

# Vector Software

## WHITEPAPER

## Using VectorCAST to Satisfy Software Verification and Validation for ISO 26262

### Purpose

This document is intended to serve as a reference to show how the VectorCAST products from Vector Software can be used to satisfy the Verification and Validation requirements specified in the ISO 26262 standard, itself derived from IEC 61508. It is not intended to be an exhaustive review of the standard, but merely to provide a high-level view of the different requirements that can be achieved using VectorCAST. For more information, please contact Vector Software.

### Introduction

Competition in the automotive industry is intense. Successful companies must constantly innovate by introducing new features, many of which contain significant amounts of software. The automobile has been transformed from primarily a mechanical device, into an integrated machine with embedded software in all major systems including: engine control, power train, suspension, braking, and entertainment.

Controlling the costs of automotive embedded systems is extremely important for automotive industry suppliers since there is a much higher volume of software than other safety-critical industries like avionics and railway.

Software testing has traditionally been very expensive, but the cost of finding software bugs now versus the direct costs and damaged product branding associated with recalls makes thorough testing a necessity in the automotive industry.

## Automotive Software Verification and Validation Standards

Currently **MISRA** and **ISO 26262** are the two software standards applying to verification and validation of vehicle-based software.

### **MISRA**

Static analysis has been a big part of the automotive application development process since the advent of the Motor Industry Software Reliability Association (MISRA) C standard. ***“Guidelines for the Use of the C Language in Vehicle Based Software”*** is a document which was first published in 1998 to promote safe use of the C language in the automotive industry. It contains rules defining a subset of the C language that is now widely accepted as a model for good programming practice. ***“MISRA C++: 2008 Guidelines for the use of the C++ Language in Critical Systems”*** was published in 2008 to define similar rules for the C++ language.

### **ISO 26262**

ISO 26262 is a Functional Safety standard currently under development, titled ***“Road vehicles -- Functional safety”***. The standard is an adaptation of the Functional Safety standard IEC 61508 for Automotive Electric/Electronic Systems. Part 6 of the ISO 26262 standard addresses the recommendations for dynamic software testing and verification as part of the standard for software development.

Recommended activities include both **unit level** and **system level testing**, such as functional tests (requirement-based tests and partition tests) and **structural coverage tests**.

## How VectorCAST Supports Compliance with the ISO 26262 Standard

The VectorCAST embedded test tools for ISO 26262 satisfies recommendations for software testing and verification specified in Part 6 of the standard for software development by supporting the creation and management of test cases to prove that the low-level software requirements have been tested.

VectorCAST is also used for a variety of robustness testing activities such as range and out-of-bounds testing.

Additionally, VectorCAST tools support capture and reporting of structural code coverage for all Automotive Safety Integrity Levels (ASIL) required by ISO 26262.

*ASIL is the automotive specific risk-based approach for determining product risk classes. Risk classes are defined as Level A through D, with ASIL D representing the highest risk.*

The VectorCAST/C++ test tools for C and C++ unit and integration testing, combined with VectorCAST/Cover for system-level test verification, provide a complete dynamic test suite for host, simulator, and target level testing.

*Please note that ISO 26262 is presently in the last stages of acceptance and specific recommendations may change before it is homologated.*

For this document, the following legend is used:

- R** Recommended activity
- HR** Highly recommended activity

ISO 26262 proposes a waterfall approach to testing, with a clear demarcation between unit testing activities and system testing activities. The document thus addresses these two levels of testing separately.

## VectorCAST during Software Unit Testing

### **Methods for Software Unit Testing**

Section 9 defines the objective of unit testing as *to demonstrate that the software units fulfill the software unit specifications and do not contain undesired functionality.*

To achieve this goal, the standard recommends the following unit testing methods to be implemented (see ISO 26262 Table 12 – Methods for Software Unit Testing).

Methods	ASIL				Supported by VectorCAST
	A	B	C	D	
1a. Requirement-Based Test	HR	HR	HR	HR	✓
1b. Interface Test	HR	HR	HR	HR	✓
1c. Fault Injection Test	R	R	R	HR	✓
1d. Resource Usage Test	R	R	R	HR	✓ <sup>1</sup>
1e. Back-to-Back Test between Model and Code (if applicable)	R	R	HR	HR	✓ <sup>2</sup>

*ISO 26262 Table 12 – Methods for Software Unit Testing*

<sup>1</sup> *If using VectorCAST/C++ under the control of the debugger or in conjunction with other tools.*

<sup>2</sup> *VectorCAST can use the data from models to generate test cases.*

### **1a. Requirements-Based Test**

Requirement-based test is a method where software requirements, such as functionality, robustness, etc., are first established. The code is then implemented, and test cases are associated with specific requirements to demonstrate the code achieves them. All testable requirements should be tested. Requirement-based test is highly recommended for all levels of ASIL, and is the cornerstone of most software standards in critical industries, including Avionics.

