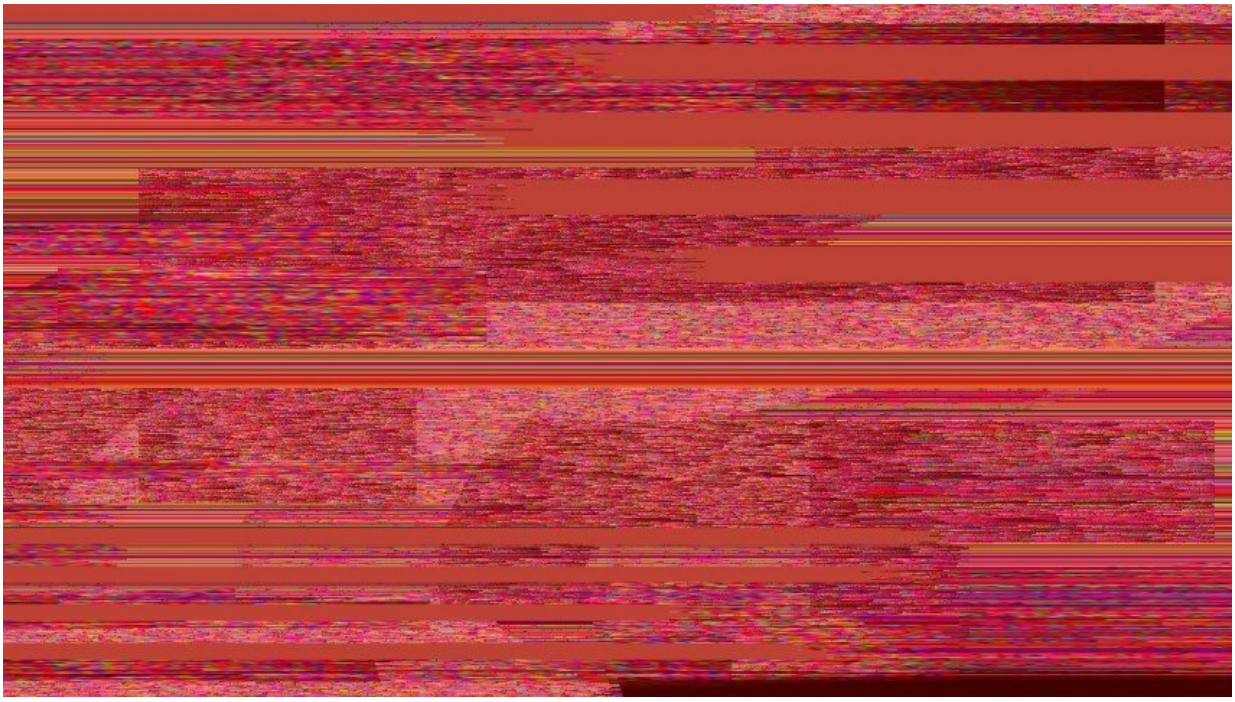
New tools could provide ironclad certainty that computer bugs are a thing of the past

May 2 2019



Credit: Stevens Institute of Technology

It's bad enough losing an hour's work when your computer crashes—but in settings like healthcare and aviation, software glitches can have far more serious consequences. In one notorious case, a computer bug caused cancer patients to receive lethal overdoses from a radiation therapy machine; in more recent headlines, flawed software was blamed for airplane crashes in Ethiopia and Indonesia.

Now researchers at Stevens Institute of Technology, in collaboration with Yale University, are developing tools that could make catastrophic computer mishaps far less likely. Led by Eric Koskinen, an assistant professor of computer science at Stevens, the work not only aims to ensure that programs run correctly in specific situations, but also uses algorithms to determine whether it's logically possible, in any circumstances whatsoever, for software to produce unwanted results.

"What we're aiming for is a 100 percent guarantee that you'll never encounter a bug," said Koskinen.

Koskinen's team, backed by over $2.5 million from the Office of Naval Research, models differences between two versions of a program. That's useful because programmers often work by building on existing software, rather than writing code from scratch, and bugs can be introduced from one version to the next. This approach is especially valuable for the military, since defense agencies frequently buy software from private contractors, then make changes in-house before deploying them in mission-critical situations.

"They need a way to confirm that they've made changes correctly in-house, and haven't introduced new problems," Koskinen said.

To prove mathematically that a computer program could never have any kind of bug, no matter what circumstances, anticipated or unimagined, Koskinen's team uses a strategy called temporal logic. Rather than scrutinize individual lines of code to look for syntactic differences, the team, including assistant professor Jun Xu, an expert in binary analysis at Stevens, looks at how a program behaves over time. The idea is to prove that no matter how long the program runs, there is no logical way for it ever to return an unwanted result.

Modeling a program's structure and behavior, rather than poring over

individual lines of code, is important because the exact same lines of code can have different effects in different contexts, just as lines of code that appear very different can accomplish the same thing. It's like studying a legal document, Koskinen explains: changing a single word might seem trivial, but can change the whole meaning of the document. Temporal logic helps to model a program's potential, gaining powerful insights into the program's real-world capabilities.

The team's approach also allows one to eliminate bugs in off-the-shelf commercial software for which the source code is unavailable. Without the source code, the team is left to compare computer programs using the binary version of the source code. "It's difficult to see if the vulnerability really has been eliminated if you can't see the source code," he said. "The techinques we are building will do that: if you have a version of the software that you trust, our techniques will be able to help you spot changes—vulnerabilities in software updates or malware inserted into executable programs—and decide whether to trust the new version."

Koskinen's team is also developing a toolkit that other researchers and members of the public will be able to use to test software—and they are scaling up their approach to work with larger programs and more complex glitches. "These are big issues that plague modern computer systems," said Koskinen. "These issues will only grow more critical—in fields like healthcare, aviation, autonomous vehicles, and many more—so it's vital we develop practical techniques to make computer -controlled systems bug-free and safe to use."

Provided by Stevens Institute of Technology

certainty-bugs.html