

Software bugtraps | Software that makes software better

Computing: Programmers are using a variety of software tools to help them produce better code and keep bugs at bay

MODERN civilisation depends on software, so it needs to be as reliable as possible. But software is produced by humans, who are fallible. The programs they create are prone to crashes, bugs and security holes. What can be done? A good way to make more reliable software may, oddly enough, be to use even more software. Programmers are increasingly calling upon bug-squashing tools for help at various stages in the software-development process. Some of these tools help programmers to work together more effectively. Other tools scrutinise the resulting software, or its underlying source code, for problems. And still others help project managers put numbers on code quality, programmer productivity, and the cost-effectiveness of fixing particular bugs, so they can decide which bugs ought to be fixed first.

Things are improving, insists Daniel Sabbah, who started programming over 30 years ago and is now general manager of IBM's Rational Software unit, which makes software-development tools. Such tools "really have gotten much better over the years," he says, though their impact is difficult for ordinary users to see, in contrast with the far more obvious improvements in hardware performance, network speeds and storage capacity. Unlike whizzy new hardware, which is quickly adopted by manufacturers, new programming tools and techniques can take several years to percolate through the software industry, he says.

Not everyone agrees with Dr Sabbah's rosy view. Even if the tools are better, the number of bugs in newly written code has remained constant at around five per "function point", or feature, says Capers Jones of Software Productivity Research, a specialist consultancy. Worse, says Mr Jones, only about 85% of these bugs are eliminated before software is put into use. Dr Sabbah responds that such numbers do not show whether software is effective—bug-free code that does not do something useful, or does it two years too late, is not much help to a business, he says. And broader metrics suggest that things are, indeed, improving: the Standish Group, a consultancy that produces a biennial "CHAOS Report" on the state of software development, found that 35% of software projects started in 2006 were completed on time, on budget and did what they were supposed to, up from 16% in 1994; the proportion that failed outright fell from 31% to 19%.

Software as a social science

According to Jim Johnson, the chairman of the Standish Group, most of this improvement is the result of better project management, including the use of new tools and techniques that help programmers work together. Indeed, there are those who argue that computer science is really a social science. Jonathan Pincus, an expert on software reliability who recently left Microsoft Research to become an independent consultant, has observed that "the key issues [in programming] relate to people and the way they communicate and organise themselves." Grady Booch of IBM Rational once tracked 50 developers for 24 hours, and found that only 30% of their time was spent coding—the rest was spent talking to other members of their team.

Programmers generally work together using a software platform called an "integrated development environment", which keeps track of different pieces of code and assembles them when required into a complete program, or "build", for testing. But many firms no longer have all their programmers and testers in the same place, or even in the same country. So it has become necessary to add features to programmer tools to allow coders to communicate with each other, request design changes, report problems and so on.

This field was pioneered by CollabNet, with the launch in 1999 of Subversion, a collaborative platform for programmers which now has more than 2.5m users. Subversion integrates with existing programming tools, including IBM's Eclipse, and offers features such as project-management features, discussion threads and support for quality-assurance engineers.

In 2007 IBM announced a similar effort called Jazz, which (as the name implies) is intended to foster creativity and collaboration among programmers. The idea is to provide a standardised way for existing programming tools to handle change requests, project updates and scheduling details for a particular project, not just code. As well as improving communication between far-flung programmers, centralising this information could also allow managers to track a project's progress more precisely.

High-level improvements in project management, and in the distribution and testing of new versions of a particular piece of software, are a useful, top-down way to improve

the quality of software. But just as important are the low-level tools that scrutinise the actual code to look for bugs, conflicts, security holes and other potential problems. Such tools, which are now proliferating, can be divided into two main types: dynamic-analysis tools, which examine software as it runs to work out where breakdowns happen, and static-analysis tools, which look at code without actually running it to look for fundamental flaws.

Analyse this

To use a mechanical analogy, dynamic analysis is like watching a machine in operation, whereas static analysis is like poring over its blueprints. “Dynamic-analysis tools say, ‘Well, you’ve got a problem on something over here,’” says David Grantges, a technical manager of application security at Verizon Business, a unit of the American telecoms giant. “Static-analysis tools say, ‘You’ve got a problem on line 123.’” The two types are complementary, and Verizon, like most firms, uses both, he says.

