TECHNICAL WHITEPAPER

Guide to Achieving Functional Safety in Industrial Automation

How to Satisfy IEC 61508 SIL Requirements

INTRODUCTION

Safety functions are increasingly being carried out by electrical, electronic, or programmable electronic systems. These systems are usually complex, making it impossible in practice to fully determine every failure mode or to test all possible behavior. Although it's difficult to predict the safety performance, testing is still essential. The challenge is to design the system in such a way as to prevent dangerous failures or to control them when they arise.

Safety is one of the key issues of today's and tomorrow's electrical/electronic/programmable electronic safety-related systems. New functionalities increasingly touch the domain of safety engineering. Each function that is required to keep a risk at an accepted level is called a safety function. To achieve functional safety, these functions need to fulfill safety function requirements (what the function does) and safety integrity requirements (the likelihood of a function behaving in a satisfactory manner). Future development and integration of the functionalities containing safety functions will further strengthen the need to have safe system development processes and to provide evidence that all reasonable safety objectives are satisfied.

With the trend of increasing complexity, software content, and mechatronic implementation, there are rising risks of systematic failures and random hardware failures. IEC 61508 includes guidance to reduce these risks to a tolerable level by providing feasible requirements and processes.

The purpose of this document is to detail how Parasoft can help software development teams meet requirements for particular SIL levels. It first introduces the idea of SIL as defined by the IEC 61508 standard. Next, it describes Parasoft's integrated solution for automating best practices in software development and testing. Finally, it presents how Parasoft can be used to fully or partially satisfy software development process requirements for particular SIL levels.

SOFTWARE INTEGRITY LEVELS

Safety Integrity Level (SIL)—as defined by the IEC 61508 standard—is one of the four levels (SIL1-SIL4) corresponding to the range of a given safety function's target likelihood of dangerous failures. Each safety function in a safety-related system needs to have appropriate safety integrity level assigned. An E/E/PE safety-related system will usually implement more than one safety function. If the safety integrity requirements for these safety functions differ, unless there is sufficient independence of implementation between them, the requirements applicable to the highest relevant safety integrity level shall apply to the entire E/E/PE safety-related system.

According to IEC 61508, the safety integrity level for a given function is evaluated based on either the average probability of failure to perform its design function on demand (for a low demand mode of operation) or on the probability of a dangerous failure per hour (for a high demand or continuous mode of operation).

The IEC 61508 standard specifies the requirements for achieving each safety integrity level. These requirements are more rigorous at higher levels of safety integrity in order to achieve the required lower likelihood of dangerous failures.

ABOUT PARASOFT'S DEVELOPMENT TESTING SOLUTION FOR C AND C++

Parasoft C/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/C++test facilitates:

» **Static code analysis.** Includes data flow, control flow, and metrics analysis.

» **Unit testing.** Create, execute, optimize, and maintain.

» **Code coverage.** Exposes what code has not been executed through testing.

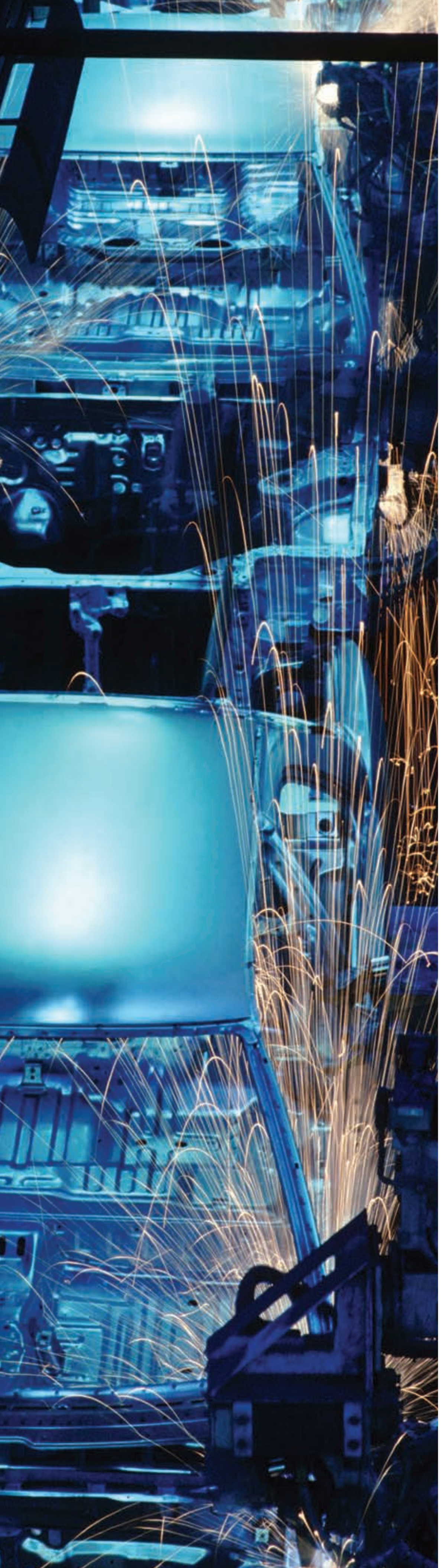
» **Requirements traceability.** Link requirements to tests and code.

