Parasoft Automotive Compliance eKit:

ISO 26262 Software Compliance with Parasoft

Parasoft C/C++test Data Sheet

ISO 26262 Software Compliance with Parasoft: Achieving Functional Safety in the Automotive Industry

Some modern automobiles have more lines of code than a jet fighter. Even moderately sophisticated cars ship with larger and more complex codebases than the same line from just a few years ago. The inclusion of multi-featured infotainment systems, driver-assist technologies, and electronically controlled safety features as standard components—even in economy models—have fueled the growth of software in the automotive industry. Additionally, the emergence of driverless technology and "connected" cars that function as IoT systems on wheels will mean even larger and more complex codebases.

All of the innovation taking place in the automotive industry, though, raises concerns over the safety, security, and reliability of automotive electronic systems. The concerns are appropriate given that the automotive software supply chain is a long convoluted system of third-party providers spanning several tiers. Consider, for example, that software developed for a specific microcontroller unit (MCU) may be integrated by a third-tier provider into a component they're shipping to a second-tier provider and so on—until a composite component is delivered for final integration by the automaker.

While not all automotive software is critical to the safe operation of the vehicle, code that carries out functional safety operations must be safe, secure, and reliable. Organizations must implement strong software quality process controls around the development of safety-critical software in accordance with ISO 26262, which is a functional safety standard for automotive software. ISO 26262 provides guidance on processes associated with software development for electrical and/or electronic (E/E) systems in automobiles. The standard is aimed at reducing risks associated with software for safety functions to a tolerable level by providing feasible requirements and processes.

In this paper, we provide background information on ISO 26262 and its goals. We also discuss some of the policy-related issues associated with developing embedded software that complies with ISO 26262. Finally, we describe how Parasoft can help automotive software development organizations achieve compliance with ISO 26262.

What ISO 26262 Covers

ISO 26262 is a functional safety standard that covers the entire automotive product development process (including such activities as requirements specification, design, implementation, integration, verification, validation, and configuration). The standard provides guidance on automotive safety lifecycle activities by specifying the following requirements:

- Functional safety management for automotive applications
- The concept phase for automotive applications
- Product development at the system level for automotive applications Software architectural design
- Product development at the hardware level for automotive applications Software unit testing
- Product development at the software level for automotive applications
- Production, operation, service and decommissioning
- Supporting processes: interfaces within distributed developments, safety management requirements, change and configuration management, verification, documentation, use of software tools, qualification of software components, qualification of hardware components, and proven-in-use argument.
- Automotive Safety Integrity Level (ASIL)-oriented and safety-oriented analyses

What ISO 26262 Does Not Cover

- Unique E/E systems in special purpose vehicles such as vehicles designed for drivers with disabilities
- Safety standards for large vehicles, such as those over 3500KB (7700 pounds) gross weight
- Hazards related to electric shock, fire, smoke, heat, radiation, toxicity, flammability, reactivity, corrosion, release of energy and similar hazards, unless directly caused by malfunctioning behavior of E/E safety-related systems
- Nominal performance of E/E systems

Software-specific Sections of ISO 26262

Part 6 of the standard specifically addresses product development at the software level. Requirements for the following development activities are specified:

- Initialization of product development
- Specification of software safety requirements
- Software architectural design
- Unit design and implementation
- Unit testing
- Software integration and testing
- Verification of software safety requirements.

Methods defined by the ISO 26262 standard should be selected depending on the ASIL (automotive safety integrity level). The higher the ASIL, the more rigorous the methods.

Part 8, section 11, describes the software tool qualification process. Tools that automate software development activities and tasks can significantly help organizations meet ISO 26262 requirements. Software tool qualification is intended to provide evidence that the tool(s) is suitable for developing a safety-related item or element. In the simplest practice, tool qualification means running the development tool on a control codebase and making sure that the product is consistent and accurate.

Qualifying Parasoft defect prevention tools and technologies involves running static analysis, flow analysis, unit tests, and any other testing practice used in your development process on a set of control code. Parasoft will consistently, accurately and objectively report errors, which ensures that the tool functions properly.

ISO 26262 Compliance and Policy-Driven Development

A particular feature that makes developing compliant embedded software so challenging is the gap between software development and business expectations. Software engineers make business-critical decisions every day in the form of their coding practices, quality activities, and engineering processes. As software permeates critical functions associated with functional safety, these engineering decisions can lead to significant business risks. E/E systems in automobiles that must conform to ISO 26262 are particularly vulnerable to risks because the standard specifies very detailed lifecycle processes throughout the approximately 400 pages intended to answer a simple, yet ambiguous, question: Is this safe?

