This is better for the novice than having a program that will not compile at all [16].

Although Python is a powerful scripting language, along with other scripting languages, it has some drawbacks. Generally, scripting languages are not as efficient as system languages at compiling and executing programs because they tend to take longer in processing. Another concern is that students will most likely still have to learn a system language because scripting languages are mainly complementary and are not broad enough to replace system languages [16]. Python's benefits are strong enough for it to be used in introductory programming classes, but it may still lack the tools needed to be used in an advanced programming course.

Despite some of its drawbacks, the popularity of Python is increasing and it is widely used. It's a simple but powerful language that can be used as a scripting or an object-oriented language. Python can also complement other languages by being plugged into languages that are system based. Example platforms that users can run Python include: Windows, MacOS, Linux, Unix, and BeOS [18].

2.4.2. Characteristics of C++

The language of C++ was developed in 1979 by Bjarne Stroustrup as an enhancement to the C language [20]. Stroustrup calls the language "a better C" because it is built to provide better support to the C language and its programming styles [21]. Primarily, C++ takes on various behaviors that C encapsulates. Characteristics that are seen in the C++ language are [21]:

- checking and coercion of argument types
- inline functions
- constants that are scoped and typed
- number of arguments that vary
- declarations as statements
- data abstraction support
- initialization and cleanup
- free store operators
- operator overloading
- coercions
- derived classes
- virtual functions
- visibility control

The benefits and advantages that C++ gives to programmers include user defined types, encapsulation, separation of interface from implementation, composition, inheritance, and run-time polymorphism [15]. C++ contains all the necessary components to support object-oriented programming. A significant advantage that C++ offers is efficiency; it is even more efficient than its predecessor, C, in many cases.

Since C++ heavily depends on object-oriented behavior, it is critically important that a programmer is well acquainted with object oriented programming in order to be productive at this language [15]. For a novice to be great at the C++ language, one major task that he/she must understand is that C++ is not a “pure” object-oriented language because it allows data and code to be used outside its class structure [15]. Another obstacle is knowing how to manipulate constructor and destructor functions. Even experienced programmers are sometimes tripped up by the functions being invoked without any explicit function calls [15]. A third issue is adjusting

to the idea that C++, in many cases, does not indicate the appropriate classes or other mechanisms that need to be externally included [15]. A broader factor that novices need to be concerned about is how intimidating C++ looks. Its appearance alone has the tendency to discourage many students [17]. Therefore, C++ is a very rich and efficient object-oriented programming language but its complexity may be an issue for students in introductory programming courses.

2.4.3. Comparing Python and C++

When comparing the two languages, C++ is more object-oriented while Python is considered more interpreted and interactive. Generally, C++ has been labeled as one of the more complex languages to learn because of its formality. Python on the other hand uses simplicity and its programming format looks more like the English language. Python also uses simpler syntax and semantics that allow students to create more advanced programs with less implementation effort [18]. It has been a suggested that Python has the potential to be a great language for introductory classes [16, 18].

A study was conducted in the School of Computing at the University of Leeds using Python in an introductory programming course during the 2002 - 2003 school year [16]. Three hundred students from a diverse range of degree programs along with different levels of experience took this twenty week course. The students learned Python during the first seven weeks and moved on to C++ for the following seven weeks. The results from this study showed that while learning Python most students were able to write simple programs pretty quickly. Many of them were also inspired to experiment with other features within this language. On the

other hand, the students felt that C++ was more complicated. More studies were performed on a sample group of students from this population, based on learning the material covered in C++ with Python. It became apparent that it was easier to learn Python's version of the material because less amount of teaching was needed. The students' attitude at the end of this course indicated that they favored the use of Python more than C++ [16]. Generally, scripting languages are more favorable to programmers than languages that are primarily object-oriented because of the significantly smaller amount of code needed to develop projects. Due to its high-level syntax and the ability to use a smaller amount of code in programs, errors are less likely to occur in a scripting language which is a positive benefit for novices.

2.5. Summary

This chapter provided background information about the behavior of novices in programming, the general usage of the IDE and command line environment, and the advantages/disadvantages of using Python and C++ languages. Novices in some cases struggle with understanding the fundamentals of programming. One particular reason is that their mental model for writing a program is different from that of an expert. This and other factors show why novices make various kinds of errors that are normally avoided by experienced programmers.

When using certain environments to write a program, a command line environment requires a programmer to use various commands as part of the procedure for programming while using a compiler independent of the editor. Integrated development environments provide user friendly tools to help programmers produce code in an efficient manner. Pedagogical IDEs in particular help novices with their ability to program in the language that is being used. Some languages are simple to use when compared to others. The scripting language, Python, uses simple syntax that is close to the English language. C++ on the other hand is more object oriented and in many cases is considered one of the more complex languages to learn because of its format.

3. STUDY

This chapter discusses the interviews and surveys done to obtain information from using both environments. It was the intent that this study would give more insight about these environments and set the background for further research in the future. The Computer Science Department has decided to change the introductory sequence for CS majors. Originally, the students were required to take the course sequence: CS 114, 124, and 325. CS 114 and 124 taught the C++ language using the IDE, Microsoft Visual Studio. CS 325 also taught the Java language but with a command line environment on the Linux platform. During the Spring 2009 semester, the department offered CS 150, which is the first course of the new sequence. This course taught the Python language with a command line environment on the Linux platform. This study allowed for interviews to be conducted with a sample of students who either have completed or are about to complete the original course sequence along with surveys for students who were enrolled in the new introductory course. The purpose for this information was to attempt to address the question of whether one environment helped students with grasping basic programming skills over the other.

3.1. Interviews: Original Sequence (CS 114, 124, and 325)

An email was issued out to the students who were taking CS 325 and/or 357 with no incentive being involved. There were ten students who agreed to participate in an interview which lasted fifteen to twenty minutes on average; one of these students was interviewed differently from the other students because this particular student also served as a mentor for the students in the new sequence. It was expected that this student would most likely have further insights about these environments than the other students. Therefore, some of the results seen in Chapter 4 will only consist of nine responses. These interviews took place in Houser Hall.

The students were asked various questions concerning their experience with using both an IDE and a command line. As stated earlier, these students learned the C++ language on the Microsoft Visual Studio environment. After the completion of the first two courses, they then begin learning Java in a command line environment on the Linux platform. It was the intent that these questions would successfully provide information about the advantages and disadvantages of both environments. The questions used for these interviews can be seen in Appendix A.

