Programming Education

Tim Hong-Chuan Lin, Jenny Zhen Yu

Department of Electrical and Computer Engineering,

California State Polytechnic University Pomona,

3801 W Temple Avenue, Pomona, CA 91768, USA

Abstract

Computer Programming is an essential skill for modern engineers since engineering products are realized or enhanced through software or firmware; examples include printers, web pages, office products, etc. Also, the curricula of engineering departments normally include at least one programming class, to prepare the students to write programs. C, Visual Basic, MATLAB etc. are examples of such languages for engineering students.

The authors have worked in industry for many years, wherefore the first author has worked in software industry for 20+ years and later has instructed several programming courses in the engineering college for another 20 years. The author knows there are many types of programming errors, from the mechanical ones that can be fixed using the right tools to more subtle style errors like stinky code etc. The author presents the errors as well as discuss the possible remedies to fix these through education..

Indexed Terms — programming errors, hard code, bad smell, programming proficiency.

1. Introduction

Our modern world consists of many computers, devices and parts made via engineering, that depend on engineering disciplines, but gradually have components or modules written in computer software or firmware and it is common understanding that economy of developed nations depend greatly on software.

Realizing the importance of such skill, the programming and software engineering courses have become part of the curricula for many engineering colleges. The level of competence and proficiency of programming skill however is important for the engineering student if they want to have that as a good asset in their repertoire.

We all know that many serious system problems were due to faulty computer software / firmware such as New York Blackout in 2003, Bank of New York overdraft of 32 billion dollars in 1985, and the ARIANE rocket failure in 1986 etc.

Since the successful software and faulty software are all developed by programmers, formerly programming students, it is the programming educators' responsibility to ensure students receiving programming education do not make faulty software, starting first from not making syntax errors, link errors etc. (which are mentioned in section 2 of this paper)

A paper from ACM SIGCSE (Association for Computing Machinery's Special Interest Group on Computer Science Education) had identified 13 syntax errors and 6 logic errors from surveying 58 schools about Java programming[1].

Programming students or even working programmers usually pay attention only to making working software, but not to making good software; whereas good software should not only be bug free but also should be easy to maintain (chapter 9 Evolution[2]).

From the educators' point of view, we know that the programming students can learn many constructs of programming *mechanically*, and know how to put them together as a working program, but not necessarily know how to verify if their programs (or some industry product) really work (see section 4.3.4 verify on a real example from student mistakes in doing their exams). They can produce *smelly code*, *hard code* (see section 3.2 of this paper) or can produce code that only works on their computer (using their C drive for example, see section 3.3.3). They also can submit the answer of their program for instructor review in some peculiar way like say just code, or just output, but not both (section 3.3 of this paper), which made the programming instructor's work hard. To show the extreme of a working but not acceptable code for educational purpose, professor Nyhoff[3] showed in his Data Structure book a small program that works but should by no means be a program (this is explained in section 3.1). All such code programs are examples of non-maintainable code which is error prone that can easily produce buggy code and then cause big problems sooner or later.

Below, Section 3.2 shows several examples of smelly code generated by the students that the instructor needs to be careful of not grading those as correct just because they generated the right result and section 3.3 shows defective submissions of programming students for the assignments and exams.

Section 4 of this paper presents some remedies to the programming problems with pair programming, instructor's fixes (educating the students on NOT doing the hard code approach, on teaching the students to do test and debug by incorporating several test cases in every programming assignment etc.

Section 5 concludes this paper.

2. Common Programming Errors (when students take the classes)

2.1 When do Programming Problems Happen?

Students of programming can have problems at the time when they take the programming class(es), or later on when they need to use their programming skills learned earlier in either a later class, capstone project in school, internship project, or programs when they start working as a software engineer.

The following lists the common errors in the first or second programming class. The students may know about syntax errors (the purpose of learning a computer language is to know, to avoid and / or to fix syntax errors), may or may not know about how to fix link errors (where to find the missing libraries), may know they have run time errors like division by zero, array out of bound.

