Choice of Language for an Introduction to Programming Course

Jeffrey L. Duffany, Ph.D.
Universidad del Turabo, Gurabo, PR, USA, jeduffany@suagm.edu

ABSTRACT

There are over 300 programming languages currently in existence today however only about a dozen of the most commonly used languages would be candidates for use in teaching an introductory programming course to engineering students. This short list would likely include languages such as C, C++, C#, Java, MATLAB, OCTAVE, R, Python, Ruby and Visual Basic. One approach would be to use a language such as C, C++, C#, Visual Basic, Java or Ruby which tend to be lower level languages with a more difficult learning curve for beginning programmers. Another approach would be to use a language like MATLAB, OCTAVE, R or Python which are higher level languages with somewhat less difficult learning curves for beginning programmers. Advantages and disadvantages each of these approaches is discussed as well as other general considerations for choosing a programming language for an introductory programming course for engineering students.

Keywords: computers, programming languages

1. INTRODUCTION

Since the mid-1970's engineering students have generally been required to take an introductory programming class in their first year of engineering school. The programming language typically chosen was Fortran. There was likely no text book and the program was probably run from punched cards. It may not have even been a full course. Most likely it was given as the second half of an engineering graphics course (where students learned how to draw). The same course was typically taken by all engineering students. For many students it would be the only exposure they have to programming during their undergraduate engineering education.

But that was then and this is now. There currently are over 300 programming languages in existence (Wikipedia 2014); most of them are unknown to (and unheard of by) the vast majority of people. There are a variety of programming languages which could be used for an introductory programming course each with its own set of advantages and disadvantages. It is important to keep in perspective why programming languages are being taught as this is an important factor in choosing one. The purpose of an introductory programming course should be to teach programming concepts and not to try to teach someone to be a professional programmer. It should teach the student a skill that can be useful in solving engineering problems. It should also be a lifetime skill that can be used in engineering courses while in school and on the job after graduation.

There does not appear to be any one single programming language that is best for teaching an undergraduate introductory programming course. Programming languages are primarily designed to meet real world needs for solving scientific problems or sharing information on the world wide web or for entertainment. Programming languages are generally not designed for the teaching programming languages (although some of them might be). Time invested in learning a language should be worthwhile which argues against "throw-away" learning languages. The choice of programming language for engineering might not be the same as for computer science.

One of the main challenges of choosing a programming language is to satisfy the needs of engineering students across multiple disciplines such as Electrical, Mechanical, Industrial, Computer, and Civil Engineers to name a few. Table 1 (ASEE 2011) shows the undergraduate enrollment across majors in 2011-2012 for universities in the United States. Clearly engineering schools will exhibit wide variations with many other disciplines (including aerospace, nuclear, biomedical, etc.) that needs to factor into the choice of programming language.

Table 1: Percentage of Students Enrolled in Various Engineering Majors (ASEE-2011)

Mechanical	Electrical or Computer	Civil	Chemical	Industrial
20%	16%	10%	7%	3%

2. ABET PERSPECTIVE

The Accreditation Board for Engineering and Technology (ABET) accredits engineering schools based on their curriculum and other factors. According to ABET (2013), engineering students, by the time they graduate, should have the following skills:

.

- an ability to design and conduct experiments, as well as to analyze and interpret data
- an ability to use the techniques, skills, and modern engineering tools necessary for engineering practice

ABET (2013) does not specifically require a course in computer programming for engineering students with the exception of Computer Engineering (before the availability of computers the ABET objectives were probably met by teaching students how to use the slide rule). However, teaching a programming course to all engineering students will contribute significantly to the meeting of the above two objectives and so it appears almost universally the case. For Computer Engineering students ABET requires in addition to the above objectives:

- An exposure to a variety of programming languages and systems
- Proficiency in at least one higher-level language
- Coverage of the fundamentals of algorithms, data structures, software design, concepts of programming languages and computer organization and architecture

As a basis for meeting these objectives it is assumed that the typical engineering curriculum has three core programming classes: introductory programming, intermediate programming, and object oriented programming. All engineering majors would likely be required to take Introductory Programming while Electrical Engineering majors perhaps would be required to take both Introductory Programming and Intermediate Programming. Computer Engineering majors would be required to take all three of the core programming courses.