3.1.1. Participants: Original Sequence (CS 114, 124, and 325)

The participants in these interviews consisted of students who were currently taking CS 325 (students who were about to complete the course sequence) or were taking CS 357 (students who have finished the course sequence); there were a couple of students who were taking both courses in the same semester. These students were mainly Computer Science majors with the exception of a couple of students. Table 1 shows demographic information about each of the participants. It was the intent to get a well diverse group of students to participate in these interviews. However, most of the respondents were Caucasian male students, a sample that is fairly representative of the student population in those classes.

Table 1: Demographic Information about the CS 114 Students

Student	Gender	Age	Ethnic Background	Major	Current CS class
ID #1	Male	21	Caucasian	CS	CS 357
ID #2	Male	22	Caucasian	CS	CS 325 and 357
ID #3	Male	19	Caucasian	CS	CS 357
ID #4	Male	19	Caucasian	Finance	CS 325
ID #5	Male	20	Caucasian	CS	CS 325 and 357
ID #6	Female	20	African American	CS	CS 325
ID #7	Female	21	Caucasian	CS	CS 357
ID #8	Male	23	Caucasian	MIS	CS 325
ID #9	Male	27	Caucasian	CS/Mathematics	CS 325
ID #10	Male	20	Caucasian	CS	CS 357

3.2. Discussion and Survey: First Course of New Sequence (CS 150)

For the students in CS 150, a focus group discussion was conducted. There were a total of thirteen students who participated in this discussion. One of the professors in the department, who did not teach this course, was the primary facilitator for the discussion which took place in room 112 of Houser Hall. The questions were related to the format of the course along with the students' experience in using the command line environment. The discussion lasted for about two hours. At the end of the discussion, the students received a questionnaire that asked them to provide more detailed information about the command line environment in addition to learning the Python language. The information that these students provided on the questionnaire represents the data shown throughout Chapter 4. This questionnaire is provided in Appendix B.

3.2.1. Participants: First Course of New Sequence (CS 150)

The participants were from a variety of fields: Computer Science, Mathematics/Math Education, Management Information Systems, Electrical and Computing Engineering, International Business, and Biology. This group was very diverse; there were seven males and six females (eleven Caucasian and two Non-Caucasian students). Table 2 provides demographic information about the thirteen participants involved in the focus group. (Note: a few students did not provide all their background information, therefore that information is noted as N/A in the table).

Table 2: Demographic Information about the CS 150 Students

Student	Gender	Age	Ethnic Background	Major	Current CS class
ID #11	Male	19	N/A	N/A	CS 150
ID #12	Female	N/A	Caucasian	MIS	CS 150
ID #13	Female	22	Caucasian	Biology	CS 150
ID #14	Male	19	Caucasian	International Business	CS 150
ID #15	Male	21	Caucasian	MIS	CS 150
ID #16	Female	20	Caucasian	MIS	CS 150
ID #17	Female	21	Caucasian	CS	CS 150
ID #18	Female	22	Caucasian	Math Education	CS 150
ID #19	Female	19	N/A	MIS	CS 150
ID #20	Male	19	N/A	ECE	CS 150
ID #21	Male	18	Caucasian	MIS	CS 150
ID #22	Male	20	Caucasian	Mathematics	CS 150
ID #23	Male	23	Caucasian	MIS	CS 150

4. RESULTS

This chapter looks at the data retrieved from the interviews and surveys conducted on students from the original course sequence and CS 150. The data provided throughout this chapter is analyzed either qualitatively or quantitatively. Section 4.1 discusses any programming experience that both groups of students possess prior to taking their introductory course. The next section looks at the students' initial response when being exposed to these environments. Section 4.3 focuses on the level of comfort each student had with their respective environment, which also consist of a subsection that talks about the amount of time it took for them to become comfortable with the environment. The fourth and fifth section compares the easiest as well as hardest attributes about Microsoft Visual Studio and the command line in Linux respectively. Section 4.6 gives statistical ratings about the programming languages used for these environments. The seventh section mainly focuses on the students who took the original course sequence since they were exposed to both environments. This section consists of three subsections that go into detail about: the easiest/hardest things about their transition from one environment to the other, which environment is their favorite to use, and which environment is the most efficient for programming. The final section, 4.8, looks at information given by one student who not only has taken the original sequence but also served as a student mentor for CS 150 students.

4.1. Programming Experience

This section provides information about whether the both groups of students had programming experience prior to taking their respective introductory courses. Out of the ten participants who took CS 114, six of them (60%) actually had prior experience in programming. The students in CS 150 had seven (~54%) to have prior experience with programming. Figure 3 shows a bar graph of this information. After a chi-square analysis was performed on these results where the calculated $\chi^2 = 0.205$, $df = 1$, $\alpha = 0.05$, and the critical value is 3.481, the distribution between these samples were not significantly different.

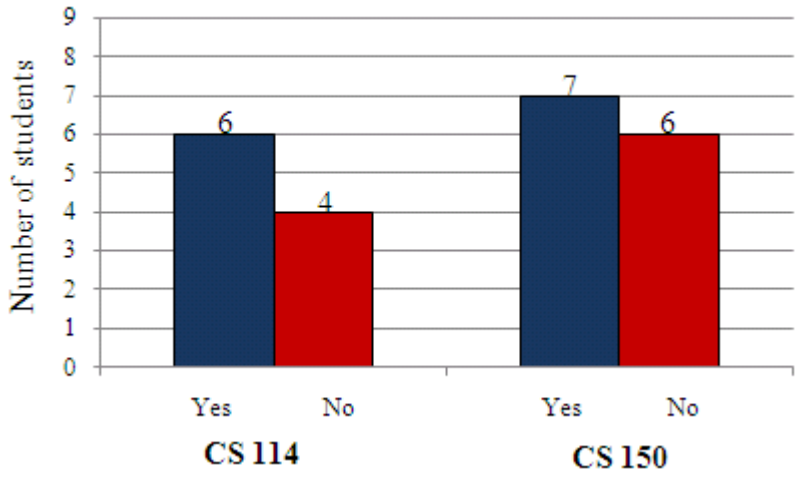


Figure 3: Programming Experience Prior to CS 114/CS 150

For the six students who took CS 114 who had prior experience in programming, these environments and languages consisted of: TI Calculator, C++ (two students), Java, Visual Basic, C and Pocket C; the seven students from CS 150 had experience with: Sway (two students), Java(JGrasp), Html, JavaScript, C++ (three students), Python, and TI Basic.

