Core Concept #1 – Algorithms and Data Structures

1.1 – Big O Notation

Say you order Harry Potter: Complete 8-Film Collection [Blu-ray] from Amazon and download the same film collection online at the same time. You want to test which method is faster. The delivery takes almost a day to arrive and the download completed about 30 minutes earlier. Great! So it's a tight race.

What if I order several Blu-ray movies like The Lord of the Rings, Twilight, The Dark Knight Trilogy, etc. and download all the movies online at the same time? This time, the delivery still take a day to complete, but the online download takes 3 days to finish. For online shopping, the number of purchased item (input) doesn't affect the delivery time. The output is constant. We call this **O(1)**.

For online downloading, the download time is directly proportional to the movie file sizes (input). We call this **O(n)**.

From the experiments, we know that online shopping scales better than online downloading. It is very important to understand big O notation because it helps you to analyze the **scalability** and **efficiency** of algorithms.

Note: Big O notation represents the **worst-case scenario** of an algorithm. Let's assume that O(1) and O(n) are the worst-case scenarios of the example above.
More: Big O Notations (video), Plain English explanation of Big O, A Beginner's Guide to Big O Notation

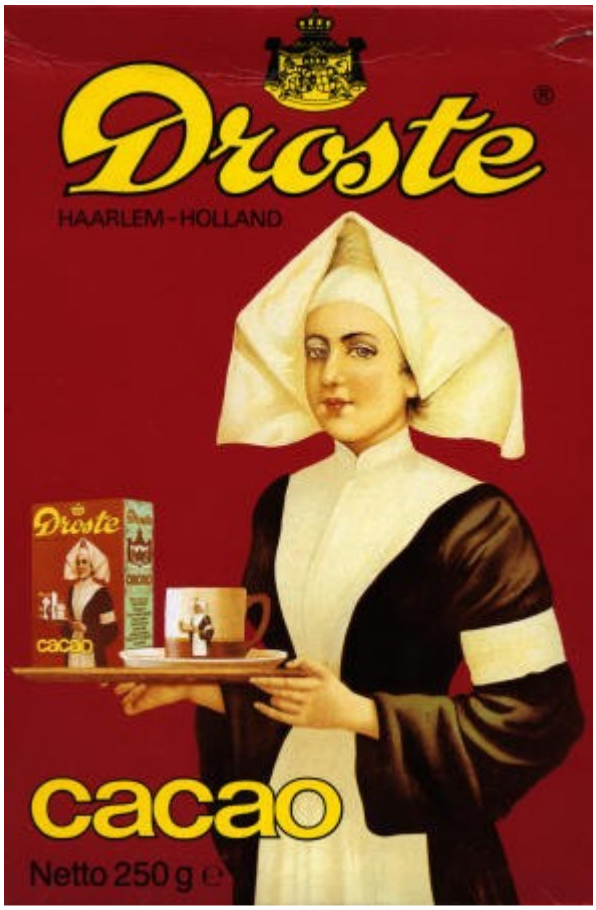
1.2 – Sorting Algorithms

More: Sorting Algorithm Animations, Beautiful and configurable visualizations of sorting algorithm

1.3 – Recursion

Someone in a movie theater asks you what row you're sitting in. You are too lazy to count, so you ask the person in front of you. You simply have to add 1 from the person's answer to get your current row number. Brilliant right? However, the person in front of you did exactly the same thing, and so on. Finally the question reaches row 1 and he answers: "I'm in row 1!". From there, the correct message (incremented by one each row) will pass all the way up to the person who asked.

Here's another example known as the Droste effect. A nurse is carrying a tray with a box of cocoa and a cup containing a smaller image of her holding the same thing, which in turn contains an even smaller version of the image, and so on.
Here are more Droste effect examples to get you drowsier.



If you still don't get what recursion is, check out… Otherwise, continue reading.

1.4 – Big Data

Let's assume you have a leak in a water pipe in your garden. You take a bucket and some sealing materials to fix the problem. After a while, you see that the leak is much bigger that you need a plumber to bring bigger tools. In the meanwhile, you are still using the bucket to drain the water. After a while, you notice that a massive underground stream has opened. You need to handle gallons of water every second.

Buckets aren't useful anymore. You need a completely new approach to solve the problem because the volume and velocity of water has grown. To prevent the town from flooding, you may need the government to build a massive dam that requires an enormous civil engineering expertise and an elaborate control system.