VectorCAST/C++ for unit testing makes requirement-based testing as efficient as possible. A test environment consists of the code under test and the associated test code, usually in the form of a driver and stubs. This environment generation is automatic with VectorCAST.

The creation of requirement-based tests is greatly facilitated within VectorCAST/C++ through the use of a GUI where the input and expected values of parameters, return values, global variables and even stubs (a piece of code that replaces dependencies so as to ensure a greater degree of isolation) can be specified. Test case creation can also be done through the import of data from a comma separated values (CSV) file.

Using VectorCAST/Requirements Gateway (a module of VectorCAST/C++) each test case can also be linked to a specific software requirement downloaded from a requirements management tool such as DOORS. Once the link is established, that information can also be uploaded to the requirements database, along with the test status information, making it even easier to monitor which requirements were met – and which were not.

### **1b. Interface Test**

ISO 26262 also highly recommends testing the interfaces of all units for all ASIL levels. This can be done in various ways, for instance by using boundary values of the interface, illegal values and median values. These values can also be combined in different ways. The goal here is to ensure that the interface is robust so as to prevent errors from occurring.

All of these activities can easily be performed with VectorCAST/C++. Extreme values can be specified according to the functional range or type, and illegal values can be easily specified. The exact range of each data type is duly tested by VectorCAST/C++ in the target environment, so the boundary values are known with precision and can be used to generate boundary test cases automatically. These can also be done in a combinatorial mode, which multiplies the number of effective test cases that are derived from a single set of data – entirely automatically.

## 1c. Fault Injection Test

Unit testing can be used to voluntarily introduce faults, including corrupt values of specified variables, in order to test the safety mechanisms embedded in the unit. It is especially recommended for ASIL D. VectorCAST/C++ offers facilities for this type of testing. The stub functionality, which also extends to library calls themselves, can be used to introduce intentional errors in the middle of a function. If greater flexibility is needed, the test case can also be run under the control of the underlying debugger so variable values can be changed at different points in time.

## 1d. Resource Usage Test

Resource usage testing at the unit level can be partially automated with the VectorCAST unit test tool for C and C++. Utilizing the VectorCAST/RSP module, tests can be run from a target board (or a simulator). Additional tools can then be used to measure resource consumption.

## 1e. Back-to-Back Test between Model and Code

VectorCAST tools have already been used by end clients to test code auto-generated from models in Simulink® and Rhapsody®. Vector Software is in contact with the publishers of these two products to make the process of unit testing auto-generated code even more seamless.

## Methods for Deriving Test Cases for Software Unit Testing

In addition to the different methods for unit testing the code, ISO 26262 also recommends four different strategies to generate test cases during unit testing. They are listed as follows (see ISO 26262 Table 13 – Methods of Deriving Test Cases for Software Unit Testing).

Methods	ASIL				Supported by VectorCAST
	A	B	C	D	
1a. Analysis of Requirements	HR	HR	HR	HR	✓
1b. Generation and Analysis of Equivalence Classes	R	HR	HR	HR	✓
1c. Analysis of Boundary Values	R	HR	HR	HR	✓
1d. Error Guessing	R	R	R	R	✓

ISO 26262 Table 13 – for Deriving Test Cases for Software Unit Testing

Requirement-based testing at unit level is central to ISO 26262 compliance, as it is highly recommended for all ASIL levels.

The standard also makes provisions for testing based on the structure of the code itself (generation and analysis of equivalence classes and analysis of boundary values) activities highly recommended whenever testing a system to levels ASIL B, ASIL C, and ASIL D.

### **1a. Analysis of Requirements**

As explained in the previous section on requirement-based tests, test cases for unit testing can be derived from low-level software requirements. In VectorCAST, such test cases can be linked to specific requirements and their status can be exported to a requirement management system, if one is present.

### **1b. Generation and Analysis of Equivalence Classes**

This strategy aims at testing the software adequately by determining partitions on the input domain necessary to exercise the software. These test cases aim at testing the program sufficiently. They can be based on either software specifications or the internal structure of the program (or both).

