A BZ Media Publication

SD Times SOFTWARE DEVELOPMENT

The Industry Newspaper for Software Development Managers

Don't Ignore Static Analysis

Complexity has become the most significant challenge to meeting time-to-market and reliability demands for software. Automated tools, such as static source-code analyzers, are needed to cope with this complexity. A static analyzer determines execution paths through code and how the values of program objects flow through these paths, potentially resulting in bad pointer references, memory leaks, buffer overflows, and many other nasty problems that are reported to the programmer. Yet most developers do not regularly use static analyzers. Let's discuss why and what can be done about it.

At a recent computer conference, a survey of engineers found that only 5 percent of developers regularly employ static analysis. I have asked a wide variety of professional software developers about their use of these tools. A common set of barriers emerges in addition to the obvious one: cost.

Occasionally, the pain of tracking down and averting false positives is cited. This complaint, however, has gone the way of the dodo, as commercial static analyzers have become very accurate. Some developers base their experience on the old—and free—Unix lint analyzer, which is notoriously verbose and inaccurate. You get what you pay for.

The more common complaints center on how static analyzers (don't) fit into the software developer's workflow. Developers are accustomed to the edit-build-debug cycle, and adding another "analyze" phase is uncomfortable. Static analyzers are usually stand-alone tools that must be invoked and managed separately from the developer's IDE. Developers also lament the execution time of static analyzers, often orders of magnitude longer than a regular compile. These inefficiencies deter usage and rob developers of an incredibly valuable tool and the resulting meaningful improvements in fielded product reliability.

INTEGRATION TO THE RESCUE

What can be done about it? The answer is quite simple. Static analyzers need to become integrated static analyzers (ISAs). An ISA introduces a new approach in which static analysis is performed within the same compiler and IDE used to build software. An ISA can generate its warnings or errors interleaved with the other standard diagnostics output by the compiler. Furthermore, common integrations between the project builder and the editor augment the usability of the static analysis tool: When a defect is reported during the build process, the user can hyperlink from the builder's output window back to the source code quickly, rectify the error, and then return to rebuilding the program.

The execution time barrier can also be squashed. The ISA analysis engine takes advantage of efficient dataflow analysis, constant propagation, and path-pruning algorithms developed over many years to perform complex compiler optimizations. In addition, the total time to build and analyze software is reduced because the compiler uses a single parsing pass of the code to perform both compilation and analysis. Finally, the integration with the IDE enables the analyzer to take advantage of the existing distributed build mechanism. The parsing pass for the project's source code is distributed across available workstation assets on the user's network, dramatically reducing the total analysis time.

Consider that a traditional analyzer takes about 10 minutes (one vendor's published results) to analyze the Apache Web server code base; the ISA requires only 30 seconds on the same PC hardware.

Finally we come to cost. Although business models vary, a common one involves charging a fee proportional to the size of the codebase put through analysis. Cost-per-line-of-code lends itself nicely to a return on investment analysis. High-quality commercial analyzers cost approximately 5 cents per line of code. Let's see if we can quantify the benefit.

ISAs reduce development time by enabling engineers to detect and resolve problems more efficiently and earlier in the development cycle. By reducing development time, products reach the market faster and stay in the market longer, translating into higher sales and profits. By increasing product quality, analyzers reduce post-sales cost associated with product failures, recalls and in-field maintenance. Furthermore, increased quality improves market positioning and reputation, enabling organizations to command higher prices, which filter directly to the bottom line.

Unfortunately, while clearly substantial, many of these benefits are difficult to quantify. So let's look at the direct cost of production software development, something that has been thoroughly researched over the years.

It is estimated that it cost US$1,000 to develop each line of code on the space shuttle. Developing software to the stringent DO-178B Level A standard (for critical aircraft systems) has been estimated at hundreds of dollars per line. On the lower end, Red Hat Linux has been estimated to cost $33 per line of code. Other estimates generally place the cost of high-quality commercial software in the range of $30 to $40 per line of code.

Yet other studies have estimated how this development time is spent. Most concur that more than half of software development time is spent debugging: identifying and correcting software defects. If we use a conservative estimate of $30 per line of code in total cost, this means that organizations conservatively spend $15 to debug each line of code.

THE EARLIER, THE BETTER

Another well understood truth is that the cost of identifying and correcting defects grows dramatically as the development cycle progresses. Some studies have shown that the time to fix a bug grows from an average of two to three hours during the coding phase to between 16 and 18 hours when a defect must be tracked down during post-integration quality assurance testing. Author Steve McConnell is often quoted for his estimates that defects cost 10 to 100 times more to fix when they escape detection during the coding phase.

Now let's consider the decrease in defect resolution time enabled by static analyzers. Some studies have shown that static analysis can reduce the number of defects found relative to manual reviews by more than 40 percent. In addition to new code, analyzers have been run on mature production code, including the Linux kernel, OpenSSL, and many others, uncovering numerous defects, including security vulnerabilities. When a defect is identified using static analysis, the most expensive part of defect resolution—tracking down the bug—is reduced to a negligible amount; the tool automatically locates defects and elucidates the offending code sequence leading to the failure. Using a conservative estimate of 10 percent for the decrease in bug fixing time enabled by an ISA, the $15 cost to debug a line of code is reduced by $1.50.

A savings of $1.50 per line of code represents a return of approximately 30 to 1 on our 5-cent investment. And we're not counting the other downstream benefits resulting from improved quality and time-to-market.

Developers worried about the cost of analyzers should instead be concerned about the cost of not using them. Static analyzers are found far too infrequently in the software developer's toolbox. A new breed of integrated analyzers is breaking down barriers. ▮

David Kleidermacher is CTO of Green Hills Software, which sells embedded development tools with an integrated static analyzer.