Improving Motivation of Students in Introductory Programming Courses

Jeffrey L. Duffany[1], Ph.D.
Universidad del Turabo, Puerto Rico, U.S.A., *jeduffany@suagm.edu*

Abstract– Research has shown that many students struggle in introductory programming classes often doing poorly or failing the course outright. However very little has been done to suggest ways for improving the situation. Given that there is little control of the variation in the level of preparation of the incoming student for this course one possible approach to improving the student performance would be to focus on increasing the motivation of the student. One method for doing this would be to provide evidence that they are learning a useful tool that will in the long run save them time and effort and potentially improve their grades. Another method is to capture their imagination and inspire the students to be self-motivated out of their own curiosity and desire to learn.

Keywords-- programming, motivation, active learning.

I. Introduction

Researchers studying students majoring in computer science have found that many fail their introductory programming class[1][2]. The average pass rate across 161 courses in 51 universities in 15 different countries on the average found only a 67% pass rate[1]. This means that around 33% of computer science majors fail introductory programming on the first try. This same result is found regardless of the programming language used in the course[1]. This seems rather counterintuitive since computer science students would appear to have a predisposed motivation to want to learn programming as there is an expectation that they may need to learn it to fulfill their degree requirements.

It is not clear exactly how this result would extrapolate to engineering students as the two groups of students are not necessarily directly comparable. The exact number may be different for engineering students but the fact of the matter is that a significant number of engineering students struggle and/or do not do well in their introductory programming class. Factors in doing well no doubt include prior preparation[6][9][10] however the result of not doing well in any class can be due other issues such as: not paying attention, not doing the assignments, not keeping up with the work, etc. Like missing the first five minutes of a movie and then not understanding fully what is happening in the movie later on. You can watch a movie over again but you easily cannot replay your programming class. In every programming class the students should be given some practice exercises either in class or for homework so they can solidify their understanding of the programming concepts and get used to the myriad of details that need to be taken care of to make things work. A computer language is a language and requires some practice to master it[18][19].

Digital Object Identifier (DOI): http://dx.doi.org/10.18687/LACCEI2018.1.1.349
ISBN: 978-0-9993443-1-6
ISSN: 2414-6390

II. MOTIVATION TO LEARN PROGRAMMING

Human behaviour is driven by motivation which can be internal or external[3][4][5]. People do things for a reason and in many cases this is internally motivated to acheieve a desired goal or a need to conform socially. On the other hand external motivation comes from a recognized authority such as the government or one's parents. Many people comply automatically even if they do not understand the reason simply to avoid the negative consequences. Take for example a charity fund. Someone may donate because they are asked to donate (external motivation) or they may wish to donate because they believe in the cause (internal motivation).

No one questions the need for engineers to study mathematics. However there appears to be a general misperception among students regarding the necessity of introductory programming course. Is it just mental calisthenics or exactly what purpose does it serve? Are all engineering disciplines being lumped in together and all being forced to take programming whether they need it or not? Is there some inherent mistrust on the part of the student as to the wisdom of whoever developed the curriculum?

According to the ABET(American Board of Engineering Technology) all engineering disciplines must be taught the modern tools of engineering but in general it does not specify exactly what those tools are[7][8]. Engineering schools can at least partially comply with ABET by requiring an introductory programming course for all engineering disciplines[7].

It is important for the students to know on the first day of class that the goal is not to try to make them a professional programmer. There is software in many products (over 100 million lines of code in a car) and that as an engineer you might have to work with software development for example in interdisciplinary teams. Students should be told about the SDLC (Software Development Life Cycle) and the fact that they may be part of the process (for example setting requirements phase or inspecting code). Engineers must have a basic idea about programming even if they never write a single line of code throughout their entire career. As engineers it will help them if they have a general knowledge about the inner workings of software. As such it is a good idea to clarify the reasons for the course and get the students to buy into it from the beginning.