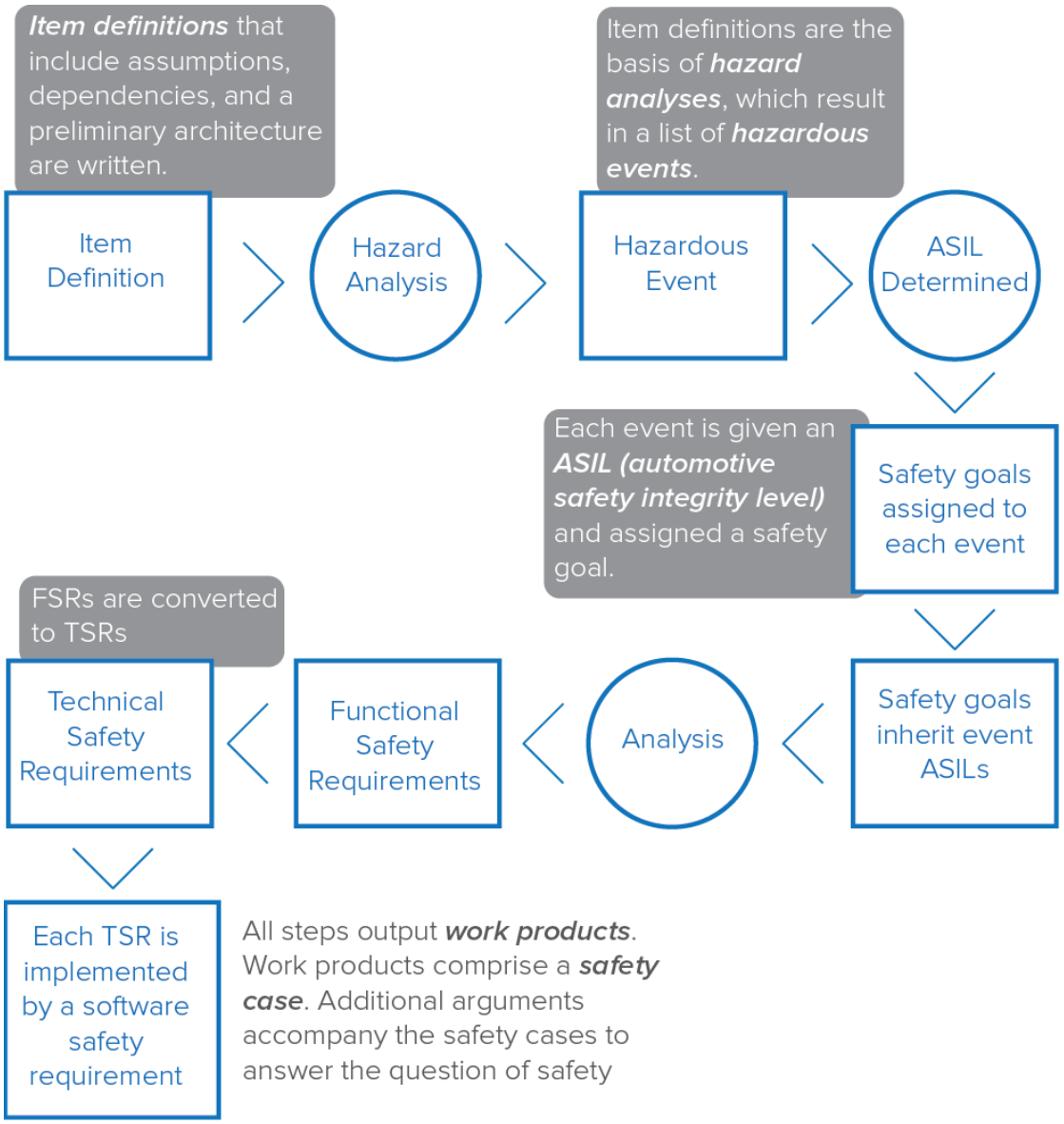


Figure 1: Software development lifecycle defined by ISO 26262

The purpose of ISO 26262 is to outline the policy surrounding the processes in Figure 1, but policies specific to the organization can be integrated at any step.

The key to reining in these risks is to align software development activities with your organization's business goals. This can be achieved through policy-driven development, which ensures that engineers deliver software according to your expectations. Policy-driven development involves:

- Clearly defining expectations and documenting them in understandable polices
- Training the engineers on the business objectives driving those policies

- Enforcing the policies in an automated, unobtrusive way

By adopting a policy-driven strategy, businesses are able to accurate and objectively measure productivity and application quality, which lowers development costs and reduces risk.