Big data describes data sets so large and complex that is impossible to manage with conventional data processing tools.

More: Big Data by TED-Ed (video), What is Big Data and Hadoop (video)

1.5 – Data Structures

Every computer scientist and programmer should at least know:

- Array
- Tree
- Stack
- Queue
- Graph
- Hash Table
- Linked List
- Heap

Core Concept #2 – Artificial Intelligence

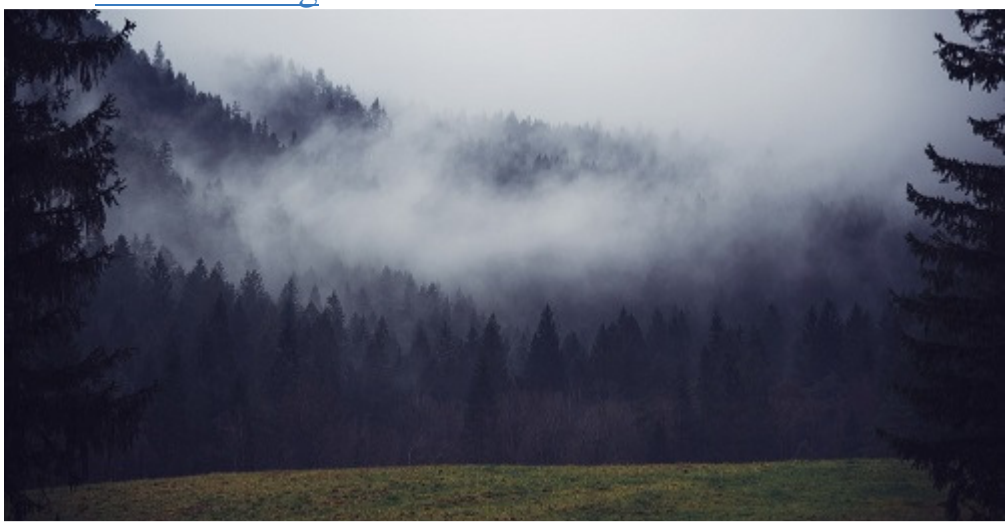
2.1 – Greedy Algorithm

Imagine you are going for hiking and your goal is to reach the highest peak possible. You already have the map before you start, but there are thousands of possible paths shown on the map. You are too lazy and simply don't have the time to evaluate each of them. Screw the map! You started hiking with a simple strategy – be greedy and short-sighted. Just take paths that **slope upwards the most**.

After the trip ended and your whole body is sore and tired, you look at the hiking map for the first time. Oh my god! There's a muddy river that I should've crossed, instead of keep walking upwards.

A greedy algorithm picks the **best immediate choice** and never reconsiders its choices.

2.2 – Hill Climbing



This time you're climbing another hill. You're determined to find the path that will lead you to the highest peak. However, there's no map provided and it's very foggy. To make your trips easier, you have downloaded a hiking app that track paths you've taken and measures your current altitude.

You climb the hill over and over again. Each time, you take the exact same path that leads you to the highest peak ever recorded, but somewhere in the **middle of your journey**, you choose a **slightly different route**.

You can also randomly choose a different starting point, which is known as **random-restart hill climbing**. So that you don't just linger around the same area and reduce your probability of getting stuck.

The hill climbing algorithm attempts to find a better solution by generating a **neighboring solution**. Each neighboring solution is generated based on the best solution so far, with a **single element modified**.

2.3 – Simulated Annealing

It's Mount Everest, the biggest challenge you've ever faced. Your goal is to reach the summit, but it's impractical to climb Mount Everest over and over again. You have one chance. You are more cautious now. Instead of always climbing upwards, you **occasionally move to a lower point and explore other paths**, to reducing your chance of taking the wrong path. The higher you climb, the lower the probability you move to a lower point and explore.

2.4 – Dynamic Programming

Dad: **Writes down "1+1+1+1+1+1+1+1 =" on a sheet of paper**
Dad: What's that equal to?
Kid: **counting and 3 seconds later** Eight!
Dad: **Writes down another "+1" on the left**
Dad: What about now?
Kid: **instantly** Nine!
Dad: Wow, how did you calculate so fast?
Kid: You just added one more!
Dad: So you didn't need to recount because you remembered it was eight before. Brilliant!

