Conversation with a Prominent Propagator: Michael Kölling

David P. Bunde
Zack Butler
Christopher L. Hovey
Cynthia Taylor

In order to improve teaching in computer science, pedagogical innovations must be propagated beyond the original creators to the larger community of CS educators. Accomplishing this requires time, energy, and planning. We believe that the CS education community must engage in a sustained effort to learn more about evidence-based strategies for increasing the adoption of educational practices. This column is part of our continuing efforts [3, 5] to capture the knowledge within our community through interviews with *prominent propagators*, those who have been successful spreading educational innovations to other instructors.

In this column, we interview Michael Kölling, Vice Dean for Education of the Faculty of Natural and Mathematical Sciences at King's College London. Michael has started and now leads several high profile projects in Computer Science Education. The first and probably best known is BlueJ [2, 9], a Java IDE aimed at introductory students that lets students create and manipulate objects through the GUI. Later came Greenfoot [6, 8], an IDE aimed at pre-college students that facilitates the creation of simulations and games. Building on BlueJ, his group has also created Blackbox [4], a collection of anonymized student code written using BlueJ for analysis by the CS Education research community.

Below are highlights of the interview, which ran approximately an hour. They have been edited for clarity and style.

Q: So how did you start getting BlueJ out to other people? How did it go beyond just you using it?

MK: It was published first in 1999, so it's now more than 20 years old. My PhD project was something very similar to BlueJ. It was an object-oriented language and environment for teaching. So we did think about it when I started as something like that, the Pascal for object orientation. It included not only the programming language, but also the development environment, and that was called Blue. Then after I finished my PhD, I thought about what I wanted to do next, and at that time, Java was just becoming very popular. But, it didn't have a very nice environment.

So I thought I can either continue developing Blue and it remains a purely academic project, or I can throw away the language and do the same again for Java. This was essentially taking half of the project, the environment, and adapting it for Java, but with a chance of actually getting some real users because Java is the language that was being used. So that's what I decided to do. Then we come to the question that you just asked: how do you get people to adopt it?

The first thing I did was write academic papers and go around to conferences and present those papers at things like the SIGCSE Symposium and the Australasian Computing Education conference. That was essentially the only thing we did to try to popularize it or generate exposure. The first couple of years, there was just small, regular uptake, a small number of institutions that started using it just based on the software itself, but it wasn't really very much yet. It wasn't booming at the time.

Q: What happened to make that change?

MK: Writing a book about it [1]. We did it a second time with Greenfoot [7]. The one big lesson I learned throughout this whole thing is that software alone gets you very little. Even if it's good, just putting software out there doesn't have a great chance of adoption. What actually makes the difference is having teaching material that shows teachers exactly what to do with the software. And having good software out there that helps people and that people want to use is a necessary prerequisite, but it's not sufficient. That will get you adoption by maybe, 10% or so of educators who are very confident and have a good insight, and understand the subject very well. What most educators really want is to be shown how to teach with it, and you can show them good teaching material, good teaching practices, and say, “This is what you can do with the software.” So we wrote a textbook that has very detailed instruction on how to teach with BlueJ. That made a huge difference.

Q: Was it just the book that made BlueJ take off?

MK: No. We also consistently, continuously worked on improving the software. I think you have to do that as well. There is no stable state in software; either you continuously update or it dies. We set up mailing lists and we tried to have communications with users and got feedback and we took a lot onboard, and we very regularly updated it. Partly, just to keep current with operating system changes—if you don't do anything, 18 months later your software doesn't run anymore. And partly, we added functionality over time, quite significantly, often based on user requests.

I also have to say, a few years later, once the adoption was high, I saw my job leading the development team as often saying "no" to everyone. A lot of people had ideas that they wanted added to the software and we had a long list of requests. Many of them when taken on their own were very sensible. But if you take them all together, it would have completely ruined the system because one of the major aims of our system was simplicity in usability. Under the hood, there was some quite sophisticated stuff there. But from a user perspective, a new inexperienced programmer, the clarity of interface and simplicity was very important to us.

Q: What’s been your motivation to continue the ongoing development work?