3. GOALS OF AN INTRODUCTORY PROGRAMMING COURSE FOR ENGINEERING STUDENTS

Bjedov (1995) asks whether should programming be taught to freshmen engineering students at all. The answer is clearly yes for a number of reasons. First of all, as already pointed out, it helps fulfill one or more ABET objectives. However, from a practical point of view, it gives the students a tool they can use for their engineering classes and also a skill they can take with them after they graduate into the workplace. At the very least the student will have an increased ability to communicate with software developers and a better appreciation of software applications that they will probably use on a day to day basis. Some of the main goals of an introductory programming should undoubtably be to:

- teach basic programming skills
- learn logical thinking
- develop problem solving abilities

In theory the basic programming skills taught in introductory programming courses should be transferable to any other programming language the student might subsequently learn. Flowcharts help teach logical thinking. In addition, many programming exercises can be viewed as a type of puzzle that needs to be solved by writing the code needed to get the solution. As an added bonus, the teaching of programming can also be used to reinforce mathematical concepts and there are many ways that this will naturally occur. For example a *for loop* can be used to reinforce the concept of a mathematical summation (e.g., adding the integers from 1 to 100). Also, functions can be written to solve mathematical formulas. For example if the student writes a computer program to solve the quadratic formula it will reinforce the mathematical concepts of real and imaginary roots.

4. CONSIDERATIONS IN CHOOSING A PROGRAMMING LANGUAGE

Some of the important considerations for choosing a programming language for an introductory programming course can be summarized as follows:

- Ease of Use (Learning Curve)
- Textbook and teaching materials (powerpoint, end of chapter exercises)
- Programming Environment
- Cost

Ease of use should factor heavily into the choice (Kelleher 2005). A steep learning curve could frustrate students and could cause them to transfer to a different major (Yacob 2012) (Mannila 2006). Another important factor is the textbook and learning materials. Most textbooks are written for a wide audience from computer science majors to professional programmers to people who are studying on their own. Some textbooks have too much depth or are too technical for an introductory course and may have been written for professional programmers or for people who have already mastered one or more programming languages. The programming environment itself should not be too overwhelming or intimidating to the novice. Many IDEs (Integrated Development Environments) are complex and unforgiving having far too much capability. These are designed for professional software developers and not for students learning the language. The cost can also be a factor especially from the student's point of view. A language that can be downloaded for free is more likely to be used on the student's home computer or laptop and used to solve problems from engineering courses or analyze lab data.

5. ANALOGY WITH SPOKEN LANGUAGES

There are some important similarities between learning a programming language and learning a spoken language. Clearly everyone needs to know at least one spoken language which is their native language. However there is little motivation in learning a second language unless you plan to use it. There is little motivation to learning Japanese unless you are planning to visit Japan or are planning to move to Japan at some point in the future. Engineering students are likely to use a programming language for several reasons including:

- solving homework problems
- solving problems on take home tests
- analyzing lab data

Unfortunately many first semester engineering students do not recognize the value of learning a programming language. It takes time and effort and practice to learn a programming language just as it does to learn a spoken language. As a result many students view their introductory programming course as either a throw-away course or a necessary evil and their main goal is just to pass the course with a decent grade. It is likely that many students never use their programming skills after the introductory course and get by using a calculator for the rest of their lives. However, even if that is the case, an introductory programming course will give them a valuable exposure to the software development process which may benefit them in unexpected ways later on.