III. Motivational Methods

Unfortunately the rather vague justifications just cited for learning programming may not be sufficient. Fortunately there are other methods that can be used to provide motivation for learning introductory programming. The two main techniques to be discussed are: (1) to illustrate tangible benefits of knowing how to program and (2) to capture the imagination or inspire the students which can lead to an inner self-motivation. In addition it must be demonstrated that the effort involved in doing this and in acquiring the skills is not excessive in terms of the effort required and the tradeoffs involved. Since students taking the course are likely to be in their first semester it may not be possible to find something they are knowledgeable about however you can plant the seed of an idea in hope that they will remember it later on. The important thing is that the student have a motivation to take the learning of programming more seriously.

Another possible alternative technique is to find a way to make programming more entertaining or to "gamify" it or otherwise make it more interactive and appealing[13][14]. Unfortunately this approach can require significant amounts of effort and may not be easily applied to any particular subject matter with equal success.

IV. Illustrative Examples

The main idea is to show students how easily written and relatively short computer programs can be used to help them solve homework problems and perform computations for lab reports in courses such as physics and chemistry. The following were chosen to be illustrative examples:

A. Potato Trajectory
B. Calculation of π
C. Electrical Circuits
D. Finding Roots
E. Probability and Statistics

These examples were chosen to be easy to program and at the same time provide tangible evidence to illustrate how programming skills are likely to benefit the student later on. There are many other examples that could have been chosen so these should be considered mainly as representative.

A. Potatoes in the Parking Lot

It is assumed that all students have seen either a football game or a baseball game, basketball game or soccer game depending on the popularity in the culture. In the game of football the ball is frequently passed or kicked and according to the laws of physics the path of the ball theoretically follows a parabolic trajectory. A discussion can be held about parabolas and how they show up in a variety of contexts as a way to engage interest and inspire the imagination.

A classic physics lab is to give a team of students a slingshot potato launcher. The team goes outside to an open field or athletic field or perhaps a parking lot with a sack of potatoes and the slingshot to study projectile motion. A team consists of ideally 3 students: two to launch the potato with the slingshot and one to make the measurements which includes total time in the air and horizontal distance traveled. The goal is to use the measurements to calculate what must have been the initial velocity v0 and the angle at which the potato was launched (theta). The angle of launch can be between 0 and 90 degrees but typically between 30 and 60 degrees. One student measures with a stop watch or cellular phone the time in the air. A rope is used to measure the total distance travelled along the ground. According to the laws of physics the vertical and horizontal displacements as a function of time (t) are given by:

$$x = v0*t*\cos(theta) \quad (1)$$

$$y = v0*t*\sin(theta) - 1/2gt^2 \quad (2)$$

A very simple computer program for calculating the values of x and y was written using R Language[15] as shown in Figure 1 below. There is nothing special about R language and many other languages could have been used[16][17].

```
potato()
{
 n<-100
 x<-rep(0,n)
 y<-rep(0,n)
 theta<-pi/10
 v0<-88
 max<-1.2*v0*sin(theta)/16
for (i in 1:n){
 t<-(i-1)*(max/n)
 x[i]<-v0*cos(theta)*t
  y[i]<-5+v0*sin(theta)*t-16*t^2}
  y[y<0]<-0
 plot(x,y,xlab="distance(ft)", ylab= "height
 (ft)", main="potato trajectory")
}
```

Figure 1. R Language Program to Plot Potato Trajectory

This short 12 line program can calculate and plot the trajectory of a potato launched from a slingshot for any initial velocity and launch angle as specified by the user and based on Equations (1) and (2). It automatically calculates when the potato will hit the ground and dimensions the plot accordingly. It also labels the plot axes and gives it a title. The user can also specify the level of detail by specifying the time interval between trajectory samples.

Digital Object Identifier: (to be inserted by LACCEI).
ISSN, ISBN: (to be inserted by LACCEI).

