Available online at www.sciencedirect.com

ScienceDirect

Procedia - Social and Behavioral Sciences 182 (2015) 388 – 394

Procedia
Social and Behavioral Sciences

4th WORLD CONFERENCE ON EDUCATIONAL TECHNOLOGY RESEARCHES, WCETR-2014

Computer programming for all: a case-study in product design education.

Giovanni J. Contreras[a]*, Kin Wai Michael SIU[a]

[a]*School of Design, The Hong Kong Polytechnic University, v810 Jockey Club Innovation Tower Hong Kong PolyU Hung Hom, Hong Kong*

Abstract

As scientific and technological innovation has become a key component of economic growth, policies geared towards reinforcing education in these areas have become a priority. Due the exponential growth of technologies based on or relaying on software, one of the key skills identified as a basic literacy for the future is computer programming. In consequence several countries are taking steps to introduce coding in education, however, the speed at which this skill is becoming essential demands its prompt introduction at all levels. While the steps being taken in primary and secondary education are widely discussed, the extent to which computer programming has been introduced in higher education is less known; are universities addressing this demand, and if so how? Taking design education as a case study, we approach these questions by asking faculty from design schools in different countries whether their respective institutions offer any computer programming courses and in which modality. After laying down the theoretical framework for the discussion, we use a mix of qualitative interviews and questionnaires to gather information and finally discuss how the steps being taken in primary and secondary education could serve as a reference in higher education. We close by reflecting upon the implications of the information obtained to develop a work force capable of not just using technology, but creating it.

Peer-review under responsibility of Academic World Research and Education Center.

Keywords: Education; Technology; Higher Education; Design Education; Computer Programming; Coding; New Literacies; ICTs

1. Introduction

The capacity of adaptation of a species as being crucial for its survival is undoubtedly one of the most important

* Giovanni J. Contreras.Tel.: +852-2766-6845; fax: +852-2774-5067.
E-mail address: giovanni.contrerasgarcia@connect.polyu.hk

Peer-review under responsibility of Academic World Research and Education Center.
doi:10.1016/j.sbspro.2015.04.799

concepts of Darwinist evolution; those species capable of adapting better to environmental changes have the best chances of surviving. This is a concept of particular importance in today's rapidly changing environment and reason why technological change matters more than ever before; history has shown repeatedly that technology not only can bring about profound social and environmental changes as did the industrial revolution, but often faster than people can adapt to them.

1.1. Technology Disrupts

The industrial revolution not only changed how production was made, but also altered the very structure of society, yet, as history has also shown, despite its importance people often underestimate the power technological change (Thornburg, 2013). We all have heard stories of how automation and other manufacturing technologies have displaced manual labour. As robots were introduced in production lines for example the skillset needed for a job often became entirely different; a welding job at a bicycles factory that once was based on manual skill and experience, suddenly required knowing how to use a computer in order to instruct a robot—that is, to program it—which then did the actual job of welding. The same happened with a number of other professions and trades like printer, travel agent, sign writer, and even car mechanics can no longer work without knowing how to use a computer. Not surprisingly automation has been blamed for displacing several types of jobs over time and will most likely continue to do so (McKinsey, 2014).

The disrupting impact of technology at the organization level is equally serious; when companies fail to successfully predict and adapt to technological change the result is often catastrophic; stories of how companies that once dominated the market like Kodak, Blockbuster, Nokia, Blackberry of Microsoft have either disappeared or struggled to survive due to unreadiness in front of quick technological changes are abundant (Bicer, 2012). Even entire industries have been fundamentally transformed or disappeared due to disrupting technologies, like the music industry. But while automation is expected to continue displacing jobs, considering that the historical trend favours jobs requiring higher knowledge of science and technology, developing a workforce with the skill to adapt to emerging technologies, or what Shapiro & Hughes (1996) call "Emerging Technology Literacy," should be a top priority at a time of fast technological change and in which economic growth is increasingly dependent on technological innovation (Shapiro & Hughes, 1996; Brinson, 2014).

1.2. STEM and Computer Programming

Even though science technology engineering and mathematics—STEM—education has been in the educational debate relatively recently, demands to strengthen these areas as means to sustain competitiveness and economic development have been pointed out for decades; in 1983 Arizona's Governor Bruce Babbit acknowledged that the US was falling behind in front of countries like Japan and Russia which were putting more emphasis on math and engineering education than their American counterparts (Oppenheimer, 2003). Discourses like this have kept appearing in different countries ever since making similar comparisons to other countries like Korea and China and making further calls to strengthening STEM education at all levels.

