Changing the Perception of Computer Science: A Visual, Activity-Based Curriculum

Ali Fatolahi
University of Ottawa

Cate Huston
Google Waterloo

Abstract

The number of students to enroll in computer science (CS) as well as the quality of the outcome of the selection process is expected to be associated with outreach programs. A successful outreach program needs to address students with different levels of knowledge and a variety of interests. It needs to balance being exciting with the educational aspect. Instructors of such programs, being seen as role models, must be strong in practical CS skills and communicate effectively. In this paper, a visual activity-based curriculum to introduce high school students to CS is presented. The curriculum is successfully tested in the context of CS outreach programs at the University of Ottawa and the results are so far extremely positive.

1. Introduction

Although Computer Science (CS) enrolment has declined in recent years, the last two years have shown a positive trend in the US **Error! Reference source not found.**. At the same time, universities have started realizing the necessity of effective outreach programs. More and more universities consider CS outreach important in order to recruit students. For outreach to be successful, it is important to focus on students who may not be considering taking CS, but would likely be successful at it. This includes students who have no prior programming experience, but may be strong in related fields, such as mathematics, and especially female students. We view it as important to break the stereotypes about CS students and give candidates a more exciting view of what their work would look like if they take CS at university, whilst remaining realistic.

The increase of enrollment has had its own problems partly because many students, who enroll in CS lose their interest in the upcoming years. This can lead to more students dropping off the program causing difficulties for universities when planning their hiring strategies. Again, a well-executed outreach program can lead students to a proper understanding of CS and hence to involve appropriate candidates to the program. In this paper, we propose a visual activity-based curriculum to introduce CS to high school students.

We also present our experience of developing and running an outreach workshop **Error! Reference source not found.** using this curriculum. The workshop was designed to engage the interest of 16-18 year-olds in taking CS at university. The workshop is one day in duration; typically students receive around 4 hours of programming time and instruction. This presents the challenge of balancing our two goals. We want to give students an idea of what they would do in a CS course but we also hope to inspire them with what it is possible to create and more importantly to make it interesting.

In order to avoid the effect of the imbalanced prior experience of different students, several examples ranging from elementary to advanced were created with a variety of flavors such as gaming, math, artwork and image processing. Since no grade was given in the workshop, an end result was defined for the students, which was to create something "awesome" from their individual point-of-view.

The workshop is a part of a wider program called CONTACT **Error! Reference source not found.**, designed to attract high school students to engineering through hands-on experience at the University of Ottawa. The hosting university also offers workshops in Chemical Engineering and Robotics, guest lecturing from professors and graduate students, as well as one-week mini-courses on a variety of topics.

In this paper, we will outline the key design choices made in the creation of this workshop, and our experiences in running it. Results suggest the workshop is having a positive impact - in post-workshop surveys, the students report an increased interest in evaluating CS at their college/university-level of study. This is in accordance with results from the university that projects an increase in enrollments in CS in comparison to other programs.

The paper is organized as follows. In Section 2, we present the history of the workshop. In Section 3, we review the related work. Section 4 contains a discussion of design choices made. Section 5 provides an analysis of the curriculum. Section 6 summarizes the results. Finally Section 7 concludes this paper with a focus on the future work.

2. Background

The spark of this idea came when one of the authors was teaching programming (using the Java programming language) in the US at the technology summer camp, *IDTech Camp*. Students taking the other project-based courses were taking home video

games, (sometimes even 3D ones!), after weeklong courses. The programming classes were disappointed to take home a text-based hangman. The difficulty was that creating a Graphical User Interface (GUI) in Java (using the Swing libraries) is prohibitively difficult to introduce to beginners.

As a result, a prototype curriculum was developed based on a design that students and instructor had created together. A programming framework was produced, with the GUI coded by the instructor, and the students coded the inner workings themselves.

Additional frameworks were developed the following summer, and subsequently the existing curriculum was overhauled with a more visual focus. Early concepts were introduced using Processing **Error! Reference source not found.** (a language and editor for teachers and artists that makes it simple to create visual applets) as a library used within Eclipse development environment. Students had the option of a final project created using Processing, or with a Java graphical user interface (GUI) framework provided. This curriculum was taught to approximately 6,000 students aged 12-18 participating in summer programming courses across the US and in Shanghai. The *Programming* course became one of the most popular in the 2009 season.

