Bertrand Meyer: Software Engineering and the Eiffel Programming Language

Charles Severance, University of Michigan

For more than 30 years, Bertrand Meyer has taught the computer science field about formal software engineering and object-oriented approaches; he continues do so through his many books, conferences, and Eiffel.

Bertrand Meyer has spent his career at the nexus of computer science education, object-oriented programming, and formal approaches to software engineering. His earliest work involved the Simula programming language:

I had the great privilege and good fortune of being exposed to Simula-67. Simula was the dominant view of object-oriented programming at that time [mid-1970s]. I got introduced to it and immediately knew it was the right way to program—it was pretty obvious, at least to me. For a number of years, I worked with Simula both to help my research work and to build systems.

Meyer's decades of work in understanding object-oriented programming patterns are reflected in his popular book, Object-Oriented Software Construction (Prentice Hall, 1988). He also cowrote the first formal software engineering paper in 1977 on the concept of Z notation (often called Zed). To watch a video of our conversation, visit www.computer.org/computingconversations.

I looked at C++. I opened the book—I closed it pretty quickly afterwards. There was also Objective-C and Smalltalk; these were all interesting developments, but they didn't really correspond to the strict software engineering standards that my colleagues and I had learned to observe.

The more recent object-oriented languages either moved toward a typeless approach or implemented a subset of the rich object-oriented concepts present in Simula. Instead of compromising on his approaches to software development, Meyer decided to use a language of his own making that could cross-compile to C and get his ArchiText project started:

We developed our own language, which was based on a book that I had written that I had never finished. It had a very precise notation for expressing algorithms, so I used it as the syntax...
Meyer and his team used his new makeshift language to quickly develop the ArchiText editor in time for the company to generate customer interest by displaying it in a booth at the inaugural Object-Oriented Programming, Systems, and Applications (OOPSLA) conference in 1986. Attendees found the programming language that Meyer had invented far more interesting than the ArchiText application he had developed using the language. According to Annie,

At OOPSLA, Bertrand had some tutorials, and that was the first place where Eiffel was really exhibited. From then on, it was clear that the focus of the company was on Eiffel and not ArchiText. We would build tools to help programmers make the most of the power of object-oriented technology.

After returning home from OOPSLA, Meyer realized he wasn’t alone in having reservations about approaches being taken by the mid-1980s object-oriented programming languages such as SmallTalk and C++. He quickly made his rough Eiffel development environment available to a few interested customers.

People who started playing with the language, even with the very primitive implementation we had at the time, started telling us that there was something absolutely new that I had never seen before, which is how easily you can change your mind. One of the major differences between Eiffel and other technologies is how easily you can have a first design—a clean design. I’m not talking about agile-style hack it and see if it works. A good design that isn’t perfect, and you realize it isn’t perfect, and you change it, and you don’t spend your entire life paying for the sins of your youth, so to speak.

A NEW TOOL

Because Eiffel provided the full breadth of object-oriented features and had formal software engineering support throughout, it provided an environment that addressed the entire software development life cycle from the initial design and specifications down to implementation, debugging, performance tuning, and the ability to evolve and extend the application over a long period of time:

One of the key distinctive features of Eiffel is that it’s a life cycle approach, so that it isn’t just for programming. It’s for analysis, design, implementation, maintenance, and testing as well. So it’s kind of a holistic view of software development. This approach still goes against the grain of the software engineering culture today. Most people think they need some kind of high-level requirements tools, some CASE tools to do analysis and design, and then Eclipse or something like Visual Studio to do implementation, and then JUnit to do testing.

What we do [in Eiffel] is integrate everything, which means that the developer doesn’t need to switch between Dr. Jekyll and Mr. Hyde all the time to switch gears. You stay in the same conceptual framework, using Eiffel’s basic ideas—classes, inheritance, single and multiple, contracts, the use of formal precise specification elements in association with every piece of software, and a few more. These ideas apply throughout the life cycle, throughout the process from the highest, most abstract levels of thinking about the system all the way down to the nuts and bolts, the nitty gritty of the software.

The company started out relatively small, building the Eiffel software and traveling around giving tutorials and educating developers about the Eiffel environment, which gave developers the benefits of formal software engineering and rich object-oriented programming.

Annie Meyer started doing office work for the eiffel.com company. Over time, as she learned new skills and the staffing changed (including one tragic incident in which a business manager was killed in a fatal airplane crash), Annie slowly took on additional responsibilities to the point where, 20 years later, she’s now the CEO and runs eiffel.com’s day-to-day operations. The company continues to develop and evolve the Eiffel programming environment as well as provide training and conferences for the Eiffel community.

A STRONG COMMUNITY

Like the Simula community in the late 1960s, the Eiffel community is small, and most programmers might not even know that the Eiffel language exists. But the Eiffel community is dedicated and passionate about a product that allows them to approach their software development projects using a powerful framework and language. According to Bertrand Meyer,

The typical Eiffel user is someone who has a difficult application for which he or she has tried something else before and failed. Eiffel is for people who...
just cannot afford for stuff to fail—reliability and quality requirements are very high, often with continuous operation.

Meyer teaches the Introduction to Programming course for computer science majors at ETH Zurich using the Eiffel language and his book Touch of Class (Springer, 2009). Using Eiffel in the class allows his students to achieve a wide range of skill levels. Using a large existing class library that supports traffic simulations, students with less experience can quickly write relatively sophisticated and engaging applications while leaving the complexity behind the abstraction. More sophisticated students can immediately peer into the nature of the abstraction and the code, which is fully documented with well-specified software designs as well as contracts for each of the modules in the class library.

For more than 30 years, Bertrand Meyer has taught computer scientists about formal software engineering and object-oriented approaches, and he continues do so through his many books, conferences, and the Eiffel language. C

Charles Severance, Computing Conversations column editor and Computer’s multimedia editor, is a clinical associate professor and teaches in the School of Information at the University of Michigan. You can follow him on Twitter @drchuck or contact him at csev@umich.edu.

Selected CS articles and columns are available for free at http://ComputingNow.computer.org.