Digital Object Identifier: (to be inserted by LACCEI).
ISSN, ISBN: (to be inserted by LACCEI).

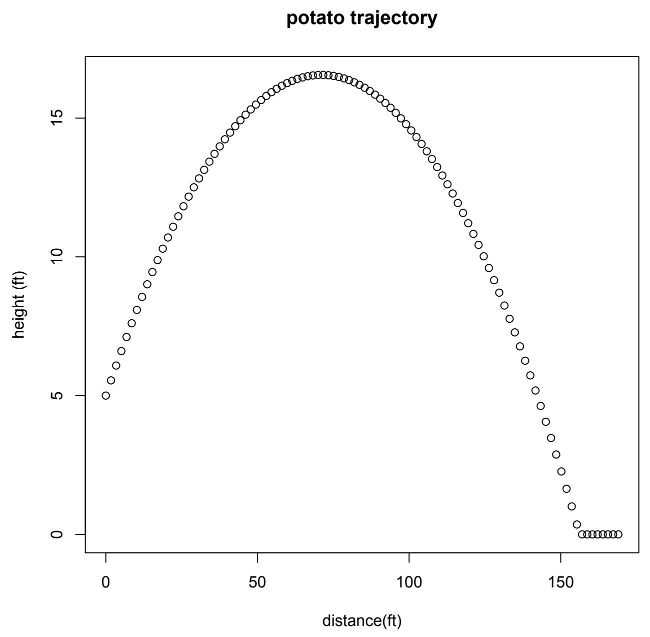

Figure 2. Potato Trajectory from Launch to Landing

The output of the program is shown in Figure 2 as a parabolic arc with the slingshot is represented as being about 5 feet off the ground. This is a fairly short computer program showing many important details (for example notice how the potato hits the ground and rolls a few feet on the ground). The student can easily add their own data and make the calculations specific to their own experiment. The student also can vary the initial velocity and launch angle and plot various trajectories on the same graph.

This base program can be modified to suit practically any variation of projectile trajectory experiment for example the effect of wind and/or wind resistance. The amount of effort to write and debug this program if fairly minimal even for a novice programmer it should only take about 20 minutes of effort. This example clearly illustrates that writing a simple program could save time and effort and possibly even lead to a better grade.

What should be emphasized to the students at this point is how this result was obtained with relatively little effort. This will plant a seed in the minds of the student hopefully that this technique can be extrapolated to other physics and other lab courses. Another observation that will show them the value of programming is the analysis of their data. In this case they can type all the data into the program itself. This works well for small amounts of data but does not scale very well. Later on in the course they will learn how to capture large amounts of data in a file that can be read into the program.

SUMMATION TERMS	PI APPROXIMATION
10	3.23231580940559 (0 DECIMAL PLACES)
100	3.15149340107099 (1 DECIMAL PLACES)
1000	3.14259165433954 (2 DECIMAL PLACES)
10000	3.14169264359053 (3 DECIMAL PLACES)
100000	3.14160265348972 (3 DECIMAL PLACES)
1000000	3.14159365358877 (5 DECIMAL PLACES)

Table 1. π approximation using Gregory-Liebniz formula

B. Calculation of π

The number π is a mathematical constant defined as the ratio of a circle's circumference to its diameter and approximately equal to 3.14159. A summation formula to approximate π is given by the Gregory-Leibniz series [20]:

$$\pi = 4/1 - 4/3 + 4/5 - 4/7 + 4/9 - 4/11 \ldots \tag{3}$$

Being an irrational number, π cannot be expressed exactly as a common fraction. In other words its decimal representation never settles into a permanent repeating pattern. Figure 3 gives the R Language code corresponding to Equation (3).

```
pi()
{
 sum<-1
 sign<--1
 for (i in 1:10){
 sum<-sum+sign/(2*i+1)
 sign<--sign}
 print(c("pi=",4*sum))
}
```

Figure 3. R Language Program to Calculate π

As individual terms of this infinite series are added to the sum, the total gradually gets closer to π (Table 1), and with a sufficient number of terms can get as close to π as desired. It converges quite slowly however as shown after 1,000,000 terms it produces only five correct decimal digits of π.