The workshop discussed here builds on this experience. Early programming concepts such as variables and assignment, conditional logic and loops are introduced in an entirely visual way, using what we describe as an "activity-based" curriculum - inspired by Don Norman's ideas about "activity-centred design" **Error! Reference source not found.**. A low ratio of students to TAs is maintained to allow students autonomy in tasks and to enable students to take on more challenging creations.

Due to the short timeframe of the workshops, we opt to use the simpler Processing editor (shown in ***Error! Reference source not found.***) rather than Eclipse. Eclipse is an extremely complex and powerful program and has a steep learning curve, hence the popularity of simplified environments such as "Dr. Java" for the first year programming courses. The Processing editor is preferred for its simplicity. It also removes some of the detailed syntax necessary to create an applet.

3. Related Work

The need to change teaching methods of computer science (CS) with regard to programming languages has been raised by many educators. Harvey Mudd credit curriculum redesign as one of the three promising practices that they have used to dramatically increase the number of women in their CS program [6]. According to Papp-Varga et al. [7], there is a methodological uncertainty in teaching methods of programming languages and none of the existing teaching methods are based on the expectations of the learners. Blum [8] mentions that the curriculum model for CS at Carnegie Mellon favored male students, those with prior experience, and was focused on the coding rather than the end result. This agrees with the fact that more parents encourage their daughters to be actresses than engineers [9] – suggesting that much work is still needed to change perceptions of Computer Science.

Papp-Varga et al. **Error! Reference source not found.** mention a number of methods for teaching the programming skills, among which the "Sample task-based" is the closest one to ours because it allows students learn by example but a major difference is that by presenting a limited set of examples it misses the component of different levels of skill and different interest groups. We make a variety of examples available so that students, once familiar with basic examples, are free to choose examples that best suit their level and interest.

Successful experiments reported by Patton **Error! Reference source not found.** are based on different types of curriculum for different groups of students based on their interest, and that teaching programming must be based on reading rather than writing; students are given partially-written programs at the beginning to aid them in the process of learning. According to Patton **Error! Reference source not found.**, users of programming languages are changing from people who have problems to be solved using computers to those who want to program computers. This demands interactive teaching methods that help students understand what they can do with a programming language rather than knowing the details of a programming language. This study affirms our choice of methods and techniques.

Ragonis et al. **Error! Reference source not found.** study approaches in teaching CS and thus conclude the importance of teaching methods and techniques as a challenge in teaching CS especially with regard to students recently graduated from high school. The authors suggest that the variety of programs and teaching methods in teaching high school students makes it difficult to propose a unique CS teacher preparation approach. The authors confirm that high school students come from a variety of levels of interests and skills in computer science and once joined in a university-level program, it is essential to employ curricula targeting multi-level students especially in first and second year courses.

Yadav et al. **Error! Reference source not found.** verify the difficulty of incorporating computation in problem solving and suggest more active learning should be involved for teaching computational thinking in different programs. The authors agree that conventional CS teaching methods focusing on programming concepts should be changed toward

training problem solvers that are able to interact with computers effectively and efficiently.

Liu et al. [13] report efforts to teach CS to students and educators. They asked teachers to develop CS teaching curriculum and taught students using game/animation development tools. They also mention the importance of quality TAs assigned to every student. Their students complained not being able to work on their material from their own residence, which asserts the importance of respecting the open-source and remotely accessible curricula that allows students working from home as well.

Bell and Lambert **Error! Reference source not found.** indicate a large distance between the fields of CS and education, which requires more practice to create working CS curricula in accordance with educational theories. The authors suggest reducing paper-based and textual contents for teaching CS. The authors also mention the importance of active learning and involving skilled TAs.

Drake and Sung **Error! Reference source not found.** encourage the usage of games for teaching programming. This is especially true board, card and dice games. These games need less effort from the instructor to teach background and have relatively simple UI. The authors also report the overwhelmingly male nature of CS enrollment. They agree that teaching CS should be objective and focused on problem-solving rather than being concept-oriented.