But of all the abilities necessary to cope with quick technological change in the future one has been identified as fundamental; the ability to understand computer programming at a deeper level. Candy (1997) for instance notices that:

> Access to the full computational power of a system may be obtained by learning the programming languages that drive them; but, for the ordinary non-computing specialist user, these languages are arcane and, thus, the extent of that power remains highly circumscribed (Candy, 1997).

Today it would be hard to deny that this vision is at least in part correct; the days in which having access to or being able to use sophisticated computer software constituted a competitive advantage are history, as more and more technological innovations are based on or rely on software, "the basic knowledge of how lines of code create the digital worlds we explore every day is becoming a fundamental digital literacy" (Pearce, 2013), a participant in a forum rightly pointed out that these days "the future of fitness may be in hardware like Fitbit. And the future of transportation may be in services like Lyft and Uber" (Loukides, 2014) because computer programming allows people to move from just being consumers of technology to become creators of it (DeLoura & Paris, 2013).

Everyone, from scientists to lawyers can benefit from understanding more about the world of computer

programming; a study conducted at Carnegie Mellon found that many people already do programming without realizing it, when doing things like creating macros or database queries, however, not knowing anything about coding puts them in disadvantage: “if they knew something about computer science, they might not have to struggle so much later" (Shein, 2014).

In research too very often it is necessary to write code, because no software exist that can do what the researchers at the forefront of their field require, whether it is in biology, medicine, economics or any other field, this requires programming skills. Thus, it has then been highlighted that computer programming is becoming the “literacy of the future” (code.org), part of an essential information literacy, a kind of knowledge that every person should have “in the present historical context of the dawn of the information age” (Shapiro & Hughes, 1996), a new component of the ‘cannon’ of studies or a core curriculum (Marucs & Davis, 2014; Wagstaff, 2012; Shein, 2014; Henn, 2014; Trovalds, 2014).

But besides the motivation of economic competitiveness, there are pedagogic benefits attached to learning computer programming that have been studied for long. Seymour Papert the developer of the LOGO language believed that when children learn computer programming, not only they become engaged, but their learning process becomes more self-directed and purposeful (Burleson, 2005).

1.3. Programming and Design

As more and more products are technological in nature, and as technology is increasingly relying on software, if there is one field that would benefit from including computer programming courses as part of a basic curriculum it is product design. This would allow students not just to understand the nature of new products better, but to understand “what are the possibilities” (Cuni, 2014). Besides, computer programming has not only been highlighted as a creative endeavour that fosters problem solving and inventiveness (Vaidyanathan, 2012), two highly valuable skills for design students.

But there is also an increasing amount of creative projects that could have not been achieved without using some sort of programing skills, renowned artist Willliam Latham for example has produced some 3D forms using techniques that demand a good understanding of computer languages and which he could have not produced by himself (Boden, Creativity and artificial intelligence, 1998), this approach to form generation is commonly called *generative art* (Boden & Edmonds, 2009) and more specifically *generative design* when applied to design projects. Generative Design is becoming an increasingly common practice in different design domains like product and graphic design, but the software tools to do generative design require that designers have a better understanding of computer science than they usually have; for example the website contextfreeart.org provides a useful online tool that can be used to create graphics with application to different design projects, however this tool is used through a computer programming type of interface.

The emergence of ‘*visual programming languages*,’—or VPLs—which can be programmed using a visual interface rather than typing code has partially solved the needs of designers without any computer programming experience (Whitley, 1997), however even this simplified tools are difficult to use precisely due to a lack of basic notions about computer programming.

In a study at the Georgia Tech, professional graphic designers were asked to modify part of a computer program; “the idea was to see whether they could turn themselves into informal programmers and figure out how to develop automated functions in Adobe Photoshop” (Shein, 2014). However, when they tried to look for information about the code they needed, they ended up going in the wrong direction and tried to use information for *Java* when what they actually needed was *Javascript*, something that probably would have not happened if they had had a higher understanding about computer languages. The team says these findings indicate that while a lot of people need to know more about computer science, the fact that they have not been exposed to this knowledge is making them waste time and bear much frustration. This has all kinds of consequences for an organization which are often unknown or neglected; prominent creativity researcher Teresa Amabile for example has shown that personal achievement and setbacks are the most important element in predicting positive and negative experiences, and best and worst days at work. This is important because as Burleson (2005) points out; for many people out there, the technologies used on a daily basis play an important role in determining these setbacks and achievements (Burleson, 2005).