With public safety, potential litigation, market position and other consequences on the line, it behooves software development teams and people in the traditional business management positions to come together on policy and implement the strategy into their software development lifecycle. Visit www.parasoft.com for more information about policy-driven development.

Parasoft Support for ISO 26262

Parasoft Development Testing Platform (DTP) facilitates the software quality tasks specified in ISO 26262, including static analysis, data flow static analysis, metrics analysis, peer code review, unit testing and runtime error detection. This provides teams a practical way to prevent, expose, and correct errors in automotive functional safety systems. DTP collects data generated by software engineering processes, such as static code analysis violations, test results, code metrics, coverage analysis, source control check-ins, defect tracking systems, etc., and generates meaningful views of the correlated and prioritized data.

The real power of DTP is the Parasoft Process Intelligence Engine (PIE), which performs an additional post analysis on the development artifacts collected in order to pinpoint risk in the code while highlighting opportunities for improving the your development processes. DTP reports the problematic code and a description of how to fix it to the engineer's IDE based on the organization's programming policy.

The specific sections of the ISO 26262, part 6: Product development: software level that can be addressed or partially addressed with Parasoft are described below. The information presented here is intended to serve as an introduction to ISO 26262 software verification and validation processes with Parasoft. Please refer to the standard and consult functional safety experts for clarification of any requirements defined by the ISO 26262 standard.

Initialization of Product Development at the Software Level

This section of the ISO 26262 – part 6 standard defines general information about the process of software development and validation.

5.4.6 Requirements for achieving correctness of software design and implementation. Methods described here apply to both modeling and programming languages.	
Requirement	Parasoft capability
Enforcement of low complexity	Reports cyclomatic complexity, essential complexity, Halsted complexity, and other code metrics
Use of language subsets	Coding standards enforcement, e.g., detection of unsafe language constructions

Enforcement of strong typing	Implicit conversions detection
Use of defensive implementation techniques	Enforces defensive programming against appropriate coding standards rules, e.g., checking the return value of malloc, checking the error code value returned by called functions, etc.
Use of established design principles	Enforcement of industry coding standards rule sets, e.g. MISRA C/C++, JSF, HIS source code metrics, etc.
Use of unambiguous graphical representation	Enforcement of specific formatting conventions
Use of style guides	Enforcement of specific coding conventions
Use of naming conventions	Enforcement of specific naming conventions

Software Unit Design and Implementation

This section defines the process of specifying and implementing software units, as well as the verification of the design and implementation.

8.4.4 Specifies the design principles for software unit design and implementation.	
Requirement	Parasoft capability
Design principles for software unit implementation, e.g. Initialization of variables, No implicit type conversions, etc.	Static analysis: • MISRA C rules • MISRA C++ rules • MISRA C 2012 • MISRA 2004 • Additional standards

Please refer to the Satisfying ASIL Requirements with Parasoft C/C++test paper for additional information about C/C++test support for specific software unit implementation design principles.

8.4.5 Specifies the verification methods for checking software unit design and implementation.	
Requirement	Parasoft capability
Control flow analysis	Control Flow Analysis
Data flow analysis	Data Flow Analysis
Static code analysis	Coding standards enforcement
Inspection of the source code	Automated peer code review module
Walkthrough of the source code	Automated peer code review module

Software Unit Testing

This section defines the process of planning, defining, and executing software unit testing.

9.4.1 Describes general information about unit test execution.	
Requirement	Parasoft capability
Unit test execution	• Unit test execution module • Reports module for presenting results
Unit test specification	• Configurable unit test generation module creates tests according to the defined specification • Test Case Explorer module presents a list of all defined test cases with pass/fail status
9.4.2 Describes methods used to specify and execute unit tests.	
Requirement	Parasoft capability

Requirement-based tests	• Automated static analysis and unit testing based on MISRA and other standards • Create custom rules based on your requirements
Unit test specification	• Maps test cases with requirements and/or defects in conjunction with the Concerto • Supports user-defined test cases created manually and tests created with the Test Case Wizard
Interface tests	Uses function stubs and data sources to emulate behavior of external components for automatic unit test execution
Fault injection tests	• Enforcing fault conditions using function stubs • Automatic unit test generation using different set of preconditions (e.g., min, max, heuristic values)

Please note that Parasoft allows for packaging test cases into groups to allow easier management of the tests (e.g., execution of the tests from a single group only).