Sael et al. **Error! Reference source not found.** report that most of problems in teaching CS are related to the fact that a programmer 1) needs to understand a problem as a machine would do and 2) have to come up with a solution as a human would do. Once this is done, the even harder step is to communicate the human-made solution with the machine. An effective suggestion is to avoid students being involved with syntax difficulties by choosing a simple language. For example, a simple framework such as Processing may be used in place of Java. Carlos et al. **Error! Reference source not found.** address the same problem from a more technical point-of-view by building an environment to help students avoiding low-level syntax errors and focus on algorithmic features instead.

Guzdial **Error! Reference source not found.** explains how the enrollment in CS has dramatically increased at Stanford in past two years. This has caused the need to create new faculty positions and requires more innovative and flexible CS curriculum. Because the large number of students comes from different levels, a portion of enrolments are from those who have a job in an area but would like to increase their chance to apply for internal jobs requiring CS knowledge. Other students are only joining university in light of the economic difficulties in hope a CS degree will help them finding a job.

A well-directed outreach program can also help students make the right choice and help better planning for CS courses of the 3rd and 4th years. As Cain Miller puts it, the need to redesigning curriculum by focusing on applications rather than concepts is inevitable **Error! Reference source not found.**.

4. Design Principles

Our workshop aims to incorporate the elements for Motivation 3.0 **Error! Reference source not found.** - autonomy, mastery, and purpose, particularly autonomy, through self-directed learning. Based on the lessons learnt from different rounds of the workshop as well as best practices suggested by the related work, we have set up a framework based on the following principles: Entirely Visual, Activity Based, Open Source, Effective TAs, Self Directed, Multi-Level and Multi-Interest.

4.1. Entirely Visual

Every exercise has a **visual outcome**, examples are shown in Appendix A. Students can compare what they produced to the one in the instructions. This provides a **visual honesty** lacking in text-based programming applications that, in the experience of the authors, students find frustrating and discouraging. By working visually, we turn the program from a "black-box" where the student often does not understand the relationship between the input and output, into a colourful one.

4.2. Activity-Based

When teaching programming, the following pattern often occurs: (1) Introduce concept. (2) Provide contrived example. Programmers do not normally follow this method for writing code. Programmers typically write code as follows: (1) Evaluate problem. (2) Apply solution, which could be known, looked up or invented depending on the experience of the programmer, and the complexity of the problem.

Contrived examples are boring, and patronizing to the student. In our curriculum, we instead present things that the student might want to do, for example - “repeat things” (loops), or “sometimes do one thing, sometimes do another” (conditionals), or using a grid (arrays, shown in Figure 2), and provide code with an explanation. For a one-day workshop, we cannot expect the students to memorize concepts and it is unreasonable to expect them to be interested to. Our focus is on showing them how we can solve problems programmatically and why that is fun.

4.3. Open Source

Everything about our workshop is Open-Sourced. The content is licensed under a Creative Commons Attribution-Non Commercial license. Similarly, the Processing library is licensed under the LGPL. Thus, students and teachers can distribute their code without concern, and more importantly - students and their teachers can download the software at home or at school free of charge, and continue to explore the modules in their own time. We have also documented the experiences on blogs for feedback on the design and content.

4.4. TAs

A low ratio of students to TAs (4 students to each TA) is important. Being stuck, and having to wait, is extremely frustrating to a student. Also, more one-on-one help from a TA makes it possible for students to tackle more challenging projects. TAs were interviewed and asked both technical questions and experience questions. Requirements were: (1) Fast problem solving; and (2) Good, clear, communication of their solution.

Communication is really important - firstly, a more effective communicator will teach students faster. Secondly, better communicators are more likely to break the stereotype of CS students as socially inept and not fun to hang out with. We look for TAs who are talented programmers but also can be good role models for the students. This includes hiring undergraduates, not just graduate students, who sometimes code infrequently.

