

Automated Programming: The Next Wave of Developer Power Tools

Laurianne McLaughlin

Hugo Liu thinks automated-programming tools can make great use of natural language. Don Batory is pushing for a model-driven development approach. Dan Cooke uses a declarative programming language to help people grasp code more easily. Today's research on automated programming tools—a class of up-and-coming technologies that aim to make development faster, better, and less expensive—boasts a wide range of ideas.

But don't be mistaken. These software pioneers say it's not a question of whether automated-programming tools will come into their own. It's a question of when.

"Automation is a big, serious problem," says Cooke, who's a Texas Tech computer science professor and the director of the university's Center for Advanced Intelligent Systems. "But I think breakthroughs are around the corner that could drastically modify how different communities produce code."

What fuels the desire for automated tools? Given the huge amount of time and money today's software takes to develop, the software industry certainly needs new ways to improve productivity and reduce cost. But automated tools also promise a big payoff in high-assurance software, says Doug Smith, who has crafted innovative automation tools at the Kestrel Institute (www.kestrel.edu), a nonprofit research group in Palo Alto, California.

"With commercial software, at least half the cost is testing for bugs," Smith says. "Auto-

ated programming comes to the rescue here"—because automated-programming tools can generate not only code but also a proof that the code solves the specified problem.

The biggest challenges in bringing automated-programming tools to the next level—mainstream use—might be people problems, not technical ones, says Batory, a computer science professor at the University of Texas, Austin (see www.cs.utexas.edu/~dsb). He notes that software researchers will have to look outside their own disciplines and tackle problems between fields.

"I believe we are geniuses in making the simplest things look complicated," he says.

The good news? Look at various automated-programming approaches, and many of the core ideas are the same, Batory says.

MIT's new Metafor

According to Batory, automated-programming tools could help a wide variety of developers, not just those in certain niches. "A lot of people could benefit now from it," he says. "You could use it just for creating different versions."

Consider what Hugo Liu, an MIT PhD candidate, has done with his Metafor project (see <http://web.media.mit.edu/~hugo/publications/papers/TUI2005-metafor.pdf>). Metafor lets programmers use natural language, instead of harder-to-learn traditional program languages, to create a program's framework. As Liu puts it, Metafor doesn't create programs—it creates the scaffolding for them. A developer enters

plain-English sentences describing desired results. For example, if you were programming a computer game set in a bar, you could type a sentence such as “The bartender serves drinks.” Metafor parses the sentences, looking for verbs, subjects, and objects. Then it maps the parts of speech into a basic code framework of objects, properties, functions, and if-then rules. Metafor can display the framework in Java, Python, or Lisp.

Liu envisions three ways to use Metafor, starting as an outlining tool for developers. “It would be very useful as a modification tool,” Liu says. “People outline an essay but don’t outline code.” As an outliner, Metafor can help you find mistakes and complete design iterations faster, he says.

Second, Liu would like to see the online-gaming world implement the technology, so that individual players could program their own parts of virtual worlds.

Finally, Metafor could aid software education. For example, students could use Metafor to make their language more precise or to learn new programming languages, Liu says.

The big challenge Liu still must solve is improving Metafor’s ability to negotiate with the user, so that you can easily “roll back” and improve your sentence when you don’t get the desired code, he says.

According to Liu, natural language makes sense for this system (as opposed to, say, a purely graphical interface) because it’s intuitive, rich, and flexible. Metafor users quickly learn to communicate efficiently with the program—for example, using succinct verbs and avoiding run-on sentences, he says.

What concerns does he hear from the software community? Some people initially don’t understand that Metafor complements programmers instead of taking their jobs. “We’re not replacing people; we’re helping people conceptualize and learn,” Liu says.

Indeed, you hear this worry from some developers when you discuss automated-programming tools, says Batory, but it’s not a big controversy.

“There is a scary part of automation that says ‘This is going to take my job

away,’ Batory says, “but it’s probably just going to take away grunt work.” Consider the “gruesome programming” some people had to do before relational databases, he says.

Or as Kestrel’s Smith puts it, developers can think about automated-programming tools the same way they’d view power tools compared to hand tools.

Prototyping at warp speed

Dan Cooke has developed an intriguing power tool for NASA software engineers. His declarative programming language, called SequenceL (www.cs.ttu.edu/~sequence), is speeding up NASA’s software prototyping.

Although the SequenceL work began some 15 years ago, the big breakthrough for Cooke’s team happened within the last four years. Originally, the language had many special cases for generating control structures, but Cooke was able to reduce that to one special case and eliminate a bunch of syntax. “This makes the code more readable to NASA engineers,” he says.

“We were able to reduce a lot of things you would derive algorithmically to a single semantic,” says Cooke. It’s now a very compact language, he adds.

For NASA, Cooke showed that using SequenceL would reduce the software prototyping time for a shuttle abort system by at least 75 percent. Now one of his PhD students is working with NASA engineers in Houston to build a software prototype.

Cooke’s team has been working with NASA for years, but mostly on language development, originally with an eye toward data mining. This project is the first work in the trenches with NASA.

What else might SequenceL do for NASA? “A big issue with NASA is the ability to modify capabilities on the fly,” Cooke says, especially when you’re discussing missions such as exploring Mars. NASA does this well now, but too slowly for future Mars trips, he says.

Cooke grapples with some of the hurdles common to automated-programming tools right now. “One of the big challenges is for automated-programming tools to produce good runtime performance and produce code that someone

can look at, deal with, and understand,” he says. The last challenge is to ensure the code is provably correct. “These are always the big three in automated programming.”

“The ultimate goal is for us to provide onboard code for NASA,” Cooke says, but he adds that that’s probably 10 years away.