A paper from ACM SIGCSE (Association for Computing Machinery's Special Interest Group on Computer Science Education) had identified 13 syntax errors and 6 logic errors from surveying 58 schools about Java programming. [1]

2.1.1 Syntax Errors

Any thing not following the language specs, such as no semicolon at the end of C instructions, left brace without matching right brace, variables starting with numbers, etc.

2.1.2 Link Errors and Load Errors

Link errors are usually caused by missing libraries (like say the main function does not exist). The author instructed computer network labs and when winsock (windows socket) library is used, there is special way of including the libraries in say Visual Studio to make the program link well (ws2_32.lib needs to be specified, the library name is cryptic, right?).

2.1.3 Run Time Errors

This can be division by zero, array out of bound, wrong data type etc., that usually can be treated by exception handling.

2.1.4 Logic Errors

This means software bugs; code that does not work as specified. This is the hardest one to fix. Most if not all software contain bugs to some extent.

3. More Advanced Programming Problems (stinky, smelly, or bad code)

Programmers can work hard to get rid of errors as mentioned in section 3. However, they may fall trap to write *bad code*, i.e. code that works, but not coded in a professional way. They could be called stinky code, smelly code etc. Sommerville mentioned in his software engineering book, code with bad smell [2] (Sommerville's writing originally referred to Fowler's book in 1999 [6])

Spaghetti code is one such example of bad smelly code (nobody likes to read spaghetti code, even though you know it works).

The programming problems mentioned in this section refer to the code that seems to work, i.e. it produces the correct answer; but it is no good in a professional way.

3.1 Spaghetti code example: From Dr. Larry Nyhoff's exercise.

Nyhoff showed an interesting example of spaghetti code in the exercise of his Data Structures Book [3] in Fig. 1: (he had asked the students to rewrite that as structured code)

```
int row = 0, col;
A: col = 0;
     If (col < n) goto B;
     goto A;
B: if (row < n) goto C;
     goto E;
C: if (mat[row][col] == item) goto D;
     col ++;
     If (col < n) goto B;
     row ++;
     goto A;
D: cout << "item found\n";
     goto F;
E: cout << "item not found\n";
F: ;
```

```
boolean found = false;
double  mat [n][n]; // declare and define
A square matrix of n x n
for (int row = 0; row < n; row ++)
      for (int col = 0; col < n; col ++)
            if (mat[row][col] == item)
                  {
                  cout << "item found";
                              found = true;
                              break;
                    }

// after finishing the nested for loops
if (!found)
   cout << "item not found";
```

Figure 1. (a) Spaghetti code (b) structured version of (a)

Explanation: The code in Figure 1(a) is the spaghetti code that follows the spec of searching for an item in a matrix, however, it is entangled with so many goto's so that the idea of the program is hidden in the moving arounds. The code in Figure 1(b) does the same thing without the confusing goto's.

3.2 Smelly code: hard code or special logic code

Smelly code, or stinky code means code that works or runs in principle, but is not considered correct way of doing programs since they used the wrong approach, used hard code, or tried with lots of pains with the code not generalizable or scalable for different situations. Followings are a few smelly code examples from the author's students' homework or exams.

- Example 1. Consider this freshmen programming assignment: Compute and display prime numbers up to n with n = 100 and display 8 in a row.

3.2.1 Find Prime Number: Problematic Hard Code approach

The normal design would be: write a loop (for loop or while loop) and check all integers k from 2 up to the integer of interest n exclusive (or 100, 200 etc.) to see if k divides the integer n (note the range would be 2 to n-1, or 2 to n / 2 (half n), or best, the square root of n).

However, a student of the author had code like Fig. 2 below. Will that be considered a correct program to generate all prime numbers up to 100 (and then 200, 300 etc.?)

```
cout << "2 is a prime" << endl;
// print 2 is a prime and next line
cout << "3 is a prime" << endl;
…..
cout << "89 is a prime" << endl;
cout << "97 is a prime" << endl;
```

Figure 2. Student code of program to generate prime numbers using brute force or hard code

This is the "hard code" approach: the student manually calculated 2, 3, 5, 7, …, 89, 97 and typed these one by one. Should this be considered a correct answer? (of course, it would be hard if not impossible to use the same approach to print all prime numbers < 1,000, < 10,000 etc.