Static analysis, being more difficult, is the younger of the two disciplines. In recent years several start-ups, including Klocwork, Fortify and Ounce Labs, have entered the field. Static analysis is best done as close as possible to the programmer, because the earlier a bug can be identified, the cheaper it is to fix. (An industry rule of thumb is that a bug which costs \$1 to fix on the programmer’s desktop costs \$100 to fix once it is incorporated into a build, and thousands of dollars if it is identified only after the software has been deployed in the field.)

In February Klocwork released Insight, a new version of its static-analysis tool that can run on a programmer’s desktop every time code is submitted for a build. The advantage of this approach, says Gwyn Fisher, Klocwork’s technology chief, is that programmers do not need to wait for a build in order to test their code. And when a whole team uses Insight, it can spot potential conflicts between code written by different programmers. Brian Chess, Fortify’s chief scientist, says such tools can also spot mistakes that programmers are known to make routinely, such as allowing “buffer overflows” and “SQL injection”, both of which can open up security holes.

Dynamic analysis involves running a chunk of code with a variety of test inputs to see if it performs as expected, and to make sure it does not do anything undesirable such as crashing, going into an endless loop or demanding more and more memory as it runs. This process can be automated to a certain extent, but guidance from the programmer or tester, in the form of test scripts, is usually required.

Both static and dynamic analysis have been around for a while, but encouraging more programmers to use them is not always easy. It is especially hard to spread these tools beyond large companies, which have the staff to support them. Veracode, a firm based in Burlington, Massachusetts, thinks the answer is to offer code testing as an online service. Chris Wysopal, the firm’s co-founder and technology chief, says that his company’s tool will broaden the market for software testing by giving smaller companies “a blood test” to check their code. At the moment, he says, “we’re where network security was in 1995, when some people didn’t even have a firewall.”

A question of priorities

Another approach is to integrate testing tools more closely with existing programming tools. If testing tools do not fit neatly into a company’s existing way of doing things, developers will not use them, notes Alberto Savoia at Agitar Software, the maker of a tool called Agitator which automatically produces test scripts for use in dynamic analysis. Seth Hallem, the co-founder of Coverity, which makes a static-analysis tool, expects greater integration between programming and testing tools in future.

But analysis tools that spot potential problems, useful though they are, can in turn cause new problems. John Viega of McAfee, a big security-software firm, used to run a start-up that sold a static-analysis tool called CodeAssure (which is now owned by Fortify). He says he did not realise how daunting such tools were to use until he tried selling them. “People would use our tool and find out that they had many reliability problems and many potential security problems, but the cost of researching and fixing them all was astronomical, so they would give up,” he says.

Not all bugs are worth fixing—but how can programmers decide which ones to concentrate on? Jack Danahy, technology chief of Ounce Labs, says the expertise required is the software equivalent of interpreting an MRI image. But his company is doing its best to automate the process, with a static-analysis tool that spots problems and estimates the risk associated with each one.

A similar risk-analysis approach is also being applied to software on a larger scale, through efforts to develop metrics for code quality and programmer productivity. Atlassian, an Australian developer of software tools, last year released Bamboo, which tracks trends in code over time, such as the number of bugs found. Veracode’s analysis service has a code-scoring tool that gives grades to code. And Mr Savoia has developed a system to assess the quality of software written in Java, which he has jokingly named “change, risk, analysis and predictions”, or CRAP. His software plug-in, which determines the “crappiness” of a particular piece of code, has been downloaded by hundreds of programmers. Given that programmers are paid a total of half a trillion dollars a year, Mr Savoia estimates, the industry needs better tools to assess the quality of their work.

To this end, America’s National Institute of Standards and Technology (NIST) is doing its best to create the software equivalent of the “generally accepted accounting principles” used in the financial world. Its Software Assurance Metrics and Tool Evaluation (SAMATE) project is intended to offer companies a way to quantify how much better their code will be if they adopt particular tools and programming languages.

Paul Black of NIST says its first report, on static-analysis tools, should be available in April. The purpose of the research is “to get away from the feeling that ‘all software has bugs’ and say ‘it will cost this much time and this much money to make software of this kind of quality,’” he says. Rather than trying to stamp out bugs altogether, in short, the future of “software that makes software better” may lie in working out where the pesticide can be most cost-effectively applied. ■