2. The Study

As it has been mentioned, calls to highlight the importance of computer literacy and computer programming from early stages of education have been made for a long time. Already in the early 1980s some schools in the US offered computer programming courses at the high school level (Oppenheimer, 2003). But while it has been pointed out that computer programming is a fundamental literacy for the future, statistical data shows that computer-programming courses remain scarce. In the US for example, computer science is the only of the fields known as STEM— science, technology, engineering and mathematics—where student participation has decreased during the last 20 years, (Wagstaff, 2012; Loukides, 2014).

Given the speed at which computer programming is becoming a fundamental literacy, a fix in higher education is necessary if readiness in front of technological change is to be achieved, waiting for new generations of students coming with a higher degree of literacy in computer programming from high school would leave several generations of university students in the limbo. However, while efforts to introduce computer programming in primary and secondary education are widely discussed, information about the actions being taken in higher education is scarcer; are universities taking steps to address this issue? What is the extent to which computer programming is being taught at universities? Do faculty support this idea? Using product design education as a case study, in this paper we approach these questions hoping to shed light into what the current situation in higher education in general may be.

In this study faculty from well-known universities in the field of design were approached using a combination of semi-structured qualitative interviews and questionnaires, the main target were professors in charge of teaching computer-related classes or those in charge of developing the curriculum for those classes, however some other professors where approached in the process of reaching the appropriate parties.

Out of all the institutions approached by the time this report was written, there were only two from which no response was received. Perhaps this was due to institutional policies discouraging faculty from speaking due fear of 'saying something wrong.' Due convenience to conduct face-to-face interviews and to avoid time-zone conflicts, universities around the same time region were approached on a first stage, table 1 shows universities from which some type of reply was obtained by the time this report was written as well as the country where they are located:

Table 1. Institutions approached.

Number of Institutions	Country
2	Spain
1	Germany
3	UK
1	Italy
1	Netherlands
2	USA

While the Royal College of Art in the UK does not offer any undergraduate courses, it was still approached as it was considered that being one of the most iconic institutions in the field of design, the information obtained it would be an important insight about what is the general view in the field.

The questions asked were organized along three main lines of inquiry aimed at knowing: 1. Whether product design students are learn programming—or if schools are teaching it, 2. Whether faculty consider this to be an important skill to learn/teach and, 3. How do they conceive the implementation of this teaching could take place. The information collected was obtained mostly online using platforms like e-mail, Facebook and Skype.

3. Where are we?

Out of all institutions from which information was obtained by the time this report was written, none of them included teaching of computer programming classes *per-se* in the standard curriculum, however one of them required students to work with programming Arduino electronic boards which something relatively similar. Later

during the 2014 'DesignEd' conference in Hong Kong it was learned that students at Tongji University in Shanghai have the same requirement. As for the rest, one school offered the same as an elective and some other schools had studio courses in which students had the opportunity to do work with Arduino as well if they wanted or if their projects demanded it (Lecownski, 2014; Cuni, 2014). Fig 2 shows the number of institutions found to teach computer programming and in which modality. As the table shows, there are 5 schools out of the 9 approached that do not report to have any type of courses where students would have the opportunity to learn about computer programming or at least something like Arduino.

Table 2. Institutions teaching computer programming.

Computer Programing	Arduino		
	Compulsory	Elective	Studio-Based
0	1	1	3

The fact that only one institution has incorporated some kind of computer programming courses—as elective—suggests that perhaps computer programming is not considered to be an important skill, however, faculty's opinion about whether students should learn computer programming suggests otherwise; while some of the professors that were contacted for this study openly stated that they did not see the need for it, some others like Cuni do support the idea of bringing computer programming closer to product design students.

In order be able to draw more definitive conclusions it is necessary to continue working on this study, however, the results obtained so far suggest that computer programing at least when it comes to higher education in product design is—at best—still incipient. The fact that almost no product design major amongst the studied institutions includes it in their standard curriculum is an important indicator of what the general situation may be, and apparently corresponds to the situation of primary and secondary education where computer programming is said not to be taught at "90% of schools" (Pearce, 2013; Shein, 2014).

3.1. Why

The incapacity of the education system to change has been blamed numerous times by technology advocates for blocking much needed reforms (Oppenheimer, 2003; Buckingham, 2007; NooNoo, 2012), as Wagstaff (2012) reckons, if we "take a look at the curriculum of many classes labelled computer science today and you'll find not much has changed from the days of dial-up modems," they only cover the basics such as learning how to type and using Word and PowerPoint (Wagstaff, 2012). Similarly Ritz (2011) acknowledges that "tradition has led many educators to teach technical expertise" (Ritz, 2011).