3.2.2 Problematic Special code or magic number approach for prime numbers

Another interesting way of doing this program is: check if the integer n is divisible by 2, 3, 5, or 7. If not, then this is a prime number. This is also a wrong way, but not that easy to discern.

There is a seemingly smarter way of doing this problem by C as in Fig. 3:

```
int main() {
// This program computes all prime numbers less than 100
// by checking if 2, 3, 5, or 7 divides that number
        int n = 100;
for (int i = 2; i < n; i ++)
        if (i == 2 || i == 3 || i == 5 || i == 7)
                cout << i << " is a prime number" << endl;
        else // i is a prime number if 2, 3, 5, or 7 does not divide that
                if (i%2 != 0 && i%3 != 0 && i%5 != 0 && i%7 != 0)
                        cout << i + << " is a prime number" << endl;
}
}
```

Figure 3 Another example with "bad smell" for computing prime numbers

Explanation. This approach turns out to be working (printing out all primes < 100) for n = 100, but does not work for n = 200 since 121, as the square of 11, is not a prime, it is not divisible by 2, 3, 5, or 7 (unless the code is now "enhanced" to check if the integer n is divisible by also 11 and 13). Special numbers or magic numbers like 2, 3, 5, and 7 are used.

3.2.3 Another special code approach example

Example 2: Consider now a different still freshmen level programming assignment: **Compute** the sum of digits for an integer of variable length. Suppose we are computing the sum of digits for 1234 and also 12345.

```
sum = 0;
while (n != 0) {
sum += n % 10; // add last digit
n = n % 10;
}
```

```
sum = 0;
If (n < 10)
sum += n% 10;
else if (10 <= n < 100)
// code to process 2 digit integer
else if (100 <= n < 1000)
// code to process 3 digit integers.
```

Figure 4 (a) Correct code, (b) Pseudo (wrong) code handling integers as 1 digit, 2 digits, 3 digits, 4 digits etc.

Explanation. The correct code is as in Figure 4(a) that employs basic constructs like while loop, and arithmetic instructions + and &. The smelly approach in Figure 4(b) uses different ways to process integers of 1 digit, 2 digits, 3 digits etc.

3.2.4 Two different ways of reverting an integer. Which way is right?

```
int rev = 0;
while (n != 0) {
rev *= 10; // multiply by 10
rev += n%10 // add last digit
n = n / 10;
}
cout << rev;
```

```
int rev =0;
while (n! = 0) {
rev = n%10;
cout << rev;
n = n/10;
}
```

Figure 5 (a) Correct code (b) Wrong code that looks simpler than (a)

Explanation: The correct way in Figure 5(a) computes the reverted integer; the wrong way in Figure 5(b) displays the last digit of the integer starting from the last digit, one by one. If say we use n = 1234, both (a) and (b) will display 4321, but (a) generates 4321 in the final variable rev, while (b) only displays 4, 3, 2, and 1 in that order without generating any number. Using optics as an analog, (a) generates a real image, while (b) generates a virtual image.

3.3 Defective submission of code

It is not always easy to receive the student programmers' code correctly. There are many possible ways, some of them are actually really funny.

3.3.1 Send a link

The student's answer to the programming question is "my answer code is uploaded to this Github link (or some other link)". The student did NOT show any output from such code; the instructor needs to download from say Github (and create a Github account if the instructor does not have yet), run, and check if the output is correct.

3.3.2 Show a folder only

The student either answered in a Word / PDF file by saying "my answers are in the attached folder (or attached source files)" or only submitted folders of code without any answers to the programming assignment, shown in Fig. 6.

(a)	(b)
Exercise 1 (Q1)	Q1
Exercise 1 (Q2)	Q2
Exercise 1 (Q3)	Q3
Exercise 1 (Q4)	Q4
Exercise 1 (Q5)	Q5
Exercise 1	

Figure 6.(a) Proper way with a solution (b) Improper way with code, but no answer in Word or PDF

3.3.3 Other peculiar ways that the student may present their answers

- Show code only in Word / PDF

The student could just copy their code of many lines (sometimes 1 or 2 page long) into the answer, without outputs and explanation. It in general obscures where the answer of the student is. There is no hint of if the code runs, or what the output may look like from the student (the student may not show the outputs of the code at all).

