TITULO ABREVIADO DO DOCUMENTO

Independent Studies Mozilla Recruiting - Report

Filipe Gonçalves

(Learning report)

Abstract—Applying for an internship position in a big, selective tech company is an inherently complex and delicate process for both parties. Learning how to deal with it, how to look different among other candidates, and how to address the right people, is an art that comes only with going through such an experience. This report addresses some of these issues, and summarizes some of the best learning experiences I've had.

Index Terms—Mozilla, recruiting, teamwork, opensource, human interaction, software development, community, résumé

1 Introduction

PURSUING a Computer Science degree is a lot like military training: it hurts - it really does - but it makes us stronger. A lot of exercises seem totally irrelevant and utterly unnecessary, not to mention blatantly and overly complex, but they build fundamental muscles that are needed on the field. The easiest way is to quit. People can easily lose track. But in the end, for those brave souls who ought to survive, it pays off. Or does it? Well, maybe it doesn't. I mean, it depends! Nowadays, it is not very likely that an average skilled software developer is unemployed. There are always opportunities for software development positions out there, and IT industries need more and more specialized technicians to help support the digital era. The comfort of having computers managing the world comes at the cost of increasing need of professional, skilled workers who know how to operate them: programmers, security experts, server administrators, database gurus, OS developers, IT technicians, datacenter engineers, and the list goes on. The point is, it is *extremely* hard for someone in the tech industry *not* to find a job, which is impressive when we look at unemployment rates in other fields. So, in that sense, yes, Computer Science *is* worth it.

What's the catch? It can be hard to find a nice place to work. With the financial chaos that mostly affects South Europe, including Portugal, workers see their incoming amount getting reduced; at the same time, prices go up. Taxes increase. People lose motivation, productivity decays, and it all adds up to failure. Now, that's a very good example of how *not* to solve a problem. As a software developer, finding a nice place to work is very similar to finding and differentiating the companies that are looking to *reduce* the cost of programmers VS. the companies that actually want to find the best programmers and are willing to *increase* that cost, because they know that good programmers make great software. Great software leads to more profit, more happy customers, and, unsurprisingly, this time, it all adds up to success.

In the next sections, I will be addressing the issue of success, namely, how I *learned* that success is a lot more than doing good at maths and algorithms courses in college. This report is a compilation of thoughts, ideas, and knowledge I gained with my experience on applying for a Mozilla internship. Over the last months, I

• *Filipe Gonçalves, nr. 70592, E-mail: filipe.a.goncalves@tecnico.ulisboa.pt, is a student in Information Systems and Computer Engineering, Instituto Superior Técnico, Universidade de Lisboa.*

Manuscript received May 30, 2014.

(1.0) Excelent	LEARNING					DOCUMENT						
(0.8) Very Good	CONTEXT x2	SKILLS x1	REFLECT x4	S+C x1	SCORE	Structure x0.25	Ortogr. x0.25	Gramm. x0,.25	Format x0.25	Title x0.5	Filename x0.5	SCORE
(0.6) Good												
(0.4) Fair	2	1	3.6	0.8	7.4	0.25	0.25	0.25	0.23	0.5	0.5	1.98
(0.2) Weak												

learned a lot about how tech companies recruiting works, what they are looking for, and how a Computer Science undergraduate can shine.

2 HITTING THE HIGH NOTES

The single, and perhaps most controversial, topic in this report is that doing good at university is just not enough. Not that there's anything wrong with the Computer Science curriculum at top universities like Instituto Superior Técnico, it's just that it's not tailored to exercise some of the most core skills of a good software developer. Typically, this is the case with soft skills. Although the course staff is fully aware of this reality, and has been working towards this issue, there will always be a set of skills and knowledge that one has to assimilate by himself - the kind of stuff that can't be taught in a classroom. A bright student with top grades on algorithms assignments can perform poorly in a job interview for a variety of reasons, the number one being that a very smart candidate is of no use to a company if he can't communicate his thoughts. I exchanged some interesting ideas with other international candidates to summer internships, and performed a few mock interviews (both as an interviewer, and as an interviewee), and I can say that the most common mistake anyone can make in a job interview is to stop talking. That's a very effective way to get a nice, shiny rejection letter.