4.2. Initial Response to Environments

These students were asked about their initial responses to their environment. For the nine students from CS 114 who provided a response, seven (~78%) had a positive response to Microsoft Visual Studio while two (~22%) had either a fair or negative response. Out of the thirteen respondents for CS 150, six students (~46%) had a positive response to the command line while the remaining seven (~54%) had either a fair or negative response. There was only one student from the CS 150 sample who was a Computer Science major. This student in particular was one of the seven that gave a fair/negative initial response about using the command line environment. Figure 4 represents the responses made by the students for their respective environment. After a chi-square analysis was performed on these results where the calculated $\chi^2 = 7.66$, $df = 1$, $\alpha = 0.05$, and the critical value is 3.841, there was a significant difference between the distribution of both samples.

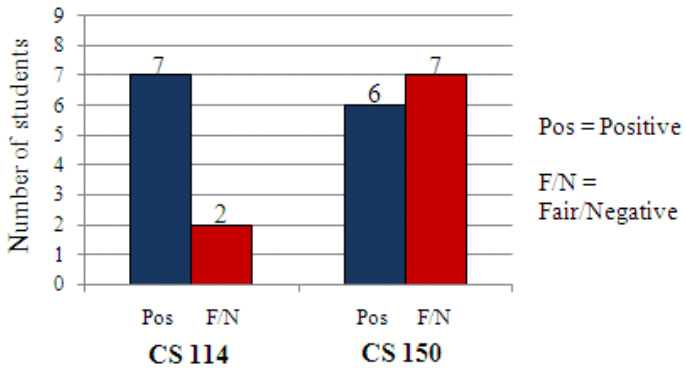


Figure 4: Initial Response from the Environments

4.3. Level of Comfort with the Environments

The level of comfort the students had with their respective environment was defined by the following categories: *not comfortable, fairly comfortable,* or *very comfortable*. There were cases where some of the students literally stated that they were “not comfortable,” “very comfortable,” etc. Other students’ responses were interpreted and categorized by the researcher

The students who took CS 114 were either very comfortable with using Microsoft Visual Studio or fairly comfortable. Seven out of the ten were very comfortable while the remaining three were fairly comfortable. Five students in CS 150 were very comfortable with the command line environment in Linux; six were fairly comfortable and two were uncomfortable. Figure 5 displays these results. After a chi-square analysis was performed on these results where the calculated $\chi^2 = 4.85$, $df = 2$, $\alpha = 0.05$ with the critical value being 5.991, the distribution between both groups were not significantly different.

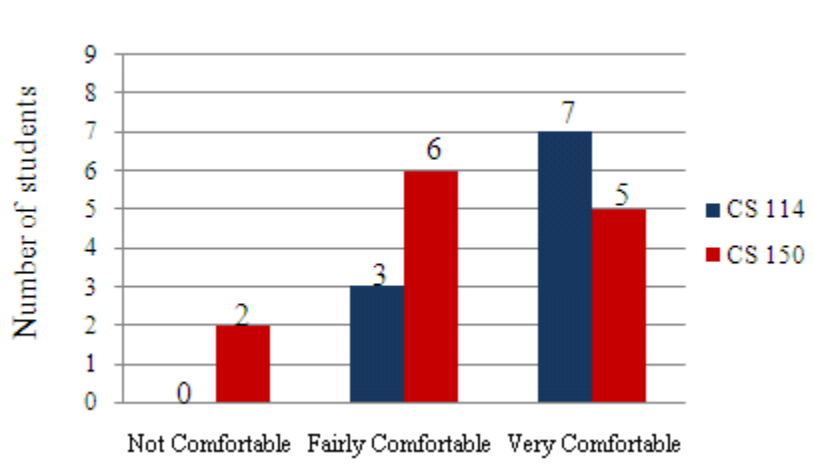


Figure 5: Students’ Level of Comfort with Their Respective Environment

A proposed threat to validity was possible from these results. The students who took CS 114 were now taking higher-level courses and had already gained further experience with using Microsoft Visual Studio (particularly in CS 124). After multiple semesters of using this environment, it seems obvious that these students should be fairly comfortable with using Microsoft Visual Studio.

4.3.1. Length of Time it took to get comfortable with the environments

To measure the amount of time it took both groups to become comfortable with their respective environment, the responses was placed into either of the two categories: *one month or less* or *more than one month*. Seven out of nine students from the original sequence became comfortable with Microsoft Visual Studio within a month's period. Their actual responses were:

- The first project (two students)
- A couple of weeks
- After understanding how to compile and execute a program
- After one month (three students)

The remaining two students fell into the category of more than one month because they became comfortable either *halfway through the semester* or *toward the end of the semester* (this student took CS 114 during the summer).

The information from the students in CS 150 showed that nine out of the thirteen students became comfortable with the command line environment within a month's period. These students stated that the amount of time it took for them to learn and understand the environment was:

- One to three weeks (five students)
- Not long at all (three students)
- After the second class

The remaining four on the other hand were placed into the more than one month category. Their response indicated that it took:

- Very long
- Two to three months
- Majority of the semester
- The entire semester

The two students who were not comfortable at all with the command line environment stated that they it took them *very long* or *the entire semester*. After a chi-square analysis was performed on these results where $\chi^2 = .537$, $df = 1$, $\alpha = 0.05$, with the critical value being 3.841, the distribution between both groups were not significantly different.

