PARASOFT®

# Satisfying EN 50128 Requirements with Parasoft:

## Achieving Functional Safety of Railway Software

With the growing reliance on software components in embedded systems, quality has become an ever-increasing concern. Long-standing quality strategies, such as testing with a debugger, are no longer efficient or sufficient. To further complicate matters, many developers cannot readily run a test program in the actual deployment environment because they lack access to the final system hardware. To address these challenges, code quality needs to be addressed throughout the development lifecycle using a synergy of proven techniques for early defect prevention, assisted by automation for implementation and monitoring.

Software for mechatronic implementations is becoming increasingly complex, raising the risks of systematic and random hardware failures. The EN 50128 standard includes guidance for reducing these risks to a tolerable level by providing feasible requirements and processes for the software aspect of railway applications. C/C++test, Parasoft's development testing solution for C and C++ software, not only facilitates the application of EN 50128, it also enables teams to produce better code for embedded systems, test it more efficiently, and consistently monitor progress towards their quality goals. With Parasoft, quality activities, such as static analysis, unit and component testing, coverage analysis, and more, are automated throughout the development cycle, including on the software engineer's desktop to prevent the injection of coding patterns associated with software defects.

Analysis results, test data, coverage, and other quality metrics generated by during manual and fully automated execution within regression and continuous integration environments can also be sent to Parasoft Development Testing Platform (DTP). DTP collects code analysis and quality data, correlates it with other data generated throughout the development lifecycle, and returns intelligent, actionable SDLC analytics that enable you to implement a true defect prevention strategy. This paper describes how Parasoft can help software development teams meet requirements for particular SIL levels as defined by the EN 50128 standard.

## About EN 50128

EN 50128 is part of a group of related European Standards that define requirements for railway applications. Other standards in the group are:

- EN 50126 “The Specification and Demonstration of Reliability, Availability, Maintainability and Safety (RAMS)”, and
- EN 50129 “Communication, Signaling and Processing Systems – Safety-Related Electronic Systems for Signaling”

EN 50128 is titled “Communication, Signaling and Processing Systems - Software for Railway Control and Protection Systems” and specifically addresses the software aspect of railway applications. The standard defines five software safety integrity levels (SIL). The lowest SIL is 0 and the highest is 4. The software integrity level depends on the possible risk resulting from a software failure. Software that carries a safety higher risk in the event of a failure is categorized with a higher SIL. For each SIL, the required techniques and measures are defined in the EN 50128 normative tables of Annex A. This document gives the information on how to satisfy or partially satisfy these requirements.

## About Parasoft C/C++test

Parasoft C/C++test is an integrated development testing solution for automating a broad range of best practices proven to improve software development team productivity and software quality, including:

- Static analysis– monitor coding standards compliance and prevent defects early in the SDLC
- Unit testing–create and execute tests on the host or target
- Coverage and analysis—identify gaps in test and code coverage
- Runtime error detection–detect memory access errors, leaks, corruptions, and more

This provides teams a practical way to prevent, expose, and correct errors in order to ensure that their C and C++ code works as expected. To promote rapid remediation, each problem detected is prioritized based on

configurable severity assignments, automatically assigned to the developer who wrote the related code, and distributed to his or her IDE with direct links to the problematic code and a description of how to fix it. For embedded and cross-platform development, C/C++test can be used in both host-based and target-based code analysis and test flows.

## Automate Code Analysis for Monitoring Compliance

Implementing a standards-based coding policy can eliminate entire classes of programming errors. Parasoft monitors compliance with standards by checking code against patterns known to result in software defects. Hundreds of built-in code analysis rules help identify improper C and C++ usage that can leave software vulnerable to security threats, while enforcing best coding practices and improve code maintainability and reusability. Sources include:

- Implementations of MISRA, MISRA 2004, MISRA C++, and MISRA C:2012 standards
- HIS source code metrics
- Guidelines from Meyers' Effective C++ and Effective STL books
- and more

Organizations can augment the coding standards with custom rules, which can be created with an intuitive graphical editor. Custom rules can enforce standard API usage and prevent the recurrence of application-specific defects after a single instance has been found.