MK: I was always interested in translating my work into real systems that people actually use in a university setting. That's difficult because typically we get funding for initial development but there's absolutely no funding for sustaining anything, even without making functional changes, just adapting to new Java versions and new operating system versions and this and that. Writing new video drivers is constant work that gives you no academic credit whatsoever. You can't write a paper about that. It's boring work, and you get no recognition, you get no funding normally for it. And we've been lucky that we had industry funding for a long time that allows us to do that.

Q: How did industry get interested?

MK: It was a bit of luck. Fairly early on, at a conference, my PhD supervisor met a senior manager from Sun Microsystems and they were trying to promote Java. And we were working on the environments and they said, “Yes, but can you do it in Java?” He was senior enough that he gave us a little bit of money. It wasn't much but it was enough to develop the small prototype and go back to them and say, "Here, look, we can do it." And then gradually our relationship developed with Sun Microsystems; they gave us more funding and ultimately to a level where we could employ a couple of people to work on it full time.

Q: How much of the development work is maintenance versus expansions and new functionality?

MK: It splits into many different things actually because a team of four of us are doing all the development, including maintenance and user support. We have support email, and we answer every support email within 24 hours. So we have this low-level work, the boring work to keep things running, and then development work. But also we are at a university and we are a research team. It's always been important to us to also do active research on the project. These same people who write the code also do research studies, they write papers, they go to conferences, they're all academic post-grads. So it splits off into maintenance and design and original development and research and presentation and we also have a number of different projects that are spinoff projects from BlueJ.

Q: What made you branch off from BlueJ into other projects, like Greenfoot?

MK: Well, BlueJ has been going for a long time. We are academics, and we want to do research and we want to write papers, but you can write papers only if you've got something interesting new to say. In the core BlueJ system, there isn't really anything to write a paper about anymore. So we have continuously tried to investigate things at the current edge of programming education.

Greenfoot was published in 2006, driven by the observation that introductory programming education was moving down the age groups. There was a large drop in enrollment in computer science in the early 2000s. The problem was that pupils had already decided before university that they didn't want to do computer science. They weren't interested. Making teaching better at university has absolutely no impact on this because they never even see it. We needed to go into schools because that's where the decision making is happening. Greenfoot is a different environment with slightly different goals, custom-made for school education.

Q: What was your process for customizing this for school education?

MK: That essentially came out of discussions either in our group or with teachers. It was the experience of actually being immersed in that community. I did a lot of workshops with teachers. I had contact with the teachers. It was this firsthand and secondhand experience, being in the community and hearing from them.

Q: How has the process of starting something new changed from when you first began this work?

MK: Yeah, it has changed a lot. Partly, the context and environment has changed, and partly things have changed for our group specifically because we have some name recognition. If we publish something, it's probably a bit easier to get some visibility than for someone who's just starting out. Although, it's still not easy—if you invent your own new thing, whatever it is, your new language, your new tool or something, it's always very hard to make people take notice. That remains the greatest challenge for tool designers.

The situation compared to 20 years ago has changed completely in that when we started, there were hardly any educational tools around. There was Pascal in the seventies. Then there was Karel the robot in the eighties and early nineties, but there wasn't an IDE. BlueJ, and Alice at roughly the same time, were the first ones that came out and for a long time there was nothing else. At that time, the real challenge was to convince teachers to use an educational system. People said, "No, you have to use Microsoft Visual Studio or Eclipse because that's what real programmers do." That has changed now and the opposite is true. There are so many educational systems that it has become hard for teachers to evaluate them and choose which one to use. You don't have the time to evaluate them all. There's a lot of competition out there now.

Q: What advice would you give to other people who want to do something like you've done?

MK: You have to have a really good system that does something brilliantly; you can't fake it. Your tool has to solve a problem that someone actually has. Teachers will not adopt and continue using a tool if that is not true. And to do that you have to listen to your users. You have to have a clear idea who your users are. You have to talk to them.

The real advice is the second step. Once you have a good idea and you have a system, what you need to do is you need to develop material and a community around it. That's what drives adoption.

Q: How do you build a community?

MK: Originally we had a mailing list, that solves part of the problem. You can at least ask questions and write to people. But that wasn't really sufficient. So quite some time ago, we developed a community website for educators who teach with our tools. But it's very hard to create engagement. So we tried to think about that and do a lot in the functional design and user interface design to encourage engagement. We decided to base it on a Wiki model that everyone can edit everything in, where we don't have a gatekeeper, but where we still try to achieve a level of quality by allowing people to improve each other's contributions and collaborate.