» **Runtime error detection.** Find memory access errors, leaks, corruptions, and more.

» **Peer code review.** Examine algorithms, design, and search for subtle errors.

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 a ssigned 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

A properly implemented coding policy can eliminate entire classes of programming errors by establishing preventive coding conventions. C/C++test statically analyzes code to check compliance with such a policy. To configure C/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. Code analysis reports can be generated in a variety of formats, including HTML and PDF.

Hundreds of built-in rules—including all implementations of MISRA standards, HIS source code metrics as well as guidelines from Meyers' Effective C++ and Effective STL books, and other popular sources—help identify potential bugs from improper C and 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.

IDENTIFY RUNTIME BUGS WITHOUT EXECUTING SOFTWARE

Parasoft's the advanced interprocedural flow 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 bugs. Defects detected include using uninitialized or invalid memory, null pointer dereferencing, array and buffer overflows, division by zero, memory and resource leaks, and 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 possible.

C/C++test greatly simplifies defect analysis by providing a complete path trace 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.

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. Parasoft DTP Change Explorer enables convenient analyses of source code deltas between specific milestones or points in development. Overlapping code delta information with static analysis or unit testing results elevates the traditional code review process to completely new level. The effectiveness of code review process is for example, 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 with zero false positives. 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.

Without requiring advanced and time-consuming testing activities, the instrumented application—additional code is added for the monitoring purposes—goes through the standard functional testing and all existing problems are flagged. The application can be executed on the target device, simulated target, or host machine. The collected problems are presented directly in the developer's IDE with the details required to understand and fix the problem (including memory block size, array index, allocation/deallocation stack trace.

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).

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 execution of the monitored application in a real target environment.

UNIT AND INTEGRATION TEST WITH COVERAGE ANALYSIS

Parasoft C/C++test's automation greatly increases the efficiency of testing the correctness and reliability of newly developed or legacy code. C/C++test automatically generates complete tests, including test drivers and test cases for individual functions, purely in C or C++ code. These tests, with or without modifications, are used for initial validation of the functional behavior of the code. By using corner case conditions, these automatically-generated test cases also 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.

Both automatically-generated and hand-written 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, such as different continuous integration phases, testing incomplete systems, or testing specific parts of complete systems.

CONFIGURABLE DETAILED REPORTING

Parasoft C/C++test's 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.

DTP

Code analysis and test results, coverage analysis, and other C/C++test data can be sent to Parasoft 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 achieve SIL requirements.

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.

ACHIEVING SIL REQUIREMENTS WITH PARASOFT C/C++TEST

The following tables describe how Parasoft C/C++test supports the software development life cycle methods required for the safety functions to achieve a given SIL. Please note that the information presented here is meant to briefly introduce C/C++test usage in the SIL-related verification and testing process. Please refer to the standard and consult functional safety experts for clarification of any requirements defined by the IEC 61508 standard. If you have any additional questions regarding how to use C/C++test in the IEC 61508 certification process, please contact your Parasoft representative.

The following markers are used in the tables presented below to indicate:

» R – functionalities matching methods recommended by the IEC 61508 standard

» HR – functionalities matching methods highly recommended by the IEC 61508 standard

» --- – no recommendation

Techniques and measures defined annex A and B of IEC 61508-3 edition 2.0 2010 that are satisfied by C/C++test functionality has been captured in the tables below.

