Revista de Sistemas de Informação da FSMA
n. 14 (2014) pp. 48-52

SISTEMAS DE INFORMAÇÃO
http://www.fsma.edu.br/si/sistemas.html

TRILHA PRINCIPAL

TEACHING PROGRAMMING

Ricardo Linden, FSMA

***Abstract* — This article discusses how university curricula could improve its performance in. terms of producing computer science graduates who are more attuned to the needs of the industry in which most of them will be employed. The evaluation systems that are currently in place in this country strengthen the tendency to regard teaching as a low priority activity completing a vicious circle of decline.**

***Keywordss* — Teaching, Programming, Universities, Evaluation.**

I. INTRODUCTION

IN the US, some of the most important employers in the computer software area are starting to hire non-graduates, even to technical jobs. Even if there is no concrete evidence that this is a major trend and no specific statistics of the jobs they are receiving, there are some signs that some companies, such as Google are considering hiring talented programmers based on their talents alone and not on whether they have a Computer Science degree or not.

According to a Google vice-president, this is due to the fact that “”the academic setting is an artificial place where people are highly trained to succeed in a specific environment” [1]. The same article points out that most of the new hires at Google are still college graduates, but we should ask ourselves whether this is an outlier or a real phenomenon.

There are some warning signs for academia that this might become a trend. In a recent article [2], Harvard Business Review asked employers to stop requiring attendance to a university as a requirement for a job. This may be dismissed as a rant, but this paper will point out that there is a major disconnect between what is taught in Computer Science courses worldwide and what is effectively needed in most jobs that involve programming skills.

There are many in academia who will argue that this topic is irrelevant, because universities are not “career schools”. Unfortunately, this kind of disconnect with the realities of job markets is also an issue, because more than 95% of our students will not become either graduate students or researchers.

Hence, I argue that we need to increase our workload on making our graduates better IT professionals. We need to make our students better programmers, who are equipped to be undeterred by highly complex environments while developing large pieces of software. In order to achieve this, when creating our syllabi, we should focus less on the small and tricky challenges and include material that is more in line with the reality of software development.

It may appear that the idea of improving the employability of our students is something that should not concern the best universities (like the research-centered, ivy-leaguers or Brazilian Grade-6 ones). However it is clear that the ability to be better team players, better communicators and better coders will benefit everyone (maybe even future professors), not just for some professionals that are supposed to drone on boring jobs.

Besides, being a coder is also a part of the life of today’s researcher. We have to implement a lot of software in order to test and even to arrive at our ideas and if we could write software that is safe, correct and readable (so that our research can use it as the foundation of future progress), our research would only stand to gain[1].

This paper is organized as follows: in section II, we discuss the current structure of programming teaching in our universities.

II.HOW UNIVERSITIES TEACH PROGRAMMING

According to three university ranking sites (TopUniversities, The Guardian and Shanghai Ranking), MIT offers either the best or second best computer science program in the world. Hence, we are going to take a look at its curriculum in order to see whether we can find a trend. Obviously, there are some differences between institutions, but as most readers will acknowledge, their syllabi are very representative of the reality of Computer Science Teaching.

Ricardo Linden is a Full Professor at FSMA, Macaé-RJ (Ricardo.linden@gmail.com)

[1] If every researcher wrote open source, high quality software, knowledge would spread faster. Hence, science as a whole may also benefit from students becoming better coders.

According to the flowchart at its website (http://www.eecs.mit.edu/docs/ug/6-3.pdf), the Computer Science and Engineering program at MIT requires as foundational work one disciple of software and one of algorithms, on top of which is added a discipline of advanced algorithms.

Algorithms, according to the MIT catalogue, offers an "Introduction to mathematical modeling of computational problems, as well as common algorithms, algorithmic paradigms, and data structures used to solve these problems. Emphasizes the relationship between algorithms and programming, and introduces basic performance measures and analysis techniques for these problems. " Advanced algorithms, on the other hand, teaches "Techniques for the design and analysis of efficient algorithms, emphasizing methods useful in practice. Topics include sorting; search trees, heaps, and hashing; divide-and-conquer; dynamic programming; greedy algorithms; amortized analysis; graph algorithms; and shortest paths. Advanced topics may include network flow; computational geometry; number-theoretic algorithms; polynomial and matrix calculations; caching; and parallel computing. ".

These syllabi are beautifully complemented by the software discipline which "Introduces fundamental principles and techniques of software development, i.e., how to write software that is safe from bugs, easy to understand, and ready for change. Topics include specifications and invariants; testing, test-case generation, and coverage; state machines; abstract data types and representation independence; design patterns for object-oriented programming; concurrent programming, including message passing and shared concurrency, and defending against races and deadlock; and functional programming with immutable data and higher-order functions."