The example above describes memoization (yes memoization **not** memorization), a top-down approach in dynamic programming, which store the results of previous computations for future use.

More: Dynamic Programming – From Novice to Advanced (TopCoder), Tutorial for Dynamic Programming (CodeChef)

2.5 – Machine Learning

Pararth Shah wrote a brilliant analogy here, but it's too long to be included.

2.6 – P vs NP Problem

P vs NP one of the most popular and important unsolved problem in the computer science field.

Say I give you a multiplication question like:

Q1: 7 x 17 = *p*
The answer is 119. Easy to solve right? What if I reverse the question:

Q2: *p* x *q* = 119 (*p* & *q* cannot be 1 & 119)
To solve Q2, assuming that you haven't seen Q1, you probably have to go through all possible numbers from 2 to 118. We are **yet to discover an efficient algorithm** that can find the factors of a number easily.

What if I ask you: Could *p* possibly be 7? You can **easily verify** the answer right? Just divide 119 by 7!
Multiplication is easy. Finding the original factors of a number is hard.

So Q1 is a P (polynomial) problem because it is **easy to solve**. Computer can easily multiply 2 super large numbers without spending significantly more computer time than small numbers.

Q2 is a NP (nondeterministic polynomial) problem because it is **easy to verify**, but **hard to solve**. Finding the factors of 119 is still fairly easy for computer to solve, but how about a 500-digit number? It's impossible for any computers right now.

Here's the important part: Are NP problems (e.g., factorization) also P problems (e.g., multiplication), just that we haven't discover the efficient way to solve NP problems? Are NP problems really hard to solve, or we just need an "aha moment" from a brilliant scientist (or you?) to come out with an efficient algorithm? Or maybe humans are too dumb? Imagine there exist machine or life that possesses much higher intelligence than human. They see us like how we see ants. Our level of intelligence is too insignificant to them. Solving P vs NP problem is like solving 1 + 1 to them!

So why is P vs NP problem important? If we are able to prove P=NP, that means all NP problems can be solved easily within reasonable computer time. We will be able to cure cancer (protein folding), break passwords (RSA), etc. It's world-changing.

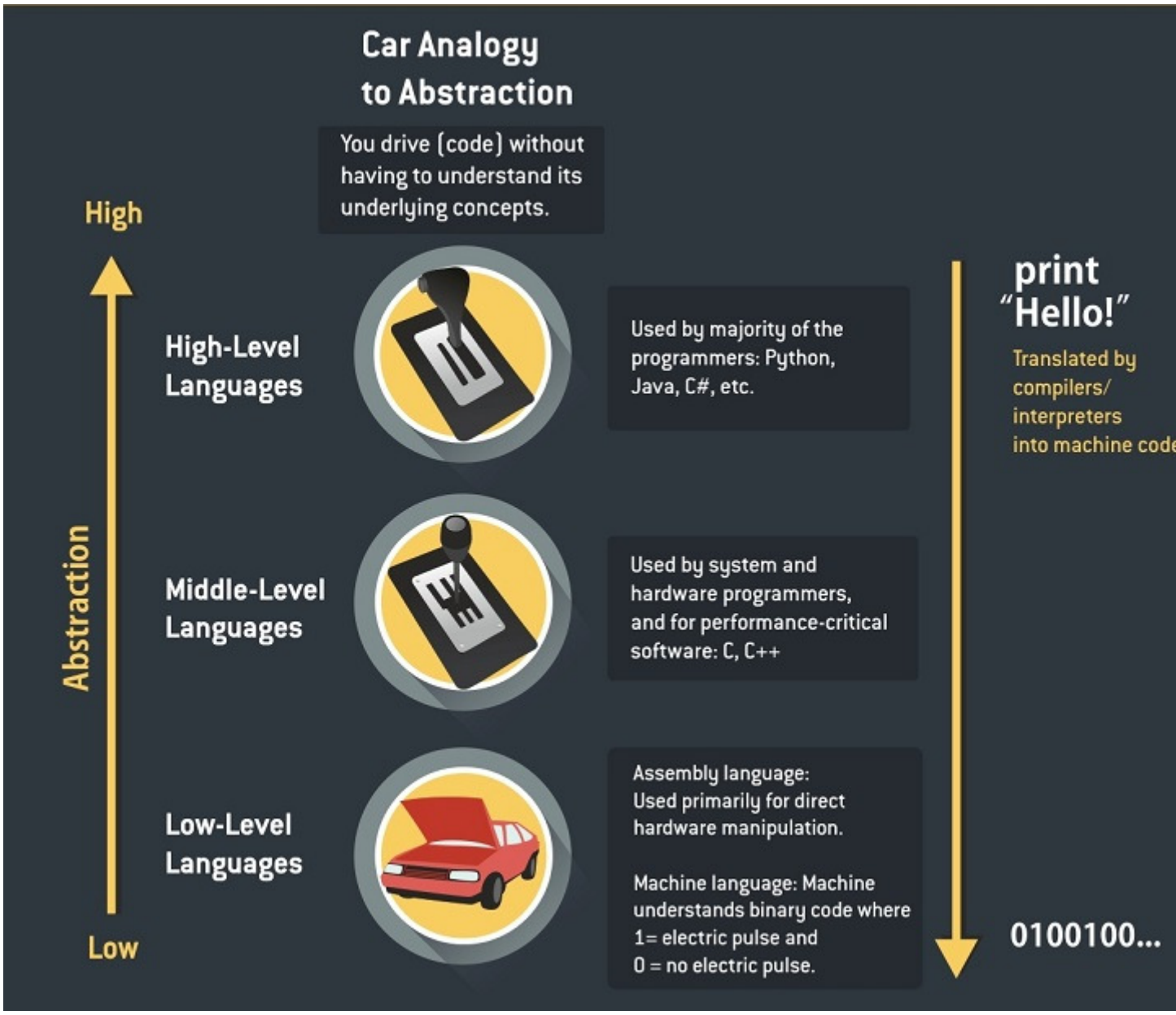
P vs NP is listed as 1 of the 7 Millennium Prize Problems by Clay Mathematics Institute. $1 million will be awarded to the first correct solution.