These types of activities are also highly automated in VectorCAST/C++, which enables users to quickly build test cases based on ranges and lists of values. These inputs can be executed in a non-combination, linear mode, or the tool can use all of the input combinations possible to run tests. Execution of these complex test cases is done automatically whether they are executing in a host, simulator, or target environment.

VectorCAST also features a partition test case generator to automatically create additional test cases on a provided domain.

### **1c. Analysis of Boundary Values**

These tests aim at removing potential software errors at parameter limits or boundaries, where it is most likely to fail. According to ISO 26262 Table 13, values that should be tested include values at, approaching and exceeding the boundaries of the interface.

This can be done easily within VectorCAST/C++, on both the range of the variable type and the functional range. It can be further automated by automatically generating MIN-MID-MAX test cases, which set all values to their minimum, median and maximum values, respectively. The minimum and maximum values are determined by testing the range of every type present in the program on the target board or simulator. Thus, using the tool on either the board or on a simulator will guarantee that the range of boundary values tested through automatically generated MIN-MID-MAX tests is valid in the system, whether it is 8, 16 or 32-bit.

These two tools can also test approaching values, illegal values, and even special values such as Not-A-Number (NaN), positive and negative infinity on floating-point variables.

## 1d. Error Guessing

According to ISO 26262 Table 13, these tests *can be based on data collected through a “lessons learned” process and expert judgment*. They are recommended for ASIL levels A, B and C, and are highly recommended if the system should be tested for ASIL D compliance. They aim to ensure that the components of the software are as error-proof as possible.

With VectorCAST, creating a test case is a straightforward matter, with no need for scripting. Signals can be raised for exceptions thrown and pointers can be voluntarily left un-initialized to see if the code will be protected against this type of occurrence. Performing “what-if” scenarios are easily generated, while manual test or script-based tools would require significant amounts of test code to be developed.

All test cases are kept outside the test harness until needed, which means that test cases can be created and deleted without the need for recompiling the code, which maximizes productivity when compared to other “similar” tools.

### Structural Coverage Metrics at the Software Unit Level

ISO 26262 strongly recommends the usage of code coverage as a metric to measure whether sufficient testing has been performed on a given unit. Since it is generally impossible to reach 100% code coverage when testing the system as a whole (using statement coverage) ISO 26262 further recommends that *at least* each line of code be duly tested during unit test.

The goal is clearly intended to be 100% coverage based on the coverage criteria selected. As stipulated by ISO 26262 Table 14, Note 4, *a rationale is to be given for the level of coverage achieved (e.g.; for accepted dead code or code segments depending on different software configurations), or else code not covered can be verified using complementary methods (e.g.; inspections)*.

At the unit level, three different criteria can be chosen for code coverage as indicated below (see ISO 26262 Table 14 - Structural Coverage Metrics at the Software Unit Testing)

Methods	ASIL				Supported by VectorCAST
	A	B	C	D	
1a. Statement Coverage	HR	HR	HR	HR	✓
1b. Branch Coverage	R	HR	HR	HR	✓
1c. MC/DC (Modified Condition/Decision Coverage)	R	R	R	HR	✓

ISO 26262 Table 14 – Structural Coverage Metrics at the Software Unit Testing

It should be noted that these levels of coverage are progressively more difficult to achieve. When using a tool like VectorCAST (which enables users to test MC/DC, Branch and Statement in isolation), the following combinations of criteria should be used in order to achieve a structural coverage level compatible with the letter – and spirit - of ISO 26262.

- *Statement coverage only for ASIL A*
- *Statement **and** Branch coverage for ASIL B and ASIL C*
- *Statement, Branch, **and** MC/DC for ASIL D*

VectorCAST/C++ (as well as VectorCAST/Cover) has a simple to use code coverage viewer. The coverage viewer indicates through symbols and color codes whether the code (a) is entirely covered (in green), (b) is partially covered (in orange), or (c) is not covered (in red). Leaving the cursor in place on any on the lines covered enables users to see which test cases cover specific lines.



