使用 Parasoft C++test 以满足 ASIL 需求

实现汽车软件的功能性安全

引言

安全功能越来越多地在电气、电子或可编程电子系统中得到实现。这些系统一般都是非常复杂的，这就使得在实际中完整地判断每个失效模式(failure mode)或测试所有可能的行为成为了不可能完成的任务。虽然预测其安全方面的性能非常困难，但测试仍然是非常有必要的。关键的挑战在于设计一种能够预防危险性失效或能在这些失效发生的时候对其进行控制的系统。

安全性将成为未来汽车开发中的一个关键因素。新功能——不仅仅在驾驶员辅助系统领域，同样在车辆动态控制以及主动和被动安全系统中——越来越多地触及到了安全性工程的范畴。未来的开发以及对这些功能的集成将会进一步提升对安全系统开发流程的需求，同时还需要提供所有合理安全性目标已满足的证据。

随着复杂性增强、软件内容以及机电层面实现的趋势，系统化的失效以及随机硬件失效的风险有显著增加的趋势。通过提供可行的需求以及流程，ISO/DIS 26262 包含了能够将这些风险降低到可接受程度的指南。

本文档的宗旨在于详细阐述 Parasoft C++test 如何帮助汽车软件开发团队满足特定 ASIL 级别的需求。本文将首先介绍 ISO/DIS 26262 标准所定义的 ASIL 的概念。然后将讨论 Parasoft C++test 是为软件开发以及测试提供自动化最佳实践的集成解决方案。最后将阐述 Parasoft C++test 如何满足或部分满足软件开发过程中特定 ASIL 的需求。

汽车软件完整性级别

IEC 61508 标准中所定义的安全完整性级别（SIL）——或在 ISO/DIS 26262 标准中所定义的汽车安全完整性级别（ASIL）是为避免不合理驻留风险，用以指定安全性措施的四个安全性度量之一（IEC 61508 中用 1-4 来表示，ISO/DIS 26262 中用 A-D 来表示），其中 4 或 D 是最紧要的级别，1 或 A 是相对最不紧要的级别。注意，安全完整性级别是特定安全功能的一种属性，而不是整个系统或某个系统组件的属性。

在安全性相关的系统中，各个安全功能都需要指定相应的安全完整性级别。根据 ISO/DIS 26262，各个危险事件中的风险是通过以下一些属性来进行评估的：

- 相应情况发生的频率，也称作“曝光频率（exposure）”
- 潜在损坏的影响，也称作“严重性（severity）”
- 可控性

根据上述三个属性的值，可以对某个特定功能性缺陷的相应安全完整性级别进行评估。这将确定该安全功能的整体 ASIL 级别。

ISO/DIS 26262 标准为达成各个汽车安全完整性级别制定了相应的需求（安全性度量）。为了使相应的危险失效发生的可能性变得更低，这些需求在越高的安全完整性级别中变得越严苛。

关于 **Parasoft C++test**

Parasoft C++test 是经广泛证明的最佳实践集成解决方案，它能有效提高开发团队工作效率和软件质量。C++test 能促进：

- **静态分析** —— 静态代码分析，数据流静态分析以及度量指标分析
- **同行代码审查**流程自动化——准备、提示以及追踪
- **单元测试**——单元测试创建、执行、优化以及维护
- **运行时错误检测**——内存访问错误、泄漏、崩溃以及其它

这为开发团队提供了一种实际的方式来预防、发现以及纠正相关的错误，从而确保其 C 以及 C++代码如预期般工作。为了促进快速修复，每个检测到的问题都基于可配置的严重级别分配进行排序，并自动分发到相应代码的开发者处，同时分配到他或她的 IDE 环境中，并配以该问题代码处的直接链接以及如何修正该问题的描述。

对于嵌入式以及交叉平台开发而言，C++test 可以同时用在基于宿主机以及基于目标平台的代码分析以及测试流程中。

自动进行代码分析以监测是否遵守标准

采用合适的编码策略可以建立预防性的编码习惯，从而消除整类编码错误的发生。C++test 对这些代码进行静态分析，以检测其是否遵守相应策略。如需配置 C++test 以增强某个团队或组织专用的编码标准策略，用户可以使用内建以及自定义规则来建立自己的规则集。代码分析报告可以按不同的形式生成，包括 HTML 以及 PDF 格式。