This is a structure very similar to the one at the university where I teach. Even though requirements may vary and some professors may be more rigorous than others, workload may be more demanding at some institutions and etc., I argue that this is a common ground for most institutions that teach Computer Programming. Let us first understand what software companies expect from their hires and we will come back to discuss whether the structure described above is actually consistent with this reality.

III. What do employers want?

It is important to understand that there is a great divide between what employers say they want and what they really want. Hence, job descriptions in ads might be somewhat deceitful (specially because many companies usually have in house training to guarantee that new hires have what they really want). Nevertheless, we may see what some important bloggers and career advisers say and discuss it.

Matt Weisfeld (2013), for instance, polled many companies and came with some interesting characteristics, which he grouped by company size.

The first important skill companies are seeking is the ability to learn. We must understand that the technology field changes rapidly and whatever programming language we teach our students is actually irrelevant, because it will be phased out in five years.

The set of skills that are dominant include strong programming logic required; sometimes specific technologies are preferred. Employers expect that programmers can learn to work in most environments.

Nevertheless, the same companies state that soft skills, such as writing, presentation and other communication skills, may ultimately be the most important skill, especially as you move up the ladder in an organization.

Reading another set of advice by programming guru Joel Splosky [4], we see that he also focuses on writing skills and also on learning non-CS subjects such as microeconomics.

The most interesting material comes from a career advice site, called MyMajors [5]. This site rates as most important abilities, reading comprehension, critical thinking, quality control analysis, active listening, systems evaluation and systems analysis, among others. It also states that customer and personal service, administration and management and design are important fields of knowledge for those interested in pursuing a career in Computer Science.

These examples are representative of many more offering similar advice. Thus we should ponder what they are saying and compare the skill set required versus those offered at our colleges.

To put it bluntly, these sites are just saying something that should be obvious: implementing Fibonacci sequence calculators, complex stack managers and similar course favorites is not a common problem faced by software companies. They deal with complex software that must actually solve a real need, be readable and maintainable, interact with other software artifacts and be thoroughly tested.

Do our colleges really deal with these issues? Analyzing the syllabi of MIT, we may come to the conclusion that the two algorithm subjects are actually intended to give a strong foundation on programming and the software syllabus is a giant step towards teaching the students the fundamentals of good software. But it is enough?

In the US, many CS majors from top universities do summer jobs and internships that oblige them to become aware of real problems, real programming and job environments, which is probably true for MIT students. Hence, it is possible MIT and other universities focus on theory in the core curriculum and practice is to be learned in summer jobs, internships etc.

This is not true in other countries. For instance, in Brazil, many, if not most, students do not get in touch with their future profession until close to their graduation time.

Besides, "outsourcing" the teaching on this important issue could mean that we, as educators, are giving up the opportunity to influence positively the job market and improve the state of the art of software in the world.

Hence I argue that what is currently offer is not enough to prepare our CS students for their future job market. Let us discuss some issues first before coming to a full proposal.

IV. What should we offer?

We clearly should offer something more. In spite of the fact that MIT and similar courses offer something towards a good background, they are not providing the tools our students need to thrive on the job market.

Critics may state that the market fluctuates and universities cannot be expected to match those changing needs. I answer by saying two things: first of all, universities should change according to what society expect because it is our job to prepare the future professionals and, more important, there is a set of skills that is unchangeable and that we are not addressing.

For instance, why don't we offer courses on business modeling? Even those that are prone to pursue a scientific career would benefit from being able to understand the nuances of a business and extract the essence of the problem that should be solved.

Teaching business modeling would also have the added benefit of helping students read fuzzy specifications (which are common throughout all bigger problems) and identify the solution through its relevant variables and their restrictions. Afterwards, using the tools available at this discipline, students can also build a conceptual model that will help them elaborate a solution.

A professor from a major Brazilian university read the manuscript of this paper and offered an example that corroborates some of the ideas above. He teaches Introduction to Optimization and sees the difficulty his students have in understanding the problems, classifying them and making an analogy with previous problems, either simpler or more tractable, so that they can arrive at a solution less painfully.

As he points out, many students nowadays see the code as an end in itself and do not spend much time thinking about their solution in regard to the stated problem. In order to help them with this issue, he uses many examples coming from a Business Dynamics course.

Notably, this professor is not from a Computer Science college, which lends support to the claim that Business Modeling is an idea that might offer instruments to better problem solving.