Most of our users are school teachers. I have no idea what kind of material they want, and what it should look like, and what format they expect: how to write a lesson plan so that someone can just take it and deliver it tomorrow. So then we thought we should get actual teachers to write the material. It's difficult, but when it happens, it's good and I think it really comes down to that your tool has to be good enough that it excites people, so that there are some teachers who are really energetic and can write material to get people excited about your systems.

Q: When do you know a project is successful?

MK: My definition of success has shifted significantly. I knew BlueJ was going to be successful quite quickly because after a few months, one or two other universities had adopted it and I thought that was success. I was really proud. I thought, there's someone who I don't know personally who is using it. And I thought that was success and I was happy with that. Then, after the first year, a handful of universities also adopted it and that was great. That's all I was aiming for when I originally wrote it. Then it became a lot bigger as time went on. The longer you do the project, and especially if it continues to rise, you aim higher over time. You also have more resources and more experience and so now you can achieve more.

Q: What advice would you give to someone with a great idea and who wants to have a big impact?

MK: One thing that's actually very hard for people to solve is that in academia, you get no funding for doing all the support work and you get no academic credit for it. And so it's difficult to put into practice. We've seen this several times with spin-offs from our systems. Because they're open source, people can take them and make variations. Every time these spin-offs have died fairly quickly, mostly because people underestimate the work it takes to keep these systems alive. Academia is not set up to provide an environment where you can easily do that. Getting the resources to keep the systems alive is challenging. You need to constantly come up with things that are novel enough that they are original research. But at the same time, you also have to manage to not ruin your system by constantly meddling with it. That's a fine line to tread.

References

[1] Barnes, D.J. and Kölling, M. 2017. *Objects First with Java: A Practical Introduction Using BlueJ*. Pearson.

[2] BlueJ: 2020. *https://bluej.org*. Accessed: 2020-07-01.

[3] Bort, H., Bunde, D.P., Butler, Z., Hovey, C.L., Spacco, J. and Taylor, C. 2020. Conversation with a prominent propagator: Leo Porter. *ACM Inroads*. 11, 1 (2020), 12–15. DOI:https://doi.org/10.1145/3381023.

[4] Brown, N.C.C., Altadmri, A., Sentance, S. and Kölling, M. 2018. Blackbox, Five Years On: An Evaluation of a Large-scale Programming Data Collection Project. *Proceedings of the 14th Annual ACM Conference on International Computing Education Research (ICER)* (August 10-12, 2018, Espoo, Finland, Aug. 2018), 196–204.

[5] Bunde, D.P., Butler, Z., Hovey, C.L. and Taylor, C. 2020. Conversation with a Prominent Propagator: Sushil Prasad. *ACM Inroads*. 11, 3 (2020), 22–24.

[6] Greenfoot: Teach and learn Java programming: 2020. *https://www.greenfoot.org/door*. Accessed: 2020-07-01.

[7] Kölling, M. 2016. *Introduction to Programming with Greenfoot: Object–Oriented Programming in Java^{TM} with Games and Simulations*. Pearson.

[8] Kölling, M. 2010. The Greenfoot Programming Environment. *ACM Transactions on Computing Education*. 10, 4 (2010), 14:1–14:21. DOI:https://doi.org/10.1145/1868358.1868361.

[9] Kölling, M., Quig, B., Patterson, A. and Rosenberg, J. 2003. The BlueJ System and its Pedagogy. *Computer Science Education*. 13, 4 (2003), 249–268. DOI:https://doi.org/10.1076/csed.13.4.249.17496.

David P. Bunde
Knox College
2 E. South St
Galesburg, Illinois 61401 USA
dbunde@knox.edu

Zack Butler
Rochester Institute of Technology
Rochester, NY 14623 USA
zjb@cs.rit.edu

Christopher L. Hovey
University of Colorado Boulder
1045 18th Street, UCB 315
Boulder, CO 80309
hoveyc@colorado.edu

Cynthia Taylor
Oberlin College
10 N Professor St
Oberlin OH, 44074
ctaylor@oberlin.edu