- Show code with hard coded folder of student's computer

The student could have a line of code like below (if the student's name is John Doe)

`JohnsFunction (C:\\JohnDoe\\ Johnsexecutable)` as an absolute path.

This code runs on the student's computer because the student has access to the folder JohnDoe; however it does not run on the instructor's computer or any other computer.

The correct way is `JohnsFunction (“..\Johnsexecutable”)` as a relative path.

- Show outputs only (without turning in the code)

The student could turn in the output, but did not turn in the source code. (the code could be hard coded to generate the expected output).

- Turn in PNG file or JPG file for the result of each question

4. Remedies or Corrections for the Problems

There were papers suggesting ways to fix software coding problems mentioned in section 2 and section 3 above [56]. There seems to be no culture or environment that shows a consistent and universal way in the programming education that educate the programming students to not only write working code (code that gets rids of the many types of errors in section 2) but also to write effective and professional code that does not have the problems mentioned in section 3. The students should also submit their answers to the instructors in a professional way. The following subsections of this section 4 shows the author's understanding of possible different approaches (as the first attempt? since the author never heard about or hardly heard about this) to bring a better programming environment academically and to have students better educated in programming, which is an essential skill for the student, and also an essential element to make the world advancing in the right way and the right speed. The author recalls a retired colleague's comment: “Engineering students used to be good in hardware skill, but poor in software skill; but they are now poor in both hardware skill and software skill” (this comment was made about 10 years ago, around 2013).

4.1 Pair Programming (Agile Development)

Pair programming of having two programmers working in a team, reviewing or testing each other's code, or practiced in agile development (chapter 3[2]) is a good way that trains the programmer or the student trying to understand the other programmer's code. The author had tried pair programming in some programming course and also had covered the concept of agile development with the sprint cycle and scrum master concept in the Software Engineering course. It seems that the students received such concepts well and could conceivably use this if they work in the software industry later.

4.2 Software Refactoring.

The good theoretical ways would be through software refactoring methodology as we can see in the 3rd, 5th and 6th references later on [4-6].

4.3 Immediate Fixes by Instructors (or Managers)

Pair programming or extreme programming practiced in agile development
The quick way of fixing the problems is through *programming education*. This means through the direct interface of instructors with the students (or managers with the employees) by telling them what they did wrong and probably show them what the right way is.

4.3.1 Spaghetti code or magic number code.

The spaghetti code or smelly code mentioned in section 3.1 and sections 3.2.1, 3.2.2, 3.2.3, 3.2.4 etc. can be straightened out through the instructor interactions with the students or via the ways in reference 3 and 5 [35].

4.3.2 Right way of submission:

The many peculiar ways of submissions mentioned in section 3.3 can be fixed via instructor interactions or via showing the students good examples like below of good one and best one in Figure 7 (in a class manual say).

Good one: Show the code (with a screen capture) and outputs (professional)

Best one: Show the code (with a screen capture) and outputs (professional) *plus explanation of the code*

```
1  //
2  //  HW1 Question1.cpp
3  //  Hw 1
4  //
5  //  Created by Anish Junnarkar on 2/17/19.
6  //  Copyright © 2019 Anish Junnarkar. All rights reserved.
7  //
8
9  #include <iostream>
10 #include <math.h>
11
12 int main() {
13    using namespace std;
14     int x;  // defining the function x
15     cout<<"Insert an integer!!"<<endl;
16     cin>>x;
17     cout<<"Your integer is x=" <<x<<endl;// x is the integer that is being show
18     cout<< "Two times your integer is "<< 2*x<<endl;//* multiplies the values
19     cout<<"The square of x is "<<pow(x,2)<< endl; //the pow function allows power
20     return 0;
21
22 }
23
24
```

```
Insert an integer!!
3
Your integer is x=3
Two times your integer is 6
The square of x is 9
Program ended with exit code: 0
```

