Software Is Too Buggy and Unreliable

08.03.05

ENLARGE

Discuss Total posts: 1

By Alan Cohen

Software bugs aren't mere inconveniences. Far too often they crash mission-critical systems, wreaking havoc on business, revenues, and the sanity of technical support departments. And as more software makes its way into our lives—at the doctor's office, at the ATM, on the cable box hooked up to the television—the glitches are multiplying.

The notion of bug-free software seems a lot like time travel, dollar-a-gallon gasoline, and a humble Donald Trump: something we'd all like to see, but won't. Yet even if we're not able to eradicate software bugs entirely, we may be able to stamp out a growing number of them.

One of the most promising antibug efforts is underway at Carnegie Mellon University. Researchers there are analyzing how programmers find and fix bugs, and they're using this information to create a debugging system. "We spend a lot of time watching people debug and write code," says Brad Myers, a professor in Carnegie Mellon's School of Computer Science and head of the research project, which is known as the Whyline. "When programmers discover behavior that they hadn't expected, they ask "why" questions, usually why something did *not* occur."

< back next >

table of contents

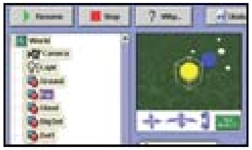
The Ten Biggest Problems in Computing and How We'll Solve Them

Software Is Too Buggy and Unreliable

Something's Amiss

In a prototype of the Whyline, a programmer watches his code execute step by step and hits a button labeled Why when he notices something amiss. The button brings up a menu of 10 to 20 items—each representing an event that could have happened as the code executed, but didn't. By clicking on an item, the programmer is in effect asking, "Why didn't this happen?"

The Whyline then highlights the relevant part of the code, along with a data flow diagram showing the value of data the program was working with. The data diagram is crucial, says Myers, because many bugs are the result of incorrect assumptions about data values.



Whyline

ENLARGE ↗

Many other bugs are the unintended results of attempts to fix bugs. By steering programmers to the root of a problem and avoiding the guesswork, the Whyline can reduce these add-on bugs. On one test, programmers using the Whyline created software with half as many bugs as programmers who did not use the tool. Now the team is attempting to integrate the Whyline into the C and Java programming languages, which may take as long as three to five years.

Major software developers are hard at work on bug extermination efforts as well. Researchers at IBM, for example, are scouring code looking for so-called *antipatterns*. "These are patterns in code that can cause problems," says Eric Naiburg, group manager for desktop products at IBM Rational. "By building tools that look for antipatterns, we can find potential problems and give recommendations on how to fix them." IBM has already incorporated antipattern technology into toolkits it offers to programmers working with its Rational software development platform. As more and more antipatterns are discovered, they'll be incorporated into future toolkits.

As part of its Security Development Lifecycle initiative, Microsoft is building tools to enable its in-house programmers to perform an in-depth analysis on code. The hope is that this will flag problems that might otherwise lead to security and reliability problems in Microsoft software. "The tools go through the code and ask questions like 'What are the inputs you're supplying?', 'What are you assuming about these inputs?', and 'Is some code down the line making an assumption about those inputs that is wrong?'," says Steve

table of contents

Lipner, the director of security engineering strategy at Microsoft. Many of these tools are now available on Microsoft's site; others will be incorporated into its commercial software development products.

< back

next >

Software Is Too Buggy and Unreliable

Crazy Ideas

In a research project spearheaded by Yale University computer science professor Zhong Shao, the focus isn't on eradicating bugs so much as verifying that software touted as bug-free is, indeed, bug-free. Shao's team is developing a system whereby software designers can create a mathematical proof that travels along with their code. In effect, the proof demonstrates that the software does what it's supposed to do.

"Most people find the idea crazy," says Shao, "but when they design code, programmers already do a lot of the same logical reasoning that is required for writing proofs. What we're trying to do is to convert their informal reasoning into a proof." Already, Shao's team has created proofs for small programs. The challenge now is scaling it up to more complex software. For a lot of programs—big operating systems like Windows XP and Linux, for example—proof checking won't be possible anytime soon. The most likely applications, Shao says, will be "small but complex low-level systems software such as embedded OS kernels, runtime systems, and device drivers."

Software bugs are in the crosshairs, but don't expect them to disappear completely. "We're going to get better at eliminating bugs, but we're not going to get perfect," says Myers of Carnegie Mellon. "Programming is inherently a human endeavor, and humans are always going to be imperfect."

table of contents

The Ten Biggest Problems in Computing and How We'll Solve Them