EDITORIAL PREFACE

Special Issue on Kids and Other Novices Learning to Code: Insights, Tools and Lessons from the Visual Programming Frontline

Steve Goschnick, Swinburne University of Technology, Melbourne, Australia

Leon Sterling, Swinburne University of Technology, Melbourne, Australia

The theme for this issue is the environments designed to help novices to learn to code, with insights about tools and lessons from some of those that design, build and support the environments.

Currently there is much discussion around the world about the place of computer programming in school curricula. There is a spreading realisation of the shortage of skilled coders to drive innovation and productivity improvements for business, government and the community. Further, that the need for coders continues to accelerate. Many questions arise from that realisation. When should education in programming start? Are children ready or is it a graduate skill? Should computer programming be in school curricula and should it be a mandatory topic?

Different countries are taking different approaches, but there is general agreement that over previous decades there have been deficiencies in the way that IT/computing has been taught at schools. There has been a lack of qualified teachers, and a lack of understanding of careers in the IT field, the activities taken by people working in the IT industry, and how IT careers are changing. There has been confusion between computer literacy and computational thinking which we believe has contributed to a lack of engagement of students at school. At the heart of all these problems has been the lack of appropriate tools to introduce novices to the idea of coding and the concepts involved.

We believe that the tide has turned. Led most notably by Scratch (Resnick et al, 2009), educators are using block structured languages and community environments, and kids are thriving. Fun, engaging environments are becoming the norm. There is much anecdotal evidence that the new environments work for both student and teacher.

Our Call for Papers for this special issue took as its title - Kids and Other Novices Learning to Code: Insights, Tools & Lessons from the Visual Programming Frontline.

It read as follows: "We are interested to hear of your research and experience with using and/or providing languages and programming environments designed to take whole new cohorts of people into Coding. Of interest is the whole span: 8 and 9 year-old school kids and younger, through to life-long learners who never thought about programming as a realistic option for themselves.

The well-known options include: Scratch, Alice, Greenfoot, AgentSheets, FLIP and various derivatives of Scratch (e.g. Snap!/BYOB, App Inventor, Blockly, FlashBlocks), derivatives of Blockly (e.g. Code.org, WonderWorkshop, Blockly Games, etc.), and various other approaches, including executable flowchart environments (e.g. LARP).

Topics include but are not limited to:

- Are colored block-based languages more effective than other approaches?
- What are the lesser known but effective language and environment options?
- Which features of current languages and environments help the student and which ones hinder them?
- Are environments that allow online collaboration and sharing with fellow learners from afar, better or not than collaboration with peers in the immediate classroom?
- Are teams, even in pairs, effective or disruptive to getting all learners to understand coding concepts?
- What data can be harvested from the coding environment, and how can it be used as feedback: to identify students facing conceptual difficulties, to improve coding exercises, and mentoring?
- Is learning to code enhanced when done in conjunction with robotics?
- How motivational is it to have a coding environment for apps that can be distributed to a large audience? E.g. App Inventor for Android.
- Is the quest to transition the new coder from a blocks-based language, to a conventional script language (e.g. Python, JavaScript) - all within the coding environment and the lessons - useful in teaching all potential learners, or simply a filter to identify future programming talent?
- When and how it useful to introduce other skills into a team project (e.g. art and design of graphic and video content), allowing individual students to build on existing strengths and weaknesses?
- How can the current crop of code learning environments and languages, be improved upon?

We had a diversity of responses, from which we have chosen three items plus a review of ScratchJr, an app available on both Android and iOS devices, aimed at children as young as six or seven years old.

The first piece is adapted from a blog post by Mitchel Resnick, the designer of Scratch. While not a formal paper in the usual sense, we felt it appropriate to include it in this special issue. The blog accurately presents the opinions of the authors, Mitchel Resnick and David Siegel, founders of the Scratch Foundation in 2013. It represents the points that they most care about. The block-based language in Scratch has gone on to directly influence other block-based language environments, including App Inventor for Android, and the Blockly open-source JavaScript library used in implementing many newer block-based language environments.