The purpose of this exercise is to stimulate the student's imagination and curiosity and create motivation for wanting to learn about computer programming. This can work out particularly well if the course is given in the spring semester and one of the classes happens to fall on March 14th (π day). It also encourages further exploration by the student. For example what other formulas can be used to approximate π? What happens when you increase the number of summation terms to 100 million or a billion. These ideas can be explored without an overbearing amount of effort to modify the code.

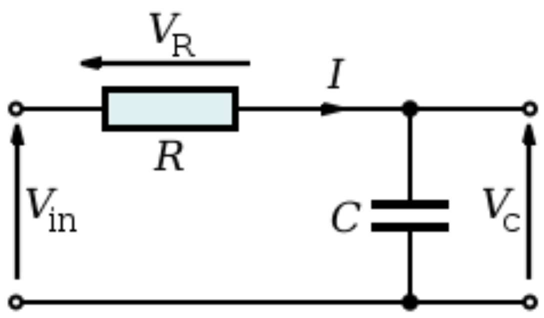

Figure 4. A Simple RC Circuit

C. Electrical Network Labs

All engineering students are required to take lab courses and many are required to take electrical networks lab including mechanical, electrical and computer engineering students. Students are required to make voltage and current measurements in the lab and then compare those to the theoretical calculations. Typically they start out with resistor circuits and usually they do node and mesh analysis and end up with either 3 (or 4) equations and 3 (or 4) unknowns. These circuits can be very tedious to set up solve and it is a very error-prone process. If you make a mistake you must go back and recalculate everything.

In the vast majority of cases the students would be far better off to write a computer program to do the calculations. Even if their professor requires they write the solutions out by hand at least they will have the output of the program to corroborate their results. The tedious and extremely error-prone substitution of variables method is replace by a simple matrix inversion. The student can perform both mesh and nodal analysis without much additional effort. They can also verify that Kirchhoff's laws are satisfied at each node and easily spot if any errors were made in in analysis. In addition they can calculate the current and voltage for each branch of the circuit to compare with their lab measurements.

A typical electrical network lab exercise is to measure the gain and phase characteristics of a typical RC circuit as shown in Figure 4 using the formulae given in Figure 5.

$$G_C = |H_C(j\omega)| = \left|\frac{V_C(j\omega)}{V_{\text{in}}(j\omega)}\right| = \frac{1}{\sqrt{1+(\omega RC)^2}}$$

$$\phi_C = \angle H_C(j\omega) = \tan^{-1}(-\omega RC)$$

Figure 5. Gain and Phase Calculation for an RC Circuit

The R language code to perform calculation for gain and phase for an RC circuit is shown in Figure 6.

```
rc()
{
 n<-1000
 r<-1
 c<-1e-6
 gain<-rep(0,n)
 phase<-rep(0,n)
 for (i in 1:n) {
 w<-2*pi*i*100
 gain[i]<-1/(1+(w*r*c)^2)^.5
 phase[i]<-(180/pi)*atan(-w*r*c)}
 plot(gain)
 plot(phase)
}
```

Figure 6. Program for RC Circuit Gain and Phase

This exercise be presented as an illustration of formula computation, storage arrays and plotting functions. Students do not have to know anything about circuits to write this program. It is included to plant the seed of an idea into the minds of the students about writing programs to help them with future classes that incorporate laboratory components.

Note once again the simplicity of the program in Figure 6. This program can be easily be edited to handle subsequent lab assignments (typically an RL circuit and RLC circuit) so you don't have to start from scratch each time you have a new lab assignment. The output of the program is shown in Figures 7 for the Gain (G) and Figure 8 for the Phase (P).

