Parasoft C/C++test is an integrated development testing solution for C and C++. It automates a broad range of software quality practices—including static code analysis, unit testing, code review, coverage analysis, runtime error detection and more. C/C++test enables organizations to reduce risks, cut costs, increase productivity, and achieve compliance with industry guidelines and standards. It can be used in both host-based and target-based code analysis and test flows, which is critical for embedded and cross-platform development.

**Automate Code Analysis for Monitoring Compliance**

A properly implemented policy-driven development strategy can eliminate entire classes of programming errors by preventing defects from entering the code. C/C++test enforces your policy by analyzing code and reporting errors directly in the developer’s IDE when code deviates from the standards prescribed in your programming policy.

Hundreds of built-in rules—including implementations of MISRA, MISRA C++, FDA, Scott Meyers’ Effective C++, Effective STL, and other established sources—help identify bugs, highlight undefined or unspecified C/C++ language usage, enforce best practices, and improve code maintainability and reusability. Development managers can use the built-in rules and configurations or create highly specialized rules and configurations specific to their group or organization. Custom rules can enforce standard API usage and prevent the recurrence of application-specific defects after a single instance has been found.

For highly quality-sensitive industries, such as avionics, medical, automobile, transportation, and industrial automation, C/C++test enables efficient and auditable quality processes with complete visibility into compliance efforts.

**Identify Runtime Errors without Executing Software**

C/C++test’s integration-time static analysis module simulates feasible application execution paths—which may cross multiple functions and files—and determines whether these paths could trigger specific categories of runtime errors. The defects C/C++test detects include:

- Using uninitialized or invalid memory
- Null pointer dereferencing
- Array and buffer overflows
- Division by zero
- Memory and resource leaks
- Various flavors of dead code

The ability to expose defects without executing code is especially valuable for embedded applications, where detailed runtime analysis for such errors is often ineffective or impossible.

C/C++test greatly simplifies defect analysis by providing a complete highlighted path for each potential defect in the developer’s IDE. Automatic cross-links to code help users quickly jump to any point in the highlighted analysis path.

C/C++test’s customizable workflow allow users to test code as it’s developed, then use the same tests to validate functionality/reliability in target environments.
Streamline Code Review

C/C++test automates preparation, notification, and tracking of peer code reviews, which enables an efficient team-oriented process. Status of all code reviews, including all comments by reviewers, is maintained and automatically distributed. C/C++test supports two typical code review flows:

- **Post-commit code review**—Automatic identification of code changes in a source repository via custom source control interfaces; creates code review tasks based on pre-set mapping of changed code to reviewers.
- **Pre-commit code review**—Users can initiate a code review from the desktop by selecting a set of files to distribute or automatically identify all locally changed source code.

Additionally, the need for line-by-line inspections is virtually eliminated because the team's coding policy is monitored automatically with C/C++test's static analysis capability. By the time code is submitted for review, violations have already been identified and cleaned. Reviews can then focus on examining algorithms, reviewing design, and searching for subtle errors that automatic tools cannot detect.

Unit and Integration Test with Coverage Analysis

C/C++test automatically generates complete tests, including test drivers and test cases for individual functions, purely in C or C++ code in a format similar to CppUnit. Auto-generated tests, with or without modifications, are used for initial validation of the functional behavior of the code. By using corner case conditions, the test cases also check function responses to unexpected inputs, exposing potential reliability problems.

Specific GUI widgets simplify test creation and management and a graphical Test Case Wizard enables developers to rapidly create black-box functional tests for selected functions without having to worry about their inner workings or embedded data dependencies. A Data Source Wizard helps parameterize test cases and stubs—enabling increased test scope and coverage with minimal effort. Stub analysis and generation is facilitated by the Stub View, which presents all functions used in the code and allows users to create stubs for any functions not available in the test scope—or to alter existing functions for specific test purposes. Test execution and analysis are centralized in the Test Case Explorer, which consolidates all existing project tests and provides a clear pass/fail status. These capabilities are especially helpful for supporting automated continuous integration and testing as well as “test as you go” development.

A multi-metric test coverage analyzer, including statement, branch, path, and MC/DC coverage, helps users gauge the efficacy and completeness of the tests, as well as demonstrate compliance with test and validation requirements, such as DO-178B/C. Test coverage is presented via code highlighting for all supported coverage metrics—in the GUI or color-coded code listing reports. Summary coverage reports including file, class, and function data can be produced in a variety of formats.

Automated Regression Testing

C/C++test facilitates the development of robust regression test suites that detect if incremental code changes break existing functionality. Whether teams have a large legacy code base, a small piece of just-completed code, or something in between, C/C++test can generate tests that capture the existing software behavior via test assertions produced by automatically recording the runtime test results.

As the code base evolves, C/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 agile development and short release cycles, and ensures the continued functionality of constantly evolving and difficult-to-test applications.
Monitor and Eliminate Runtime Memory Errors

Runtime error detection constantly monitors for certain classes of problems—such as memory leaks, null pointers, uninitialized memory, and buffer overflows—and makes results available immediately after the test session is finished. The reported problems are presented in the developer’s IDE along with details about how to fix the errors (including memory block size, array index, allocation/deallocation stack trace etc.). This not only improves the quality of the application—it also increases the skill level of your development staff.

Coverage metrics are collected during application execution. These can be used to see what part of the application was tested and to fine tune the set of regression unit tests (complementary to functional testing).

Test on the Host, Simulator, and Target

C/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.

C/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. 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 preset tool options.

C/C++test can be used with a wide variety of embedded OS and architectures, by cross-compiling the provided runtime library for a desired target runtime environment. All test artifacts of C/C++test are source code, and therefore completely portable.
Supported Host Environments

Host Platforms
Windows / Linux / Solaris UltraSPARC

IDEs
- ARM Workbench / ARM Development Studio / ARM ADS
- Eclipse IDE for C/C++ Developers
- Green Hills MULTI
- IAR Embedded Workbench
- Keil µVision
- Microsoft eMbedded Visual C++ / Microsoft Visual Studio
- QNX Momentics IDE (QNX Software Development Platform)
- Texas Instruments Code Composer
- Wind River Tornado / Wind River Workbench

Host Compilers
Windows: Microsoft Visual Studio / GNU gcc/g++ / Green Hills MULTI
Linux 32 and 64 bit processor: GNU gcc/g++ / Green Hills MULTI
Solaris: Sun ONE Studio / GNU gcc/g++ / Green Hills MULTI

Target/Cross Compilers

Build Management
GNU make / Sun make / Microsoft nmake / ElectricAccelerator

Continuous Integration
Hudson / Jenkins / ElectricAccelerator

Source Control
AccuRev SCM / Borland StarTeam / CVS / Git / IBM Rational ClearCase / IBM Rational Synergy / Microsoft Team Foundation Server / Microsoft Visual SourceSafe / Perforce SCM / Serena Dimensions / Subversion (SVN)