Why do companies care about this so much? Because software development is a lot more about human relationships than it is about programming. Big, complex projects are carried out collaboratively. This means that people have to communicate and coordinate their work. Decisions are made by a team, not by one single person. Moreover, great ideas can become even greater when shared with other developers. I think we can all agree here that hiring John, a reasonably smart student who is passionate about programming and enjoys having a good chat with other developers, is a much better choice than hiring Andrew, who is blazingly smart and can implement a recursive descent parser for C declarations in about 15 minutes, and despite that being impressive, he never really saw that he was tackling the wrong problem, because he didn't care to ask for more details. Oh, and he didn't talk during these 15 minutes, all the interviewer could hear was the sound of keyboard typing. No one wants to hire a candidate that can only communicate with the compiler, no matter how smart he/she is.

Thus, the most important thing to keep in mind is that it is of great importance to show good communication skills - thinking out loud. Demonstrate a good, structured reasoning. Make sure we are solving the right problem. Practice foreign languages, especially English. Learning new languages is a good way to improve communication skills. Reading, starting a blog, attending writing and communication seminars, learning how to talk in public - these are all great ways to get better at communicating, because afterall, that's what life is all about.

3 INTERACTING WITH THE COMMUNITY

A good lawyer tries to keep up to date with the latest laws. He always knows about the fresh and complex criminal cases being investigated. He hangs out with other talented lawyers. The same is true for doctors. And for researchers. And, of course, for software developers, or aspiring software developers. No one can become a rockstar without having other rockstars teaching him. In the end, it is a matter of *interacting with the community*.

However, interacting with the community is a double-edged sword, it can be counterproductive: the right communities must be chosen. From my experience, communities like Quora [1] and StackOverflow [2] are just the right kind of communities any good software developer, or aspiring software developer, should follow. These two complement each other: quora is typically focused on the not-so-technical questions. Almost anything can be asked in quora. This is where I had the joy to talk to other candidates, as well as with experienced software developers at known companies that were kind enough to give out precious hints on how to behave during an interview. StackOverflow is very different, but equally important: it is all

about technical questions. At StackOverflow, if a question sounds primarily opinion-based, is unclear, or too broad, it is closed and eventually deleted. This happens because StackOverflow targets specific programming questions where there is no space for philosophical discussions; an answer is either right or wrong, and one can win the debate simply for *being right*, not because his arguments are stronger, or because people like him.

Participating in communities like StackOverflow and Quora is important, and there is more to this than the simple eye meets. First and foremost, these communities are accessed by thousands of brilliant developers all over the globe. Second, actively answering and commenting on questions creates the right environment for debate. These communities, especially StackOverflow, can be harsh for newcomers (and sometimes, even for oldtimers). In the beginning, I had a lot of my answers being downvoted along with a comment from other users on why my answer was not precise enough and lacking technical detail. Or why I was wrong when I said that it's impossible to do X without using Y. Or why I completely missed the point of the question, and answered a different question. Or why my solution is overly complicated, slow, and doesn't work for case A, B, C, and D. Or why there's a better data structure for the problem at hand. Like I said, it's a hard community to be part of, but after some hard work, I became an active user with a considerable reputation on both communities. The bottom line is that constructive and sincere comments from other people on *why* our participation is suboptimal are a very good learning tool. People are generally very honest behind the anonimity of Internet, which is a strong advantage in this specific case - no one at these communities has a problem on criticizing a bad answer. *The only real mistake is the one from which we learn nothing*, says John Powell. I learned from my mistakes. In particular, I learned how to deal with criticism to my work, how to control my willingness to reply with not-so-nice words, and how to improve my knowledge based on those critics. It's been a wild, exhausting ride, but I'm sure these communities of great people have considerably contributed to improve my soft skills. I feel like I achieved something here: I can honestly look at my own work and criticize it as if it was written by someone else. Basically, I achieved evenhandedness, one of the most core soft skills anyone can have. That's something that requires a lot of effort, and is absolutely crucial when evaluating our weaknesses (and trying to improve on them).