The screenshot displays the VectorCAST code coverage viewer interface. At the top, a status bar shows coverage metrics: Statements 91%, Branches 86%, Pairs 63%, and Subprogram Coverage 100%. The main area shows a C++ code snippet with coverage indicators on the left margin. The code is as follows:

```
1 0 (T) Add_Included_Dessert
1 1 (T) (F) if({
1 1.1 (T) (F) Order -> Entree == HUOGUO &&
1 1.2 (T) (F) Order -> Salad == CAESAR &&
1 1.3 (T) ( ) Order -> Beverage == MIXED_DRINK )
1 2 * ) { Order->Dessert = PIE;
1 4 ( ) (F) else
1 4.1 (T) (F) if({
1 4.2 ( ) (F) Order -> Entree == BAOZI &&
1 4.3 ( ) ( ) Order -> Salad == GREEN &&
1 5 ( ) ( ) Order -> Beverage == WINE )
1 5 * ) { Order->Dessert = CAKE;
int Place_Order(int Table,
int Seat,
struct order_type Order)
{
struct table_data_type Table_Data;
2 0 (T) Place_Order
2 1 * Table_Data = Get_Table_Record(Table);
2 2 * Table_Data.Is_Occupied = v_true;
2 3 * Table_Data.Number_In_Party = Table_Data.Number_In_Party + 1;
2 4 * Table_Data.Order[Seat] = Order;
/* Add a free dessert in some cases */
2 5 * Add_Included_Dessert(&Table_Data.Order[Seat]);
```

In the case of targets with sufficient memory and liberal timing constraints, code coverage can even be animated – literally “replaying” how the code coverage was achieved during execution. For more restricted environments, code coverage can also be run in modes that save on memory space and/or have less of an impact on timing issues. The VectorCAST code coverage instrumentation and data collection has several options that allow the user to customize to maximize resource efficiency for special applications.

### **1a. Statement Coverage**

Statement coverage attempts to cover individual lines of code in a given unit. For instance, the following line would require a single test case to be covered.

```
if(i < 10 && j == 0 || k > 12)
```

It should be noted that although a single test case evaluating to false at this line will not cover additional lines that may be contained within that ‘IF’ statement. Likewise, if this ‘IF’ statement has an ‘ELSE’ statement attached to it, a test case evaluating to true will not cover the contents of that ‘ELSE’ statement. However, either the true or false test case will cover the ‘IF’ statement per se.

VectorCAST/C++ (as well as VectorCAST/Cover) supports Statement coverage, either stand-alone or in combination with Branch and/or MC/DC coverage. Furthermore, the automatic test case generation capabilities in VectorCAST/C++ can help engineers devise test cases seeking to maximize statement coverage. For instance, automatically generated test cases based on the basis paths (the unique paths in the code) will often provide a high degree of statement coverage.

### **1b. Branch Coverage**

Branch coverage concentrates on the state of decision points, which can be either true or false. For instance, the following line of code would require two test cases to be covered – one where the ‘if’ statement returns true, and another where it returns false.

```
if(i < 10 && j == 0 || k > 12)
```

VectorCAST/C++ (as well as VectorCAST/Cover) fully supports Branch coverage, either stand alone, or in combination with other criteria of code coverage. To comply with ISO 26262, VectorCAST can produce both Statement and Branch levels of coverage during a single test execution.

**1c. MC/DC (Modified Decision/Condition Coverage)**

MC/DC requires the greatest number of tests to accomplish the coverage requirement. This level of coverage demonstrates that all sub-conditions that are part of a conditional statement can independently affect the outcome of the conditional statement itself. For instance, in the case of the following statement:

```
if(i < 10 && j == 0 || k > 12)
```

One should demonstrate that by changing the value of 'i' while keeping the value of other sub-conditions stable, the end value will change.

This task can be very arduous even for the most experienced engineer. However, VectorCAST provides a very efficient way of doing this type of testing through its VectorCAST/MCDC module. A truth table is automatically generated which indicates clearly which test case pairs are required to achieve MC/DC coverage, and then flags which test cases and test case pairs have been provided.

Unit: manager  
Subprogram: Add\_Included\_Dessert  
Condition: # 1  
Source line: 17  
Actual Expression is: ( Order -> Entree == HUOGUO && Order -> Salad == CAESAR && Order -> Beverage == MIXED\_DRINK )  
Condition "a" (Ca) is: Order -> Entree == HUOGUO  
Condition "b" (Cb) is: Order -> Salad == CAESAR  
Condition "c" (Cc) is: Order -> Beverage == MIXED\_DRINK  
Simplified Expression is: ( a && b && c )

Row	Ca	Cb	Cc	Rslt	Pa	Pb	Pc
*1	T	T	T	T	5	3	2
2	T	T	F	F			1
*3	T	F	T	F		1	
*5	F	T	T	F	1		