For 12345:

Microsoft Visual Studio Debug Console

```
This program computes and displays the sum of the digits of the integer that you input

Please enter an integer to find the sum of its digits:
12345

The sum of the digits for 12345 is: 15
```

```
Pseudocode & Explanation:

Main function
    Display (This program computes and displays the sum of the digits of the integer that you
            input)
    Display (Please enter an integer to find the sum of its digits:) //ask user to input a number

    Int first Number = x
    Int sum = 0

    While loop (while x is not equal 0)
        Sum = sum + (x % 10) // this function keep adding the reminders of the inputted
            number when divided by 10
        x = x/10 // this function changes the value of x every time to the answer of the numbe
            divided by 10
    end loop
    display (The answer: the sum of the digits of the inputted number)
```

Figure 7 (a) Good one (b) Best One

Explanation. This would be a godsend if a student turned in the answers like this; it is very rate; but we hope the education or programming education can make more of this kind happen.

4.3.3 Test and Debug (to fix logic errors mentioned in section 3)

The student wrote code to compute say 5! (factorial five) since the program says (write a program or function to compute 5!). They did not know or bother to see if the code works more generally to compute say 10!, or 2! (or what happens if the user input -1, intending to compute the non-existing (-1)!.

The student also in general is not familiar with taking the advantage of the debugging tools (say the symbolic debugger in Visual Studio or in Java Eclipse) to locate and fix the logic error in the code that produces wrong outputs.

```
25
        1 reference
26      public void mul(Complex c1)
27      {
28          // use the formula (a+bj) * (c+dj) = (ac-bd) + (ad+bc)j
29          rP = rP * c1.rP - iP * c1.iP;
30          iP = rP * c1.iP + iP * c1.rP;
31      }  ≤1ms elapsed
32
        1 reference
33      public void mulfix (Complex c1)
34      {
35          // use the formula (a+bj) * (c+dj) = (ac-bd) + (ad+bc)j
36          Complex ccopy = new Complex();
```

No issues found

Autos

Name	Value	Type
c1	{Complex}	Complex
c1.iP	-1	double
c1.rP	1	double
iP	-1	double
rP	2	double

```
32
        1 reference
33      public void mulfix (Complex c1)
34      {
35          // use the formula (a+bj) * (c+dj) = (ac-bd) + (ad+bc)j
36          Complex ccopy = new Complex();
37
38          ccopy.rP = rP * c1.rP - iP * c1.iP;
39          ccopy.iP = rP * c1.iP + iP * c1.rP;
40          rP = ccopy.rP;
41          iP = ccopy.iP;
42      }  ≤1ms elapsed
43
        3 references
44      public void print()
45      {
```

No issues found

Autos

Name	Value	Type
ccopy	{Complex}	Complex
ccopy.iP	0	double
iP	0	double

Figure 8 (a) code of mul with iP = -1 (wrong), (b) code of mulfix with iP = 0 (correct)

Explanation: These two pictures show using debugging windows that the variable iP can have the wrong value -1 or right value 0 when multiplying 1 + i and 1 -i.

4.3.4 Verify

The following command in fig. 9 from a common software tool produces funny *(wrong*) result of trying to calculate the remainder of 23! divided by 17.

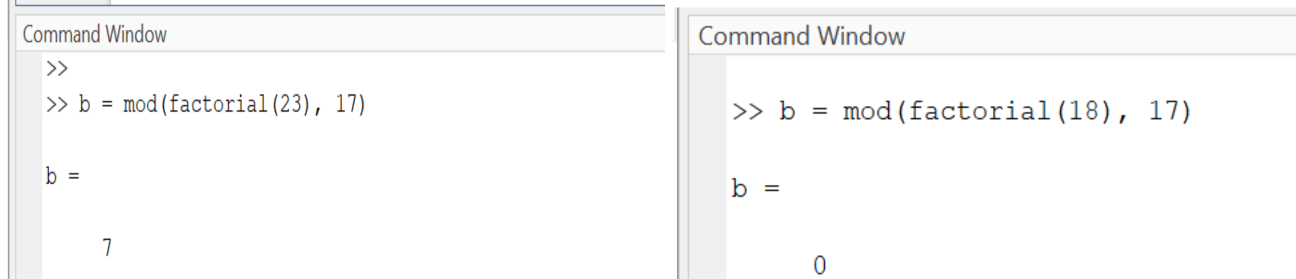

Figure 9. Outputs from command window (a) wrong with answer = 7, (b) right with answer = 0

Explanation. This was a midterm question where students answered wrong by using the tool carelessly without verification. (those students trying manually with mathematics or coding in C / Java etc. got the right answer 0).