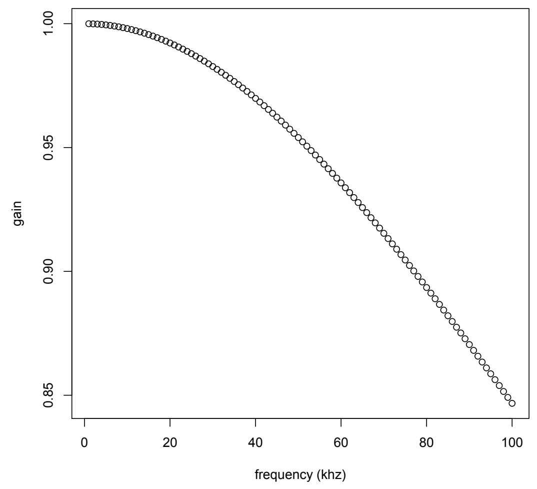

Figure 7. The Transfer Function |H| Gain for an RC Circuit

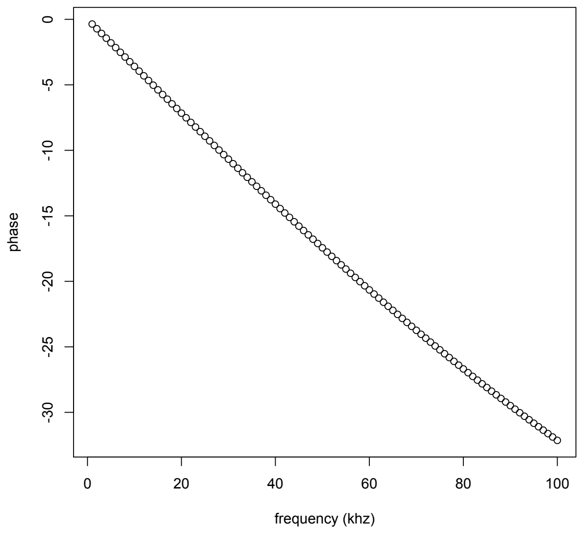

Figure 8. The Transfer Function Phase ϕ_C for an RC Circuit

Figure 7 shows the Gain (G) decreasing with frequency in a typical low-pass filter effect as the impedance of a capacitor (Z_c) decreases to zero at high frequencies. Correspondingly Figure 8 shows the phase shift ϕ_C going from zero to -30 on its way asymptotically to -90 degrees at high frequencies. The phase shift ϕ_C can be measured with an oscilloscope but getting an accurate reading is not easy. Students will benefit by making these calculations before going into the lab so they have a good idea of the results they should expect.

D. Finding Polynomial Roots

The factoring polynomial equations shows up in many engineering problems including circuit and system analysis, dynamical systems and system stability analysis. There are many techniques that can help to factor a polynomial (find the roots of polynomials) including the quadratic formula, polynomial division and Descartes rule of signs. Polynomial roots can be real, imaginary, rational or irrational. In some cases where it may be difficult to find the roots it can be helpful to plot the graph. This can be used with mathematics as well as engineering courses.

Equation (4) shows an example of a quartic polynomial:

$$f(x) = x^4 - x^3 - 6x^2 + 4x + 8 \tag{4}$$

whose 4 real roots are located at x = 2, x = 2, x = -1, and x = -2. There is a double root at x=2 and two negative roots one at x = -1 and the other at x = -2.

```
function ()
{
 n<-200
 x<-rep(0,n)
 y<-rep(0,n)
 a<-1
 b<--1
 c<--6
 d<-4
 e<-8
 for(i in 1:200){
    x[i]<-(i-100)/40
    y[i]<-a*x[i]^4+b*x[i]^3+c*x[i]^2+d*x[i]+e
 }
 plot(x,y)
}
```

Figure 9. R Language Program to Plot a Quartic Polynomial

Figure 9 shows a computer program for plotting a quartic polynomial written in R Language. It can easily be adapted to other degree polynomials. To use it just take the coefficients of the polynomial and enter them in the program as variables a,b,c,d, and e as indicated in Figure 9. The result of running the program on Equation (4) is shown in Figure 10.