Pa => a pair was satisfied (1/5)  
1/5  
Pb => a pair was satisfied (1/3)  
1/3  
Pc => no pair was satisfied  
1/2

### **Execution on Target Board or Simulator**

ISO 26262 clearly indicates a strong preference for executing unit test cases in an environment that is as close as possible to the actual environment where the program will run in the device. As mentioned in section 9.4.5, *the test environment for software unit testing shall correspond as far as possible to the target environment. If the software unit testing is not carried out in the target environment, the differences in the source and object code, and the differences between the test environment and the target environment, shall be analyzed in order to specify additional tests on the target environment during the subsequent test phases.*

VectorCAST/C++ in conjunction with the VectorCAST/RSP (Runtime Support Package) is the ideal tool to comply with this section of ISO 26262. Through the use of the RSP module, VectorCAST can automatically execute test cases on the end target (or simulator in the case of resource-poor environments). VectorCAST/RSP is specifically designed to interface with a unique set of compiler, board, debugger and RTOS (if any) combination, which enables it to run test cases on the actual target with little to no user input. The re-running of test cases (regression testing) at the command line is also 100% automatic.

### **VectorCAST during Software Integration and Testing**

Section 10 defines particular *integration levels* that are *tested against the software architectural design*. These integration levels are based on the hierarchical architecture of the software – from the very bottom (testing individual units) to the very top (testing the software as a whole).

Although ISO 26262 does not make the distinction, it is advantageous to consider the testing of modules (units that are combined to represent a functional process, but not the whole software) as *module testing* or *integration testing* (which is the term used on [www.vectorcast.com](http://www.vectorcast.com)). A test performed on the entire applications is often referred to as a *system test*.

In ISO 26262, the process should be to progressively integrate different units together based on the particular hierarchy of the software *until the embedded software is fully integrated* at system level (section 10.4.1). It thus suggests that both module testing and system testing shall be undertaken.

**Methods for Software Integration and Testing and for Deriving Test Cases**

Just like during the unit testing phase, a number of methods are recommended for software integration testing. These are listed below (see ISO 26262 Table 15 – Methods for Software Integration Testing).

Methods	ASIL				Supported by VectorCAST
	A	B	C	D	
1a. Requirement-Based Test	HR	HR	HR	HR	✓
1b. Interface Test	HR	HR	HR	HR	✓
1c. Fault Injection Test	R	R	HR	HR	✓
1d. Resource Usage Test	R	R	R	HR	✓ <sup>3</sup>
1e. Back-to-Back Test between Model and Code (if applicable)	R	R	HR	HR	✓ <sup>4</sup>

ISO 26262 Table 15 – Methods for Software Integration Testing

Likewise, the methods for deriving test cases for software integration testing are listed in below (see ISO 26262 Table 16 – Methods for Deriving Test Cases for Software Integration Testing).

Methods	ASIL				Supported by VectorCAST
	A	B	C	D	
1a. Analysis of Requirements	HR	HR	HR	HR	✓ <sup>5</sup>
1b. Generation and Analysis of Equivalence Classes	R	HR	HR	HR	✓ <sup>5</sup>
1c. Analysis of Boundary Values	R	HR	HR	HR	✓ <sup>5</sup>
1d. Error Guessing	R	R	R	R	✓ <sup>5</sup>

ISO 26262 Table 16 – Methods for Deriving Test Cases for Software Integration Testing

These methods are exactly the same as those defined in the unit testing section. In this section, we will focus on highlighting the useful features of VectorCAST that specifically address integration testing.

<sup>3</sup> If executing test cases from the control of the debugger.

<sup>4</sup> VectorCAST can use the data from models to generate test cases.

<sup>5</sup> Applicable to VectorCAST/C++ only.

### ***How to build an integration test environment***

VectorCAST/C++ can be used to run integration test cases during module/integration test. Its usage is the same as that specified during the unit testing phase – the only change is that several files that comprise a functional process are used to construct the test harness instead of a single file.

VectorCAST/C++ offers real integration testing – the files that are put together become, in essence, a larger, integrated module. Thus, if tests are executed on one function in a first unit, and if this unit itself calls another function in the second unit, both functions will be tested and both will generate code coverage. As in the case of unit testing, there is no script to write in order to generate an integration test environment – everything is automatically generated.

