What Makes a Good Engineer?

One of the major concepts that people talk about in engineering is abstraction. Abstraction is when you use a product that someone else spent a lot of time developing, and you don’t have to spend time caring how it works.

An example is gravity: physics majors think of gravity as something like F = G\*m1\*m2/r2 but engineers think of it as F = 9.81m. We’ve abstracted away the complicated equation because other people can prove that on the surface of the earth, it simplifies to 9.81m.

Another example is high level programming languages, like C++, compared to hardware-specific assembly/binary code. Somewhere, at some point in time, somebody spent a lot of time writing a program (called a compiler) that converts a high level language into assembly languages for you. If you write a program in C++, you have abstracted away the assembly language because you can just run you text through this other guy’s compiler program and he’ll write the assembly for you. Every chip/machine/operating system will have a different assembly language; so, being able to write code in one universal language, like C++, and have that automatically translated into the real code for any computer or phone is a huge advantage of abstraction.

So abstraction is good because it saves you a lot of time and energy. Don’t reinvent the wheel.

Sometimes, however, abstraction can blind us from reality. Compilers try to optimize your C++ into the best assembly code possible, but it’s never going to be as good as the assembly code written by an engineer who understands a machine’s native assembly language. If you work for NASA, F = mg is just not going to cut it.

For example, try compiling the code “Ben is awesome” into 27 different languages at www.ackuna.com/badtranslator (seriously though, try this website). Eventually, Bing translates “Ben is awesome” (which represents C++ in this analogy) to

“Хорошее платье” see image

which is actually Russian for “nice dress”. Obviously you would never compile your program 27 times in series but still, if your goal was to compliment me, “nice dress” is not be the most optimized option.

So abstractions sometimes lead to results that are not 100% perfect. The question is, do you care? If you are designing playground equipment\* gravity can just be 9.81, no problem.

If you want a program for cars that can run on a small microprocessor, sense when accidents are occurring, and do so with speed and 100% accuracy, an assembly code compiled from C++ may not be good enough for you. Especially if you are going to sell many cars with the same microprocessor hardware, it makes sense to learn that particular assembly language and write the program at the low level, where you can understand and check for problems yourself.

So, after all that, here is my main point. I think there are three qualities that make up a good engineer:

1. Knowing the abstractions that people use in your engineering field.
2. Being able to decide when an abstraction is acceptable or not good enough.
3. Having the ability to go under the hood when you decide the abstraction not good enough.

Ninety nine percent of the work required to fix a problem occurs in 1 and 3, but the value of your final result is disproportionately dependent on 2: on how you decide to solve problem.

This is really what people are debating when they ask which is better, a new engineer or an experienced engineer. A new engineer has no experience to draw on when deciding how to best solve the problem. Chances are, they will either not make a working solution or they will waste lots of (fill in the blank) to reach a working solution. Old engineers have the opposite problem, they can be counted on to find a decent solution, but they are less likely to research new ways of solving the problem because they have already found a best way in prior experiences. But best ways can change.

Engineering programs are inherently good at teaching students number 1. That’s the entire point.

Engineering programs are inherently bad at teaching students number 2, because every class in engineering is about a specific type of abstraction. The decision is therefore easy: for problems in that class, the abstraction can be applied. And trick questions are usually pretty obvious.

Engineering programs are differentiated by how well they teach number 3.

There is also a network effect. If a school has a reputation and culture of teaching the subject in greater depth, more students will come who are motivated to learn at this level and these students will help motivate one another.

This is why I want to learn chemistry. I want to be able to go under the hood of electronics. If the part is not good enough, I want to make it better.

This is also what communication is all about. How can you agree on the best way to tackle a problem when everyone has had different experiences with the accuracy of 1 and difficulty of 3?

So obviously we should just take a class on decision-making and we will all become much better engineers, right? No, because the only way to make a good decision is to know why a particular method will or won’t work and what the alternatives may be. This paragraph isn’t going anywhere

\* for use on earth.