6. COMPARISON OF PROGRAMMING LANGUAGES

There are over 300 programming languages currently in existence today (Wikipedia 2014). However only about a dozen of the most commonly used languages would be candidates for use in teaching an introductory programming course to engineering students. This short list would likely include languages such as C, C++, C#, Java, MATLAB, OCTAVE, R, Python, Ruby and Visual Basic. Of these possibilities our discussion will focus mainly on C++, Visual Basic, MATLAB, and R Languages. C, C++, C#, Visual Basic, Java and Ruby are lower level languages with a more difficult learning curve for beginning programmers. MATLAB, OCTAVE, R and Python are higher level languages with less difficult learning curves for beginning programmers. The remainder of this section gives a summary of the advantages and disadvantages each of these languages with regard to its use for an introductory programming course (except for OCTAVE which is very similar to MATLAB).

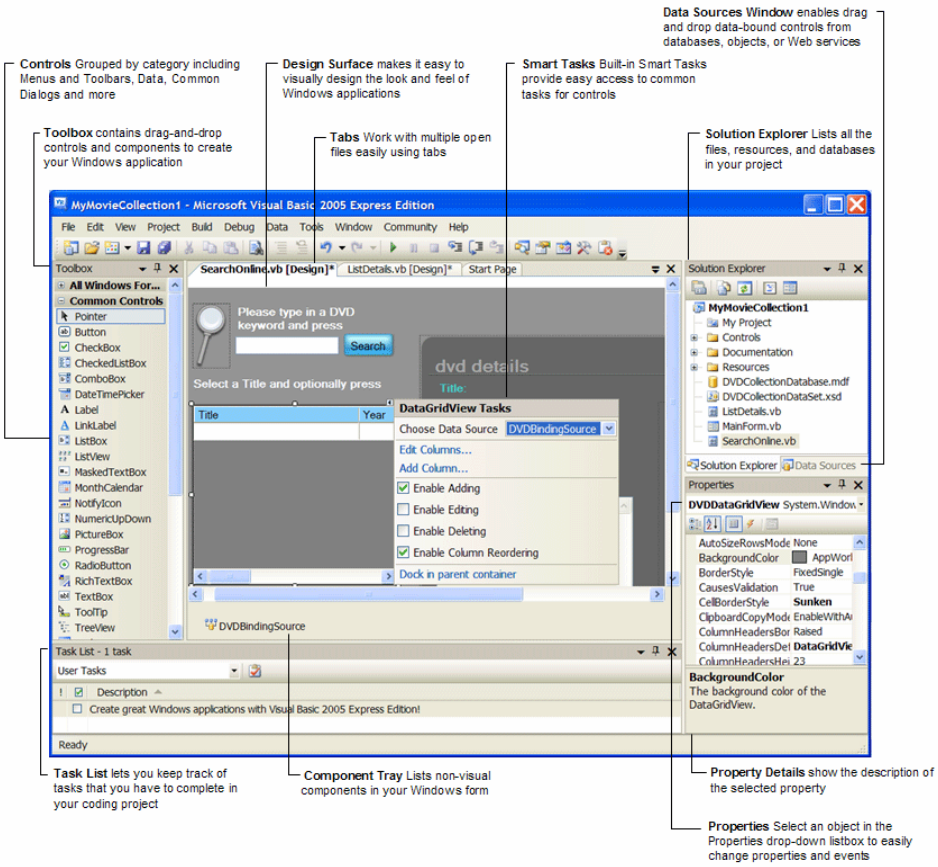

Figure 1: Visual Studio Integrated Development Environment

C Language (Kernighan 1988)

During the 1980s you could teach programming concepts with C language and the classic book on the subject (Kernighan 1988). The programming environment consisted of a text editor to enter the program and a C compiler program (cc). You would write a program using an editor (such as vi) and name your text file *program.c*. You would then compile the program using the C compiler using the command line *cc program.c > a.out* which stored the binary executable in the *a.out* file. To run the program you typed *a.out* into the unix shell command prompt *$* and the program would print its output on the terminal command line interface. To help build confidence the first program usually just printed something on the screen (e.g., *printf("hello kitty");* which prints *hello kitty* on the screen. Writing a C language program is somewhat syntax intensive which beginning programmers might find to be somwhat burdensome. All variables must be defined before using them (int, char, float, etc.), header files must be specified for each library to be used (e.g., stdio.h), preprocessor directives specified and all statements must end in a semicolon. Failure to observe any syntactical constraint will lead to a failure in code compiliation.