4.4. Microsoft Visual Studio: Easiest/Hardest attributes

This section provides information about what the students who took CS 114 perceived as easy as well as difficult about using Microsoft Visual Studio. When asked the easiest things about using Microsoft Visual Studio the students stated:

- Syntax Highlighting (two students)
- Provided feedback about the program (one student)
- Code Layout (four students)
- Easy to manage projects (two students)
- Compiling and executing a program with just a click (four students)
- Easy interface (one student)
- Debugging (two students)
- File uploading (one student)
- Error detection (one student)

When asked about the hardest of using Microsoft Visual Studio, the students stated:

- Nothing (one student)
- Placing files in the correct location in order to compile a program (one student)
- Understanding exception messages (one student)
- Debugger was not clear (very sensitive to errors) (three students)

- Understanding the errors detected by the compiler (one student)
- It took a while to find everything the environment had to offer (one student)

4.5. Command Line Environment: Easiest/Hardest attributes

This section provides information about what the students from CS 150 believed to be easy as well as difficult when using the command line environment in Linux. When asked about the easiest things of the command line environment, the students stated:

- Easy to understand (four students)
- Using macros from the keyboard to invoke commands onto the command line (one student)
- Lightweight environment (one student)
- Error Messaging (one student)

When asked about the hardest things about the environment, the students responded by saying:

- Adapting to the environment (two students)
- Learning skills on their own (teaching themselves) (one student)
- Switching between Linux and Windows (one student)
- Knowing short cuts through this environment (one student)
- Not being able to use a mouse (one student)
- Error messages were sometimes cryptic (one student)
- Remembering all the commands in the environment (one student)

Note: Some of the responses given fell into the same category as other responses; a couple of students did not provide a response to these questions.

4.6. Python vs. C++ (Statistical Ratings)

As part of the interview, the students were asked to rate on a scale of 1-10 (1 being the lowest, 10 being the highest) their programming skills in their respective language. This data give potential information about whether the environments may have an influence on the ratings given. However, there was a possible threat to validity because the students who took CS 114 have had multiple semesters of the C++ language while the CS 150 students only had one semester. To look at the significance of comparing the two groups and how they rated their skills in the languages respectively, a two-sample *t* test was performed. The averages were computed from the ratings given from both groups in order to perform the test. The average rating given by students who used C++ was a 6.94; the CS 150 students gave an average of 5.23.By using an alpha value of 0.05 (95% confidence interval), the results indicated that these averages, with a p-value of 0.016, were significant. Figure 11 shows an individual value graph plot of these results.

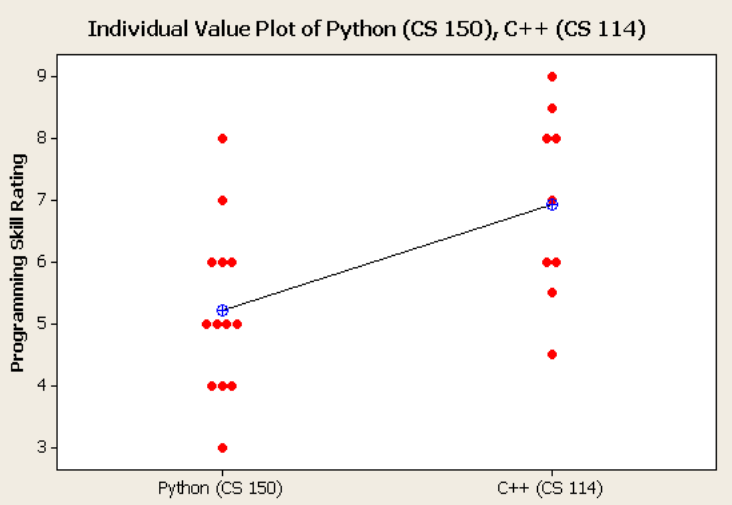


Figure 6: Skill Ratings for the Languages of Python and C++

4.7. Data Concerning Students who took the Original Course Sequence

The next sections, Section 4.7.1, 4.7.2, and 4.7.3 focus only on the students who took the original course sequence since they have experience with using both environments. Section 4.7.1 explores how well these students transitioned from Microsoft Visual Studio to the command line environment in Linux. In Section 4.7.2, the students are asked about their favorite environment to use when programming and the reasons that led them to make their decision. Section 4.7.3 looks at what the students saw to be the more efficient environment to use.

4.7.1. Transition Between Environments

Since these students have been exposed to both an IDE in CS 114 and 124 and the command line environment in CS 325, they were asked to express what was easy as well as difficult about the transition from one environment (IDE) to the other (command line). The easy things about the transition were:

- Same language was being used/Syntax did not change (four students)
- Already knew how to use the command line environment (four students)
- File traversal was easier (one student)
- Code copying and pasting into the environment (one student)

The difficult things about this transition were:

- Compiling on the command line (one student)
- Inputting arguments on the command line (one student)
- None of the compilers provided detailed debugging options (one student)
- Lose syntax highlighting (two students)
- Lose the ability to click a button to compile and execute a program (one student)
- Learning all of the commands (five students)
- Remembering to manually add “include” statements to the program (one student)

4.7.2. IDEs vs. Command Line: Favorite

When asked about their favorite environment to use and the factors that lead them to their decision, out of the ten students interviewed, eight of them (80%) chose the IDE to be their favorite while the remaining two (or 20%) chose the command line. Figure 12 shows a graph of these results.

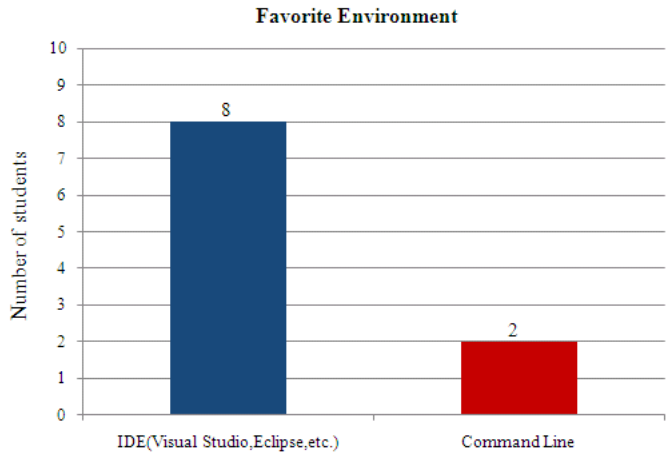


Figure 7: Favorite Programming Environment

The following table, Table 3, shows a list of factors that caused the students to decide their favorite environment:

Table 3: Factors that Lead to Favorite Environment

Microsoft Visual Studio (IDE)	Eclipse (IDE)	Command Line
Has a powerful debugging tool (one student)	Likes the interface (one student)	Command Line is better for smaller projects (one student)
Likes the interface (one student)	Easy to use (one student)	Quicker to use (one student)
Easy to use (one student)	Easier to understand (one student)	Takes up less space on the computer (one student)
Does everything it needs to do (one student)	More user-friendly (one student)	
Easier to learn (one student)		
Great for basic programming (one student)		

4.7.3. IDEs vs. Command Line: More Efficient

In addition to the information asked in Section 4.6, the students were asked which programming environment was more efficient for programming. Out of the nine students who responded to this question, six of them (~67%) believed that the IDE was more efficient while the remaining three (or ~33%) said the command line. Figure 13 shows a graph of these results.