9.4.3 Defines methods that should be used to create test cases.	
Requirement	Parasoft capability
Analysis of requirements	Parasoft Concerto module reports on requirement status, activity, and other parameters
Generation and analysis of equivalence classes	• Uses factory functions to prepare sets of input parameter values for automated unit test generation • Uses data sources to efficiently use a wide range of input values in tests
Analysis of boundary values	• Automatically-generated test cases (e.g.

	heuristic values, boundary values) • Employs data sources to use a wide range of input values in tests
Error guessing	• Uses the function stubs mechanism to inject fault conditions into tested code • Flow Analysis results can be used to write additional tests

9.4.4 Defines the methods for demonstrating the completeness of the test cases.	
Requirement	Parasoft capability
Statement coverage	Code Coverage module
Branch coverage	Code Coverage module
MC/DC (modified condition/decision coverage)	Code Coverage module

Note that ISO 26262 Part 6, Point 9.4.4 states that if instrumented code is used to determine the degree of coverage, it may be necessary to show that the instrumentation has no effect on the test results. This is achieved by running the tests on non-instrumented code.

9.4.5 Defines the requirements for the test environment.	
Requirement	Parasoft capability
Test environment for unit testing shall correspond as far as possible to the target environment	Unit test execution on both target device and simulator to perform tests in different environments (e.g. software-in-the-loop, processor-in-the-loop, hardware-in-the-loop

Software Integration and Testing

10.4.2 Describes general information about executing software integration tests.	
Requirement	Parasoft capability
Integration tests	• Automated test driver and test case generation • Multi-metric test coverage analysis
10.4.5 Defines methods for demonstrating completeness of integration testing.	
Requirement	Parasoft capability
Function Coverage	Code Coverage module
10.4.7 Defines requirements for the integration test environment.	
Requirement	Parasoft capability
Test environment for software integration testing shall correspond as far as possible to the target environment	• Flexible stub framework • Service virtualization module is available to thoroughly mimic complete system

Summary

Developing ISO 26262-compliant software for E/E systems in automobiles is no easy feat. But Parasoft eases the burden by offering a broad range of analysis tools and, more importantly, enabling you to automatically monitor compliance with your development policy—bridging the gap between development activities and business processes. Development teams can also generate configurable test reports that contain a high level of detail, which helps facilitate the work required for the software verification process.

About Parasoft

Parasoft researches and develops software solutions that help organizations deliver defect-free software efficiently. By integrating development testing, API testing, and service virtualization, we reduce the time, effort, and cost of delivering secure, reliable, and compliant software. Parasoft's enterprise and embedded development solutions are the industry's most comprehensive—including static analysis, unit testing, requirements traceability, coverage analysis, functional and load testing, dev/test environment management, and more. The majority of Fortune 500 companies rely on Parasoft in order to produce top-quality software consistently and efficiently as they pursue agile, lean, DevOps, compliance, and safety-critical development initiatives.

Contacting Parasoft

Headquarters

101 E. Huntington Drive, 2nd Floor
Monrovia, CA 91016
Toll Free: (888) 305-0041
Tel: (626) 305-0041
Email: info@parasoft.com

Global Offices

Visit www.parasoft.com/contact for contacting Parasoft in EMEAI, APAC, and LATAM.

About the Authors

Adam Trujillo, Technical Writer, Parasoft
Cynthia Dunlop, Lead Technical Writer, Parasoft

PARASOFT®

PARASOFT C/C++test™

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.

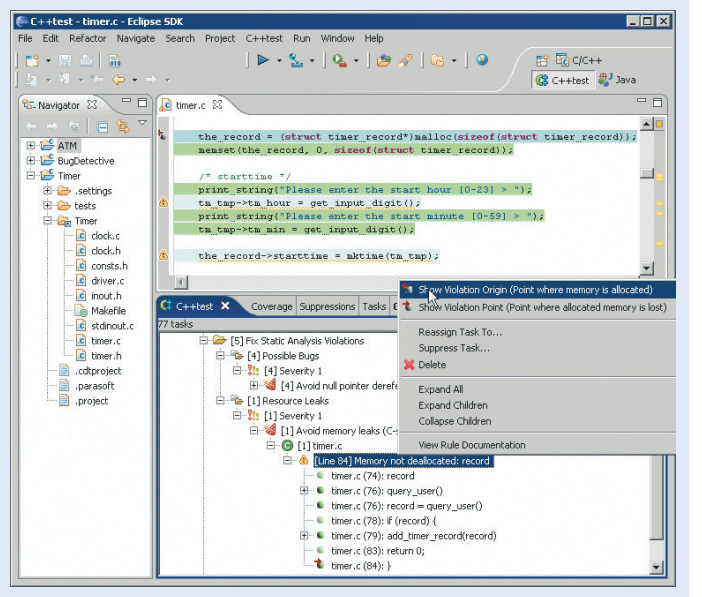
Identify Runtime Errors without Executing Software

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 sup-ports 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 se-lecting 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 func-tional 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.

