Infographic Explained & FAQ

#1. What do you mean by “writing very specific instructions to a dumb, yet obedient machine”?

Imagine this: You have to instruct a kid to shower. The kid only knows how to follow your instructions. So you ask the kid to:

1. Walk into the bathroom.
2. Turn on the shower.
3. Stand under the shower.
4. Take the soap.
5. And so on…

Oh wait, the kid didn’t even remove his/her clothes before entering the shower! That’s how programming is. Computer only does what you **exactly** tell it to do. It doesn’t know how to assume and never think about the consequences.

#2. What are some common misconceptions about programming?

* You don’t have to be a genius or good in math to be a programmer. You just need to be have good problem solving and logical thinking skills.
* Programmers do spend significant amount of time in front of the computer, but that doesn’t mean that programmers are good at fixing friends’ and family members’ PC.
* There’s no best programming language to learn. The answer varies greatly depending on your requirements, preference, time constraint, etc. Programming languages are like religions and some developers are more religious.

#3. Explain more about high-level and low-level programming.

Using the car analogy, high-level language is like driving an auto car. You don’t have to worry about shifting gears (similar to [garbage collection](http://en.wikipedia.org/wiki/Garbage_collection_%28computer_science%29) & [memory management](http://en.wikipedia.org/wiki/Memory_management) in programming). It’s easier to learn and drive (code) an auto car.

Middle-level language is like driving a manual car. You can shift gears (more control), but it’s harder to learn and drive.

Low-level language is like the car engine. You don’t have to understand how the car really works in order to drive. Similarly, you don’t have to learn how to code in [assembly language](http://en.wikipedia.org/wiki/Assembly_language) and [machine code](http://en.wikipedia.org/wiki/Machine_code) (literally impossible).

To sum up, computer is built on layers of [abstractions](http://en.wikipedia.org/wiki/Abstraction_(computer_science)). You drive a car through abstractions – gear shift, gas pedal, brake pedal, steering wheel, etc. The complexity of the vehicle and all the little details are hidden from you.

#4. Why is it hard to estimate software development time?

Here’re some great answers on [why are software development task estimations regularly off by a factor of 2-3](http://www.quora.com/Engineering-Management/Why-are-software-development-task-estimations-regularly-off-by-a-factor-of-2-3).

#5. What is the fixing mysterious bug about?

Fixing [software bug](http://en.wikipedia.org/wiki/Software_bug) is extremely common, which you will encounter daily. However, sometimes it would take hours or even days (yes, weeks and months are also possible) to fix a bug. Here are some reasons why:

* The code is badly written and it’s written by someone else. He/She might have left the company and you are unable to approach them anymore.
* Only occurs under certain conditions which are hard to replicate. For example, the bug only occurs when your website receives heavy traffic and reach certain threshold.

Now, pick your first programming language here: [Which Programming Language Should I Learn First [Infographic]](http://carlcheo.com/startcoding).