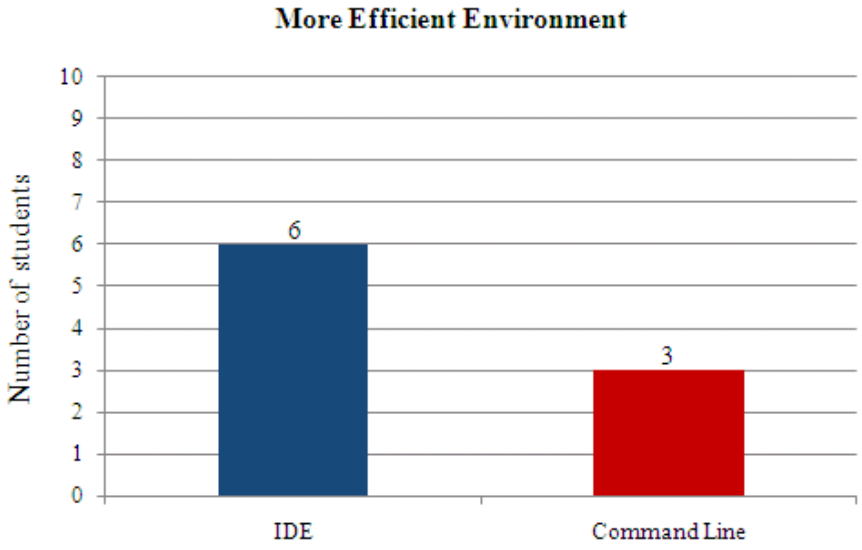


Figure 8: More Efficient Environment

Some of the reasons why certain students believed that the IDE was more efficient was because of *better file organization, code can be generated without being written,* and *easier format layout for programming.* Reasons why the command line was believed to be efficient was due to the *user not being distracted by features* in order to stay focused on the program.

4.8 Student Mentor Interview

This student was asked additional questions about the students' performance with the command line environment in CS 150 along with his experience with the environments. In particular, this assistant stated that the CS 150 students were becoming comfortable and "fluent" with VIM. Another response was that on average the students were comfortable in learning and applying the basic commands. A third observation seen was that the saved files (programs) were very easy to find when using VIM as opposed to Microsoft Visual Studio. This mentor believed that Microsoft Visual Studio required learning how to use the file directory in order to find particular programs that were saved in the environment.

5. ANALYSIS AND DISCUSSION OF RESULTS

When comparing the amount of programming experience between both groups, the results were close. The students from both the original sequence (60%) and new sequence (~54%) had a fair amount of programming experience prior to taking either CS 114 or 150. The chi-square analysis showed that the amount of experience between both groups was evenly distributed.

This question in general required a binary yes or no answer. Each student was asked whether or not they had any programming experience prior to enrolling in their respective course but this introduced another question that concerned how much prior experience did each student have. Just because a student answered yes to this question did not really measure if their programming experience was based on a one day programming assignment, months of programming, or even years of experience. However, the scope of this question and how it was related to this study showed that the students from both groups were evenly distributed with prior experience.

The initial responses to these environments contrasted much more. The majority of the students (~78%) who took CS 114 had a positive response to their initial experience with Microsoft Visual Studio. The CS 150 students roughly had half (~46%) to respond in the same way about the command line environment. By looking at the chi-square analysis from these results, the distribution showed that the amount of students who had a positive initial response to

Microsoft Visual Studio was significantly greater than those students who responded positively to the command line. This contrast could be based on the visual appearance of the IDE being more attractive than the command line. By looking at Table 3 that contains a list of reasons that lead some students to favor IDEs over the command line, the students mostly provided visual reasons. For example, these students typically gave responses such as: liked the interface, was easy to use, and it was more user-friendly as the reason for choosing the IDE over the command line. Seemingly, the interface of the Microsoft Visual Studio (and Eclipse) may have been more attractive and enjoyable to the students in CS 114. On the other hand, the appearance of the command line environment along with less than half of these students not having previous programming experience may have been the critical reason why the students in CS 150 gave a significantly lower positive initial response.

When analyzing the level of comfort these students have with these environments, the students who used Microsoft Visual Studio were at least fairly comfortable with using the environment. Between the three categories shown in Section 4.3, there were more students to say that they were very comfortable with Microsoft Visual Studio than those who were fairly comfortable. In addition, there were no students who claimed to be uncomfortable with the environment. On the other hand, the CS 150 students were almost even in the categories of very comfortable and fairly comfortable. In addition, there were a couple of students who were still uncomfortable with the command line.

However, the chi-square analysis indicated that there was nothing significant about the differences from these distributions. By observing the actual data from Figure 5, the differences in the amount of students who were not comfortable, fairly comfortable, or very comfortable

between both groups only varied by at most three students (Fairly comfortable: 3 students who took CS 114, 6 students who took CS 150, varied by three students).

By looking at the two categories used to compare the amount of time it took the students to become comfortable with these environments, a chi-square analysis showed that there is no significant difference between the two groups. However, the most frequent response by the students concerning the time frame for becoming comfortable with their environment were within one month or less. For the seven students from CS 114 who took less than a month to get comfortable with Microsoft Visual Studio, three students, which was the most frequent amount, stated that it took them about one month. Out of the nine students from CS 150 who also fell into the same category, five of them (most frequent) stated it took them one to three weeks. The second most frequent amount of students to become comfortable with Microsoft Visual Studio in a month or less was two students who stated that their confidence came after the first project; for the students in CS 150, the next frequent amount was three students who stated that it took them not long at all to get comfortable with the command line. In contrast, the amount of time it took the students who were not comfortable after one month varied. There were no students who reached their comfort level in a similar amount of time. Overall, the majority from both groups became comfortable with their respective environment within one month or less.

