

### Automate C/C++ Development Best Practices with Parasoft® C++test™

Parasoft C++test is an integrated solution for automating a broad range of best practices proven to improve software development team productivity and software quality. C++test enables coding policy enforcement, static analysis, comprehensive code review, and unit and component testing to provide teams a practical way to ensure that their C and C++ code works as expected. C++test can be used both on the desktop under leading IDEs as well as in batch processes via command line interface for regression testing. C++test integrates with Parasoft's GRS reporting system, which provides interactive Web-based dashboards with drill-down capability, allowing teams to track project status and trends based on C++test results and other key process metrics.

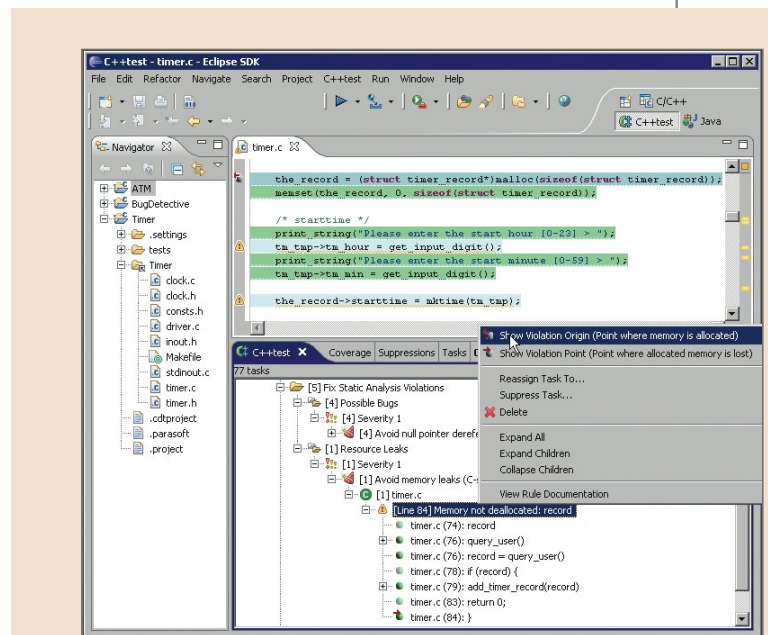
#### Identify Runtime Bugs without Executing Software

C++test BugDetective simulates application execution paths—which may cross multiple functions and files—and determines whether these paths could trigger runtime bugs. Defects detected include using uninitialized memory, null pointer dereferencing, division by zero, and memory and resource leaks. Such defects may also point to missing requirements for the specific use cases corresponding to highlighted execution paths.

BugDetective's ability to expose bugs without executing code is especially valuable for users with legacy code bases lacking robust test suites or embedded code (where runtime analysis and detection of such errors is not effective or possible).

#### Automate Code Analysis for Compliance

A properly implemented coding policy can eliminate entire classes of programming errors by establishing preventive coding conventions. C++test statically analyzes code to check compliance with such a policy. To configure C++test to enforce a coding standards policy specific to their group or organization, users can define their own rule sets with built-in and custom rules. Hundreds of built-in rules—including guidelines from MISRA, Ellementel, Meyers' Effective C++ and Effective STL books, and other popular sources—help identify potential bugs from improper C/C++ language usage, enforce best coding practices, and improve code maintainability and reusability. Custom rules, which are created with a graphical RuleWizard editor, can enforce standard API usage and prevent the recurrence of application-specific defects after a single instance has been found.