C++ (Stroustrup 2013)

Fast forward to the present. The command line interface has largely been replaced with a powerful integrated development environment (IDE) that includes an editor, compiler, and a debugger program all built into one user interface similar to Figure 1. Unfortunately, the IDE can be rather complex and unforgiving for the novice. In addition C++ must follow all of the syntactical constraints of C plus all the additional constraints introduced by the C++ language. These syntactical constraints are not very difficult once they are mastered however they can be intimidating for the first year engineering student. The significant advantage of C++ language is the availability of university-level textbooks written specifically for engineers, scientists and engineering students (Bronson 2010).

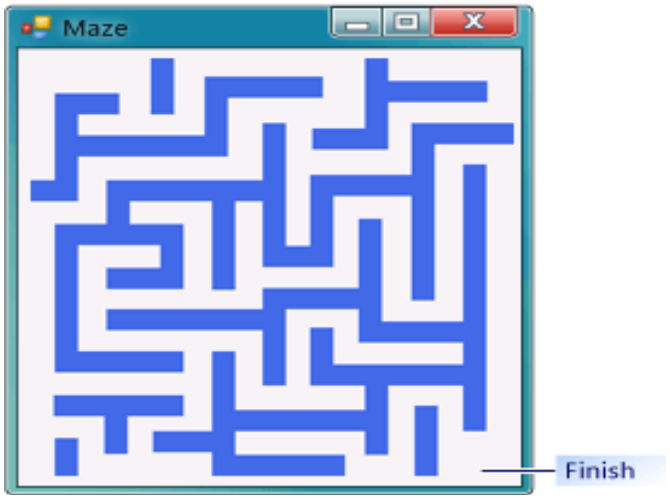

Figure 2: Visual Basic Maze Tutorial

Visual Basic (Microsoft 2013)

Visual Basic is Microsoft's version of the Basic Language (Microsoft 2013). It is an object oriented language similar to C++ that allows part of the code to be "written" by drag and drop of various objects into various containers. There are several modes of operation most notably the Windows Form Application and the Windows Console Application. The Windows Console Application looks very similar to Figure 1 and can be used very much like C or C++ for teaching programming concepts. Figure 2 shows the use of the Windows Form Application used to create a maze by instantiating a large number of box-shaped objects and then creating event handlers to detect whether the mouse is in the white area or not. Visual Basic applications are heavily oriented towards designing user input forms and web applications. Students tend to end up spending a disproportionate amount of time modifying object properties instead of focusing on more traditional programming skills. Many textbooks are available for Visual Basic (Zak 2013) but few if any appear to be written for engineering students.

Java (Gosling 1996)

Java is similar to C and C++ yet not particularly well suited for scientific or engineering problem solving. Java is perhaps better suited to a more advanced class in object oriented programming or web applications.

MATLAB (Moler 2004)

MATLAB features an interactive command line interface as well as the ability to write programs. MATLAB is relatively easy to learn and to program and has very powerful data handling, manipulation and graphical capabilities (Moler 2004). Figure 3 shows a MATLAB graphical display. There are a number of textbooks for MATLAB that teach the student skills for solving engineering problems, analyzing data and visualizing data (Pratap 2005). The main drawback is that it is not free. MATLAB programming skills will generally carry over to other languages but the transition is more difficult to more syntactically intensive languages such as C++.