## Identify Runtime Bugs without Executing Software

Parasoft Flow Analysis simulates feasible application execution paths that may cross multiple functions and files and determines whether these paths could trigger specific categories of runtime bugs. Defects detected 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 bugs without executing code is especially valuable for embedded code where detailed runtime analysis for such errors is often not effective or even possible.

## Code Metrics Analysis

Measuring features of the code is a critical part of understanding the risk associated with your software. C/C++test calculates critical code metrics, such as Cyclomatic Complexity, Inheritance Depth, Nested Block Depth, and more to help organizations identify brittle or overly-complex code that could impede agility or reuse. You can configure thresholds so that team members are alerted when metric values are outside of acceptable ranges, which frees the team to focus on analyzing and improving the problematic code–tasks that truly require human intelligence.

## Streamline Code Review

Code review is known to be the most effective approach to uncover code defects. Unfortunately, many organizations underutilize code review because of the extensive effort it is thought to require. C/C++test automates preparation, notification, and tracking of peer code reviews, enabling a very efficient team-oriented process. Status of all code reviews, including all comments by reviewers, is maintained and automatically distributed by the C/C++test infrastructure. C/C++test supports two typical code review flows:

- **Post-commit code review**. This mode is based on automatic identification of code changes in a source repository via custom source control interfaces, and creating 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 for the review, or automatically identify all locally changed source code.

The effectiveness of team code reviews is further enhanced through C/C++test's static analysis capability. With the team's coding policy monitored automatically, reviews can focus on examining algorithms, reviewing design, and searching for subtle errors that automatic tools cannot detect.

## Monitor the Application for Memory Problems

Application memory monitoring is the best known approach to eliminating serious memory-related bugs without generating noise. The running application is constantly monitored for certain classes of problems—like memory leaks, null pointers, uninitialized memory, and buffer overflows—and results are visible immediately after the testing session is finished.

Parasoft instruments the application and executes standard functional testing to expose existing problems and collect coverage metrics. The combination of test results, coverage, and other quality activities provide visibility into how well the application was tested. This runtime error detection allows you to:

- Identify complex memory-related problems through simple functional testing—for example memory leaks, null pointers, uninitialized memory, and buffers overflows
- Collect code coverage from application runs
- Increase the testing results accuracy through execu-tion of the monitored application in a real target environment

## Unit and Integration Test with Coverage Analysis

C/C++test can help you quickly create and automatically execute complete tests, including test drivers and test cases for individual functions. Use the tests for initial validation of the functional behavior of the code and define corner case conditions to check function responses to unexpected inputs, exposing potential reliability problems.

Test creation and management is simplified via a set of specific GUI widgets. 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.

Test cases can be used to produce a regression test base by capturing the existing software behavior via test assertions produced by automatically recording 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).

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. 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.

### Test on the Host, Simulator, and Target

C/C++test automates the complete test execution flow—including test case creation, cross-compilation, deployment, and execution—and loads 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 linked with, or their name or location on disk.

### Configurable Detailed Reporting

HTML, PDF, and custom format reports can be configured via GUI controls or an options file. The standard reports include a pass/fail summary of code analysis and test results, a list of analyzed files, and a code coverage summary. The reports can be customized to include a listing of active static analysis checks, expanded test output with pass/fail status of individual tests, parameters of trend graphs for key metrics, and full code listings with color-coding of all code coverage results. Generated reports can be automatically sent via email, based on a variety of role-based filters. In addition to providing data directly to the developers responsible for the code flagged for defects, C/C++test sends summary reports to managers and team leads.

### Development Testing Platform

Code analysis and test results, coverage analysis, and other C/C++test data can be sent to Parasoft Development Testing Platform (DTP) where it's correlated with data generated by third-party analyzers, source control, defect tracking, and other infrastructure components and processed by DTP. The result is actionable, intelligent analytics that not only provide visibility into the risk associated with the application under test, but also the traceability required to demonstrate EN 501287 compliance.

Defect review and correction are facilitated through automated task assignment and distribution. Each defect detected is prioritized, assigned to the developer who wrote the related code, and distributed to his or her IDE with full data and cross-links to code. To help managers assess and document trends, centralized reporting ensures real-time visibility into quality status and processes. This data also helps determine if additional actions are needed to satisfy internal goals or demonstrate regulatory compliance.