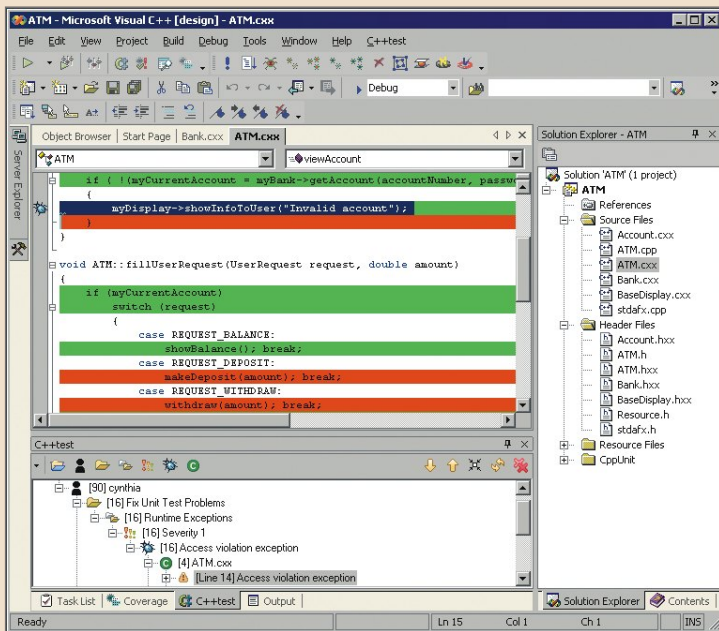
C++test's BugDetective identifies critical bugs without executing the code (Eclipse version shown)

#### Benefits

- **Increase team development productivity** – Apply a comprehensive set of best practices that reduce testing time, testing effort, and the number of defects that reach QA.
- **Achieve more with existing development resources** – Automatically vet known coding issues so more time can be dedicated to tasks that require human intelligence.
- **Build on the code base with confidence** – Efficiently construct, continuously execute, and maintain a comprehensive regression test suite that detects whether updates break existing functionality.
- **Gain instant visibility into C and C++ code quality and readiness** – Access on-demand objective code assessments and track progress towards quality and schedule targets.
- **Reduce support costs** – Automate negative testing on a broad range of potential user paths to uncover problems that might otherwise surface only in "real-world" usage.

## Features

- Static analysis of code for compliance with user-selected coding standards
- Graphical RuleWizard editor for creating custom coding rules
- Static code path simulation for identifying potential runtime errors
- Automated code review with a graphical interface and progress tracking
- Automated generation and execution of unit and component-level tests
- Flexible stub framework
- Full support for regression testing
- Code coverage analysis with code highlighting
- Full team deployment infrastructure for desktop and command line usage



Automatically generated tests capture software behavior (for regression testing) and expose defects that impact software reliability. Test coverage is highlighted in the IDE editor. (Visual Studio .NET plugin shown)

This type of static analysis virtually eliminates the need for line-by-line inspections during peer code reviews. Reviews can then focus on examining algorithms, reviewing design, and searching for subtle errors that automatic tools cannot detect.

## Enable Effective and Comprehensive Team Code Review

The innovative Code Review module, which automates preparation, notification, and tracking of peer code reviews, addresses the known shortcomings of this very powerful development practice. C++test automatically identifies updated code by scanning the source control system, matches the code with designated reviewers, and tracks the progress of each review item until closure. With the Code Review module, teams can establish a bulletproof review process—where all new code gets reviewed and all identified issues are resolved.

## Automate Unit and Component Testing for Instant Verification and Regression Testing

C++test's automated testing helps establish the correctness and reliability of newly developed or legacy code. C++test automatically generates complete tests, including test drivers and test cases for individual functions, in a format similar to CppUnit. By using corner case conditions, these automatically generated test cases check function responses to unexpected inputs, exposing potential reliability problems. To verify functional correctness of code, additional tests can be added by extending the generated test cases or by using a manual test wizard. A multi-metric test coverage analyzer, including statement, branch, path, and MC/DC coverage, helps users gauge test suite efficacy and completeness, and demonstrate compliance with test and validation requirements, such as DO-178B. These automated testing capabilities are especially helpful for supporting automated continuous integration and testing as well as "test as you go" development.

C++test also facilitates the development of a robust regression test suite that detects if incremental code changes break existing functionality. Whether users have a large legacy code base, a small piece of just-completed code, or something in between, C++test can generate tests that capture the existing software behavior via test assertions. As the code base evolves, C++test reruns these tests and compares the current results with those from the originally captured "golden set." It can easily be configured to use different execution settings, test cases, and stubs to support testing in different contexts (e.g., different continuous integration phases, testing incomplete systems, or testing specific parts of complete systems). This type of regression testing is especially critical for supporting short release cycles and ensuring the continued functionality of constantly evolving and difficult-to-test applications.

