Rafael Castro Gonçalves Silva

Cover Letter

I'm a working computer scientist and a developer.

Prelude

This is text about myself, so you can know a little bit about me. I did this text in an informal style of writing, so you can imagine me saying what is written here. Also, notice this text is a work in progress.

How I'm like

Usually I describe myself as nerd, progressist and cachorrista (a brazilian neologism for dog person). Let me be more detailed about those things:
I enjoy nerd stuff like video games, fantasy movies and technology in general. I know, very typical. Second, by progressist I mean someone that agrees with progressism: the idea we should embrace changes that can cause improvements in our life quality (e.g., weed legalization). And last: I like dogs a lot.

Of course, I'm not just three things.

Another thing that I'm is playful, and I'm far from the stereotype of a serious adult. Sure I can have serious conversations, but I just find life better to live if I can joke every now and then. Though I enjoy making jokes, I don't like it when it offends someone that did nothing to deserve it I hate when I see someone being humiliated if the person don't really deserve that. I can't watch *The Office* because I fell like throwing up when I see Michael humiliating his employees that have done nothing to deserve it. Search for *The Office — Try My Cookie Cookie* on YouTube and you will know what I'm talking about.

More on me:

I usually avoid things that cause me stress. If I have the option to spend money so I won't be bothered, I do it. Sometimes I just quick the thing that is causing me pain or anguish. I don't mean that I'm a quitter, I just question myself: do I need this? And depending on the answer I do benevolent thing to myself. For example, I disliked my first job in the first few weeks, so I quitted after just two months working there. It was there right call, so I could move on to the next thing.

I'm eager to learn and learning is quite important to me. This characteristic of mine made me want to avoid to work in the software industry. My vision was partially correct I would say: to a company the best tools are the ones that you can easily find employees to hire for. But, usually, those tools are not really the best ones for the job and are the ones that I don't want to learn. I learned Ruby for my job, but don't think I was an interesting thing to learn. However, I have learned very interesting things in my current job. So I don't see working in the industry with the same pessimist eyes as before.

But my opinion, in general, stays the same: developers should be eager to learn the best technology

for this or that specific problem. I think the problem lives on the fact that programmers most of the time are stuck with the same languages and tools, they should more often look out there and see that languages aren't all the same, and they didn't stop innovating in the 90s. In conclusion about that, I rather work in a place where I'm continuously learning interesting things. There are some very interesting new languages like Rust that I wish to see in the industry more often. However, I understand that you don't see a Rust programmer under every rock like JavaScript programmers.

I'm usually very communicative and I don't fill ashamed of asking people to teach me about something. I don't pretend to know something that I don't. Not only that, but I enjoy working with teams that do *real team work* with members support each other and support each other.

My life in a nutshell

I was borne in 1993 in the city of São Paulo (Brazil), but when I was three years old I moved to the State of Santa Catarina (SC), in the south of Brazil and I have been living here since then. I lived in some cities of SC, but at the moment I live in Joinville. In 2012, I started my undergrad in Computer Science, here in Joinville, at the Santa Catarina State University. For the first time in my life I noticed that I was doing something that I enjoyed. I loved the courses of Automata Theory, Compilers, Formal Methods and Computability Theory. I guess I discovered myself in theoretical computer science. After my undergrad, I started my masters at this same university.

Academic me

At the end of my undergrad I discovered that programs can be certified in respect to a specification using proofs. I already knew Curry-Howard correspondence, but when I found proof assistants like Coq and Isabelle, and languages with dependend types like Agda and Idris, I was very surprised to see what people were doing with it. For example, I found out CompCert, a C compiler with a fully verified backend. Come on, this is the coolest piece of software ever made! I started the masters and my advisor had experience with type systems for programming languages, specifically type inference algorithms. So I decided that my masters would be using a proof assistant and would have something related to type inference.

Long story short: I did a Coq certification of algorithm W using Hoare logic using a monad. Google *Monadic W in Coq* and you will find the paper.

After the finishing my masters, I decided to look for a PhD, but things didn't go well so far in this matter. In addition, a pandemic started so this plan is delayed.

