Computer exterminators: Nancy Lynch and Stephen Garland rid software of bugs. (Photograph by Nathan Whitehorn)

TR Related Articles
- The Rules of Innovation
- 7 Startups Graduate with Honors
- 5 Patents to Watch
- Emerging Technologies That Will Change the World

On the Web
- James Paulson Homepage
- Nicolas Gisin Homepage
- Paul Alivisatos Homepage
- Stephen Chou Homepage

10 Emerging Technologies That Will Change the World

February 2003

Software Assurance

Computers crash. That's a fact of life. And when they do, it's usually because of a software bug. Generally, the consequences are minimal—a muttered curse and a reboot. But when the software is running complex distributed systems such as those that support air traffic control or medical equipment, a bug can be very expensive, and even cost lives. To help avoid such disasters, Nancy Lynch and Stephen Garland are creating tools they hope will yield nearly error-free software.

Working together at MIT's Laboratory for Computer Science, Lynch and Garland have developed a computer language and programming tools for making software development more rigorous, or as Garland puts it, to "make software engineering more like an engineering discipline." Civil engineers, Lynch points out, build and test a model of a bridge before anyone constructs the bridge itself. Programmers, however, often start with a goal and, perhaps after some discussion, simply sit down to write the software code. Lynch and Garland's tools allow programmers to model, test, and reason about software before they write it. It's an approach that's unique among efforts launched recently by the likes of Microsoft, IBM, and Sun Microsystems to improve software quality and even to simplify and improve the programming process itself.

Like many of these other efforts, Lynch and Garland's approach starts with a concept called abstraction. The idea is to begin with a high-level summary of the goals of the program and then write a series of progressively more specific statements that describe both steps the program can take to reach its goals and how it should perform those steps. For example, a high-level abstraction for an aircraft collision avoidance system might specify that corrective action take place whenever two planes are flying too close. A lower-level design might have the aircraft exchange messages to determine which should ascend and which should descend.

Lynch and Garland have taken the idea of abstraction further. A dozen years ago, Lynch developed a mathematical model that made it easier for programmers to tell if a set of abstractions would make a distributed system behave correctly. With this model, she and Garland created a computer language programmers can use to write "pseudocode" that describes what a program should do. With his students, Garland has also built tools to prove that lower levels of abstractions relate correctly to higher levels and to

simulate a program's behavior before it is translated into an actual programming language like Java. By directing programmers' attention to many more possible bug-revealing circumstances than might be checked in typical software tests, the tools help assure that the software will always work properly. Once software has been thus tested, a human can easily translate the pseudocode into a standard programming language.

Not all computer scientists agree that it is possible to prove software error free. Still, says Shari Pfleeger, a computer scientist for Rand in Washington, DC, mathematical methods like Lynch and Garland's have a place in software design. "Certainly using it for the most critical parts of a large system would be important, whether or not you believe you're getting 100 percent of the problems out," Pfleeger says.

While some groups have started working with Lynch and Garland's software, the duo is pursuing a system for automatically generating Java programs from highly specified pseudocode. The aim, says Garland, is to "cut human interaction to near zero" and eliminate transcription errors. Collaborator Alex Shvartsman, a University of Connecticut computer scientist, says, "A tool like this will take us slowly but surely to a place where systems are much more dependable than they are today." And whether we're boarding planes or going to the hospital, we can all appreciate that goal. —*Erika Jonietz*

Others in SOFTWARE ASSURANCE	
RESEARCHER	**PROJECT**
Gerard Holzmann Bell Labs	Software to detect bugs in networked computers
Charles Howell Mitre	Benchmarks for software assurance
Charles Simonyi Intentional Software	Programming tools to improve software
Douglas Smith Kestrel Institute	Mechanized software development

Print Version Email to Friend Write Us Add your thoughts Order Reprints

Continued
Single-page view
1 2 3 4 5 6 7 8 9 10 11