4 Teamwork and Open Source

Interacting with people is way harder than interacting with a compiler. Managing teams of humans makes Lisp macros and C++ metaprogramming positively *trivial*. Over the course of time, it came to my attention that teamwork experience is a plus. For someone who is still studying full-time, it can be hard to gain teamwork experience. Sure, college projects attempt to address this by arranging students into groups. This is good and desirable, but the experience is quite different from what happens "in the jungle": in college, groups are usually very small (3 elements), and students get to choose their group - usually, their best friends. That's not how it works in real world, and that's why everyone should be doing open source. Open source projects are a great opportunity to evolve - students can learn both hard and soft skills. The soft skills experience comes from interacting with the team. Big projects, like CPython [3], the *de facto* Python implementation, are developed by hundreds of interested and dedicated programmers on their spare time. Joining the mailing list, contributing to the code base, and participating in discussions is an amazingly rich experience. All sorts of ideas and feature proposals are discussed there; it is a very good exercise to try and submit some ideas. New and constructive ideas are usually the source of hot debate. This shaped my ability to stand for my arguments, and it certainly improved my vocabulary as a non-native English speaker.

5 The Résumé

Having an appealing résumé is challenging. After a bit of studying and reading, I turned

the Mozilla's recruiter phone call about the internship offer into another opportunity to learn and asked why Mozilla decided to interview me in the first place, and what exactly in my résumé scored points. As it turns out, lots of things in the answer were not new for me at that point. Listing impressive projects, along with my achievements, was the first good sign. Not everyone has implemented a user-level thread library in C, or written a compiler for a custom, in-house language just for the fun of it, or contributed to a Python Just-In-Time compiler. Moreover, from those who did, not everyone is comfortable enough to talk about it to the point of including it in a résumé, since people expect to get asked about what they list. My BS certificate from IST with a GPA of 18/20 scored a few more points, and the final touch was my experience in interacting with online communities and my blog. About a year ago, I started a blog. Listing it in my résumé made the person reviewing it curious enough to go and check it out (this is why it is never a good idea to have an outdated blog where the last post was 6 months ago - that sounds like a dead project, and conveys the idea that the candidate doesn't get things done and can easily lose motivation).

There are also a couple of other details that are just as important, for example, waxing rhapsodic about how I was moved to tears of joy by *Structure and Interpretation of Computer Programs* [4], or even *Compilers - Principles, Techniques & Tools* [5]. Apparently, this was funny enough to make the reviewer laugh a little bit and remember about me while looking at other candidates. I was "that crazy guy who got really excited with a compilers book". They just couldn't forget about me. What about English skills? Does the résumé indicate good communication skills? This includes neatness and orderliness. A disorganized résumé rife with grammatical errors where nothing lines up is a pretty big red flag for a disorganized thinker or just general sloppiness.

That being said, the reasons aforementioned seem to have been enough to persuade Mozilla into interviewing me. An organized, well written and concise résumé with impressive and respectful projects, showing a little bit of passion interleaved with a personal touch, is all it takes. And honestly, it's not *that* hard, and it surely pays off to spend a few time reading about how to write good résumés, and how to use some common buzz words that look better in paper than others.

6 Conclusion

My application to a Mozilla summer internship brought me a lot of new, interesting knowledge on how the tech industry recruiting process works. With respect to soft skills, I believe the last couple of months were the most intensive learning period: I interacted with a relatively large amount of people in different countries working at software companies, I learned what employers are looking for, I earned the quality of fair-mindedness, and I had the chance to iteratively build a good and quality résumé.

This activity has given me the opportunity to come closer to the job market, and at the same time, provided me with some very interesting, insightful and useful knowledge. It has given me a clear, concise and realistic vision on the typical recruiting process, something that every Computer Science student should be familiar with.

The bottom line is that taking the time to build great soft and hard skills can be very time and brain consuming, but undoubtedly, it gives access to a freeway pass to the world of great employment opportunities, in companies where going to work every day is a pleasure and not an obligation.

References

[1] http://www.quora.com
[2] http://www.stackoverflow.com
[3] http://www.python.org
[4] Hal Abelson and Geral Jay Sussman, *Structure and Interpretation of Computer Programs* (Cambridge, MA: MIT Press, 1985). Second Edition, 1996
[5] Aho, Lam, Sethi and Ullman, *Compilers - Principles, Techniques & Tools* (Boston, MA: Pearson Education, 2007). Second Edition

In this type of document (technical), the Conclusion should start with a summary of the subject addressed and then should highlight the results.