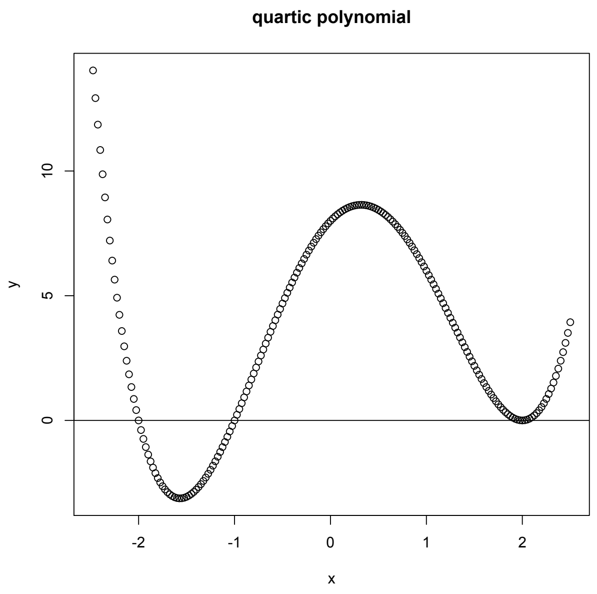

Figure 10. Plot of quartic polynomial in Equation (4).

Figure 10 clearly shows there are real roots f(x) = 0 somewhere around x= -1 and x = -2 and possibly x =2. The program can be used iteratively to identify or shed light on the location and possibly the nature of the roots (real, imaginary, irrational).

E. Statistics and Data Analysis

Frequently in university lab courses such as physics or chemistry data is taken during the lab and then analyzed somehow usually according to some mathematical formula. Using a calculator can work for small datasets but is very cumbersome and tedious for larger datasets. If an error is made the whole calculation may have to be done over again. Using a computer program can be much more efficient especially if any data entry or computation error is made.

Suppose you had for example a dataset containing the height (in cm) and weight (in kg) of a sample of 10 adults. Generally speaking the taller a person is the more they weigh. The students are asked to plot the data and draw a linear regression line that minimizes the mean squared error between the regression line and all of the data points. The following R Language program shows how the data can be entered into two vectors "height" and "bodymass".

```
regression ()
{
height <- c(176, 154, 138, 196, 132, 176, 181,
 169, 150, 175)
bodymass <- c(82, 49, 53, 112, 47, 69, 77, 71, 62,
 78)
plot(bodymass, height, pch = 16, cex = 1.3, col =
 "blue", main = "HEIGHT PLOTTED AGAINST BODY
 MASS", xlab = "BODY MASS (kg)", ylab = "HEIGHT
 (cm)")
abline(lm(height ~ bodymass))
}
```

Figure 10. R Language Program for Data Analysis

This R Language program creates the plot shown in Figure 11. The "plot" function creates the scatterplot of the original data, the "lm" function (linear model) does the linear regression and the "abline" function draws the straight line representing the linear regression. The linear regression line represents a predictive model in that given the height of someone you can predict their weight and vice versa.

V. DISCUSSION

In the previous section five examples (A-E) were given to illustrate how simple computer programs could be written to solve homework problems or to analyze lab data in courses that engineering students must take. It is likely that in general writing a program will save the student time in the long run allowing them to produce better results and in the bottom line help them to get better grades. That in and of itself should provide a positive motivation for learning the discipline of computer programming. Also there is a software reuse factor whereby the student can reuse code to solve similar problems thus saving even more time and effort.