## Satisfying EN50128 Requirements with Parasoft

The EN 50128 standard defines requirements for supporting tools used for development and verification of railway software. Parasoft C/C++test can be categorized as tool class T2, which is defined as a tool that "supports the test or verification of the design or executable code, where errors in the tool can fail to reveal defects but cannot directly create errors in the executable software." As required for tools in class T2, C/C++test provides a user's guide that clearly defines the behavior of the tool and gives instructions on its use.

The tables below present techniques and measures recommended and/or mandated by the EN 50128 standard, normative Annex A, that can be satisfied or partially satisfied using Parasoft C/C++test. For each technique / measure, a brief information is given on how to comply or partially comply with it using the Parasoft C/C++test capabilities. The following markers are used in the tables presented below to indicate that for given SIL:

| | |
|---|---|
| M | Technique is mandatory |
| R | Technique is recommended |
| HR | Technique is highly recommended |
| - | No recommendation |

## A.1 Clauses Tables

### Table A.4 – Software Design and Implementation

| Technique/measure | Reference | SIL 0 | SIL 1 | SIL 2 | SIL 3 | SIL4 |
|---|---|---|---|---|---|---|
| **Modular Approach** | | HR | M | M | M | M |
| Use the Coding Standards and Code Metrics modules to verify that the code has proper modularity by checking coupling between objects metric, usage of global variables, number of function parameters, code documentation and more. | | | | | | |
| **Components** | Table A.20 | HR | HR | HR | HR | HR |
| Use the Coding Standards module to verify the robustness and maintainability of the software components. See Table A.20 for details. | | | | | | |
| **Design and Coding Standards** | Table A.12 | HR | HR | HR | M | M |
| Use one of the pre-defined test configurations for Coding Standards (MISRA C, MISRA C++, JSF++, etc.) or prepare custom test configuration using an appropriate set of built-in and user-defined coding standards rules – including C/C++ best practices, avoiding dangerous code, conforming to naming and formatting conventions for readability, and many more. See Table A.12 for details. | | | | | | |
| **Structured Programming** | | R | HR | HR | HR | HR |
| Use the Coding Standards and Code Metrics modules to verify the structural complexity of the software components (e.g., by keeping Cyclomatic and essential complexity metric values for functions at an appropriate level). | | | | | | |
| **Language Subset** | | - | - | - | HR | HR |
| Use the Coding Standards module with a customizable configuration of coding standards rules to verify that the code applies the desired C/C++ language subset–use one of the pre-defined configurations (e.g. MISRA C, MISRA C++, JSF++, Ellemtel Coding Standards, etc.) or prepare a custom C/C++ subset. | | | | | | |
| **Object-oriented Programming** | Table A.22 | R | R | R | R | R |
| Use the Coding Standards and Code Metrics modules to verify that the code conforms to the rules for object-oriented programming. See Table A.22 for details. | | | | | | |

## Table A.5 - Verification and Testing

| Technique/measure | Reference | SIL 0 | SIL 1 | SIL 2 | SIL 3 | SIL4 |
|---|---|---|---|---|---|---|
| **Static Analysis** | Table A.19 | - | HR | HR | HR | HR |
| Perform Static Analysis using a customizable set of Coding Standards, Control Flow and Data Flow Analysis Rules. See Table A.19 for details. | | | | | | |
| **Dynamic Analysis and Testing** | Table A.13 | - | HR | HR | HR | HR |
| Perform Unit Test execution (on different levels: from file-scope to project-scope) and Application Monitoring – both with Code Coverage Analysis. See Table A.13 for details. | | | | | | |
| **Metrics** | | - | R | R | R | R |
| Use Code Metrics and Coding Standards modules to calculate a wide range of code metrics, including complexity metrics, coupling between objects, depth of inheritance, etc. | | | | | | |
| **Traceability** | | R | HR | HR | M | M |
| Connect Unit Tests with requirements, development tasks or defects by inserting the appropriate information as metadata in code comments. | | | | | | |
| **Test Coverage for Code** | Table A.21 | R | HR | HR | HR | HR |
| Use the Coverage Module to report Code Coverage for the executed Unit Tests and for the functional tests executed by monitoring the running application. See Table A.21 for details. | | | | | | |
| **Functional/Black Box Testing** | Table A.14 | HR | HR | HR | M | M |
| Execute the Unit Tests prepared to verify the functionality of the developed code. See Table A.14 for details. | | | | | | |
| **Interface Testing** | | HR | HR | HR | HR | HR |
| Execute the Unit Tests that were automatically generated using a wide range of input values for the tested functions (boundary values, characteristic parameter values, user-defined values for given types, etc.) Use external data sources to define a single Unit Test with a number of input combinations. | | | | | | |