## Supporting Embedded and Cross-Platform Development

For embedded and cross-platform development, C++test can be used in both host-based and target-based code analysis and test flows. On the host, developers can verify code by using C++test for coding policy enforcement, static analysis, comprehensive code review, and unit and component testing for “test as you go” verification, as well as regression testing. External dependencies for code under test are automatically replaced by stubs that can be used to mock up realistic runtime conditions without actually accessing the referenced hardware or software.

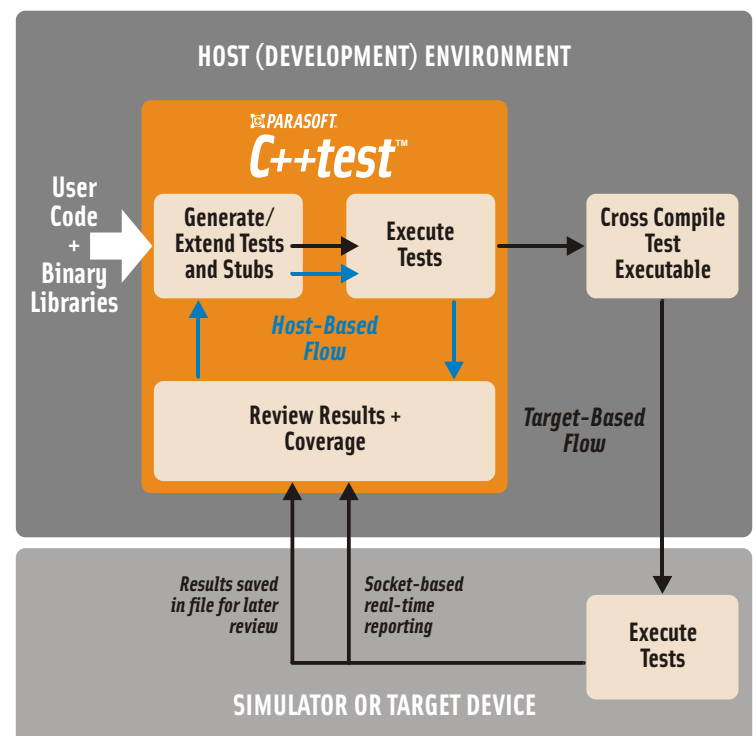
By enabling extensive host-based testing, C++test allows users to start verifying code as soon as it is completed—even if the target hardware is not yet built or available for testing. As a result, the majority of the problems with the application logic can be exposed early—when error detection and remediation is easiest and fastest—and target testing can focus on verifying the interface between the hardware and the software. Moreover, host-based tests are much easier to automatically run and maintain, enabling developers to check the validity of their platform-independent code without tying up additional embedded development tools.

When developers are ready to test on a simulator or the actual target, the test suite that was generated and refined on the host can be reused to validate software functionality on the target. The stubs that were previously used can now be replaced by the real code or system interfaces for integrated system testing—without changing the test code. C++test also provides a built-in capability to automatically capture the test outcomes from execution and turn them into “golden” data sets for subsequent regression testing.

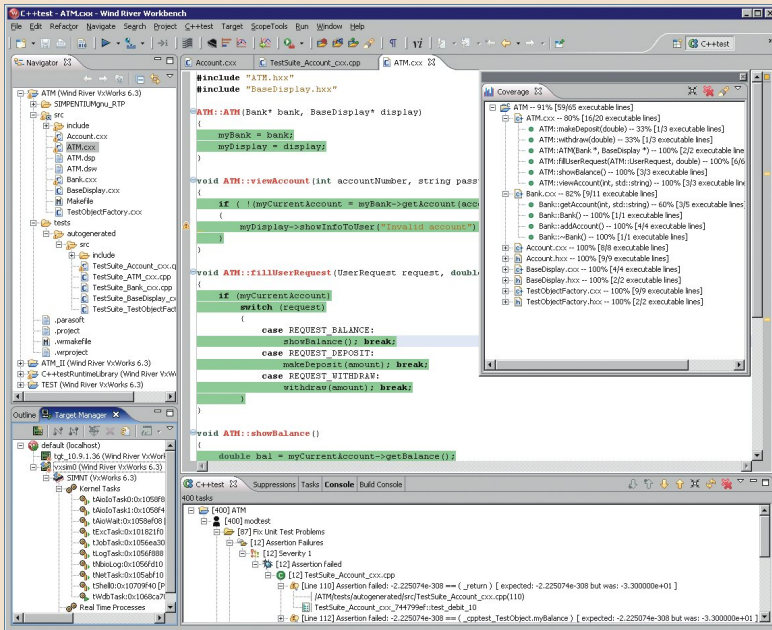
C++test automates the complete test execution flow, including test case generation, cross-compilation, deployment, execution, and loading results (including coverage metrics) back into the GUI. Testing can be driven interactively from the GUI or from the command line for automated test execution, as well as batch regression testing. In the interactive mode, users can run tests individually or in selected groups for easy debugging or validation. For batch execution, tests can be grouped based either on the user code they are liked with, or their name or location on disk.