However, the contradictory position amongst faculty previously presented suggest that the issue is more than just a matter of speed; in a previous unpublished study by these authors, it was found that most design professors—and professional designers—conceive the computer mainly as a modelling tool. So perhaps the issue is more a matter of awareness and a matter of perception about what the computer is and what the computer can do.

In this respect we might have even gone backwards from the time when computers were first introduced in schools; "Somewhere in the mid-1990s we lost our way" asserts Dan Crow, the education system did not pay attention to the expansion of computing and the internet, and instead remained focused on teaching students "how to write Word documents" instead of fomenting an entrepreneurial attitude (Crow, 2014).

When computers were first introduced to schools there wasn't much software available for purchase, so schools used to teach how to write software in languages like BASIC. This was positive because it helped kids to see computers more as creative tools rather than production tools; "whatever you wanted it to do, it would do it—if you learned how to speak its language" (Thornburg, 2013). However, as ready-made computer software became widely available by the late 1980's, word processors and computerized spreadsheets took a central role and made of computer science a less important aspect of computer literacy. Thornburg (2013) also blames partially the economics of education for this; "as software evolved to follow education's next trajectory—and its dollars—it became increasingly focused on helping students succeed on standardized tests, turning technology away from creative construction" (Thornburg, 2013).

The same is true in the case of higher education, most of those who attended university during the 80's and even

early 90's would agree that few software was available and using a computer at that time required of a higher degree of computer science literacy; in the case of product design, students have far more tools available today than 20 years ago, design tasks have been computerized, but the degree of computer-science literacy students have acquired far from increasing may have even diminished.

3.2. Filling the gap

Already in the late 70's parents who realized the prominent role of computer programming in the future knew that waiting for the education system to make necessary changes was probably not the smartest option; as Oppenheimer (2003) recounts, many took to independent—and costly—computer camps which trained their pupils in languages like LOGO, BASIC and Pascal (Oppenheimer, 2003). These days independent organizations and advocacy groups like Coursera, Code academy, Code.org, Girls who code and others have moved in to fill the gap and are bringing programming courses to groups of all kinds; and ages and circumstances, some of which are doing an excellent job—code.org for instance offers some courses in 30 different languages. (Marucs & Davis, 2014; DeLoura & Paris, 2013; Crow, 2014). But while independent and private organizations may be doing an admirable job in filling the gap, they can only be a partial fix to a broken education system (Vaidyanathan, 2012).

In order to truly favour the creative economy this knowledge must be part of mainstream education (Wagstaff, 2012). This is where the actions being taken in primary and secondary education around the world can serve as a reference for higher education; Israel, considered "Start-Up" nation for example, made computer science part of the national high school curriculum since the 1990's, In the UK a new curriculum including programming and computing skills in primary schools has just entered in effect since September this year (Chapple, 2013), and in the US not only states like Texas and California are fighting for the recognition of computer programming as if it was another foreign language but initiatives like the CS-K10 and the Computer Science Education Act are aiming at creating "clearly defined computer science education standards" and ways to evaluate them (Henn, 2014; Wagstaff, 2012).

To determine if similar initiatives could be successfully applied to higher education further inquiry into the subject is necessary. It could be that in primary and secondary education everyone is more used to work in unison due to governmental control, however, if computer programming is ever going to be a common literacy in higher education, it will be necessary to create similar national frameworks to make that a reality.

4. Conclusion

In a landscape where the race for innovation is of such intensity that in order to have a competitive edge it is no longer enough to have access to cutting edge technology but to develop it, computer programming has been insistently highlighted as an essential skill that university graduates across all disciplines should possess. As most of today's products are becoming technological in nature, this skill appears particularly important for product design students because; today often technology itself is the product and software is an increasingly essential part of technology.

In this paper we have presented data from an ongoing study looking at the presence of computer programming courses in product design majors from different universities around the world. As we have shown, the speed at which design schools have incorporated this instruction as well as the position of faculty with respect to the need for teaching this skill at the undergraduate level leaves doubts as to whether the education system can react on time in front of the challenges posed by our rapidly changing modern world. Considering the speed at which computer programming is becoming a fundamental type of literacy, today more than ever, not understanding what the minimum technological literacy of any university graduate should be could result in a dangerous incapacity to adapt to quick and sudden changes brought about by emerging and disrupting technologies, rendering many of today's professionals illiterate.

While in some cases private and independent organizations are moving in to fill this gap, it would be a mistake to expect all students to find ways to fix a broken education. Looking at the steps being taken in primary and secondary education to implement computer programming classes on a larger scale is advisable; as we hope to have highlighted, technological change conveys an often underestimated disrupting potential that demands serious attention.

