



EE TIMES ONLINE

Global news for the creators of technology

Static analysis stomps on bugs

By Richard A. Quinnell
March 6, 2008

Static source code analysis tools have evolved from simple syntax checkers to powerful tools for identifying flaws in the complex interactions of large code bases. Until recently, however, they were mainly used by quality assurance teams to evaluate code during integration builds near project completion.

The latest product introductions are now moving these tools back into the hands of developers to help detect software errors much earlier and before they propagate. Klocwork's Insight and GrammaTech's CodeSonar Enterprise both address developer needs by providing utility even when many code segments are still missing.

Software development teams have two types of tools available for automating the detection of errors in their code. One type uses dynamic analysis, which watches code as it is being executed. The other type uses static analysis, which algorithmically examines code for errors. Both types have advantages and limitations.

Dynamic analysis is good at finding runtime errors such as resource leaks and dynamic memory corruption. Developers can also be certain that any errors a dynamic analysis tool reports are real, because they were found during actual code execution.

But to use dynamic analysis effectively, the code must be thoroughly exercised, which requires the use of test cases. Thus, the effectiveness of dynamic analysis at finding errors depends on the quality of the tests being run. Further, because dynamic analysis tools work with running software, they come into play only late in development, when code is already written and first being integrated.

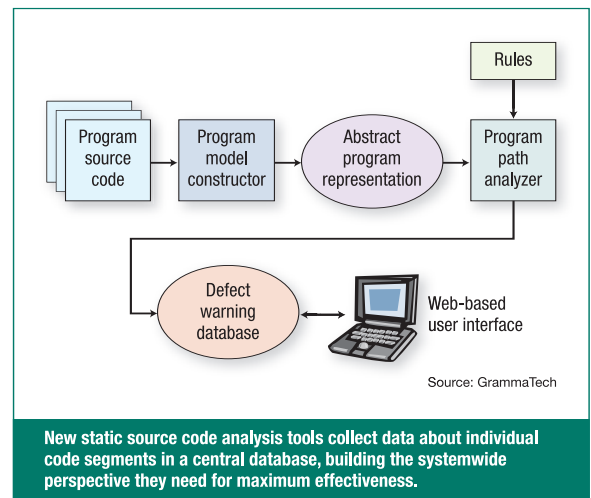
Static analysis uses an algorithmic approach to examine source code for

errors, identifying problem areas for programmers to examine more closely. This algorithmic approach eliminates the need for test cases; the algorithms alone determine how effectively the analysis discovers errors. But the approach also raises the possibility of false identification: code flagged as being in error that will, in fact, execute correctly. If they generate too many false positives—exhibit low “precision,” in the tool vendors’ terms—static tools can overwhelm users with follow-up tasks, obscuring the real errors.

The two tool types are complementary in that each excels at finding errors that cause the other difficulty, but dynamic analysis has seen more use among developers. This is partly because early static analysis tools were little more than syntax checkers that developers used to find relatively simple coding and style errors.

In the last decade, however, static analysis tools have become more productive as research has yielded more-effective algorithms. They have gained an ability to identify a host of subtle errors (see table on back), many of which only manifest as execution problems during task interleaving in a preemptive multitasking environment.

One of the side benefits of the new static analysis capabilities is the enhanced ability to find weaknesses in code that malicious users could exploit to circumvent security safeguards. It is easy for developers to underestimate software security vulnerability, because they expect code to be operated normally. The algorithmic approach of static analysis tools, however, has no



expectations, only procedure, and so will identify potential problems without bias.

Forward in development cycle

Until recently, static source code analysis tools were useful only late in the development process, at the integration build stage, when they had full access to all code segments. A number of recent releases, however—including Klocwork's Insight and GrammaTech's CodeSonar Enterprise—have added features to put static analysis tools into the hands of developers for use as code is still being generated.

This new generation of static source code analysis tools utilizes the enterprisewide software development environment to combine the efforts of development teams working on different parts of the same project (see figure). By allowing the peer-to-peer exchange of information regarding analysis scans of code segments, the tools gather the wider context needed for precision in error detection.

This wider context, coupled with automatic modeling of missing code, quickly

- Locking errors
- Null pointer dereference
- Use after free
- Double free
- Array indexing errors
- Stack overrun
- Heap overrun
- Return pointers to local variables
- Insecure use of user data
- Uninitialized variables
- Invalid use of negative values
- Underallocations of dynamic data
- Memory leaks
- File handle leaks
- Network resource leaks
- Unused values
- Unhandled return codes
- Use of invalid iterators
- File-based race conditions

Source: EE Times

Representative errors discoverable using static source code analysis.

builds a sound basis for developers to check their code against, even while the project remains incomplete. Errors caught at this stage are much easier and cheaper to correct than those caught later in development. Further, catching errors early prevents them from propagating through the system to affect the behavior of code developed later.

This early use of static analysis, however, must be handled with an understanding of the limitations stemming from its partial view of the code. “It is important to remember that in the early stages of software development, the tools are imprecise and can miss interprocedural effects,” cautioned Paul Anderson, vice president of engineering at GrammaTech.

Anderson added, however, that the results improve as the body of analyzed code grows. Frequent early use of static analysis can also help train developers to recognize weaknesses in their individual coding styles and adapt their approach to prevent repeating the same types of errors.

Because tools in this new generation are put to use by the entire project development team, they can build a history of analysis results to help identify new problems as they arise. Klocwork’s Insight, for instance, saves data from each analysis run, allowing developers to track flagged errors throughout the development cycle, said Gwyn Fisher, the company’s chief technical officer.

The tool also allows authorized senior developers to tag errors as false positives or as irrelevant, Fisher added, so that future analysis runs do not report them. This helps the development team concentrate on real errors and makes more visible any newly introduced errors or ones that arise when interacting code sections are analyzed together.

When to go static

How early in the development process and how often to use static code analysis is a question best answered on a project-by-project

basis. “Different teams are prepared to accept different levels of false positives,” said GrammaTech’s Anderson. “A safety-critical project may be willing to tolerate a 10:1 false:true error report ratio, while for other projects a 50:50 ratio is a problem because of wasted time tracking down false errors.”

Klocwork’s Fisher said, “Some of our customers start using the tool on day one, when the tool starts relatively weak. They simply tolerate the extra false positives. Others prefer to take the cost of waiting for the architecture to be fully developed, even though then, the defect density is high.”

Ultimately, project teams must decide for themselves how to balance the cost of false positives early in the process against the cost of finding and correcting errors late in the process. However teams choose to handle static analysis, though, the tools have proven to be a valuable asset in finding code defects and security weaknesses. Now that the tools are back in the developers’ hands so defects are found earlier, static source code analysis can help lower the cost of development, as well.

Coverity Inc.
(800) 873-8193
www.coverity.com

Fortify Software Inc.
(650) 358-5600
www.fortifysoftware.com

GrammaTech Inc.
(408) 246-9100
www.grammatech.com

Klocwork Inc.
(866) 556-2967
www.klocwork.com

The Mathworks Inc.
[www.mathworks.com/
products/polyspaceclientc](http://www.mathworks.com/products/polyspaceclientc)

Source: EE Times

Static source code analysis tool vendors.