Note here in Figure 9(a), it says 23!%17 = 7; but 17 is a factor of 23!, so 23!%17 should be zero, not 7.

So this is a tool error that many engineering students do not recognize as an error

Note: If you replace 23 by 18 in the command, you'll get the correct answer. This is possibly due to overflow in the tool used (if you do the same thing in C# / Java etc. it works fine)

4.3.5 Enhance

After the student achieved the level with the capability of writing code properly, presenting the programming results effectively, knew how to test and debug, and could verify the correctness of the answer, she / he won't be perfect unless they did the last step of enhancement, to make the code professionally looking and also making the reading of their results an enjoyable experience.

To achieve this level, the student is expected to, not only present the outputs, turn in the code, but also need to have some documentation of the program, either inside the program (the coding comments), or outside the program as a standalone document (such as detailed answer of a programming assignment in Word or PDF

4.4 Instructor's documented guides and examples / Company's guidebooks

This is enhancement of section 5.2 to have the instructors document such bad and good examples in a so called guidebook or manual for the students, so that the students have constant access to the good and bad examples and learn from that.

4.5 Chapters on programming books or web sites addressing such issues

The commercial programming books usually show examples of working code, but generally do not show examples of good working code in contrast with bad working code. We hope in the future, programming books or web sites cover these as well.

5 Conclusion

Programming is a very essential skill. The errors mentioned in section 2 as common errors and the advanced errors like bad smelly code should be avoided through education, from the instructors, from the books, or any other ways like internet. Yet students can acquire programming skills through formal education from courses or informal ones by learning themselves (such as picking up a language not instructed in school).

To really learn programming, the students need not only to know the basics and essentials, to get familiar with these, but also need to avoid errors (just like car drivers need to know how avoid accidents). A good percentage of programming instructions should be allocated for being conscious about errors (in the course curriculum), and know how to avoid and to fix errors, so that errors caused by software bugs can be reduced to a minimum. (there are many incidents from software bug, including the famous rocket problem caused by code bug).

By the way, the learning objectives defined by the authors' ECE department, include the followings that are related to programming: successfully practicing in the Electrical Engineering profession with solid theoretical and hands-on Knowledge of Circuits, Electronics, **Computer Software,** Hardware, Communications and Electrical Power.

Based on the learning objectives, student outcomes were defined, such as an ability to identify, formulate, and solve complex engineering problems by applying principles of engineering, science, and mathematics. With the proper programming education, this helped the students not only to solve problems via programming but also knowing the ways to verify and fix errors, they can really grab and make the programming skill an asset in their repertoire of professional expertise.

It should be a lifelong process of acquiring good habit of writing professional code.

Reference

1. "Identifying and correcting Java programming errors for introductory computer science students"; Maria Hristova, Ananya Misra, Megan Rutter, Rebecca Mercuri; ACM SIGCSE 2003 proceedings, pages 153-156.
2. "Software Engineering", 10th edition, Ian Sommerville, Pearson, 2015
3. "ADTs, Data Structures, and Problem Solving with C++ 2nd Edition", Larry Nyhoff, 2004, Pearson
4. "A survey of software refactoring", IEEE Transactions on Software, volume 30, issue 2, pages 126-139,
5. "Straightening Spaghetti-Code with Refactoring?", Markus Pizka, Software Engineering Research, 2004
6. "Refactoring: Improving the Design of Existing Code. " Boston: Addison-Wesley (now Pearson). Fowler, M., K. Beck, J. Brant, W. Opdyke, and D. Roberts, 1999