4.5. Self-Directed

Related to the low ratio of students to TAs, was the self-directed nature of the workshop. The curriculum is available online, thus there is no need to share printed copies, or wait for an instructor to explain. The early modules give students familiarity with Processing, however students with prior programming experience can quickly skip ahead to more advanced examples, such as the fractal shown in Figure 3. Where possible the modules are written to be stand-alone, so no specific path is necessary. Having a good selection of modules and an open-ended project description (“make something beautiful and/or interesting”) means that students don’t have to work on the same "final" thing as their friends, if they don’t want to.

4.6 Inclusive of Interests and Levels

Processing is designed as a tool for artists, and there are several modules involving creating fractals and showing students how to create patterns. Creating games is often used to engage children and teenagers in wanting to learn to code, but by allowing this alternative, artistic track, we hope to broaden the reach of our workshop. However, for students interested in games, there is a framework for a simplified game of Pac-Man[1]. We plan to add more games, such as Brick Breaker.

We have a mix of modules of varying challenges; for example, the most tricky parts of the fractals and patterns is the mathematics, so students less comfortable writing code can work on those. An open-ended attitude means that there are always additional challenges, for example - animation! Students with prior knowledge will often define their own challenge, or even look elsewhere on the Internet and explore the 3D capabilities of Processing.

We deliberately construct an open learning environment - students are free to search elsewhere and are directed to resources like the reference on the Processing site, and Wikipedia. We have found that this encourages them to use alternative resources on the Internet to create things outside the curriculum, in a way which we hope will continue their interest in exploring Processing, and programming in general, after the workshop. Students proud of their projects are welcome to upload pictures of them to Facebook or other Social Networks, spreading our message that programming is awesome.

5. Analysis

In this section, we review our principles and verify if the curriculum contributes to those. This analysis is performed with regard to three principals that students have declared as their most favourite aspects of the workshop. These are 1) Entirely Visual, 2) Activity-Base and 3) Multiple Levels/Interests.

Entirely Visual: We had to accompany some code or provide algorithmic guidelines to embark the thinking process. Nevertheless, all examples are accompanied by a visual output and even the ratio of lines of code per example is also kept at a very minimum level that is thirteen lines per each example in average. Table 1 lists the lines of code per every visual output.

Activity-Based: The curriculum keeps introducing concepts along activities. Concepts are only introduced if necessary to help understanding the problem or designing the solution. Table 2 shows the ratio of concepts introduced for every example. According to Table 2, every example introduces roughly one new concept, while creating an

[1] Pac-Man is a trademark of Namco Bandai Games Inc

impressive visual output and solving a realistic problem, not a contrived one.

Multiple Interests/Levels: Table 3 shows how examples vary from different levels. According to Table 3, as the level of difficulty increases the number of examples of each level decreases because more advanced problems take longer to be solved. Students at advanced levels are also encouraged to look for new concepts on the Internet. With funding, we hope to increase the number of examples available at every level.

Table 1. Lines of Code per Visual Output

Example	*Code Lines*
Create an Applet	0
Size and Background	5
Drawing Shapes	5
Saving Things for Later	12
Making the Image Change	15
Sometimes Do This/That	18
Repeating Things	20
Responding to Mouse Clicks	4
Responding to Keyboard Input	28
Saving Lots of Values	22
Working with a Grid	31
Applet to Web Page	0
Pretty Web	4
Marble Fractal	18
Sierpinski Triangle	13
Average	13

Table 2. New Concepts per Visual Output

Example	*Concepts*
Create an Applet	Applet vs. Canvas (1)
Size and Background	Brackets, RGB (2)
Drawing Shapes	Drawing Shape (1)
Saving Things for Later	Variables (1)
Making the Image Change	Framing (1)
Sometimes Do This/That	Conditions, MOD (2)
Repeating Things	Iteration (1)
Responding to Mouse Clicks	Mouse Clicks (1)
Responding to Keyboard Input	Keyboard (1)
Saving Lots of Values	Array, Type Cast (2)
Working with a Grid	2D Arrays (1)
Applet to Web Page	HTML Applet (1)
Pretty Web	0
Marble Fractal	Recursion (1)
Sierpinski Triangle	Square Root (1)
Average	1.13