Another important issue is testing. MIT offers it as part of a bigger discipline, and as a result it probably does not cover all aspects of unit testing, mocking, integration testing, interface testing and other intricacies of the problem of software quality assurance. In fact, the concept of quality is also complex and administration majors tend to dwell on it for a long time, not because they are nitpickers, but because the concept is so relevant that it requires conscious analysis.

Concepts related to complex systems development are also absent from our curricula. We seldom tell our students to develop systems that involve more than two persons and never give them assignments that require tens of thousands of lines of code. Nevertheless, when a student becomes a professional, he will probably have to deal with a much more complex reality. Why don't we try to offer them a taste and the ability to reflect on that?

We should also consider reviewing the syllabi from the subjects we teach. For instance, our network classes still teach the seven layers of OSI model and thoroughly discuss TCP/IP implementation. On the other hand, the intricacies of keeping your software secure are forgotten. The consequences of this education are well documented by David Rice [6] and amount to billions of dollars and many lives (do not worry – he also spreads the blame on companies malpractice, but we cannot forget our share).

Last but not least, we do not dwell on soft skills. With a few exceptions, we do not worry about spelling and grammar errors in the papers student give us, we do not teach them how to prepare a presentation and how to talk in public[2]. Some would say that these are skills that are acquired naturally, but why can't college professors do their share? I know that you are all very busy, but these are important issues that should be put ahead of many tasks on our to-do list.

We can summarize these ideas in the following proposals which are all intended to turn our graduates into better professionals:

- Graduation projects should be divided in two parts. The first one would be an application of no less than 5.000 lines a student should develop by himself. This should include a complete solution to a business problem (which could also be a game, an educational tool or another complete app). The second part would be an application of no less than 30.000 lines that should be developed by a group of no less than five students. These application should also be developed using all the group work tools that are common to the business environment, such as version control tools, project control software and such;
- No student should graduate without developing at least one full app for a cell phone, one app for the internet and a standalone app. Students should be encouraged to solve a real problem of the community (and hence, work on their entrepreneurship skills), but apps as simple as a hangman game should help, if the interface is properly designed and they really work. The web app should also include database access and data storage, in order to make the students understand work with and improve this feature. There is no need for something fancy – keeping scores at the hangman game and allowing players to compare themselves with others will suffice;
- Network disciplines should be about hacking – attack and defense. This would increase the students' proficiency on networks and also their fun;
- No student should be allowed to graduate without amassing at least 500 points in Stack Overflow or an equivalent technical help site. This would give them writing skills while also fostering the ability to understand other people's problems, analyzing and solving them accordingly, while developing the

[2] The professor who kindly read and commented this paper is an exception to this case. He regularly offers a seminar course called "Reading, writing and researching" that purports to teach masters and PhD students how to read, write and do research (as well as present it in talks). This is an interesting example of complementation to the formal education.

programming skills while solving the problems. Besides, these sites are also scouting ground for many companies, so students who excel at these sites would also have a leverage at the job market. This could also be used with in-class activities: you could encourage students to elp each other to solve homework assignments using a common tool (Moodle, Wiki, or others) and offer them bonuses for performance, which would result in teacher workload reduction, personal network creation and better writing skills for those who answer questions;

- Parallel programming should be about creating parallel working apps, and professors could appeal to a common desire to do better and make students create distributed online apps that help solve any important problem that requires massive processing power (such as FightAids@Home [7]);
- Teachers should consider more tasks that are interdisciplinary in nature. For instance, calculus should be integrated with programming by creating programs that solve integrals using approximations. Besides, professors should be more thoughtful about the soft skills that are fundamental to performance in the work environment. Students should lose points because of grammar mistakes or bad presentation;
- Students should be required to use at least one team work technique that is common in the workplace. For instance, teams should be oriented to use Scrum and make their meetings daily and use the tools of group work. This is not only a suggestion for programming classes – study groups could be mandatory in all disciplines, with the group management also being a topic of evaluation;
- Students should receive academic credits for good participation in programming competition such as Brazilian Programming Olympics or Topcoders. No one enrolls and performs well in these competitions without achieving a high level of programming craftsmanship. Hence, such competitions should be used as motivational tools;
- Mandatory courses should include psychology 101 and human resources management. In spite of the many legends about computer "nerds", no one is ever going to work alone in this trade and basic comprehension of human factors may vastly increase the students' productivity;
- No student of computer science should ever graduate without some business understanding and some business modeling skills. Programming is not performed in a vacuum and our students should be ready to deal with the complexities of the business world.

This is neither a final nor a complete proposal. We have many forums where such ideas can be debated. In Brazil, we have, for instance, the national conference of the Brazilian Computer Society, which has specific forums on education. Besides, these proposals should be adapted to local realities, after consultations with leading software companies about their needs and their realities.