More: P vs. NP and the Computational Complexity Zoo (video), Simple Wikipedia

Read Also: The Lord of the Rings Analogy To Programming Languages [Infographic]

Core Concept #3 – Computer Architecture and Engineering

3.1 – How do computers work?



Computers work by adding complexity on top of complexity. When you drive a car, you don't necessarily have to understand how the car's engine works. The complex details are hidden.

So how do computers turn binary code, the 0's and 1's into programs? Here's an excellent video that uses dominoes to visualize how computers perform binary calculations at the most basic, fundamental level:

More: Interactive explanation on how computer works

3.2 – Halting Problem

More: The Freeze App Analogy, Simple Wikipedia
Computer architecture and engineering is a huge topic which includes subfields like operating system, compiler, and more.

Core Concept #4 – Concurrency

Let's say you work as a secretary in company A. You have to answer phone calls, arrange meetings, typing documents, etc. You always have to **switch back and forth** between your tasks based on priority. Every time the phone rings, you have to stop whatever task you are working on.
Concurrency is a property of programs and systems that allow tasks to run in **overlapping** time periods.

4.1 – Parallelism

Eventually, you can't cope with your job because there's too much data entry tasks. You complain to your boss and he happily hires a data entry clerk to handle your data entry tasks.

Parallelism allows 2 or more tasks to run at the same time, provided that the machine has multiprocessing capability.
However, the implementation of concurrency concepts also introduces more potential problems such as race condition.

4.2 – Race Condition

This is what will happen if you allow concurrent transactions in a banking system and race condition isn't handled:

- You have $1000 in your bank account.
- Someone transfers $500 to you and you withdraw $300 from ATM.
- Imagine both transactions are performed at the same time, both transactions will see $1000 as your current balance.
- Now, transaction A adds $500 to your account and you have $1500. However, transaction B also sees $1000 as your current balance and it completes a millisecond later, it deducts $300 from $1000 and updates your account balance as $700.
- You now have $700 instead of $1200 because transaction B overwrites transaction A.
- This happens because the banking system isn't aware of other ongoing transactions.

So, what can you do to handle the above situation? One really simple way is mutual exclusion.

4.3 – Mutual Exclusion (Mutex)

Now, whenever there's an ongoing transaction, the system will lock the account(s) involved in the transaction.

This time, the moment when transaction A occurs, your account is locked. You can't withdraw money from ATM. It unlocks only when transaction A completes.