TEST AUTOMATION

C/C++test Functionality	SAFETY INTEGRITY LEVEL (SIL)			
	SIL 1	SIL 2	SIL 3	SIL 4
Design and Coding Standards				
Use of coding standard to reduce likelihood of errors *(Table B.1: 1)*	---	R	HR	HR
No dynamic objects *(Table B.1: 2)*	R	HR	HR	HR
No dynamic variables *(Table B.1: 3a)*	---	R	HR	HR
Online checking of the installation of dynamic variables *(Table B.1: 3b)*	---	R	HR	HR
Limited use of interrupts *(Table B.1: 4)*	R	R	HR	HR
Limited use of pointers *(Table B.1: 5)*	---	R	HR	HR
Limited use of recursion *(Table B.1: 6)*	---	R	HR	HR
No unstructured control flow in programs in higher level languages *(Table B.1: 7)*	R	HR	HR	HR
No automatic type conversion *(Table B.1: 8)*	R	HR	HR	HR
Dynamic Analysis and Testing				
Test case execution from boundary value analysis *(Table B.2: 1)*	R	HR	HR	HR
Test case execution from error guessing *(Table B.2: 2)*	R	R	R	R
Test case execution from error seeding *(Table B.2: 3)*	---	R	R	R
Equivalence classes and input partition testing *(Table B.2: 6)*	R	R	R	HR
Structural test coverage (entry points) 100 % *(Table B.2: 7a)*	HR	HR	HR	HR
Structural test coverage (statements) 100 % *(Table B.2: 7b)*	R	HR	HR	HR
Structural test coverage (branches) 100 % *(Table B.2: 7c)*	R	R	HR	HR
"Structural test coverage (conditions, MC/DC) 100 % *(Table B.2: 7d)*"	R	R	R	HR

Functional and Black Box Testing				
"Equivalence classes and input partition testing, including boundary value analysis *(Table B.3: 4)*"	R	HR	HR	HR
Performance Testing				
Avalanche/stress testing *(Table B.6: 1)*	R	R	HR	HR
Static Analysis				
Boundary value analysis *(Table B.8: 1)*	R	R	HR	HR
Checklists *(Table B.8: 2)*	R	R	R	R
Control flow analysis *(Table B.8: 3)*	R	HR	HR	HR
Data flow analysis *(Table B.8: 4)*	R	HR	HR	HR
Error guessing *(Table B.8: 5)*	R	R	R	R
"Formal inspections, including specific criteria *(Table B.8: 6a)*"	R	R	HR	HR
Walk-through (software) *(Table B.8: 6b)*	R	R	R	R
Symbolic execution *(Table B.8: 7)*	---	---	R	R
Design review *(Table B.8: 8)*	HR	HR	HR	HR
Static analysis of run time error behaviour *(Table B.8: 9)*	R	R	R	HR
Modular Approach				
Software module size limit *(Table B.9: 1)*	HR	HR	HR	HR
Software complexity control *(Table B.9: 2)*	R	R	HR	HR

SOFTWARE DESIGN AND DEVELOPMENT

C/C++test Functionality	SAFETY INTEGRITY LEVEL (SIL)			
	SIL 1	SIL 2	SIL 3	SIL 4
Software Architecture Design				
Fault detection *(Table A.2: 1)*	---	R	HR	HR
Diverse monitor techniques (with separation between the monitor computer and the monitored computer) *(Table A.2: 3c)*	---	R	R	HR
"Diverse redundancy, implementing the same software safety requirements specification *(Table A.2: 3d)*"	---	---	---	R
Use of trusted/verified software elements (if available) *(Table A.2: 8)*	R	HR	HR	HR
Support Tools and Programming Language				
Suitable programming language *(Table A.3: 1)*	HR	HR	HR	HR
Strongly typed programming language *(Table A.3: 2)*	HR	HR	HR	HR
Language subset *(Table A.3: 3)*	---	---	HR	HR
Certified tools and certified translators *(Table A.3: 4a)*	R	HR	HR	HR
Tools and translators: increased confidence from use *(Table A.3: 4b)*	HR	HR	HR	HR
Detail Design				
Structured methods *(Table A.4: 1a)*	HR	HR	HR	HR
Formal design and refinement methods *(Table A.4: 1c)*	---	R	R	HR

