Improving Software Quality with Static Code Analysis

By Jay Abraham, MathWorks

Engineers in automotive, aerospace, and other industries must ensure the reliability and quality of increasingly complex high-integrity software systems. To minimize software defects that can lead to catastrophic system failures, injuries, or fatalities, development of many of these systems is governed by standards such as DO-178C, ISO 26262, and IEC 61508.

To improve code quality, many development and testing teams complement traditional software verification activities with static code analysis using Polyspace code verifiers, which use formal methods with abstract interpretation to verify C, C++, or Ada code. This approach enables engineering teams to:

▪ Measure code complexity and verify compliance with coding standards
▪ Prove, via formal methods, that the software will not fail due to certain types of run-time errors
▪ Perform change impact analysis to determine how changing code in one area may affect the reliability or functionality of code in another

Static Code Analysis with Polyspace Code Verifiers

Static code analysis automates time-consuming and error-prone manual code review processes. Unlike dynamic testing, which requires the software to be executed, static code analysis is performed directly on the source code, enabling quality checks to begin before the code is ready for integration and test. During analysis, Polyspace tools calculate complexity metrics and check the code for compliance with development standards, including MISRA C®, MISRA C++, and JSF++.

Polyspace employs sophisticated formal methods that enable deep analysis of the code. Less advanced static analysis tools perform rudimentary checks, such as verifying that variables have been initialized, and are more likely to produce false positives and false negatives. When these types of tools fail to identify an error, the resulting false negative gives the team a false sense of security, and can lead to deployment of the software with the error intact. Conversely, false positives waste time because they force the team to conduct unnecessary reviews or implement unnecessary changes.

Because Polyspace code verifiers are based on formal methods and can prove that code is free of certain run-time errors, they are particularly valuable on development projects that require safety or reliability certification. Certification authorities can review Polyspace results to verify that the software has been mathematically proven to be free of these errors.

Measuring Code Complexity and Checking Compliance with Coding Standards

Code metrics such as cyclomatic complexity provide a valuable perspective when assessing the quality of software in high-integrity systems. By quantifying the number of linearly independent paths, or the decision logic, these metrics provide insight into the complexity of functions and software components. Polyspace tools provide cyclomatic complexity and other code complexity metrics that engineers can use to evaluate each version of their software. A web-based dashboard provides an overview of the metrics, as well as access to more detailed results. By setting thresholds for code complexity, teams can see clearly when certain software quality objectives are not being met.
Coding standards often address problems that stem from the wide latitude that certain languages give to software developers. Languages like C and C++ allow extraordinarily complex coding constructs. For example, in the C language, the following code for dereferencing a pointer is valid:

```c
*****pointer = 100;
```

Although this line of code can be compiled, a scenario calling for an operation that dereferences a pointer five times is unlikely and risk-prone.

Coding standards such as those established by MISRA were developed to improve software quality by eliminating risky and inadvisably complex code. For example, these standards forbid accessing fixed-size arrays with a pointer because this operation runs the risk of reading from or writing to memory outside the array’s bounds.

Polyspace code verifiers check for compliance with code standards such as MISRA. If the full set of MISRA coding rules is not required, engineers can configure Polyspace code verifiers to check for a subset of the rules. The Polyspace code complexity and code rule compliance reports can be used as artifacts to support the certification process for DO-178C, ISO 26262, and other standards.

**Proving the Absence of Errors**

Polyspace tools use formal methods with abstract interpretation to pinpoint where a certain class of run-time errors may occur and to prove the absence of these errors in the source code (see "How Abstract Interpretation Works").

During their analysis, Polyspace code verifiers identify all code elements and constructs that may fail due to a run-time error. Then, using abstract interpretation, they assign each element one of four classifications: red—proven to fail with a run-time error; gray—unreachable; green—proven to be free of certain run-time errors; or orange—unproven, or code that may fail under some circumstances. An additional classification, purple, is used to indicate a coding rule violation (Figure 2).
How Abstract Interpretation Works

Abstract interpretation is a formal method used to mathematically prove that source code is free of certain run-time errors, including arithmetic overflow, divide by zero, and out-of-bounds array access.

To better understand abstract interpretation, consider the following math problem:

\[ (-5477 \times 6.82 \div 13)^2 \]

Without a calculator, it would take some time to find the answer to this problem. However, it is immediately apparent that the answer is neither zero nor any negative number. Some important properties of the final result are known before evaluation of the entire expression.

Abstract interpretation works in a similar way. By applying mathematical theorems that define rules for analyzing complex software, it can precisely determine properties of the software without evaluating the entire problem space. Instead of analyzing each state of a program, which can become computationally prohibitive for complex software, abstract interpretation represents the software states in a mathematical abstraction that is then interpreted to determine whether the software satisfies certain rules or principles, including those that can be used to identify specific run-time errors or prove their absence.

Performing Impact Analysis

Understanding how code changes made in one area can affect the operation of code in another area is vital to ensuring that updates do not undermine the quality of the system. Polyspace code verifiers reveal how new functionality or rework on existing functionality will affect legacy code.

For example, suppose that a development team has identified those code elements in a real-time embedded system that are proven to be reliable, and those elements that are unreachable or potentially problematic under certain conditions. In the next build, the team modifies some existing functions and adds new ones. Polyspace code verifiers accelerate analysis of these changes, using incremental review, for example, to identify any conditional branches in the original code that are now unreachable. Rather than reviewing the entire build again, the engineers can use the web-based dashboard to identify coding rule violations, run-time defects, and dead code resulting from the most recent changes (Figure 3).
Accelerating Development with Static Analysis

Polyspace code verifiers are invaluable for early identification of coding rule violations, run-time defects, and problems caused by code changes. Problems identified early are significantly easier and less expensive to address than those found late in development. With Polyspace code verifiers integrated into the development and testing process, teams can further accelerate development by focusing their code reviews on those areas identified as unproven or unreachable. Additionally, they can streamline the DO-178C, ISO 26262, or IEC 61508 certification process by submitting Polyspace results as artifacts to obtain credit for verification activities required by the certification authority. Certification and qualification kits are available for Polyspace code verifiers, facilitating the use of Polyspace results for projects that must meet certification standards.

Products Used
- DO Qualification Kit
- IEC Certification Kit
- Polyspace Bug Finder
- Polyspace Code Prover

Learn More
- Code Verification and Run-Time Error Detection Through Abstract Interpretation
- Video: Proving the Absence of Run-Time Errors 3:44

See more articles and subscribe at mathworks.com/newsletters.