Table 3. Levels of Examples

Example	*Concepts*
Create an Applet	Basic
Size and Background	Basic
Drawing Shapes	Basic
Saving Things for Later	Basic
Making the Image Change	Intermediate
Sometimes Do This/That	Basic
Repeating Things	Basic
Responding to Mouse Clicks	Intermediate
Responding to Keyboard Input	Intermediate
Saving Lots of Values	Intermediate
Working with a Grid	Intermediate
Applet to Web Page	Basic
Pretty Web	Advanced
Marble Fractal	Advanced
Sierpinski Triangle	Advanced

6. Results

The workshop was held three times with roughly 90 students in total. At the end of each workshop, the students fill in a survey about their experiences. Overall, the response has been extremely positive.

The objective of the workshops was to encourage students to choose CS/engineering as their future program of study. Results from workshops based on the evaluation forms filled out by students show that more than 85% indicate they would be interested in participating in the events and programs organized by the Faculty of Engineering. Of this 85%, 55% have expressed their interest to be contacted about the program, while of the 45% who have said they would not be interested in being communicated with 11% have mentioned distance as the reason. More specifically, 62% have said they either liked it or loved it. This may seem low but the reader should not that students were randomly selected and assigned to workshops by the school administration and not based on their own interest.

According to the comments given in the feedback forms, what students enjoyed most was the interactive context of the workshop. They discover working with objects and formulas could be easy and that they can really see what they have developed.

To 14% of students the workshop seemed not be helpful and for 24% of them the workshop was interesting but did not encourage them specifically towards programming. 60% have said the workshop encouraged them to consider CS as their future university program.

Reports of the hosting university show the number of applications increased by 4.22% and 4.41% for Software Engineering and Computer Science, respectively compared to the previous year. The interesting statistic is the acceptances, which increased by 116.67% and 104.55% for Software Engineering and Computer Science, respectively. It is also understood that this is not a Faculty wide trend, since the increase for the other programs is far

less than the one of CS. In time, the university will be able to see how much of this increase resulted from the workshops and how mush is caused by other components of the CONTACT program as well as other factors including external ones.

7. Conclusion

We introduced a visual activity-based curriculum for teaching CS to high-school students in this paper. The curriculum is designed to motivate students to consider CS as their university-level program. This is in line with the increased focus of universities in extending and improving their outreach programs.

The curriculum follows several principles to involve students actively and to let them work individually toward a visual goal. Students can choose among a variety of examples based on their level of skill and their interests. The content is available online and the tools used are open-source. Results show that the workshop has encouraged the majority of students to consider CS as their future program. This is aligned with the official statistics of the hosting university that suggest over 100% increases in CS acceptances over the past year.

The future work is focused on developing more examples and content. Also, we aim at creating a more effective strategy to create a network of students involved in workshops and continue interacting with them. Finally, it would be interesting to see if a similar approach could also help reducing the number of 3rd and 4th year drop-offs. We have demonstrated that by revising a non-activity-based CS curriculum to an activity-based one, we can appeal to a wider variety of people, there are an increasing number of examples of this (e.g. Carnegie Mellon and Harvey Mudd). Outreach should be just the beginning.

8. Acknowledgements

The authors would like to thank Professor Marcel Turcotte of the School of Electrical and Computer Science of the University of Ottawa for supporting the workshop and the research process associated with it.

10. References

[1] Peter Harsha, “CRA Taulbee Report: CS Enrollments, New Majors Up For 2nd Straight Year”, Computing Research Policy Blog, March 24th, 2010

[2] Cate Huston, (2010), “Processing Workshop”, http://www.catehuston.com/workshop. (Access date: 13 March, 2011).

[3] uOttawa, (2010), CONTACT program, http://www.engineering.uottawa.ca/en/contact.

[4] Processing, (2010), http://www.processing.org. (Access date: 13 March, 2011).