What sticking points must be overcome before a tool such as SequenceL could go mainstream?

“The language community always wants to know if the approach is correct,” Cooke says. “A lot of people in the language community come with their own perspective. The NASA people we’re working with in Houston don’t have a favorite tool; they just want the best one.”

Compared to object-oriented approaches, SequenceL is “almost an opposite approach,” says Cooke. “We don’t consider reuse. With the object community, the goal is to specify an object and not worry about it. We want to actually specify it and derive it, not just reuse it.”

However, he says, “You’re trying to satisfy the same objective. In the OO approach, there are limits. Specifications in Java or whatever else don’t tell you everything about the object. When you develop in an environment you don’t know very well, it becomes a problem. In SequenceL, you can look at the specification and know what’s being done at a high level. It’s not tied to a functionality that you already need a preconceived knowledge about.”

If you think of his development effort as a scientific endeavor, what’s done now is the fundamental work, according to Cooke. The work in the trenches with NASA is crucial to improve SequenceL, he says. “You need a real application to test the fundamental against. That really makes a huge difference in clarifying what we can do and the future path.”

Automation’s sweet spots

NASA and US military agencies have been some of the first beneficiaries of automated-programming tools, through projects funded at universities such as Cooke’s and at research organizations such as the Kestrel Institute.

Take the Planware project (www.kestreltechnology.com/solutions/planware.php)—one of Doug Smith's projects funded by DARPA (the US Defense Advanced Research Projects Agency), for which he created a domain-specific software generator. In this case, the domain is planning and scheduling software.

Smith believes that domain-specific projects are a sweet spot for automated programming. "It lets a broad group of users create customized code."

Planware lets a systems analyst describe the planning problem and get code. "Someone who knows about scheduling can work with it and doesn't need a master's in computer science," he says.

Planware is partly graphical, letting users select a series of activities such as getting cargo to a destination. Its requirement language is very specialized, developed after Kestrel spent some 10 years synthesizing planning algorithms, says Smith. The magic of Planware, he says, is that the language is just right for the domain. The US Air Force and Lockheed Martin are working toward deploying it, he adds.

A similar Planware system might work for managing networks. "We're looking at niche markets where the end user executes very complicated rules," Smith says.

Kestrel is also working on using automated tools to create high-assurance software. For example, under US National Security Agency sponsorship, Kestrel is creating smart-card applets that generate code and proofs. "That's very high-assurance software and proofs generated at the same time," he says.

What are the challenges for high-assurance software done with automated tools? First, getting your specification right and describing it right, says Smith. But you can evolve the description and regenerate the code. Second, you're tying into theorem provers, which require high skill levels, he says. "That's been one turn-off." Kestrel is looking at alternative approaches to achieve the same end.

In another intriguing effort, the NSA-funded Protocol Derivation Assistant Project, Kestrel and Stanford researchers

are working on how to generate and compose authentication protocols, such as key exchange and user authentication.

"This area has been incredibly difficult," Smith notes, with researchers finding flaws in protocols only after years. By applying automated-programming methods to this task, the researchers aim to generate protocols and proofs of correctness. Their work has included constructing a family tree of protocols; in doing so, they found a significant flaw in the GDOI (Group Domain of Interpretation) protocol.

Modeling school

Don Batory likens his preferred development approach to buying a computer online: He'd like developers to be able to check off program features they do and don't want.

This model-driven development approach focuses on higher-level specifications of programs in domain-specific languages and increased automation. In MDD, models represent programs, where each model is a specification, written in a domain-specific language, about certain program details. A program specification would typically include multiple models, and models can be computed from each other. Programs are constructed by transforming high-level models into executable code.

The US Army has used some of Batory's technology for simulators testing command-and-control structures.

"People are always interested in what's the next programming paradigm beyond object orientation," says Batory. "I believe this is the kind of way you want to deliver programs. You declaratively specify what you want. This type of technology has been used for years in other disciplines," he says. "We start with a simple program and layer in more detail."

Music and movie creators take the same approach, laying down a track and then adding to it, or shooting a scene and adding in effects, Batory says. The big payoff? "You can build customized programs quickly." Batory feels his approach will also reduce complexity for people designing a family of programs.

However, thinking of programs as

objects remains controversial, he says. "People thought of this idea 40 years ago. But if you pick up a book on software design, you won't see these ideas. It's not being taught."

Indeed, there's a chicken-and-egg problem with automated-programming tools and universities, Kestrel's Smith says. Universities aren't hiring many faculty members in this field because companies haven't been hiring in this field.

Microsoft's point of view

In the commercial-software world, automated tools are still the exception rather than the rule, but some automation ideas are making their way into use, Smith says.

"Industry is kind of groping bottom-up, folding ideas into current practice," he says. "Report generators and graphical interface generators are out there being used."

Look broadly, and you'll see some progress: Consider the Object Management Group's evolving MDA (model-driven architecture) standard, says Smith. (OMG, a nonprofit consortium, works on specifications for interoperable enterprise applications; see www.omg.org/mda.) Much current university research work should complement this effort, he says.

Then there's Microsoft, edging into automation and domain-specific languages in Visual Studio 2005 and its software factories development approach, Smith notes. (For more info on software factories, see <http://msdn.microsoft.com/vstudio/teamsystem/workshop/sf/default.aspx>.) In some ways, Microsoft's vision is similar to MDA, where you have a model for things such as a graphical interface or communications, and you stitch together the models and apply transformations, he says.

Automated programming is making its way into the commercial software world slowly now, in small pieces. But don't mistake slowness for disinterest. Microsoft and a host of other companies have much to gain from the production speedups that automated programming technologies could deliver. ☞