## Table A.6 – Integration

| Technique/measure | Reference | SIL 0 | SIL 1 | SIL 2 | SIL 3 | SIL4 |
|---|---|---|---|---|---|---|
| **Functional/Black Box Testing** | Table A.14 | HR | HR | HR | HR | HR |
| Execute the Unit Tests prepared to verify the functionality of the developed code. See Table A.14 for details. | | | | | | |

### Table A.7 – Overall Software Testing

| Technique/measure | Reference | SIL 0 | SIL 1 | SIL 2 | SIL 3 | SIL4 |
|---|---|---|---|---|---|---|
| **Functional/Black Box Testing** | Table A.14 | HR | HR | HR | HR | HR |
| Execute the Unit Tests prepared to verify the functionality of the developed code. See Table A.14 for details. | | | | | | |

### Table A.8 – Software Analysis Techniques

| Technique/measure | Reference | SIL 0 | SIL 1 | SIL 2 | SIL 3 | SIL4 |
|---|---|---|---|---|---|---|
| **Static Software Analysis** | Table A.19 | R | HR | HR | HR | HR |
| Perform Static Analysis using a customizable set of Coding Standards, Control Flow and Data Flow Analysis Rules. See Table A.19 for details. | | | | | | |
| **Dynamic Analysis and Testing** | Tables A.20, A.14 | - | R | R | HR | HR |
| Perform Unit Test execution (on different levels: from file-scope to project-scope) and Application Monitoring – on the host, simulator and/or target platform. See Table A.13 for details. | | | | | | |

## A.2 Detailed Tables

### Table A.12 – Coding Standards

| Technique/measure | Reference | SIL 0 | SIL 1 | SIL 2 | SIL 3 | SIL4 |
|---|---|---|---|---|---|---|
| **Coding Standard** | | HR | HR | HR | M | M |
| Use the Coding Standards module to verify the code against selected coding standard (e.g. MISRA C, MISRA C++, JSF++ or a custom rule set containing built-in and/or user-defined coding rules). | | | | | | |
| **Coding Style Guide** | | HR | HR | HR | HR | HR |
| Use the Coding Standards module to verify the code against formatting and naming conventions using a selection of the pre-defined and/or user-defined rules. | | | | | | |
| **No Dynamic Objects / No Dynamic Variables** | | - | R | R | HR | HR |
| Use the Coding Standards module to verify the code against coding rules related to using dynamic objects. | | | | | | |
| **Limited Use of Pointers** | | - | R | R | R | R |
| Use the Coding Standards module to verify the code against coding rules related to using pointers. | | | | | | |

| | | | | | | |
|---|---|---|---|---|---|---|
| **Limited Use of Recursion** | | - | R | R | HR | HR |
| Use the Coding Standards module to verify the code against coding rules related to using recursion. | | | | | | |
| **No Unconditional Jumps** | | - | HR | HR | HR | HR |
| Use the Coding Standards module to verify the code against coding rules related to using unconditional jumps. | | | | | | |
| **Limited Size and Complexity of Functions, Subroutines, and Methods** | | HR | HR | HR | HR | HR |
| Use the Coding Standards and Code Metrics module to verify the code against coding rules and code metrics related to the size of the functions (number of statements, number of lines, number of logical lines, etc.) and the complexity of functions (cyclomatic complexity, essential complexity, etc.) | | | | | | |
| **Entry/Exit Point Strategy for Functions, Subroutines, and Methods** | | R | HR | HR | HR | HR |
| Use the Coding Standards module to check that each function has only a single exit point. | | | | | | |
| **Limited Number of Subroutine Parameters** | | R | R | R | R | R |
| Use the Coding Standards module to verify the code against pre-defined number of function parameters. | | | | | | |
| **Limited Use of Global Variables** | | HR | HR | HR | M | M |
| Use the Coding Standards module to verify the code against coding rules related to using global variables. | | | | | | |