[5] [10] Don Norman, (2010), “Activity-Centered Design: Why I like my Harmony Remote Control”, http://jnd.org/dn.mss/activity-

[6] Christine Alvarado and Zachary Dodds. “Women in CS: An Evaluation of Three Promising Practices.” In Proceedings of ACM Technical Symposium on Computer Science Education (SIGCSE). March, 2010

[7] Zsuzsanna Papp-Varga, Péter Szlávi, László Zsakó, “ICT teaching methods – Programming languages” Annales Mathematicae et Informaticae 35 (2008) pp. 163–172

[8] Lenore Blum, “Transforming the Culture of Computing at Carnegie Mellon” in Computing Research News, vol. 13, No. 5, November, 2001, p.2.

[9] Denise Dubie, “Mommas don't let their babies grow up to be engineers”, IT World, February 9th, 2009.

[10] Peter C. Patton, “New Methods for Teaching Programming Languages to both Engineering and Computer Science Students”, In Proceedings of Midwest Instruction and Computing Symposium 2004.

[11] Noa Ragonis, Orit Hazzan and Judith Gal-Ezzar, “A study on attitudes and emphases in computer science teacher preparation” in the Proceedings of the 42nd ACM technical symposium on Computer science education 2011, pp 559-564.

[12] Aman Yadav, Ninger Zhou, Chris Mayfield, Susanne Hambrusch and John T. Korn, “Introducing computational thinking in education courses”, in the Proceedings of the 42nd ACM technical symposium on Computer science education 2011, pp 465-470.

[13] Jiangjiang Liu, Cheng-Hsien Lin, Ethan Philip Hasson and Zebulun David Barnett, “Introducing computer science to K-12 through a summer computing workshop for teachers”, in the Proceedings of the 42nd ACM technical symposium on Computer science education 2011, pp. 389-394.

[14] Tim Bell and Lynn Lambert, “Teaching Computer Science majors about teaching Computer Science”, in the Proceedings of the 42nd ACM technical symposium on Computer science education 2011, pp. 541-546.

[15] Peter Drake and Kevin Sung, “Teaching introductory programming with popular board games”, in the Proceedings of the 42nd ACM technical symposium on Computer science education 2011, pp. 619-624.

[16] Mara Sael, Jacob Perrenet, Wim M. G. Jochems and Bert Zwaneveld. “Teaching Programming in Secondary School. A Pedagogical Content Knowledge Perspective”, Informatics in Education, 2011, Vol. 10, No. 1, pp. 73-88.

[17] Carlos R. Jaimez-González, Christian Sánchez-Sánchez, J. Sergio Zepeda-Hernández, A Web Platform for Creating and Administering Interactive Online Tutorials, In Proceedings of International Canada Education Conference 2011, pp. 88-92.

[18] Dangers of Escalating Enrollements, http://computinged.wordpress.com/2011/04/13/guest-post-eric-roberts-on-the-dangers-of-escalating-enrollments/, June 23, 2011.

[19] Hollywood Spurs Surge in Computer Science, http://www.nytimes.com/2011/06/11/technology/11computing.html, June-28-2001.

[20] Daniel Pink, “Drive: The Surprising Truth About What Motivates Us”, Riverhead, 2009.

Appendix A – Sample Outputs

Figure 1. Pretty Web

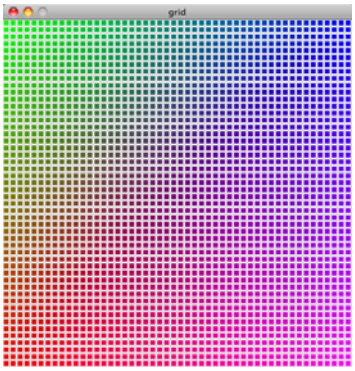

Figure 2. Grid

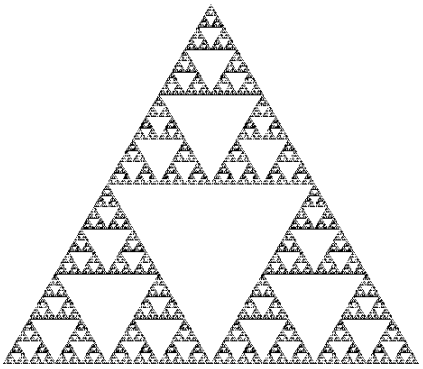

Figure 3. Fractal