Python (van Rossum 1993)

Python is similar in many regards to MATLAB and R languages and also C but is more powerful and somewhat more complex syntactically. Python has recently been viewed as a language for beginners and even children however that appears to mainly as a result of a book designed to appeal to this particular audience (Briggs 2012). It would perhaps be better suited for an introductory programming course for Computer Science students as opposed to engineering students. Python is free to download and use. One of the main drawbacks of Python is the lack of a university-level textbook specifically geared towards engineering students.

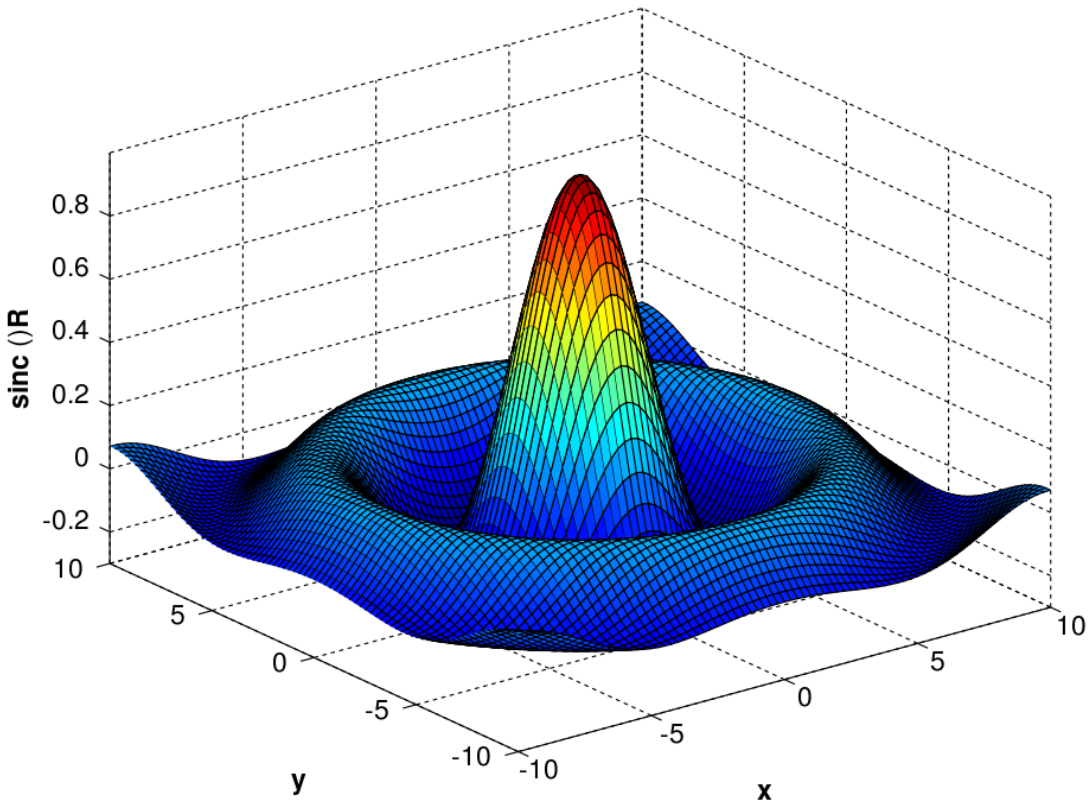

Figure 3: MATLAB 3D Wireframe Graphical Display of the Sinc Function

R Language (Ihaka 1996)

R language is an interpreted language (Knell 2013) similar to MATLAB in many regards. Commands can be entered interactively (Figure 4) or programs can be written and stored in text files. R differs from C language in that it does not require preprocessor directives, header files, variable declaration or function prototypes; nor does it require a semicolon or other delimiter at the end of each statement. Variables are typed automatically according to the data that is assigned to the variable. If you create a variable x and assign it the integer value 5 then x is defined as an integer. This can of course lead to programming errors with mixed type expressions but for introductory programming students the advantages appear to far outweigh the disadvantages. By contrast C++ will not even compile if you attempt to take the square root of an integer, even if that integer is a perfect square.