The second piece is 'Lessons From The Design of Three Educational Programming Environments: Blue, BlueJ and Greenfoot', a full length paper by Michael Kolling. Kolling is a pioneer in providing environments for learning coding. He is in the unusual and valuable position of

having been centrally involved in designing and developing three programming environments for teaching programming to novices, over a period of more than twenty years. The involvement includes cultivating and servicing input from communities of learners, and teachers, for ongoing improvements to both BlueJ and Greenfoot. BlueJ, introduced in the year 1999, was aimed at 1st year undergraduate students, teaching them 'objects first' in the Java language. Kolling soon realised that many 1st year undergraduate students were learning their first programming language well before beginning their tertiary studies. He then designed Greenfoot to address the needs of students and teachers at mid level high school.

In his paper, Kolling outlines that history, often in relation to other environments, notably Scratch and Alice. Significantly, he distils his hard-won advice gained in the design decisions of the three environments, and in supporting the large user communities of two of them. The advice is well suited to current or would-be designers of new learning environments for coding. It is equally well suited to those wanting to choose between existing environments in teaching programming/coding to a particular target user base.

His sub-headings within the section The Lessons Learned, are good to repeat here as the briefest of a summary: Visualization, interaction, simplicity; Know your target group; Do one thing well; Say no (to features requests beyond the target group); Do not be afraid to make decisions; Availability; Support material matters; The importance of motivation; Taking control (and the learner taking ownership); Visualization of program execution; The power of community; Teacher communities.

His own endeavours have seen the target user group for such environments go from first year university programming novices to high school student novices. Further he has observed the block-based languages successfully targeting primary school kids as young as seven.

There is now a new cohort of students that go from block-based environments for their first foray into coding, then learn a text-based syntax language later on if they need to take programming more seriously. Some of the new tools do both styles of coding in the one environment, demonstrating a two-way approach to coding, sometimes in dual side-by-side windows.

Towards the end of his paper Kolling introduces his own new work on Greenfoot, with a technique he calls frame-based editing that combines aspects of both block and text code editing in a single merged approach. The new version aims to enable novices who are first introduced to coding in the new environment, to be able to continue on in it, taking their coding as far as they want to go.

The last full paper is titled: UDOO App Inventor: Introducing novices to the Internet of Things. Apps have been a great motivator for kids wanting to learn to code, and App Inventor for Android - a joint project between Google and the MIT Media Lab - has been arguably the most successful approach to date. This paper is from a lesser known source and presents an extension to App Inventor called UDOO, that adds some new blocks to the language, to deal with new hardware. This has been possible since the original source code of App Inventor is open source, allowing the UDOO developers to incorporate App Inventor code into their device, giving it features beyond a standard Android. The paper shows how they use App Inventor and the UDOO hardware to connect it to all sorts of sensors, all from within the familiar block-based language of the App Inventor environment. UDOO App Inventor allows novices to get direct experience of coding sensors and firing up actuators in the real world beyond the touchscreen of their mobile devices.

Finally, we present a review of the ScratchJr (junior) App, the app-based coding environment aimed at introducing and teaching very young children how to code in the Scratch block-based

language. Unlike the Scratch environment, a web-based technology, ScratchJr is available on both Android devices and the iPad, outside the browser.

It is a pleasure and a privilege to contribute to the active discussion about coding in the curriculum. At a personal level, we have been advocating for coding on the Australian scene for some time, e.g.: (Goschnick, 2015) and Sterling, (2015). We look forward to the continuing conversation.

Steve Goschnick
Leon Sterling
Editors-in-Chief
IJPOP

REFERENCES

Goschnick, S. (2015). Want your kids to learn another language? Teach them code. *The Conversation*. Retrieved from https://theconversation.com/want-your-kids-to-learn-another-language-teach-them-code-47409

Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., & Kafai, Y. et al. (2009, November). Scratch: Programming for all. *Communications of the ACM*, *52*(11), 60–67. doi:10.1145/1592761.1592779

Sterling, L. (2015). An education for the 21st century means teaching coding in schools. *The Conversation.* Retrieved from https://theconversation.com/an-education-for-the-21st-century-means-teaching-coding-in-schools-42046

Steve Goschnick is an Adjunct Professor in the School of Design, at Swinburne University of Technology, Australia. His research interests include people-oriented programming, intelligent agents, user interaction design, programming environments, data modeling and meta-models. He developed and taught foundational masters level subjects at The University of Melbourne for 10 years, into the MIS, MIT and MBIT courses, up until 2013. He has consulted widely in industry and been called upon as an expert witness in both programming and interaction design. He is the founding co-editor-in-chief of the International Journal of People-Oriented Programming.