When observing the easiest attributes seen from these environments, the students in CS 114 most frequently mentioned Microsoft Visual Studio's ability to automatically format and align code along with being able to compile and execute a program with just a click of the button (each statement was said by four students). The second most frequent response consisted of three statements: syntax highlighting, managing projects, and debugging (each statement was said by

two students). The remaining attributes that made Microsoft Visual Studio easy was stated by one student per statement. For the command line, the ideal of the environment being easy to understand was the most frequent response by the students in CS 150 (said by four students). The remaining responses only consisted of one student per response.

When looking at what the students saw as difficult about these environments, the most frequent response from the CS 114 students was that the debugger was unclear (said by three students). The number of times this statement was said almost equaled the amount of students who stated that debugging was actually an easy attribute. The other difficult attributes about Microsoft Visual Studio had only one student per statement. For the command line, there was only one statement that was said by more than one student which concerned adapting to the environment (two students). The other statements consisted of one student per statement.

The relevance of obtaining skill ratings from these students based on their programming language was to determine if there were any potential confounding factors. By looking at the programming languages used on these environments, the CS 114 students, who used C++, gave an average skill rating of 6.94. The CS 150, who used Python, gave an average skill rating of 5.23. These results were interesting because C++ is generally thought to be a much harder language than Python. However, one confounding factor was that the students in the original sequence have already had multiple semesters of C++ while the students in Python only had one. Another possible factor could be the environments. If so, then this implies that the IDE was having a better effect on how the students are learning to program.

As far as transitioning from Microsoft Visual Studio to a command line environment, the most frequent statements concerning the easiness of the transition were: the syntax did not

change due to the same language being used or the students' already knew how to use the environment because of previous experience (each statement was said by four students). But when observing the difficult aspects about this transformation, the most frequently stated issue was the idea of learning all of the commands (said by five students); only one student from CS 150 stated this as a hard attribute of the command line. This however was not surprising since the students typically began programming using an IDE that could perform various kinds of actions via a button click. This introduces the question of whether students who use an IDE actually understand the activities involved in the procedure for programming. Rather than just knowing to click a certain button option on the interface to perform an action, do they understand what is going on behind the scenes? There are also other statements made by these students that suggest the possibility of them being "spoiled" by the IDE which caused them to lose out on the full understanding of programming. For example, inputting arguments on the command line, compilers not providing detailed debugging options, no syntax highlighting, and remembering to manually add certain statements to a program are things that can help a programmer understand the basics of programming. By manually performing these actions, the programmer can get a more hands-on feel for programming. In many cases the programmer who uses an IDE is prevented from manually performing certain actions as stated above and therefore could potentially lack the understanding of what is actually occurring in the program.

When looking at what these students chose as their favorite environment, majority of them (~80%) liked the IDE. Since some of these students were currently in CS 357, which teaches the Java language, there was a mixed preference between Microsoft Visual Studio and Eclipse. As stated earlier, the major reasons for these students liking the IDEs seems to be based

on their visual appearance with the exception of a couple of reasons which were: *it has a powerful debugging tool,* and *great for basic programming*. The two students who chose the command line as their favorite gave reasons such as: *the command line is better for smaller projects, it is quicker to use,* or *it takes up less space on the computer.* It seemed that these students liked the command line because it performed in an efficient manner.

In addition, these students were asked about the more efficient environment out of the two. The majority of these students chose the IDE; only three students chose otherwise. Some of the reasons why these students thought the IDE was more efficient was because it had better file organization, the code could be generated as well as written, and it was easier to layout the format for programming. The major reason why the other students chose the command line was because it allowed the user not to be distracted by different features.

5.1 What Do These Results Imply about these Environments?

By only observing what the students said in regard to these environments without focusing on any other factors, it seemed that the IDE was the better environment. The criteria used to obtain these results show that more students in CS 114 who used the IDE had a higher rate of positive responses, they were more comfortable with this environment, only two of these students took more than a month to get comfortable, and more easy attributes were stated about this environment. However, summarizing this data does not give accurate proof that an IDE is better than the command line. One problem is that both student samples are not that big, therefore it is not best to say that just because the students from the original sequence had more students to have a positive initial response, a greater level of comfort, and more things to say about the easy attributes than hard attributes does not prove that an IDE is better. Based on the chi-square analyses performed on much of these results, most of the criteria, with the exception of the initial response to the environment, had an even distribution between both samples. Another factor is the visual appearance of the IDE. Many reasons why the students favored the IDE were because of its visual features. But does appearance alone make an IDE better than the command line. In order for the IDE to be more suitable than the command line, it must help the programmers, in particular novices, with acquiring the basic programming skills that will help them become successful programmers. Therefore, it is better to say that the stated hypothesis for this study, *an IDE is actually a better learning environment compared to a Command Line/Console for novice programmers to use*, could not be proven based on these results, because

there were no significant factors, with the exception to the initial responses to the environment, that made one environment better than the other. Rather, there is a question of whether these environments may be equally matched in suitability for programmers, especially for novices.

6. THREATS TO VALIDITY

There were a few observations from these interviews that posed as threats to validity for this study. The first observation was that the students who took the original course sequence of CS 114, 124, and 325 had more experience with using the C++ language unlike the students in CS 150 who only had one semester of Python. It was somewhat expected that the programming skill ratings for C++ students would be higher than the students who use Python due to more exposure. This observation relates to the second observance, which is that the students involved in the original sequence also had at least two semesters of exposure to Microsoft Visual Studio compared to the students in the new sequence who had only one semester of exposure to the command line.

A third observation is that the students who took the original course sequence lacked diversity when compared to the CS 150 students. The students interviewed were mostly Caucasian males. The information retrieved from these students was not invalid but it would have been better if this information could have come from a pool of diverse students. However, this sample was a good representation of the student population of CS 325 and 357. From the sample, 80% of the students were Caucasian male; the population showed that ~79% of the students were Caucasian male. In the sample, only 10% of the students were Non-Caucasian while the larger population was ~13% Non-Caucasian. When concentrating on the difference in gender, 80% of the student sample consisted of male students while the population of males was

~89%.The percentage of females in the sample was 20% while the population showed a percentage of ~16%.