So mutual exclusion solves the problem right? Yes, but nobody wants to get rejected by the ATM every time there's an ongoing transaction.

Let's modify the solution a little bit.

4.4 – Semaphore

4.4.1 – Binary Semaphore

Now, let's set different priority levels for different types of transactions. Say cash withdrawal request has a higher priority than bank transfer. When you withdraw money from ATM, transaction A (the bank transfer) will stop and allow transaction B to carry on first because it has higher priority. It will resume after transaction B is completed.

4.4.2 – Counting Semaphore

Binary semaphore is simple. 1 = ongoing transaction. 0 = waiting. On the other hand, counting semaphore allows more than 1 process running at the same time.

Let's say you're a locker room manager for a spa. There are 30 lockers. You have to keep track of the number of keys you have each time you receive or hand out a key, but you don't exactly know who they are. If all lockers are full, others have to queue up. Whenever someone is done, he/she will hand over the key to the first person in the queue.

4.5 – Deadlock

Deadlock is another common issue in concurrency system.

Let's use the same banking system analogy with a different scenario. Just keep in mind that access to a bank account is locked whenever there's an ongoing transaction.

- Peter transfer $1000 to you (transaction A) and you transfer $500 to him at the same time (transaction B).
- Transaction A locks Peter's account and deducts $1000 from Peter's account.
- Transaction B locks your account and deducts $500 from your account.
- Then, transaction A tries access your account to add the $1000 from Peter.
- At the same time, transaction B also tries to add your $500 to Peter's account.

However, since both transactions aren't completed, both can't access the locked accounts. Both wait for each other to complete. Deadlock.

Here's a real life example:

Boy: Let her approach me first.
Girl: Let him approach me first.
And there dies a budding love story

Core Concept #5 – Computer Security

5.1 – Computer Hacking

Hacking is similar to breaking into a house. Here are some of the popular hacking techniques:

5.1.1 – Brute-force Attack

Try hundreds and thousands of different keys. An experienced burglar will try the most commonly used keys first.

A brute-force attack **tries every possible passwords**, and usually starts by guessing commonly used passwords like “123456”, “abcdef”, etc.

5.1.2 – Social Engineering

A couple just moved in next door. They are really nice and helpful. They often invite you over for dinner. One day, you mentioned that you are going for a two-week vacation soon. They happily offered to take care of your dog. You left a spare key for them. Since then, you have not heard any news about them.

Social engineering is **tricking users** into revealing their private information.

5.1.3 – Security Exploit

A burglar checks every possible entries to find the easiest way (weakness) to get in. Maybe your second-floor windows is left open, who knows?

5.1.4 – Trojan Horse

A burglar pretends to be a plumber and you unlock the door for him. He fixes your leaking pipe and everything looks perfectly normal. After he left, you discovered that your jewelry is missing.

A trojan horse is malware program that **pretends** to be useful or helpful and runs malicious code in the **background**.

5.1.5 – Rootkit

Your door lock is jammed and you call a locksmith. He fixes your door lock and secretly duplicates another key.

A rootkit gains **administrator** or **root access** of a computer through various ways like social engineering, then disguise as necessary files that is hard to detect by antivirus software.

5.1.6 – Distributed Denial-of-service Attack (DDoS)

Here’s a bookshop analogy.

Imagine 100 people visit your little bookshop at the same time. Your bookshop is occupied and others can’t come in. You can’t ask any of them to leave because they don’t seem to be coming in groups. They probably don’t know each other at all. Most of them seem to be genuinely interested to buy books. Some even ask you where are the book shelved. Someone at the counter just pay you in pennies.

People keep coming in and out for hours. All of them look perfectly normal. At the end of the day, you've only made one book sale. Remember the guy who pay you in pennies?

DDoS attempts to bring a site or service down by **flooding** it with visitors.
IPViking, a live cyber-attack monitoring site/Imgur

5.2 – Cryptography

Cryptography is the study and application of secure communication. Here are 2 of the most widely used cryptographic protocols:

5.2.1 – Symmetric cryptography

Say Alice and Bob want to send each other stuff. To make sure nobody can see their stuff, they lock it with a box. They make 2 identical (symmetric) keys for the lock and meet up to share the keys beforehand.