Defensive programming *(Table A.4: 3)*	---	R	HR	HR
Modular approach *(Table A.4: 4)*	HR	HR	HR	HR
Design and coding standards *(Table A.4: 5)*	R	HR	HR	HR
Structured programming *(Table A.4: 6)*	HR	HR	HR	HR
Use of trusted/verified software elements *(Table A.4: 7)*	R	HR	HR	HR
Forward traceability between the software safety requirements specification and software design *(Table A.4: 8)*	R	R	HR	HR
Software Module Testing and Integration				
"Dynamic analysis and testing *(Table A.5: 2), (Table A.9: 4)*"	R	HR	HR	HR
Data recording and analysis (*Table A.5: 3)*	HR	HR	HR	HR
Functional and black box testing *(Table A.5: 4) (Table A.7: 4)*	HR	HR	HR	HR
Interface testing *(Table A.5: 7)*	R	R	HR	HR
Test management and automation tools *(Table A.5: 8)*	R	HR	HR	HR
Forward traceability between the software design specification and the module and integration test specifications *(Table A.5: 9)*	R	R	HR	HR
Formal verification *(Table A.5: 10)*	---	---	R	R
Modification				
Reverify changed software module *(Table A.8: 2)*	HR	HR	HR	HR
Reverify affected software module *(Table A.8: 3)*	R	HR	HR	HR
Revalidate complete system *(Table A.8: 3)*	---	R	HR	HR
Regression validation	R	HR	HR	HR
Software Verification				
Formal proof	---	R	R	HR
Static analysis	R	HR	HR	HR
"Dynamic analysis and testing *(Table A.5: 2), (Table A.9: 4)*"	R	HR	HR	HR
Forward traceability between the software design specification and the software verification (including data verification) plan	R	R	HR	HR
Backward traceability between the software verification (including data verification) plan and the software design specification	R	R	HR	HR

PROGRAMMABLE ELECTRONICS INTEGRATION

C/C++test Functionality	**SAFETY INTEGRITY LEVEL (SIL)**			
	SIL 1	**SIL 2**	**SIL 3**	**SIL 4**
Hardware and Software				
Functional and black box testing *(Table A.6: 1)*	HR	HR	HR	HR
Forward traceability between the system and software design requirements for hardware/software integration and the hardware/software integration test specification *(Table A.6: 3)*	R	R	HR	HR

SOFTWARE SAFETY REQUIREMENTS SPECIFICATION				
C/C++test Functionality	**SAFETY INTEGRITY LEVEL (SIL)**			
	SIL 1	**SIL 2**	**SIL 3**	**SIL 4**
Parasoft DTP, Reporting and Analytics Dashboard				
Forward traceability between the system safety requirements and the software safety requirements *(Table A.1:2)*	R	R	HR	HR
Backward traceability between the safety requirements and the perceived safety needs *(Table A.1:3)*	R	R	HR	HR
Computer-aided specification tools to support appropriate techniques/measures above *(Table A.1:4)*	R	R	HR	HR

Note that C/C++test can monitor an application running in the production environment on a target device or on a simulator.

SUMMARY

Parasoft C/C++test helps software development teams meet the functional safety requirements of IEC 61508. A broad range of analysis types—including coding standards compliance analysis, data and control flow analysis, unit testing, application monitoring, workflow components, and peer code review process—together with the configurable test reports containing high level of details, significantly facilitates the work required for the software verification process.

TAKE THE NEXT STEP

Request a demo to see how your software development team can satisfy SIL requirements 1 through 4 and achieve functional safety in industrial automation.

ABOUT PARASOFT

Parasoft helps organizations continuously deliver quality software with its market-proven, integrated suite of automated software testing tools. Supporting the embedded, enterprise, and IoT markets, Parasoft's technologies reduce the time, effort, and cost of delivering secure, reliable, and compliant software by integrating everything from deep code analysis and unit testing to web UI and API testing, plus service virtualization and complete code coverage, into the delivery pipeline. Bringing all this together, Parasoft's award winning reporting and analytics dashboard delivers a centralized view of quality enabling organizations to deliver with confidence and succeed in today's most strategic ecosystems and development initiatives — security, safety-critical, Agile, DevOps, and continuous testing.