数百条的内建规则——包括 MISRA、MISRA 2004、全新的 MISRA C++标准的实现，HIS 源码度量指标、Meyer 在《Effective C++》以及《Effective STL》等书中推荐的以及来自其它主流源码的准则——有助于从使用不当的 C/C++ 语言识别潜在缺陷，实施最佳编码实践，以及提升代码的可维护性和可重用性。通过图形化的 RuleWizard 编辑器创建的自定义规则，可以增强标准 API 的使用并防止单个缺陷被发现后类似的应用程序缺陷重复出现。

在不运行软件的情况下识别运行时缺陷

作为 C++test 先进的数据流静态分析模块，BugDetective 能模拟可行的应用程序执行路径——这些路径可能会跨越多个函数和文件——并判断这些路径是否会触发特定类别的运行时缺陷。能够发现的缺陷包括使用未初始化或无效的内存、空指针解引用、数组及缓存溢出、除零、内存和资源泄漏以及各种无效代码（dead code）。这种不运行软件即可发现运行时错误的能力对于嵌入式代码是尤其有价值的，因为在嵌入式开发中要进行详细的运行时分析以检测这些错误往往是无效或不可能的。

由于在开发者的 IDE 环境中提供了对潜在缺陷的完整路径追踪，C++test 极大地简化了缺陷分析过程。自动化到代码的交叉链接能帮助用户迅捷地跳转到某段高亮显示的分析路径中的任意一点上。

合理化的代码审查

代码审查被认为是发现代码缺陷最有效的方法。不幸的是，由于认为代码审查会耗费大量额外的精力，很多组织都未充分使用代码审查。C++test 的代码审查 (Code Review)模块能自动地准

备、提示以及追踪同行代码审查，从而建立起非常高效的面向团队的审查流程。所有代码审查的状态，包括审查者所添加的所有注释都会被 C++test 的基础设施[1]加以维护并进行自动分配。C++test 支持两种典型的代码审查流程：

- **提交后代码审查**。该模式是通过自定义的源码控制器接口来自动识别代码库中的代码更改而实现的，并基于预设的更改代码与审查者的映射关系创建代码审查任务。
- **提交前代码审查**。用户可以在桌面上选择某一文件集并分配到审查者进行审查，或者自动地检测所有本地更改源代码的方式来初始化代码审查。

团队代码审查的效果可以通过 C++test 的静态分析能力进一步地增强。由于自动监视团队的编码策略，从而消除了对逐行审查代码的必要性。一旦代码被提交进行审查，相应的违例就已经被发现并清除了。这样，审查者就可以更加注重对算法的检查，对设计本身的审查，以及查找那些自动化工具无法检测到的隐蔽的错误。

监测应用程序以发现内存问题

应用程序内存监测是用于消除严重的内存相关缺陷并具有零误报率的最著名方法。对正在运行的应用程序进行监测以查找某类特定问题——诸如内存泄漏、空指针、未初始化内存以及缓存溢出等——测试过程一旦结束后，结果将会立即可见。

无需进行高级并且耗时的测试活动，被插装的应用程序——为进行监测添加了额外的代码——将通过标准的功能测试并标注出所存在的所有问题。应用程序可以在目标设备、模拟的目标平台或宿主机上执行。收集到的问题将直接显示在开发者的 IDE 中，并提供了解以及修复该问题的详细信息（包括内存块大小、数组索引、分配/释放堆栈的追踪等）。

覆盖率度量指标将在应用程序执行时进行收集。这些指标可以用来说明哪部分应用程序经过了测试，以及调整回归单元测试用例集（作为功能测试的补充）。

运行时错误检测允许用户：

- 通过简单的功能测试查找复杂的内存相关问题——例如内存泄漏、空指针、未初始化内存以及缓冲溢出等
- 通过应用程序运行来收集代码覆盖率
- 通过在真实目标环境中执行被监测的应用程序来提升测试结果的精确性

具有覆盖率分析的单元及集成测试

C++test 的自动化能力极大地提升了测试新开发或既有代码的正确性和可靠性的效率。C++test 自动地生成完整的测试套件，包括测试驱动以及用于各个独立函数的测试用例，这些测试套件都是以类似 CppUnit 格式的纯 C 或 C++代码的形式提供的。这些测试套件，无论是否进行修改，都可以用于对代码的功能行为进行初始化验证。通过使用边界用例条件，这些自动生成的测试用例同样能检测函数对未预期输入的相应状况，从而发现潜在的可靠性问题。