5.2.2 – Asymmetric cryptography

Sharing identical keys works fine among 2 people. What if Alice want to exchange stuff with another guy named Carl, and Alice doesn't want anybody to see their stuff too? Alice can't use the same lock and key that she shared with Bob, else Bob can unlock the box easily!

Of course Alice can share a completely new and different lock and key with Carl, but what if Alice wants to exchange stuff with 10 different people? She will need to keep and manage 10 different keys!

So Alice come out with a brilliant solution. Now, she only maintains **one** key (private key). She distribute the same padlocks (public key) to her friends. Anyone can close the padlocks (encrypt), but only she has the key to open (decrypt) them. Now, anyone can

send stuff to Alice using the padlock she distributed, and Alice no longer have to manage different keys for different people.
If Alice wants to send something to Carl, she will ask for Carl's padlock (public key) so that she can use it to lock (encrypt) her stuff and send it to Carl.

The basic principle is: everyone has their own private key to decrypt message, and they will provide senders their own public key for message encryption.

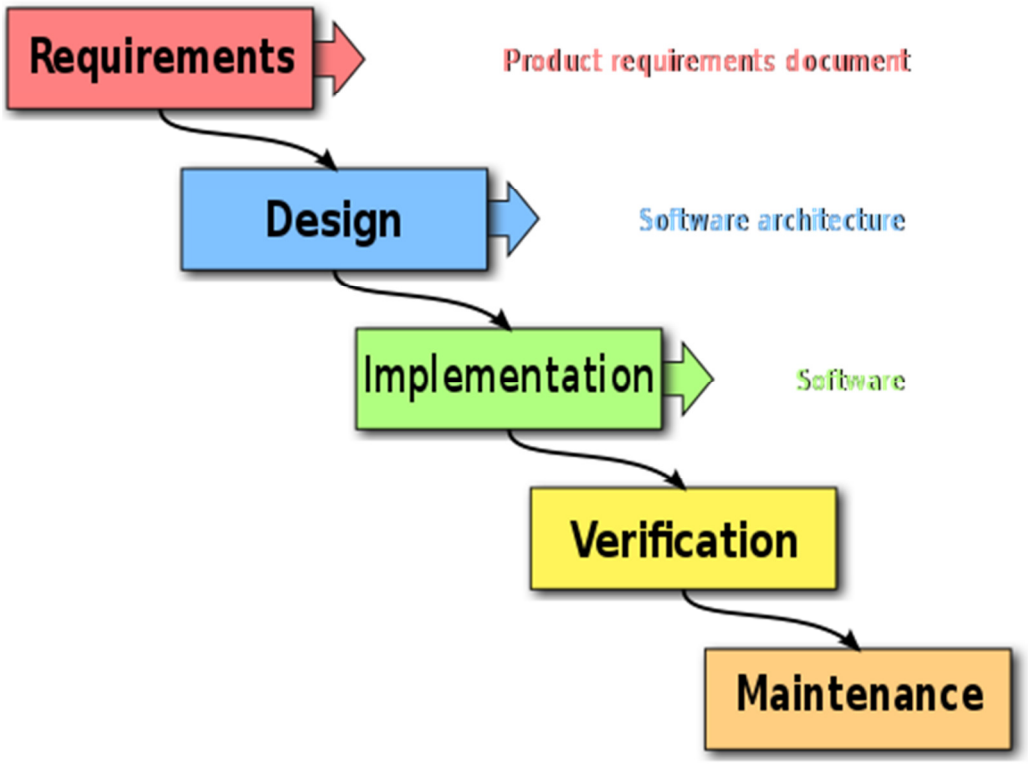
More: Public Key Cryptography: Diffie-Hellman Key Exchange (video)

Read Also: Before They Were Famous – Early Posts From Larry Page, Linus Torvalds, and More

Core Concept #6 – Software Development Methodologies

6.1 – Waterfall Development

You figure out *everything* you need to do and document them (requirements). Like a waterfall, there's no way to go back up unless you start over again. You move on to next phase only when current phase is completed.