## Table A.13 – Dynamic Analysis and Testing

| Technique/measure | Reference | SIL 0 | SIL 1 | SIL 2 | SIL 3 | SIL4 |
|---|---|---|---|---|---|---|
| **Test Case Execution from Boundary Value Analysis** | | - | HR | HR | HR | HR |
| Execute the Unit Tests that were automatically generated using built-in boundary values and/or user-defined boundary values (defined by factory functions). | | | | | | |
| **Test Case Execution from Error Guessing** | | R | R | R | R | R |
| Execute the Unit Tests created manually (using the Test Case Wizard and/or by directly writing test case source code) based on the testing experience and knowledge of the system under test. | | | | | | |
| **Test Case Execution from Cause Consequence Diagrams** | | - | - | - | R | R |

| | | | | | | |
|---|---|---|---|---|---|---|
| Execute the prepared Unit Tests with errors manually seeded in the code. | | | | | | |
| **Equivalence Classes and Input Partitioning Testing** | | R | R | R | HR | HR |
| Execute the Unit Tests prepared using equivalence classes of inputs. This can be achieved by generating test cases using pre-defined factory functions serving the appropriate values or by preparing data sources with the appropriate combinations of test input values. | | | | | | |
| **Structure-Based Testing** | Table A.21 | - | R | R | HR | HR |
| Perform Unit Testing to achieve a given level of test coverage for the code – see also Table A.21. Use the Stubs functionality so the execution of the tested code is driven to increase code coverage. | | | | | | |

## Table A.14 – Functional and Black Box Testing

| Technique/measure | Reference | SIL 0 | SIL 1 | SIL 2 | SIL 3 | SIL4 |
|---|---|---|---|---|---|---|
| **Test Case Execution from Cause Consequence Diagrams** | | - | - | - | R | R |
| Execute the Unit Tests, Integration Tests and System Tests created to verify the cause consequence diagrams created for the software. Use the Stubs functionality to verify the behavior of the software functions under test in given combinations of basic events. | | | | | | |
| **Boundary Value Analysis** | | R | HR | HR | HR | HR |
| Execute the Unit and Integration Tests that were automatically generated using built-in boundary values and/or user-defined boundary values (defined by factory functions). | | | | | | |
| **Equivalence Classes and Input Partitioning Testing** | | R | HR | HR | HR | HR |
| Execute the Unit Tests prepared using equivalence classes of inputs. This can be achieved by generating test cases using pre-defined factory functions serving the appropriate values or by preparing data sources with the appropriate combinations of test input values. | | | | | | |
| **Process Simulation** | | R | R | R | R | R |
| Perform memory and coverage monitoring of the tested application executed on the host, simulator or target platform | | | | | | |

## Table A.19 – Static Analysis

| Technique/measure | Reference | SIL 0 | SIL 1 | SIL 2 | SIL 3 | SIL4 |
|---|---|---|---|---|---|---|
| **Control Flow Analysis** | | - | HR | HR | HR | HR |

| | | | | | | |
|---|---|---|---|---|---|---|
| Use the inter-procedural Control Flow Analysis module to find unreachable code and other problems. Use the Coding Standards and Code Metrics modules to detect poorly-structured code (e.g. by defining a limit for the essential complexity metric). | | | | | | |
| **Data Flow Analysis** | | - | HR | HR | HR | HR |
| Use the inter-procedural Control Flow Analysis module to find variables in use before being initialized, buffer overflows, resource leaks etc. Use the Coding Standards and Code Metrics modules to detect poorly-structured code (e.g., by defining a limit for the essential complexity metric). | | | | | | |
| **Walkthroughs/Design Reviews** | | HR | HR | HR | HR | HR |
| Use the Code Review module to automate the process of reviewing the developed source code. | | | | | | |
| **Process Simulation** | | R | R | R | R | R |
| Perform memory and coverage monitoring of the tested application executed on the host, simulator or target platform | | | | | | |