## Unit Testing Support for Embedded Development

- True unit (function/class) and component testing
- Automated generation of complete structured tests in C or C++ source
- Support for data sources
- Interactive execution of single tests or arbitrary groups of tests from the GUI
- Automated generation of regression tests via capture of actual test results after execution
- Uniform environment for test execution on host and target
- Completely customizable test flow and runtime support
- Coverage analysis for statement, block, branch/condition, and path metrics
- HTML and XML reports with test results
- GUI/desktop and command line modes



C++test's customizable workflow allows users to test code as it's developed, then use the same tests to validate functionality/reliability in target environments



The C++test plugin for Wind River Workbench provides Workbench users with easy access to C++test's full code analysis and unit testing capabilities

At the same time, C++test allows full customization of its test execution sequence. In addition to using the built-in test automation, users can incorporate custom test scripts and shell commands to fit the tool into their specific build and test environment. Its runtime library can also be customized and cross-compiled for a wide variety of target operating systems. This unparalleled flexibility enables users to realize their desired test flow without being constrained by the preset tool capabilities.

## C++test Plugin for Wind River® Workbench

The fully integrated C++test plugin for Wind River Workbench provides Workbench users easy access to C++test's full code analysis and unit testing capabilities under their IDE. The complete target-based test execution flow, including test case generation, cross-compilation, deployment, execution, and loading results back into the GUI, can be automated via the direct interface with Workbench facilities and customized as needed. Tests can be debugged using the Workbench debugger.

## Supported Environments

### Platforms

- Windows NT/2000/XP
- Linux kernel 2.4 or 2.6 or higher with glibc 2.2 or higher and an x86-compatible processor
- Linux kernel 2.6 or higher with glibc 2.3 or higher and an x86\_64-compatible processor (32-bit compatibility package is required)
- Solaris 7, 8, 9, 10 and an UltraSPARC processor

### IDEs with Plug-in Support

- Eclipse 3.1, 3.2 (32-bit)
- Visual Studio .NET 2003 and 2005
- Wind River Workbench 2.5+ (for embedded development)

## Host Compilers

- Windows: Microsoft Visual C++ 6.0, .NET, .NET 2003, or 2005, GCC 2.95.x, 3.2.x, 3.3.x, 3.4.x; Green Hills MULTI for Windows x86 Native v4.0.x
- Linux (x86 processor): GCC 2.95.x, 3.2.x, 3.3.x, 3.4.x, 4.0.x, 4.1.x
- Linux (x86\_64 processor): GCC 3.4.x, 4.0.x, 4.1.x
- Solaris: GCC 2.95.x, 3.2.x, 3.3.x, 3.4.x, 4.0.x, 4.1.x, Sun C++ 5.3 (Sun Forte C++ 6 Update 2), Sun C++ 5.5 (Sun ONE Studio 8), Sun C++ 5.6 (Sun ONE Studio 9), Sun C++ 5.7 (Sun ONE Studio 10) Sun C++ 5.8 (Sun ONE Studio 11); Green Hills MULTI for SPARC Solaris Native v4.0.x

## Target Compilers

- Wind River GCC 3.4.x and DIAB 5.4+
- GCC 2.95.x - 4.1.x cross-compilers
- Green Hills 4.0.x

[www.parasoft.com](http://www.parasoft.com)

### Contact info:

Parasoft Corporation, 101 E. Huntington Dr., 2nd Flr., Monrovia, CA 91016

Ph: (888)305.0041, Fax: (626)256.6884, Email: [info@parasoft.com](mailto:info@parasoft.com)