Another observation relates to the fact that the CS 150 students were not all Computer Science majors. Only one student was actually a Computer Science major. Even though these students provided a great amount of information about the command line in Linux, there was a chance that the sample's representation of students may not have been a good representation of the CS 150 population. Currently, the representation of the CS 150 student population is unavailable.

A final observation was that these results represented only a sample of students. This leaves open a question of whether or not the results would differ if a totally different sample of students are interviewed. An alternative approach to resolving problem is to increase the sample to include all of the students who are (or were) taking these courses in order to make the results even stronger.

7. FUTURE WORK

From these interviews, the overall objective was to provide background information on which environment was potentially more suitable for learning to program, specifically for novice programmers. It was the intent that this information would provide the groundwork for the study or studies that needed to be done concerning this question; much of this future work would be done as part of dissertation research. As an immediate future work, questionnaires will be generated for similar courses offered in the Fall. In addition, the second course of the new sequence, CS 250, will be offered for the first time where the students enrolled in this course will be studied.

Another future work is creating a hands-on study where students will be asked to write a program using either of the two environments. A key variable for this work would be *Ease of Use*, which could be measured through components such as: *the easiness of writing the assignment in the environment, time to complete the assignment, was the right output produced*, and *how easy was it to detect and correct errors along with compiling and executing the program*. This kind of study could potentially provide stronger evidence about the suitability of either environment.

Another idea is to study whether the functionality as well as the visual appearance of an IDE robs the programmer from developing the necessary skills for programming. As stated earlier in this study, most of the students in the original sequence picked the IDE (Microsoft Visual Studio or Eclipse) as their favorite environment. However, the results of their transition

from an IDE to the command line showed that the attributes they saw as difficult were things that typical command line programmers are comfortable at using.

As stated in Chapter 6, most of the students in the sample and population of CS 325 and 357 were Caucasian males. This raised a question of whether there were ethnicity as well as gender related differences in learning to program, which is another potential future work. If there are actual differences, are these differences the reason for the lack of diversity in upper level Computer Science courses?

Another future work is to perform more interviews and surveys with a different sample of students to see if there are actually some significant differences among the groups based on the criteria used for this study. In addition, another alternative work is to involve more student mentors who help students in lower level courses that use these environments. While in the lab, these mentors could take notes based on a set of criteria that would be used to obtain the necessary data needed to explore which environment is more suitable.

8. CONCLUSION

The intent of this research was to set the background for further studies in determining which environment is more appropriate for a novice programmer to use when learning to program, whether it is an Integrated Development Environment or a command line environment. The results from these interviews/surveys, however, did not indicate anything that would prove one environment is more suitable than the other. Instead, this research has generated questions when studying these environments such as: are these environments equally suitable for novice programmers, is an IDE's functionality a detriment to a programmer learning to program, and are there differences in how a student learns to program based on ethnicity and/or gender. In addition, there are future studies that can be done to strengthen or weaken the results found in this research such as: an empirical hands-on study that allows for students to experiment with both environments, or change/increase the student sample and perform this study again to see if there are actual differences that would make one environment potentially more suitable than the other. A main contribution that is made from this study is that IDEs may be more attractive and appealing to users due to their user-friendly background but what is "attractive" does not necessarily mean that it is "better" or "more suitable" for novice programmers who are in the process of learning to program. An environment may be more engaging and interesting to students, but that may or may not translate into better programming abilities in terms of writing, debugging and using code, and it may not imply the kind of deeper understanding that should lead to better performance in more advanced courses and assignments.

BIBLIOGRAPHY

[1] Hristova, Maria and Misra, Ananya and Rutter, Megan and Mercuri, Rebecca. "Identifying and correcting Java programming errors for introductory computer science." *SIGCSE '03* (ACM), 2003: 153-156.

[2] McCracken, Michael, Almstrum, Vicki, Diaz, Danny, Guzdial, Mark, Hagan, Dianne, Kolikant,Yifat Ben-David, Laxer, Cary, Thomas, Lynda, Utting, Ian and Wilusz, Tadeusz. "A multi-national, multi-institutional study of assessment of programming skills of first-year CS students." In *ITiCSE-WGR '01: Working group reports from ITiCSE on Innovation and technology in computer science education*, 125-140. New York, NY, USA: ACM, 2001.

[3] Wiedenbeck, Susan, Fix, Vikki and Scholtz,, Jean. "Characteristics of the mental representations of novice and expert programmers: an empirical study." *Int. J. Man-Mach. Stud.*, Vol. 39(5), 1993: 793-812.

[4] Jeffries, R. "A comparison of the debugging behavior of expert and novice programmers." *American Educational Research Association.* New York, 1982.

[5] Gugerty, Leo and Olson, Gary M. "Comprehension differences in debugging by skilled and novice programmers." *First workshop on empirical studies of programmers on Empirical studies of programmers.* Norwood, NJ, USA: Ablex Publishing Corp., 1986. 13-27.

[6] Nanja, Murthi and Cook, Curtis R. "An analysis of the on-line debugging process." *Empirical studies of programmers: second workshop.* Norwood, NJ, USA: Ablex Publishing Corp., 1987. 172-184.

[7] Holt, Robert W., Boehm-Davis, Deborah A. and Shultz, Alan C. "Mental representations of programs for student and professional programmers." *Empirical studies of programmers: second workshop.* Norwood, NJ, USA: Ablex Publishing Corp., 1987. 33-46.

[8] Adelson, B. “When novices surpass experts: the difficulty of a task may increase with expertise.” *Journal of Experimental Psychology: Learning, Memory, and Cognition.* Vol. 10, (3), 1984. 483 – 495. Reprinted in Human Factors in Software Development (2nd ed.); Bill Curtis (ed.). Washington, DC. IEEE Computer Society Press, 1985. 55 – 67.

[9] Jackson, J., Cobb, M. and Carver, C. "Identifying Top Java Errors for Novice Programmers." *Frontiers in Education, 2005. FIE '05. Proceedings 35th Annual Conference.* Oct. 2005. T4C-T4C.

[10] Jadud, Matthew C. "Methods and tools for exploring novice compilation behaviour." In *ICER '06: Proceedings of the second international workshop on Computing education research*, 73-84. New York, NY, USA: ACM, 2006.