完全通过一组具体的 GUI 实用程序来简化测试的创建和管理。图形化的测试用例向导（Test Case Wizard）能帮助开发者为选中的功能快速创建黑盒功能测试套件，而不必要担心它们之间的内部运作或嵌入数据的关联性。数据源向导（Data Source Wizard）能协助参数化测试用例以及桩函数——使开发者以最少的努力提升测试范围和覆盖率。桩函数视图（Stub View）能促进桩函数的分析以及生成，在该视图中，将显示出代码中所用到的所有函数并允许用户为任何在测试范围内不可用的函数创建桩函数——或为某个特定测试目的更改既有函数功能。测试执行以及分析

[1] 译注：测试基础设施(Infrastructure)指机构中与测试相关的各种硬件、软件、服务及数据通信设施等的总和。

集中在测试用例浏览器（Test Case Explorer）中，该浏览器统一管理所有既有项目测试套件并提供明确的通过/失败状态。这些功能对于支持自动化的持续性集成和测试“随工程进度而测试”的开发是非常有帮助的。

通过自动记录运行时测试结果的测试断言来捕捉既有的软件行为，自动生成和用户编写的测试用例都可以用来生成回归测试用例的基础。随着代码库的演进，C++test 重新运行这些测试并将当前测试结果与最初捕捉到的“黄金集”进行比较。C++test 可以轻松地配置为使用不同的执行设置、测试用例以及桩函数以支持在不同环境中的测试（例如不同的持续性集成阶段、测试未完成的系统或测试已完成系统中的某个特定部分）。

多重度量指标测试覆盖率分析器，包括语句、分支、路径以及 MC/DC 覆盖率，能帮助用户评估测试套件的效率以及完成度，同时还能显示测试以及验证需求的遵从性，例如 DO-178B。测试覆盖率是通过所支持的所有覆盖率度量指标中高亮显示的代码来表示的——在 GUI 中或以不同颜色标注列举出的报告。覆盖率摘要报告包括文件、类以及函数数据可以不同的格式生成。

可配置的详细报告

C++test 的 HTML、PDF 以及自定义格式的报告可以通过 GUI 控件或选项文件来进行配置。标准的报告中包含代码分析以及测试结果通过/失败概要信息、已分析文件列表以及代码覆盖率概要信息。这些报告可以通过自定义设置以包含当前可用的静态分析检测列表、对于测试中扩展测试输出的通过/失败状态、关键度量指标的趋势图参数以及带颜色标注的所有代码覆盖率结果的完整代码列表。通过基于角色的邮件发送过滤器，生成的报告可以自动通过电子邮件进行发送。除向相关标注缺陷代码的开发者直接发送数据之外，C++test 还会向管理人员和团队负责人发送概要报告。

高效的团队部署

C++test 能建立一种高效的流程来确保软件验证任务能够集成在团队的既有工作流程中并自动化进行——使开发团队的精力能够更加注重于真正需要人类智慧的任务。缺陷的审查以及纠正是通过自动化任务指定及分配实现的。每个发现的缺陷都将进行优先级排序，指定到该段代码的相应开发者处，并分发到他或她的 IDE 中，同时提供到该段代码的交叉链接以及完整数据。为帮助管理人员评估及记录趋势，集中报告机制能确保提供质量状态以及过程的实时可视化报告。这些数据同时还能帮助确定是否需要额外措施来满足内部目标或证明符合质量策略。

团队工作流程模块

将 Parasoft C++test 与 Parasoft Concerto 相集成，能帮助团队针对既定策略以及所遵守的流程标准来配置工作流程，包括 CMMI 以及 SPICE。Concerto 提供完善的追踪能力，这些追踪能力包括源代码、自动化测试、对所有工程部件（需求、缺陷/增强以及任务）的手动测试。通过追踪以及分析软件开发度量指标以及过程，Concerto 能够判断哪些事务是耗费人力和时间的，同时使得开发团队能够快捷地发现特定的问题以及问题区域。

使用 C++test 满足 ASIL 需求

ISO/DIS 26262 标准要求对于给定 ASIL 属性的安全功能的软件开发生命周期中应该使用多种不同的方法。下面将逐一描述 Parasoft C++test 中能用来有效地实现这些方法的功能。请注意，此处所提供的信息只是对 C++test 在 ASIL 相关的软件确认与认证过程中的一个简要介绍。如需澄

清 ISO/DIS 26262 标准所定义的任何需求，请具体参考该标准或向功能安全性专家进行咨询。如果关于如何在 ISO/DIS 26262 确认与认证过程中使用 C++test 你还有其它的问题，请联系你的 Parasoft 代表。