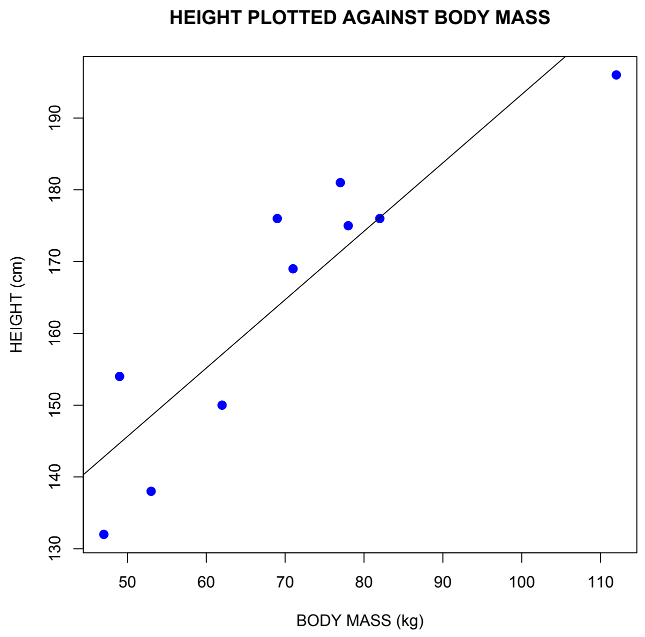

Figure 11. Body Mass vs. Height Regression Analysis

Ultimately the student must learn to judge whether it is worthwhile to write a program in any given case or not. Suppose it takes 3 minutes to solve a problem by pencil and paper but it takes 10 minutes to write a generic program to solve the same problem. Assume that it takes 30 seconds to enter the data in to program but it solves the entire problem in less than one second. If I am only going to solve this problem once then it probably does not make much sense to write a program for it. However if I am going to have to solve 10 of these problems then it could easily make a sense to just bite the proverbial bullet and write the program.

Another factor at work is that the paper and pencil approach assumes you never make a mistake and that is clearly an erroneous assumption. Human beings get tired and are more likely to make a mistake when fatigued however computers do not suffer the same disadvantage. If you make a mistake with the paper and pencil approach you have to do all the calculations over again. With the programming approach you can simply fix the error and run the program over again.

There is also another factor that needs to be considered and it is called the learning curve phenomenon. When you first start programming it might take you 30 minutes to write a simple program which could have a significant impact on the above analysis regarding whether to even write a program. If that is the case it would almost never make sense to take the programming approach. However this does not recognize the fact that that as your programming skills improve the time it takes to write and debug a program decreases which changes the whole dynamic.

VI. SUMMARY AND CONCLUSIONS

This paper describes a method for motivating students to learn programming by either (a) showing them how it can save time and effort and perhaps even lead to better grades or (b) by stimulating their intellectual curiosity.

Research has shown that many students struggle in introductory programming classes[1] often doing poorly or failing the course outright. However very little has been done to suggest ways for improving the situation. Introductory programming is one of those classes that builds on itself so if you do not get it the first time you may be in for serious difficulties later on. Students from all engineering disciplines take introductory programming and there is a wide range of abilities of the students coming in the class. Some may have studied some programming in high school or learned it on their own[6]. Others may have no exposure to programming at all and that can be rather intimidated. Recognizing this fact and assuming that we want students to be successful it would be unfortunate if a student transferred out of engineering because of the introductory programming class. Most high schools understand the need for college preparatory courses in physics and chemistry and math to help students be successful in college but not all high schools require programming and even today it is still more of an elective.

Given that there is little control of the variation in the preparation of the incoming students one possible approach to improving the student performance would be to focus on increasing the motivation of the student. One method for doing this would be to provide evidence that they are learning a useful tool that will in the long run save them time and effort and potentially improve their grades. Another method is to capture their imagination and inspire the students to be self-motivated out of their own curiosity and desire to learn.