Acknowledgements

We would like to thank the Research Grants Council of Hong Kong for the support through the Hong Kong PhD Fellowship Scheme as well as the support of The Hong Kong Polytechnic University, without which this work would have not been possible.

References

Bicer, S. (2012, 09). *Adapt or Die.* Retrieved from Business War Blogspot: http://businessaswar.blogspot.hk/2012/09/adapt-or-die.html

Boden, M. A. (1998). Creativity and artificial intelligence. *Artificial Intelligence, 103*(1), 347-356.

Boden, M. A., & Edmonds, E. A. (2009). What is generative art? *Digital Creativity, 20*(1-2), 21-46.

Brinson, S. (2014, 10). *How to live a creative life that matters.* Retrieved from DIY Genius: http://www.diygenius.com/how-to-live-a-creative-life-that-matters/

Buckingham, D. (2007). *Beyond Technology.* Cambridge UK: Polity Press.

Burleson, W. (2005). Developing creativity, motivation, and self-actualization with learning systems. *International Journal of Human-Computer Studies, 63*(4), 436-451.

Candy, L. (1997). Computers and creativity support: knowledge, visualisation and collaboration. *Knowledge-Based Systems, 10*(1), 3-13.

Chapple, C. (2013, 09). *UK Govt Outlines Computer Science Curriculum.* Retrieved from Develop: http://www.develop-online.net/news/uk-govt-outlines-computer-science-curriculum/0115853

Crow, D. (2014, 02). Why Every Child Should Learn to Code. *The Guardian*.

Cuni, B. (2014, 09). Computer Programming in Design Education. (G. Contreras, Interviewer)

DeLoura, M., & Paris, R. (2013, 12). *Don't Just Play on Your Phone, Program It.* Retrieved from Whitehouse.gov: http://www.whitehouse.gov/blog/2013/12/09/don-t-just-play-your-phone-program-it

Henn, S. S. (2014, 01). *Computers Are The Future, But Does Everyone Need to Code?* Retrieved from All Teach Considered NPR: http://www.npr.org/blogs/alltechconsidered/2014/01/25/266162832/computers-are-the-future-but-does-everyone-need-to-code

Lecownski, S. (2014, 10). Computer Programming in Design Education. (G. Contreras, Interviewer)

Loukides, M. (2014, 01 22). *The Reson Everyone Should Learn to Code.* Retrieved from Radar: http://radar.oreilly.com/2014/01/the-reason-everyone-should-learn-to-code.html

Marucs, G., & Davis, E. (2014, 06). *Do We Really Need to Learn to Code.* Retrieved from The New Yorker: http://www.newyorker.com/tech/elements/do-we-really-need-to-learn-to-code

McKinsey. (2014, 12). *Automation, jobs, and the future of work.* Retrieved from McKinsey: http://www.mckinsey.com/insights/economic_studies/automation_jobs_and_the_future_of_work

NooNoo, S. (2012, 12). The Third Revolution. *The Journal*, pp. 4-7.

Oppenheimer, T. (2003). *The Flickering Mind.* New York: Random House.

Pearce, K. (2013, 12). *Why You Should Learn to Code.* Retrieved from DIY Genius: http://www.diygenius.com/learn-to-code-online/

Ritz, J. (2011). A Focus on Technological Literacy in Higher Education. *Journal of Technology Studies*, 31-40.

Shapiro, J., & Hughes, S. (1996, 03). Information Literacy as a Liberal Art. *Sequence, 31*(2), pp. 1-6.

Shein, E. (2014, 02). Should Everybody Learn to Code? *Communications of the Association for Computing Machinery, 57*(2), pp. 16-18. Retrieved from Communications of the ACM.

Thornburg, D. (2013, 01). How Disruptive Technologies are Leading the Next Great Revolution. (S. Noonoo, Interviewer)

Trovalds, L. (2014, 06). Linux creator Linus Trovalds: I do not believe everyone should learn to code. (D. Love, Interviewer) Retrieved from The Beat.com: http://venturebeat.com/2014/06/09/linux-creator-linus-torvalds-i-do-not-believe-everybody-should-learn-to-code/

Vaidyanathan, S. (2012, 09). Should Kids Learn to Code in Grade School? *Mind Shift*.

Wagstaff, K. (2012, 07). Can We Fix Coputer Science Education in America? *TIME*. Retrieved from TIME.

Whitley, K. (1997). Visual Programming Languages and the Empirical Evidence For and Against. *Journal of Visual Languages & Computing*, 109-142.