## Table A.20 – Components

| Technique/measure | Reference | SIL 0 | SIL 1 | SIL 2 | SIL 3 | SIL4 |
|---|---|---|---|---|---|---|
| **Information Hiding** | | - | - | - | - | - |
| Use the Coding Standards module to verify the code against selected coding rules related to information hiding. | | | | | | |
| **Information Encapsulation** | | R | HR | HR | HR | HR |
| Use the Coding Standards module to verify the code against selected coding rules related to information encapsulation. | | | | | | |
| **Parameter Number Limit** | | R | R | R | R | R |
| Use the Coding Standards module to verify the code against pre-defined number of function parameters. | | | | | | |

## Table A.21 – Test Coverage for Code

| Technique/measure | Reference | SIL 0 | SIL 1 | SIL 2 | SIL 3 | SIL4 |
|---|---|---|---|---|---|---|
| **Statement** | | R | HR | HR | HR | HR |
| Use the Coverage module to report Statement Coverage for the executed Unit Tests and/or monitored application runs – on the host, simulator and/or target platform. | | | | | | |
| **Branch** | | - | R | R | HR | HR |

| | | | | | | |
|---|---|---|---|---|---|---|
| Use the Coverage module to report Decision/Branch Coverage for the executed Unit Tests and/or monitored application runs – on the host, simulator and/or target platform. | | | | | | |
| **Compound Condition** | | - | R | R | HR | HR |
| Use the Coverage module to report Condition Coverage for the executed Unit Tests and/or monitored application runs – on the host, simulator and/or target platform. | | | | | | |
| **Path** | | - | R | R | HR | HR |
| Use the Coverage module to report Path Coverage for the executed Unit Tests and/or monitored application runs – on the host, simulator and/or target platform. | | | | | | |

## Table A.22 – Object-oriented Software Architecture

| Technique/measure | Reference | SIL 0 | SIL 1 | SIL 2 | SIL 3 | SIL4 |
|---|---|---|---|---|---|---|
| **Object-oriented Detailed Design** | Table A.23 | R | R | R | HR | HR |
| Use the Coding Standards and Code Metrics modules to verify that the developed object-oriented code conforms to the defined set of rules and/or patterns. See Table A.23 for details. | | | | | | |

## Table A.23 – Object-oriented Detailed Design

| Technique/measure | Reference | SIL 0 | SIL 1 | SIL 2 | SIL 3 | SIL4 |
|---|---|---|---|---|---|---|
| **Depth of inheritance limited by coding standards** | | R | R | R | HR | HR |
| Use the Code Metrics module to verify that the limit for depth of inheritance – as defined by the used coding standard – is followed by the developed source code. | | | | | | |
| **Overriding of operations (methods) under strict control** | | R | R | R | HR | HR |
| Use the Coding Standards and Code Metrics modules to verify that the rules for overriding of operations–as defined by the used coding standard–are followed by the developed source code. | | | | | | |
| **Multiple inheritance used only for interface classes** | | R | HR | HR | HR | HR |
| Use Coding Standards and Code Metrics modules to verify that the rules for using multiple inheritance – as defined by the used coding standard–are followed by the developed source code. | | | | | | |

## Summary

Parasoft C/C++test helps railway software development teams to fully or partially satisfy software development and verification process requirements defined by the EN 50128 standard. When results of the broad range of testing and code analysis types automated through C/C++test are processed in DTP, intelligent analytics are reported that significantly reduces the work required to for software verification.


## About Parasoft

Parasoft researches and develops software solutions that help organizations deliver defect-free software efficiently. To combat the risk of software failure while accelerating the SDLC, Parasoft offers a Development Testing Platform and Continuous Testing Platform. Parasoft's enterprise and embedded development solutions are the industry's most comprehensive—including static analysis, unit testing, requirements traceability, coverage analysis, API testing, dev/test environment management, service virtualization 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.


© 2016 Parasoft Corporation
All rights reserved. Parasoft and all Parasoft products and services listed within are trademarks or registered trademarks of Parasoft Corporation. All other products, services, and companies are trademarks, registered trademarks, or servicemarks of their respective holders in the US and/or other countries.