The R Language environment is extremely portable as everything is stored in an .RData file which can easily be copied to USB or emailed. R has very good graphics capability which is very flexible to customize to the user's needs. R is capable of producing graphics similar to MATLAB using the RGL (R interface to Open GL) graphics package. A number of university-level textbooks exist for teaching R Language as a first programming language (Knell 2013) however many of these are oriented to teaching statistics. R Language is free to download and use.

Ruby (Matsumoto 2005)

Ruby is projected as being a fun and easy language (Matsumo 2005). This is perhaps true in a relative sense as compared to some other languages (if you are already an experience programmer) as syntactically Ruby is more complex than R and MATLAB. Although likely not the best choice for an introductory programming course it would perhaps be better suited for a more advanced class in web development. Ruby is also free to download.

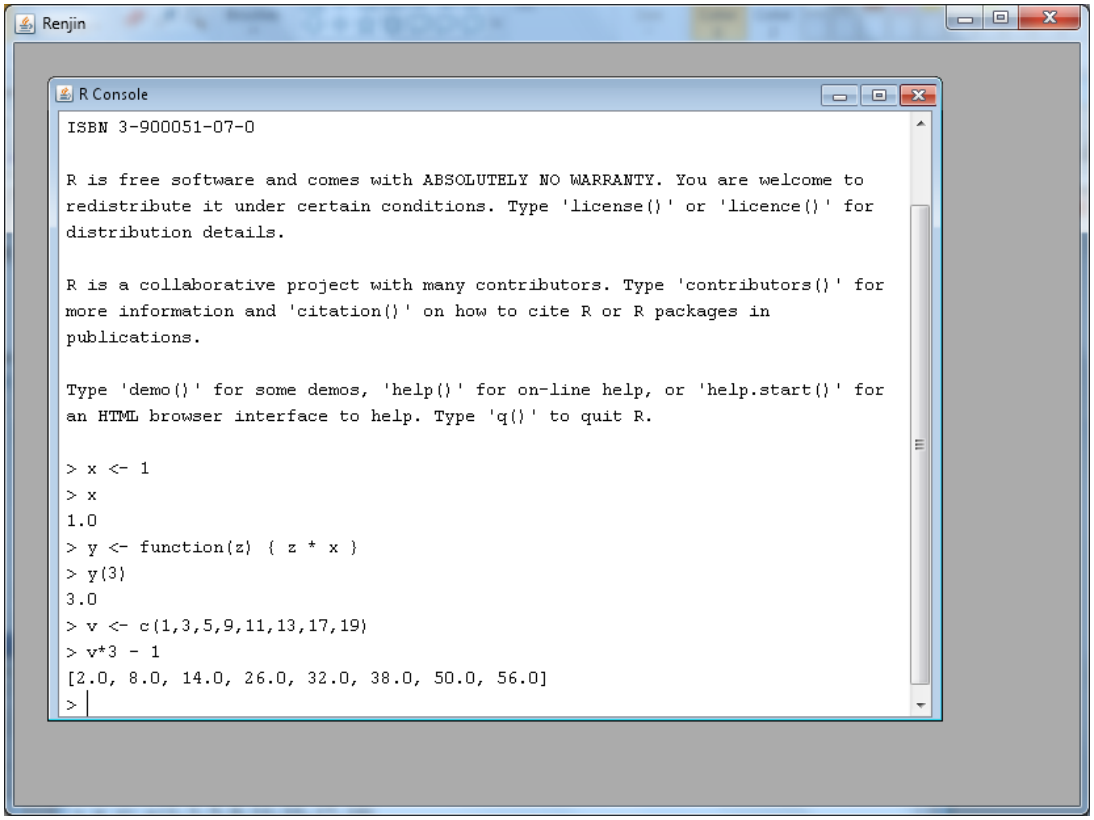

Figure 4: R Language Interactive Programming Environment

7. DISCUSSION

There are various ways to approach the choice of an introductory programming language. One way is to choose a language that is relatively easy to learn and has the highest benefit to cost ratio in terms of the effort involved. This can be considered as the lowest common denominator approach to choosing a programming language which would attempt to benefit all engineering majors as equally as possible. Programming languages that fit into this category are MATLAB, OCTAVE, R and PYTHON. These all fall in the category of higher level languages. All of these have built-in facilities for graphical display of data which is an extremely important advantage since students will want to use this for analyzing lab data. The main drawback for MATLAB is that it is not free. OCTAVE is the free version of MATLAB which does not have as many features as MATLAB and may not always work as well as MATLAB since it is free. MATLAB is very well supported in terms of teaching materials. There are a number of books on R however there do not appear to be any that are specifically focussed on science and engineering as you find with MATLAB and C++. R textbooks usually focus on statistics or data mining.

An alternative approach is to use a more lower level (general purpose) programming language such as C, C#, C++, Java or Visual Basic and provide a more rigourous understanding of programming concepts. These languages tend to have a more difficult learning curve and many syntactical constraints which can end up being a pedagogical impediment to learning the concepts of programming. The benefit behind this approach is that the student will hopefully be able to transfer their skills to other languages especially the easier languages that they might use for their courses. The drawback to this approach is that it tends to benefit the electrical and computer engineering students who go on to take more advanced courses in these languages. For this group of languages C++ has an advantage in that there are many books that teach C++ in the context of Science and Enginering (Bronson 2010). The drawback of Visual Basic is that there do not appear to be very many university level tetbooks to choose from and none of them are particularly geared toward engineering students (Zak 2003).