The basic idea is that instructors should perhaps take a different approach from what they have been doing. Instead of going directly to teaching the programming language the instructor might spend some time proactively to illustrate to the student that it may be beneficial and that they are learning a tool. Many students say they never write code after the introductory programming course. The result is that you see many students still engaged in tedious paper and pencil hand calculations using their programmable calculator from high school with the drudgery of the calculations interfering with the learning process itself. Instructors should enable students by discussing several options that are either free or very inexpensive for example Microsoft Visual Studio free version, R Language MATLAB or Octave[9][15][16][17]. Instructors could also include examples of programming that can help students in their other courses for example in physics or chemistry or other courses they are likely to take.

Recently there has been a trend towards what is called "active learning"[11][12][18] where the passive teaching techniques are reduced to a minimum while the student is more "actively" engaged in the learning process through a variety of techniques. However for active learning to work the student must be motivated to learn in the first place.

REFERENCES

[1] C. Watson and F. Li, "Failure Rates in Introductory Programming Revisited", ITiCSE '14 Proceedings of the 2014 Conference on Innovation & Technology in Computer Science Education, Pages 39-44.

[2] J. Bennedsen and M. Caspersen, "Programming in Context: A Model-First Approach to CS1," Proceedings of the 35th SIGCSE Technical Symposium on Computer Science Education, Norfolk, 3-7 March 2004, pp. 477-481.

[3] C. Maharaj, E. Blair, S. Chin Yuen Kee, "The motivation to study: an analysis of undergraduate engineering students at a Caribbean university", Journal of Further and Higher Education, Volume 42, 2018 - Issue 1, Pages 24-35.

[4] Yacob, A. and Saman, M., "Assessing Level of Motivation of Learning Programming Among Engineering Students", Frontiers in Education Conference (FIE) 16(3), 2006, p. 211-227.

[5] L. Faessler, H. Hinterberger, M. Dahinden and M. Wyss, "Evaluating student motivation in constructivistic, problem-based introductory computer science courses", Proceedings of ELEARN 2006 p. 1178-1185.

[6] Y. Qian and J. Lehman, "Correlates of Success in Introductory Programming: A Study with Middle School Students", Journal of Education and Learning; Vol. 5, No. 2; 2016 ISSN 1927-5250 E-ISSN 1927-5269.

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

[8] "Criteria For Accrediting Computing Programs", (2013). Accreditation Board for Engineering and Technology (ABET), October 26, 2013.

[9] 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.

[10]R. ALTURKI, "Measuring and Improving Student Performance in an Introductory Programming Course", Informatics in Education, 2016, Vol. 15, No. 2, 183–204 183 © 2016DOI: 10.15388/infedu.2016.10

[11]R.M. Felder and R. Brent, "Active Learning: An Introduction." ASQ Higher Education Brief, 2(4), August 2009.

[12]Bonwell, C.; Eison, J. (1991). Active Learning: Creating Excitement in the Classroom AEHE-ERIC Higher Education Report No. 1. Washington, D.C.: Jossey-Bass. ISBN 1-878380-08-7.

[13]Domínguez, A., Saenz-de-Navarrete, J., de-Marcos, L., Fernández-Sanz, L., Pagés, C., Martínez-Herráiz, J.J. "Gamifying learning experiences: Practical implications and outcomes". Computers & Education. 63. pp. 380-392. http://dx.doi.org/10.1016/j.compedu.2012.12.020.

[14]de Freitas, S.&Oliver, M., 2006. "How can exploratory learning with games and simulations within the curriculum be most effectively evaluated?", Computers & Education, 46(3), 249-264.

[15]Crawley, Michael J. Statistics: An Introduction using R. Wiley, 2nd edition, 2014. ISBN 978-1-118-94109-6.

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

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

[18]Duffany, J.L. "Active Learning Applied to Introductory Programming", LACCEI 2015 Conference, Santo Domingo, Dominican Republic.

[19]Duffany, J.L. "Choice of Language for an Introductory Programming Course", LACCEI 2014 Conference, Guayaquil, Ecuador.

[20] Wikipedia page on PI: https://en.wikipedia.org/wiki/Pi.