下述符号在以下表格中用以标记：

- **(+)** – 与 ISO/DIS 26262-6 标准中所推荐的功能相匹配的方法
- **(++)** – 与 ISO/DIS 26262-6 标准中所强烈推荐的功能相匹配的方法

C++test 功能性描述中参考了 ISO/DIS 26262-6 中的相应需求或方法，例如（表 10：1f）参考了 ISO/DIS 26262-6，表 10 中的方法 1f。

编码标准一致性——静态代码分析

C++test 功能	ASIL A	ASIL B	ASIL C	ASIL D
编码标准一致性模块——通用				
使用静态代码分析验证软件实现（表 10：1f）	(+)	(++)	(++)	(++)
分析类型				
使用代码度量指标（例如全复杂度、基本复杂度等）来增强代码的低复杂度的策略（表 1：1a）	(++)	(++)	(++)	(++)
使用编码标准来增强仅使用语言子集的策略，例如避免不安全的结构（表 1：1b）	(++)	(++)	(++)	(++)
增强特定命名习惯的策略（表 1：1h）	(++)	(++)	(++)	(++)
增强特定编码习惯的策略（表 1：1g）	(+)	(++)	(++)	(++)
增强特定格式习惯的策略（表 1：1f）	(+)	(++)	(++)	(++)
增强业界认知的编码标准规则集的策略，诸如 MISRA、JSF、HIS 源码度量指标等（表 1：1e）	(+)	(+)	(+)	(++)
使用合适的编码标准规则以增强保守性的编程的策略，例如检查 *malloc* 函数的返回值、检查调用函数的返回值错误代码等（表 1：1d）		(+)	(++)	(++)
特定编码标准指南				
查找隐式类型转换以增强强类型化的策略（表 1：1c；表 9：1g）	(++)	(++)	(++)	(++)
查找函数中的多重退出点（表 9：1a）	(++)	(++)	(++)	(++)

C++test 功能	ASIL			
	A	B	C	D
报告可能的变量初始化问题（表 9：1c）	(++)	(++)	(++)	(++)
查找无条件跳转（表 9：1i）	(++)	(++)	(++)	(++)
报告动态对象的不安全使用（表 9：1b）	(+)	(++)	(++)	(++)
查找变量名的二义性（表 9：1d）	(+)	(++)	(++)	(++)
报告隐藏的数据流（表 9：1h）	(+)	(++)	(++)	(++)
查找全局变量的使用（表 9：1e）	(+)	(+)	(++)	(++)
报告递归函数（表 9：1j）	(+)	(+)	(++)	(++)
报告不安全的指针使用（表 9：1f）		(+)	(+)	(++)
增强保守性实现技术的策略——例如，检查 *malloc* 函数的返回值、检查调用函数返回值的错误代码（表 1：1d）		(+)	(+)	(++)

Bug Detective – 静态数据以及执行流程分析

C++test 功能	ASIL			
	A	B	C	D
Bug Detective – 通用				
使用数据流分析验证软件实现（表 10：1e）	(+)	(+)	(++)	(++)
使用控制流分析验证软件实现（表 10：1d）	(+)	(+)	(++)	(++)
使用变量的可能值的抽象表示来分析源代码——语义分析（表 10：1g）	(+)	(+)	(+)	(+)
Bug Detective – 特定规则示例				
报告未初始化变量的使用（表 9：1c）	(++)	(++)	(++)	(++)
报告错误指针使用（表 9：1f）		(+)	(+)	(++)

自动同行代码审查

C++test 功能	ASIL A	ASIL B	ASIL C	ASIL D
自动同行代码审查模块 –通用				
使用自动同行代码审查模块对源代码进行审查（表 11：1e）	(+)	(++)	(++)	(++)
使用自动同行代码审查模块对源代码机型推演（表 11：1f）	(++)	(+)		

单元测试

C++test 功能	ASIL A	ASIL B	ASIL C	ASIL D
单元测试模块 – 通用				
单元测试执行以及报告执行的单元测试结果（需求 9.4.1）	对某些 ASIL 无相应特定的推荐			
添加或不添加插装的单元测试执行能力以验证诸如覆盖率插装不会影响测试结果（需求 9.4.4）	对某些 ASIL 无相应特定的推荐			
在生产环境中的目标设备或仿真器中执行单元测试（需求 9.4.5）	对某些 ASIL 无相应特定的推荐			
自动化单元测试用例生成模块				
生成使用边界值的自动化单元测试用例（表 13：1c）	(+)	(++)	(++)	(++)
生成使用推测值的自动化单元测试用例（表 12：1c）	(+)	(+)	(+)	(++)
使用工厂函数为生成自动化单元测试用例准备输入参数值（表 12：1b）	(+)	(+)	(+)	(+)
测试管理模块				
将测试用例与需求和/或缺陷进行映射——与团队工作流程（Team Workflow）模块一起使用（表 12：1a）	(++)	(++)	(++)	(++)
使用用户定义的测试用例——无论手动编写的或使用测试用例向导创建的——测试给定需求的特定原子用例（atomic cases）（表 13：1a）	(++)	(++)	(++)	(++)