8. ADDITIONAL PROGRAMMING LANGUAGE CHOICE CONSIDERATIONS

According to Table 1 the engineering major with the most number of students is Mechanical engineering with 20%. Electrical and Computer engineering combined ranks second in number of students with 16% while Civil, Chemical and Industrial engineering combine for another 20%. In terms of fairness to the most number of students the argument can be made that the choice of programming language should be based on the lowest common denominator of benefit across all of the engineering majors. In other words, the language that is easiest to learn and has the highest benefit to the student for the effort expended in learning the language. This would suggest a language such as MATLAB, OCTAVE, R or Python be chosen for the introductory programming course. Secondary considerations are the cost of the language itself (MATLAB can be rather expensive for a large number of students) and the availability of supplementary teaching materials (OCTAVE does not seem to be supported as well as MATLAB in this regard).

Another possibility to get around the problem of choosing a language is to have different sections of the introductory programming course using different languages depending on the student's major. For example there could be one or more sections that use C++ if the student is majoring in electrical or computer engineering and separate sections using MATLAB (or similar) language for all other majors. To increase flexibility any student should be allowed to choose which section they prefer to enroll in as it is increasingly common for some students to have already taken a programming class in high school or learned a programming language on their own.

The decision to have multiple sections using different programming languages could be impacted by the number of engineering students at a particular university as it might not be practical for a smaller university with a smaller number of students to implement this approach. However in a large university environment where there are a large number of engineering students this type of specialization might be the best choice to meet the needs of the students across all engineering majors. If uniformity of course sections is preferable for administrative or other reasons (ability to change majors) a hybrid approach is also possible where each section teaches both a high level and a low level language.

One final consideration on the choice of programming language would be the size of the classroom where the programming course is taught and the number of students enrolled in each section of the course. As the number of students increases in each section the ability of the instructor to give individual attention to each student diminishes proportionately. There can also be a wide variation of the abilities of the students within a given section of a course. In a university where there is a relatively large number of students (for example > 15) in each section it might be preferable to choose a higher level programming language as the lower level languages will potentially require more individual attention from the instructor.