[11] Chabert, Joan M. and Higginbotham, T. F. "An investigation of novice programmer errors in IBM 370 (OS) assembly language." In *ACM-SE 14: Proceedings of the 14th annual Southeast regional conference*, 319-323. New York, NY, USA: ACM, 1976.

[12] Spohrer, James C. and Soloway, Elliot. "Novice mistakes: are the folk wisdoms correct?" *Commun. ACM*, Vol 29(7) 1986: 624-632.

[13] Bayman, Piraye and Mayer, Richard E. "A diagnosis of beginning programmers' misconceptions of BASIC programming statements." *Commun. ACM*, Vol 26(9) 1983: 677-679.

[14] Mciver, Linda. "The effect of programming language error rates of novice programmers." In *12th Annual Workshop of Psychology of Programmers Interest Group (PPIG), Corigliano*, 181-192. Cozenza, Italy: A.F. Blackwell & E. Bilotta (Eds). Proc. PPIG 12, 2000.

[15] Biddle, R.L. and Tempero, E.D. "Teaching C++ Experience at Victoria University of Wellington." *IEEE*, 1995: 274-281.

[16] Jenkins, Troy. *The First Language - A Case for Python.* University of Leeds, March 06, 2004.

[17] Hitz, Martin and Hudec, Marcus. "Modula-2 Versus C++ as a First Programming Language Some Empirical Results." *SIGCSE '95.* Nashville, TN: ACM, 1995. 317 - 321.

[18] Zelle, John M. "Python as a First Language." *Wartburg College.* http://mcsp.wartburg.edu/zelle/python/python-first.html (accessed February 6, 2009).

[19] Python Software Foundation. *About Python.* 1990 - 2009. http://www.python.org/about/ (accessed February 6, 2009).

[20] Stroustrup, Bjarne. “A History of C++: 1979 – 1991.” New York, NY: ACM, 1996.

[21] Stroustrup, Bjarne. "An Overview of C++." *SIGPLAN Notices*, 1986: 7-18.

[22] Olan, Michael. "Dr. J vs. The Bird: Java IDE's One-On-One." *Consortium for Computer Science in Colleges*, 2004: 44-52.

[23] Reis, Charles, and Robert Cartwright. "Taming a Professional IDE for the Classroom." *SIGCSE'04.* Norfolk, Virginia: ACM, 2004. 156-160.

[24] Swartz, Fred. "IDEs." *Java Notes.* 2007. http://leepoint.net/notes-java/tools/10ide.html (accessed February 12, 2009).

[25] Chen, Zhixiong, and Delia Marx. "Experiences with Eclipse IDE in Programming Courses." *Consortium for Computing Sciences in Colleges*, 2005: 104-112.

[26] Rigby, Peter C., and Suzanne. Thompson. "Study of Novice Programmers using Eclipse and Gild." *Eclipse'05.* San Diego, CA: IBM, 2005. 105-109.

[27] Lecky-Thompson, Guy. "Console/Command Line Programming." *Suite101.com.* April 7, 2006. http://computerprogramming.suite101.com/article.cfm/consoleprogramming (accessed January 21, 2009).

[28] IDM Computer Solutions, Inc. *Text Editor, Hex Editor, PHP Editor, HTML Editor, Programmer's Editor UltraEdit/UEStudio/UltraCompare/UltraSentry.* 2009. http://www.ultraedit.com/ (accessed March 19, 2009).

[29] Lecky-Thompson, Guy. "Windows and Win32 Programming." *Suite101.com.* July 28, 2006. http://computerprogramming.suite101.com/article.cfm/the_windows_programming_paradigm (accessed January 21, 2009).

APPENDIX A. CS 325 AND 357 INTERVIEW QUESTIONS

This appendix includes the questions that were asked of the students who were taking either or both CS 325 and 357. These interviews were done during the final week of the Spring 2009 semester.

CS 325 and 357 students

1. Demographic questions: Gender, age, major, current CS class, ethnic background …

2. When you started out in CS 114, you used Microsoft Visual Studio and the C++ language.

    a. Was that your first programming experience? If not, what was your first language and environment?

3. What was your initial response to the programming environment you had in CS 114?

4. What was the easiest thing about your programming environment in CS 114?

5. What was the hardest thing about your programming environment in CS 114?

6. How comfortable are you now with this programming environment?

7. How long did it take to get comfortable using this environment?

8. If you started out with Visual Studio in CS 114, how did moving from Visual Studio to a command line environment in CS 325 and 357 change the way you think about programming?

    a. What were some of the easiest things about the transition?

    b. What were some of the hardest things about the transition?

For students who have learned more than one programming environment and/or language, these additional questions are planned. (Students who have completed the CS introductory sequence will have learned more than one programming language, thus they will get these questions.)

9. Which is your favorite programming environment?

    a. Why? What factors led you to this?

10. Which environment do you feel is more efficient for programming?

11. Which programming environment makes it easier to debug code?

12. Did the particular programming languages have an effect on your decisions? (For example, is Python easier to learn than C++ or vice versa?)

13. If you had learned C++ in a Linux environment, or Python in a Visual Studio type of environment, how do you think that would have changed your learning experience?

- **a.** How do you think that would change your understanding of the language?
- **b.** How do you think that would change your understanding of the programming process?
- **c.** Would you go back and change it if you could? What would you change?

14. On a scale of 1 – 10 (1 being extremely poor, 10 being extremely well), how would you rate your overall programming skills in:

- **a.** C++?
- **b.** Java?
- **c.** Any other language learned?

APPENDIX B. CS 150 SURVEY QUESTIONS

This appendix includes the questions that were asked of the students who were taking CS 150 after a focus group discussion that took place during the final week of the Spring 2009 semester.

Questions for CS 150 students

1. Demographic questions: Gender, age, major, current CS class, ethnic background …

2. When you started out in CS 150, you used the Linux environment and the Python language.

    a. Was that your first programming experience? If not, what was your first language and environment?

3. What was your initial response to the programming environment you had in CS 150?

4. What was the easiest thing about your programming environment in CS 150?

5. What was the hardest thing about your programming environment in CS 150?

6. How comfortable are you now with this programming environment?

7. How long did it take to get comfortable using this environment?

8. On a scale of 1 – 10 (1 being extremely poor, 10 being extremely well), how would you rate your overall programming skills in:

    a. Python?

    b. Any other language learned?