Professional me

I had just finished my masters, I couldn't start the PhD and I had no professional experience. I decided that the least I could do is to solve the latter. Looking for a job was quite sad because I don't really see many jobs opportunities that I find interesting. I managed to find a job that looked quite convenient for logistic reasons and I decided to take it. As I said, I didn't like it in the first few weeks. The problem was so many, just to get you a glance: it was a startup following this stupid trend of startups that think to be a family. More on that latter.

So I quitted that job and started looking for another one. I found a place that some colleagues from the university were working at and they said this was a good place to work. The company is an outsourcing one, they provide they service as teams of programmers. So, I applied and I was hired. This company is called Magrathea. Yes, the planet that makes planets.

I was allocated to a team working in project that is quite interesting. Being brief: it's a distributed system with concurrency scenarios that must process events in whatever order, so it can keep a materialized view correctly updated. At the moment I'm writing this text I'm still working in this project and the experience so far is great. I find very satisfactory to work with a team of people that have a higher level of abstract thinking and can write non-trivial algorithms. Working with them made me discovery how it's like to work with team that help each other in order to kill a dragon. Now I can say that I can be a good professional.

I and my teammates joke that after this project we don't know what to do because everything else looks quite boring. So, if you are an employer reading this text looking for to hire me, it's better you have something fun to offer me.

I know I don't have much experience as a developer, but now I at least see the possibility of an interesting professional life for me. Though I still wish to do a PhD, I don't see this is an issue for my professional life. If you do, you missed the chance of stop reading this text long ago.

Things that I like

TODO

The last cool thing that I did as a job

TODO

Some thoughts on what is a good programmer

TODO

My thoughts on Software Engineering in practice (Rant Section)

TODO

Range against Object Orientation (Rant Section)

My opinion on Object Orientation (OO) is in opposition with the common believe. OO programming doesn't provide more abstraction, or better organization, or elegance or simplicity, it just provides the illusion of those things. Let's begin stating what OO is: an object is an encapsulated state which you can only interact with by sending messages, in in other words OO is the association between data and functions. There are big problems with that:

- States being passed all around. In the original concept the messages in OO should send copies of states, not references of states. But this is not how 99% of OO programming languages are implemented. This is a big problem because it's harder to debug the code and to implement parallelism becomes painful. OO is just global variables with extra steps.
- OO has nothing to do with better organization. One way to implement OO is through classes, which is basically a way to define types. Well, types and module systems allows you to organize your code because you can give names for some specific kinds of data and group them together. Nothing to do with OO.
- OO doesn't provide better abstraction. Abstraction is hiding irrelevant information (one possible definition). When people are presented OO they are showed trivial examples like a class representing something real like the customer, and the teacher says this is how OO allows you to abstract a concrete thing because your customer class is some sort of model of the real life customer which

doesn't have the irrelevant information. But in practice OO programs very often have classes that doesn't represent anything real, like services, manager, factories and other doers. Those classes are there to fill a gap and because OO demands you to have the association between data and functions. So at end of the day the promised of abstraction is not fully delivered, and you end up with many words in your program that have nothing to do with the domain in matter.

So what should I do instead of OO? Should I use functional programming? The opposite of OO is non-OO, or what was usually called procedural programming. We can combine procedural programming with imperative and functional programming, as we can also combine these with OO. So, in conclusion I must say that I actually write OO code, but not by my choice. Most systems out there are objected oriented and I, as humble employee, have the job of programming in them. But of course I would like to see more people stop using OO.

Why dynamic typed languages still exists?

Let me be straight here: dynamic typed languages should have died last century. It's like coal power plants, why they still exist? Both are obsolete technologies with no good excuse to still exist.

I know, there is a - quote and quote - big debate between dynamic and static typing. But this debate shouldn't exist. Every single facility that dynamic typing gives, you can also have with static typing if the person implementing it is not lazy.

- Oh, duck typing? Meh, universal polymorphism is much better.
- Oh, not writing types? Please, are we still living in caves? Haven't you heard of type inference?
- Oh, no compilation step? Well, you can tweak the compiler for compilation speed (in development). Unless you need to recompile everything, this shouldn't be a problem in development. And nowadays we have fast computers that are affordable.

Static typing is just a more advanced technology!

Let's not make a debate about it, ok?

Range against company's bullshits (Rant Section)

TODO