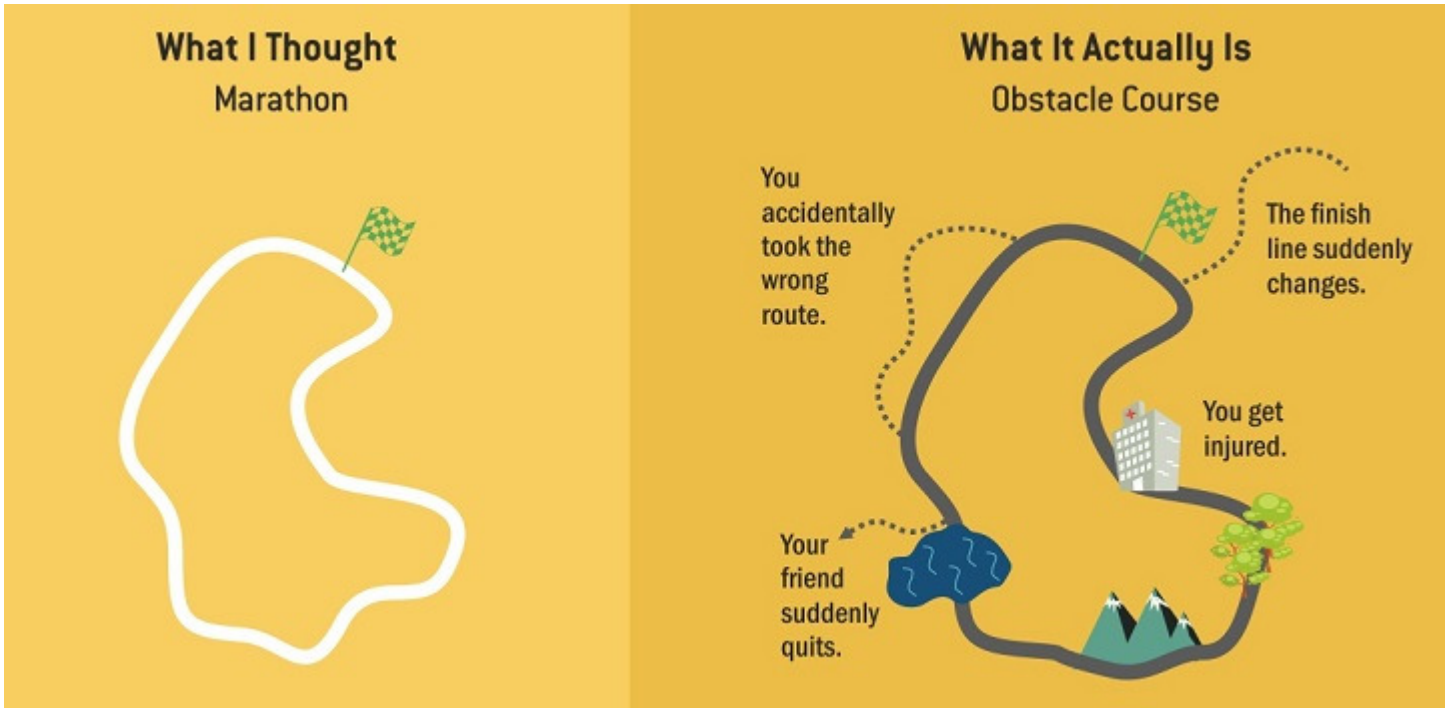
6.2 – Agile Development

You figure out *some* of the things you need to do at the beginning. Then, continuously improve, evolve, collaborate and adapt as the development goes on.
Here are some of the popular implementations of agile development methodology:

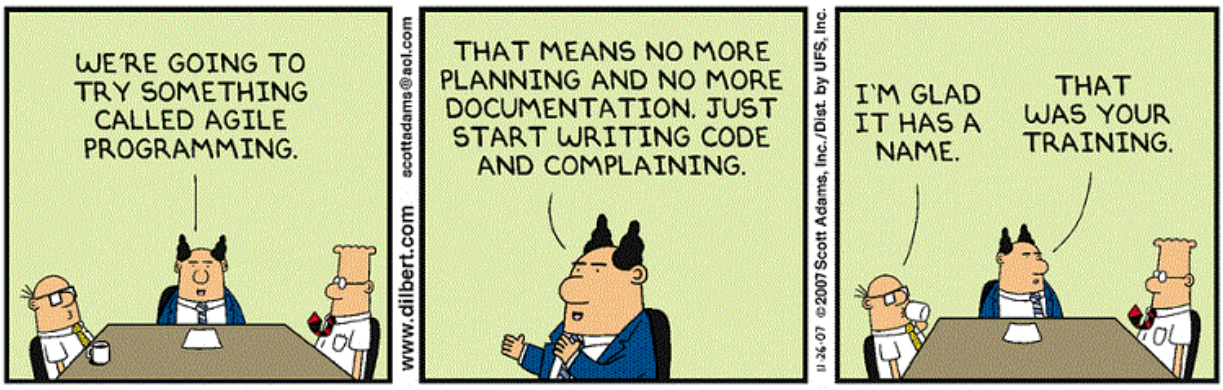
- Scrum
- Extreme programming
- Kanban

6.3 – Software Development In The Real World

So you graduated. You write good and beautiful code (hopefully), everything is perfect so far. Let me introduce you cowboy coding, a software development methodology that isn't taught in college.
Next, you wonder why you are bad at estimating development time:



And methodologies are often implemented wrongly:



Dilbert

Dilbert

So here you go. Computer science in the nutshell.

Feel free to suggest any new computer science theories or concepts to add, those that you think is important and often confusing.

Common Beginner's Questions

Q1 – What is the difference between computer science and programming?

Computer scientist is like a physicist and programmer is like an engineer.

Q2 – What is programming?

Writing very specific instructions to a dumb, yet obedient machine.

What does it mean? Imagine you have to teach a kid how 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 computer works. You have to tell the computer what it exactly needs to do. It doesn't know how to assume and never think about the consequences.

Q3 – Why you shouldn't interrupt a developer when he/she is in the zone?

Getting into the zone is like falling asleep. Imagine you are waking up a person who is close to falling asleep in few more seconds. Now he/she has to spend more time to fall back into sleep!

Q4 – What is the difference between Java and JavaScript?

They are not related at all.

Java and Javascript are similar like car and carpet are similar.

Q5 – What is the difference between JavaScript and JQuery?

JQuery is a library built on top of JavaScript.

Javascript is the ugly nerd and jQuery is the wizard who turns him into the handsome quarterback.

Q6 – What is the difference between a framework and library?

You call library. Framework calls you.

Q7 – How many lines of code does an average software engineer write per day?

It's impossible to tell. The number can even be negative, when developers are paying technical debts.
Measuring software productivity by lines of code is like measuring progress on an airplane by how much it weighs.

Q8 – What is object-oriented programming?

Objects are nouns, methods are verbs.

Objects are like people. They're living, breathing things that have knowledge inside them about how to do things and have memory inside them so they can remember things. And rather than interacting with them at a very low-level, you interact with them at a very high level of abstraction, like we're doing right here.

Here's an example: If I'm your laundry object, you can give me your dirty clothes and send me a message that says, "Can you get my clothes laundered, please." I happen to know where the best laundry place in San Francisco is. And I speak English, and I have dollars in my pockets. So I go out and hail a taxicab and tell the driver to take me to this place in San Francisco. I go get your clothes laundered, I jump back in the cab, I get back here. I give you your clean clothes and say, "Here are your clean clothes."

You have no idea how I did that. You have no knowledge of the laundry place. Maybe you speak French, and you can't even hail a taxi. You can't pay for one, you don't have dollars in your pocket. Yet I knew how to do all of that. And you didn't have to know any of it. All that complexity was hidden inside of me, and we were able to interact at a very high level of abstraction. That's what objects are. They encapsulate complexity, and the interfaces to that complexity are high level.

Q9 – What is an application program interface (API)?

At restaurants, you order food (call API) from the menu (APIs). Once your food is ready (API response is ready), the waiter will serve you the food.

The basic idea is: you ask for what you want and the system returns you a response, without exposing what's happening behind the scene.

Q10 – What is the difference between SQL and NoSQL database?

NoSQL databases store information like you would recipes in a book. When you want to know how to make a cake, you go to that recipe, and all of the information about how to make that cake (ingredients, preparation, mixing, baking, finishing, etc.) are all on that one page.

SQL is like shopping for the ingredients for the recipe. In order to get all of your ingredients into your cart, you have to go to many different aisles to get each ingredient. When you are done shopping, your grocery cart will be full of all the ingredients you had to run around and collect.

Wouldn't it be nicer if there was a store was organized by recipe, so you could go to one place in the store and grab everything you need from that one spot? Granted you'll find ingredients like eggs in 50 different places, so there's a bit of overhead when stocking the shelves, but from a consumer standpoint it was much easier/faster to find what they were looking for.