Benefits

- **Increase 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.
- **Increase code quality**—Efficiently construct, continuously execute, and maintain a comprehensive regression test suite that detects whether updates break existing functionality.
- G**ain unprecedented visibility into the development process**—Access on-demand objective code assessments and track progress towards quality and sched-ule 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.

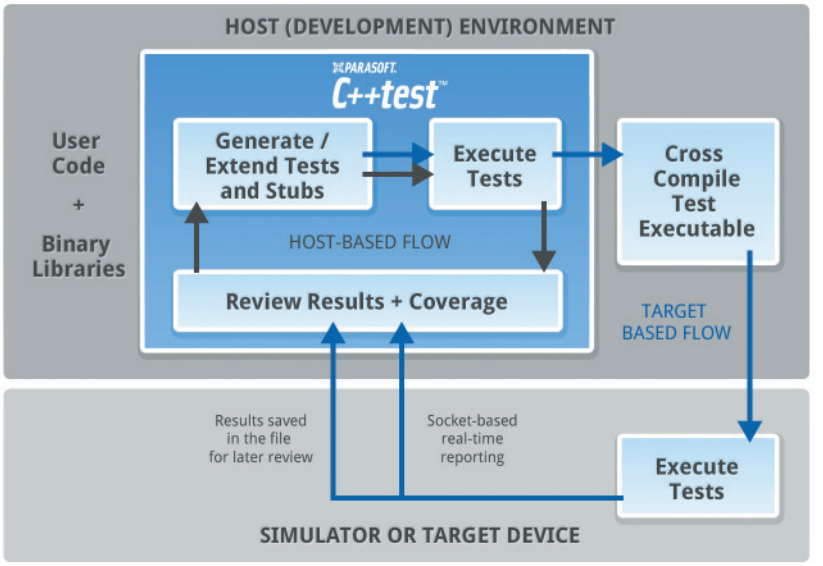
Key Features

- Static code analysis to reduce risks at each stage of development:
  - Integration-time analysis
  - Continuous integration-time analysis
  - Edit-time static analysis
  - Runtime static analysis
- Graphical rule editor for creating custom coding rules
- Automated generation and execution of unit and component-level tests
- Flexible stub framework
- Full support for regression testing
- Code coverage analysis with code highlighting
- Runtime memory error detection during unit test execution and application-level test-ing exposes hard-to-find errors, such as:
  - memory leaks
  - null pointers
  - uninitialized memory
  - buffer overflows
- Increase test result accuracy through execution of the monitored application in a real target environment
- HTML, PDF, and custom format reports:
  - Pass/fail summary of code analysis and test results
  - List of analyzed files
  - Code coverage summary
  - Reports can be automatically sent via email, based on a variety of role-based fil-ters
- Full team deployment infrastructure for desktop and command line usage

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
Altera NIOS GCC / ADS (ARM Development Suite) / ARM for Keil uVision / ARM RVCT / ARM DS-5 GNU Compilation Tools / Cosmic Software 68HC08 / eCosCentric GCC / Freescale CodeWarrior C/C++ for HC12 / Fujitsu FR Family SOFTUNE / GCC (GNU Compiler Collection) / Green Hills MULTI / IAR C/C++ for ARM / IAR C/C++ for MSP430 / Keil C51 / Microsoft Visual C++ for Windows Mobile / Microsoft Embedded Visual C++ / QCC (QNX GCC) / Renesas SH SERIES C/C++ / STMicroelectronics ST20 / STMicroelectronics ST40 / TASKING 80C196 C / TASKING TriCore VX-toolset C/C++ / TI TMS320C2000 C/C++ / TI TMS320C54x C/C++ / TI TMS320C55x C/C++ / TI TMS320C6x C/C++ / TI TMS470 / TI MSP430 C/C++ / Wind River GCC / Wind River DIAB

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)

PARASOFT

USA PARASOFT HEADQUARTERS / 101 E. Huntington Drive, Monrovia, CA 91016
Phone: (888) 305-0041 / Email: info@parasoft.com