### ***Reusing test cases from unit testing***

The advanced regression capabilities of VectorCAST can also be very helpful during integration test. Basically, it is possible to reuse part or all of the test cases designed during unit test and apply them seamlessly to integration testing. The code may have changed, but the test cases will run without any problem (so as long as the function interface remained stable, otherwise the test case will either be discarded or flagged as no longer relevant). This capability to import and export test cases is a great time-saver.

### ***Testing software modules (libraries) from third-party suppliers***

Additionally, VectorCAST/C++ can be used to test libraries, even without access to the code. This may also be useful when integrating software components that are coming from third parties, such as suppliers and contractors. Building a test case around a library file is just as straightforward as building a test harness around actual code and the testing activity can be conducted in a very similar fashion.

### ***System testing***

Once the integration reaches system level, the usual build system usually will take over, and test case input will be done through other means, such as signal generation, pushing buttons at a console, or maybe with a simulator. During system testing, VectorCAST/Cover is used to capture the code coverage (see two sections below).

**Structural coverage metrics at the software architectural level**

Function coverage is defined by each function being called at least once during the test campaign. Call coverage requires at least one execution of every function call. Section 10.4.6 further stipulates that the unspecified software component must be identified and removed or de-activated.

Methods	ASIL				Supported by VectorCAST
	A	B	C	D	
1a. Function Coverage	R	R	HR	HR	✓
1b. Call Coverage	R	R	HR	HR	✓ <sup>6</sup>

ISO 26262 Table 17 – Structural Coverage Metrics at the Software Architectural Level

<sup>6</sup>To be supported by VectorCAST in 2012.

VectorCAST/Cover (as well as VectorCAST/C++) captures function coverage. Furthermore, call coverage can be achieved indirectly by achieving partial statement coverage which stimulates all the calls of each function present in the integration or system test environment.

**Executing Integration Tests**

As described previously for unit testing, ISO 26262 clearly indicates a strong preference for executing integration test cases in an environment that is as close as possible to the actual environment where the program will run in the field. This is quite achievable with both VectorCAST/C++ and VectorCAST/Cover.

One frequent question asked about VectorCAST/Cover is how the tool exports the code coverage data from different targets. In fact, the tool can export the coverage data through three broad methods: (a) save to a file (particularly useful in case a file system is present or a simulated file I/O can be established), (b) send through a port (such as a serial port), and (c) store the data in a memory buffer that is then saved with the help of the debugger or another means.

**Reporting**

ISO 26262 specifies creation of a number of documents, such as a Software verification specification and a Software verification report.

The VectorCAST products produce a variety of test artifacts based on unit test, integration test, or system test. These reports can be generated in either Text format or HTML. They can be saved outside VectorCAST, and have been used to comply with a variety of standards such as: IEC 61508, CENELEC, DO-178B, etc.

## Certification

Both VectorCAST/C++ and VectorCAST/Cover can be certified for compliance activities with ISO 26262. Please contact Vector Software for more information.

## Other Tools

In conclusion, VectorCAST/C++, VectorCAST/Cover, and VectorCAST/Requirements Gateway can automate testing and code coverage activities in a way that makes complying with ISO 26262 requirements much more efficient. All of these tools can export their individual reports in HTML or Text, which have been used successfully in the past to comply with a number of demanding industrial standards.

VectorCAST/RSP is the module that enables the execution of a test harness on simulator or a target board. The process is entirely automated, so test cases can be executed individually or as a group by a simple click of a mouse or the command line. The execution itself requires no user input.

In addition, VectorCAST/Manage provides a completely automated regression testing facility for all VectorCAST generated tests.

## Successes in Safety Critical Markets

The VectorCAST tools have been used successfully for over fifteen years by companies developing safety and mission critical applications such as: Military, Avionics, Aerospace, Medical Devices, Railway, Automotive, and Industrial Controls. Hundreds of companies use VectorCAST to satisfy the embedded software testing requirements that will soon apply to the automotive market with ISO 26262.

### About Vector Software

Vector Software, Inc., is the leading independent provider of automated software testing tools for developers of safety critical embedded applications. Vector Software's VectorCAST line of products, automate and manage the complex tasks associated with unit, integration, and system level testing. VectorCAST products support the C, C++, and Ada programming languages.

### Vector Software, Inc.

1351 South County Trail, Suite 310  
East Greenwich, RI 02818 USA  
P: 401.398.7185  
F: 401.398.7186  
E: [info@vectorcast.com](mailto:info@vectorcast.com)  
W: [vectorcast.com](http://vectorcast.com)