9. SUMMARY AND CONCLUSIONS

There does not appear to be any single programming language that is best for teaching an introductory programming course for all engineering students. One approach is to use a general purpose programming language such as C, C#, C++, Java or Visual Basic to provide a rigourous understanding of programming concepts. These languages tend to have a more difficult learning curve and more syntactical constraints which can end up being a pedagogical impediment to learning the concepts of programming. The benefit of this approach is that the student will hopefully be able to transfer to their skills to other languages especially the easier languages that they might use for their courses. The other approach is to choose a language that is easier to learn and has the highest benefit to cost ratio in terms of the effort involved. This can be considered as the lowest common denominator approach to choosing a programming language which would attempt to benefit all engineering majors as equally as possible. Programming languages that fit into this category are MATLAB, OCTAVE, R and PYTHON. All of these have built-in facilities for graphical display of data which is an important advantage since students will be able to use these capabilities for both their homework and for analyzing lab data.

REFERENCES

"Criteria For Accrediting Engineering Programs (2014-2015)", Accreditation Board for Engineering and Technology (ABET), October 26, 2013 .

"Criteria For Accrediting Computing Programs", (2014-2015). Accreditation Board for Engineering and Technology (ABET), October 26, 2013.

Yoder, B. L., (2011) "Engineering by the Numbers", ASEE (American Society for Engineering Education, http://www.asee.org/papers-and-publications/publications/college-profiles/2011-profile-engineering-statistics.

Kelleher, J.C. and Pausch, R., (2005) "Lowering the Barriers to Programming: A Taxonomy of Programming Environments and Languages for Novice Programmers", ACM Computing Surveys. 37(2) 83-137.

Bjedov, G. and Anderson, P.K., (1995). "Should Freshmen Engineering Students Be Taught a Programming Language", Frontiers in Education (FIE) Conference, Salt Lake City,1996.

Yacob, A. and Saman, M., (2012). "Assessing Level of Motivation of Learning Programming Among Engineering Students", International Conference on Informatics and Applications (ICIA) , Malaysia, 425-432.

Mannila, L. Peltomaki, M. and Salakoski, T., (2006). "What About a Simple Language? Analyzing the Difficulties in Learning to Program", Computer Science Education 16(3), 211-227.

Kernighan, B.W. and Ritchie, D.M., (1988), *The C Programming Language*, 2nd edition, Prentice Hall.

Stroustrup, B., (2013) *The C++ Programming Language*, 4th edition, Addison-Wesley.

Bronson, G. (2010). *C++ For Scientists and Engineers,* 3rd edition, Course Technology/CENGAGE Learning.

Pratap, R. (2005). *Getting Started With MATLAB 7: A Quick Introduction For Scientists and Engineers"*, Oxford University Press.

Briggs, J.R., (2012), *Python For Kids: A Playful Introduction to Programming*, No Starch Press.

Matsumoto, Y. and Flanagan, D. (2005) *The Ruby Programming Language*, O'Reilly Media.

Knell, R. (2013), *Introductory R: A Beginner's Guide to Data Visualization and Analysis*, Robert Knell.

Zak, D. (2013) *Programming with Microsoft Visual Basic 2012*, Course Technology/CENGAGE Learning.

http://www.mycplus.com/featured-articles/hello-world-programs-in-300-programming-languages/

Wikipedia(2014), Hello World Program Examples, https://en.wikipedia.org/wiki/Hello_world_program_examples

Visual Basic Language Specification 11.0 (2013), Microsoft, June 7, 2013.

Gosling, J. and McGilton, H. (1996), "The Java Language Environment", Sun Microsystems White Paper, 1996.

Moler, C. (2004), "The Origins of MATLAB", MathWorks, www.mathworks.com.

van Rossum, G. (1993), "An Introduction to Python for UNIX/C Programmers". Proceedings of the NLUUG najaarsconferentie (Dutch UNIX users group).

Ihaka, R. and R. Gentleman (1996). "R: A language for data analysis and graphics," Journal of Computational and Graphical Statistics , **5** , 299-314.

Authorization and Disclaimer

Authors authorize LACCEI to publish the paper in the conference proceedings. Neither LACCEI nor the editors are responsible either for the content or for the implications of what is expressed in the paper.