C++test 功能	ASIL			
	A	B	C	D
使用数据源有效地为功能性等效需求的原子用例（atomic cases）提供多重输入（表 13：1a）	(++)	(++)	(++)	(++)
功能桩函数				
使用桩函数根据给定需求中的设置来控制经执行测试的流程（表 13：1a）	(++)	(++)	(++)	(++)
使用功能桩函数为自动化单元测试执行替换用户接口（表 12：1b）	(++)	(++)	(++)	(++)
使用桩函数在测试中提供故障条件（表 12：1c）	(+)	(+)	(+)	(++)
覆盖率模块				
分析 MC/DC 代码覆盖率（表 14：1a）	(+)	(+)	(+)	(++)
分析分支代码覆盖率（表 14：1b）	(+)	(++)	(++)	(+)
分析语句代码覆盖率（表 14：1c）	(++)	(++)	(+)	(+)

应用程序监测

C++test 功能	ASIL			
	A	B	C	D
应用程序监测模块——通用				
监测运行中的应用程序并报告运行时问题（需求 10.4.2）	对某些 ASIL 无相应特定的推荐			
在生产环境中的目标设备或仿真器中监测应用程序（需求 10.4.7）	对某些 ASIL 无相应特定的推荐			
覆盖率模块				
分析功能覆盖率（表 17：1a）	(+)	(+)	(++)	(++)

总结

Parasoft 的 C++test 能帮助汽车软件开发团队实现或部分实现 ISO/DIS 26262 标准中定义的汽车软件完整性级别的软件开发过程需求，并满足嵌入式程序的标准作业程序（SOP）。包括编码标准一致性分析、数据以及控制流分析、单元测试、应用程序监测、工作流程组件以及自动化同行代码审查过程在内的广泛的分析手段在内广泛的分析类型——与包含高级别细节的可配置测试报告能显著地促进软件验证过程中所需进行的工作。

关于 **Parasoft**

在过去 20 年中，Parasoft 一直致力于研究应用程序中软件错误的原因及表现。我们的方案将我们的研究成果嵌入到提升质量的可持续性软件开发过程中。这能带来更好的编码基础，更加可靠的功能单元以及鲁棒性更好的业务流程。无论你是在设计面向服务架构（SOA），升级既有系统或是改进质量流程——你都可以使用我们专业且屡获殊荣的产品来提高你的开发效率和应用程序的质量。如需更多信息，请访问：http://www.parasoft.com。

联系 **Parasoft**

全球总部
101 E. Huntington Drive,
2nd Floor Monrovia, CA 91016
USA
Toll Free: (888) 305-0041
Tel: (626) 305-0041
Fax: (626) 305-3036
Email: info@parasoft-embedded.com
URL: http://www.parasoft-embedded.com
URL: http://www.parasoft.com

欧洲总部
Parasoft SA
Kielkowskiego 9, Krakow 30-703
Poland
Phone +48 12 290 91 01
Fax +48 12 290 91 02
Email: info-pl@parasoft.com

中国总部
Rm. 902, Golden Magnolia Plaza, W. 1 Dapu Road
Huangpu District
Shanghai
Phone (021) 60932819
Email: info-china@parasoft.com

Parasoft SA (法国)
Chateau de Sainte Assise, 77240 Seine Port

France
Phone (33 1) 64 89 26 00
Fax (33 1) 64 89 26 10
Email: sales@parasoft-fr.com

Parasoft 德国公司
Mähringer Weg 45
89075 Ulm
Germany
Phone +49 89 4613323-0
Fax +49 89 4613323-23
Email: info-de@parasoft.com

Parasoft 英国公司
The Lansdowne Building
2 Lansdowne Road
Croydon.
CR9 2ER
United Kingdom
Phone +44 (0)208 263 6005
Fax +44 (0)208 263 6100
Email: sales@parasoft-uk.com

其它地区
请浏览http://www.parasoft.com/contacts