V. National Evaluation Systems

It is not just in quantum physics that an observer interferes with the observed system. In human environments, this is also always true – whenever you establish a set of quality markers, the persons under evaluation will strive to achieve the goals you gave them.

In Brazil, for instance, there are three main criteria for college evaluation: infrastructure, which is outside the control of professors, a national exam that happens every three years and publication records.

The national exam includes some programming questions. In 2014 it required the students to implement a solution to a Sudoku game using recursion and backtracking. This is pretty challenging for the short period of time the students have available, but it is not even close to a real problem. Actually, it is exactly the opposite – a token problem that is dear to the heart of college professors, even though it does not require any analytical thinking in terms of translating a problem description into a solution.

The problems of evaluating publication records are well known to anyone who has ever heard the sentence "publish or perish" and I will not dwell on them. Suffice it to say that teachers who are evaluated by their publication records tend to see students as obstacles to their "real" goals.

Nowadays it is very common for full professors to refuse teaching undergraduation courses because it does not contribute to his or her career. In the current reality, this is fully understandable – the only class a sane professor who understands the evaluation model should ever want to teach are senior level and masters level, where he can lure graduate students to work with him and increase his productivity, as measured by the powers that be.

In addition, we are in a context in which professors are hired based on their degrees and not on their teaching or research qualifications. In Brazil, for instance, colleges have to have a minimum percentage of masters and doctors in order to qualify as good or excellent. Based on this criterion, a person whose qualification is having implemented the Windows file system is considered to be less apt for employment than a recent graduate from a master's program with no real software development experience. Colleges have some leeway in this issue, but not enough.

Adding to the previous issue, we have another and most important one: professors are not evaluated by their teaching. No one is sacked or even warned for giving bad classes, being boring or not giving enough professional guidance to their students. Hence, the conclusion is obvious: go for publication, not teaching.

Even though this is a description of what happens in Brazil, it could easily compared with a similar reality in other countries.

The need to change this process is quite obvious. We need to make teaching an important part of the job, giving incentives (both monetary and career-wise) to professors who teach well evaluated undergraduate courses. Professional education should be included in the evaluation processes and measured, because of its centrality in the broader context that we have described..

VI. Conclusion

I understand that many of these proposals are offensive to many professors, who think that our teaching should be more about creating the theoretical foundation on which the students may later develop their programming skills. Nevertheless, those offended should consider the fact that professors are seen as living in an ivory tower and maybe this is not totally unfair.

One of our jobs is to provide postsecondary education to our students and provide good professionals to the society at large. This is also not a small part of our job – it should be remembered that more than 95% of our students will never set foot in a graduate program. Hence, we should provide them a course that honors their investment of time and money.

I am not proposing that we turn Computer Science colleges into trade schools. None of the proposals above suggests that theoretical work is not necessary. All are additions and enhancements of current course offerings.

The idea of teaching for the job market is neither selling out, nor a shameful proposition that should be easily dismissed by college professors. There is a reason students pay high tuitions or the State funds expensive institutions. This means that college and university professors cannot restrict themselves to a regime of academic self absorption.

It is important that we take pride in the success of our students in the business world. Every time one of them succeeds, it should be a reflection on the work we put into their education. The well being of the computer world at large should be paramount to our endeavors. As long as it is not, software companies and the whole society will suffer with the under par products of our undergraduate programs.

REFERENCES

[1] NISEN, M., "Google Has Started Hiring More People Who Didn't Go To College ", available at http://www.businessinsider.com/google-hiring-non-graduates-2013-6, 2013;

[2] McAFEE, A., "Stop Requiring College Degrees", https://hbr.org/2013/02/stop-requiring-college-degrees/, 2013

[3] WEISELD, M., "What Skills Employers Want in a Software Developer: My Conversations with Companies Who Hire Programmers", available at http://www.informit.com/articles/article.aspx?p=2156240 ,2013

[4] SPLOSKY, J., "Advice for Computer Science College Students", available at http://www.joelonsoftware.com/articles/CollegeAdvice.html, 2005

[5] MyMajors, "Computer Programmer Career", available at http://www.mymajors.com/career/computer-programmers/skills/ , 2014

[6] RICE, D., "The Real Cost of Insecure Software", Addison-Wesley Publishing, USA, 2007

[7] BARBOSA, G. C.; SILVA, B. Z.; DALPRA, H. L. O; VILARINO, I. F., PAIVA. M. M., ARBEX, W., "Computação distribuída e colaborativa aplicada à biomedicina com o FightAIDS@Home", Revista de Sistemas de Informação da FSMA n 8(2011) pp. 2 – 7