Parasoft C++test
使用评估报告

——编码规则有效提高软件代码质量

PARASOFT®

导 读

编码规则帮助您提高代码质量，生成一致代码，防止易错编码风格。Robert Buckley 对 MISRA-C 和编码规则添加了一个注释。 本文作者 JunHo 和 YoonKyu 是三星电子软件实验室的工程师。

编码规则一致性检查

为克服严峻的软件开发挑战并同时减少开发成本，软件工程领域已经形成了自己的规则惯例，如需求工程分析，设计技术，制程开发，等等。许多规则惯例都应用于开发的实际执行阶段，如编码规则，代码重构，代码检查，静态分析。其中，编码规则是基础，它能够很好地提高代码可靠性，帮助不同的开发人员都生成一致的代码，并防止易出错编码方式的出现。

三星电子重点着眼于通过定义并强制执行内部编码规则来提高代码质量。我们 QA 部门使用了一个编码标准一致性检查器来实现这个目的，但我们并没有规范全部地将这个初始检查工具应用到我们的软件开发过程中去。因为这个工具功能不强，所以我们只是在最后的审核阶段才偶尔用一下它。因此我们只看到了它对代码质量的提高起到了一丁点儿的作用。

最近，我们评估了 Parasoft 的 C++TEST，并应用它到我们正在进行的开发项目“MOBILE”中。在本文中，我们将从中学习到的经验和大家一起分享。在本文中，一个“编码规则条目”是指一个在公司编码规则和一致性文档中描述的总的描述条，一个“编码规则”（或“规则”）是指一个在自动化编码规则工具中制订的具体的编码规则。

三星电子是一个主要的消费电子公司。尽管三星以主营硬件起家，软件也迅速成为了我们一个主要的关注中心，正如大多数其他的消费电子厂商一样。这个“MOBILE”项目是三星的一个电子 C/C++开发项目，它是要开发出一个用于移动设备的可重用、可扩展的面向对象的软件框架。我们的 QA 部门是一个独立出来专门测试三星电子开发出来的软件的。我们投入了相当多的时间来评估自动化工具，以求能最大地减少重复的工作量。

C/C++编码标准

这个 MOBILE 项目使用一套源于总的三星编码规则并针对这个项目特别指定的编码规则。这套 MOBILE 项目规则可以通过改变语言变量和其他开发约束条件来进行改写。比如，在 MOBILE 项目中的

上海市打浦路 1 号金玉兰广场西楼 902 室
Rm. 902, Golden Magnolia Plaza, W. 1 Dapu Road, Huangpu District 200023 Shanghai, P.R. China
www.parasoft.com

一些编译器不支持对异常情况的处理。因此，当一个对象的构造器被调用的时候，不可能侦测到资源分配失败。为解决这个问题，我们在 MOBILE 项目中采用了大家熟知的 two-phase object construction（二阶段对象构造）技术（在 MOBILE 项目编码规则中有描述）：将一个对象的初始化分为对象分配阶段和资源分配阶段，以通过一个值的方式返回异常情况（见代码清单 1）。

```
class ResourceManager
{
    ResourceManager(); // allocate only object
    result Construct(); // allocate resources
                        // 'result' contains error code
};
int main()
{
    // Two phase construction
ResourceManager aObject;
if (aObject.Construct() == FAIL)
printf("Resource allocation is failed");
}
```

代码清单 1

另外，这个 MOBILE 项目需要对编码规则进行严格恪守。这个项目主要目标是建成一个软件框架给其他开发人员使用；它必须是一致的、组织良好的，以使得软件开发能够很好地在这个框架上进行。越多的项目涉及进来，就越需要一个自动化的工具。这就是为什么 MOBILE 项目要采用一个编码规则检察器。

Parasoft 的 C++TEST(www.parasoft.com)提供了自动的 C/C++单元测试，和自动化的编码规则检查。我们选择 C++TEST 作为我们的编码规则检查器，是因为它对于我们的大多数考虑来说是最有效的解决方案。

C++TEST 的一个明显的特点就是它的基于图形化界面的规则。如图 1 显示了对规则“每个全程变量必须进行初始化”的图形化界面规则描述，在分析源代码时，每当发现一个“全程变量”，这条规则便会评估逻辑组件。如果以下条件中的任何不相符合，程序就会报告有一个代码违规：

上海市打浦路 1 号金玉兰广场西楼 902 室
Rm. 902, Golden Magnolia Plaza, W. 1 Dapu Road, Huangpu District 200023 Shanghai, P.R. China
www.parasoft.com

- 侦测到的全程变量是一个外部的声明
- 它没有进行初始化
- 它的类型不是一个数组或类

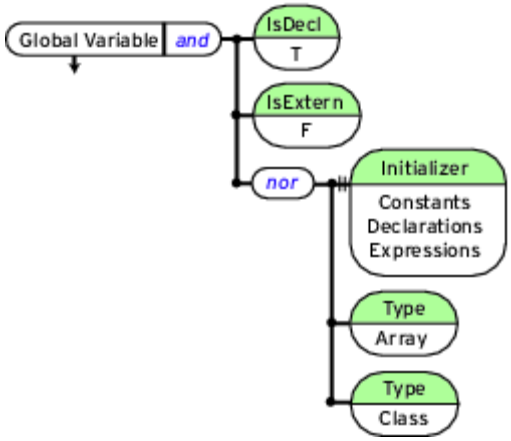

图 1: Parasoft C++test 中基于图形化界面的编码规则

图形化界面简化了规则创建。大多数的 C++代码检查器在创建规则时需要编写脚本；这有一定的难度，并要求更多的 C++编程知识。

因为现有的条件是图形化显示的，所以基本图形化界面的规则能被容易地理解和执行。通过基于图形化界面的规则可以有更好的可扩展性，因为通过图形化界面只有预定义的节点和条件可以选择。

选择一个编码规则检查器

我们的选择标准包括产品使用特点，但不是特点的细节（规则选择，规则执行，可量测性）。我们通过以下途径来建立这些标准：以前的使用编码规则检查器的经验，项目开发部门的反馈，产品评估报告。

能很灵活地修改规则：大多数编码规则检查器包括预执行的规格。拥有内含的规则可以减少规则执行的工作量。但是这些规则通常并不完全符合我们的编码风格。而且，因为大多规则中是简单地执行，不实的报错是常见的事，因而使得结果不具可靠性。我们需要一个简单的方法来自定义规则以排除异常情况，添加新的规则或者修改已存在的规则。类似于 LINT 的工具可以检查一些编码规则项，但缺少规则自定义的特点。尽管有些检查器也能为规则自定义提供参数修改，但我需要更多的灵活性来修改详细规则。

上海市打浦路 1 号金玉兰广场西楼 902 室
Rm. 902, Golden Magnolia Plaza, W. 1 Dapu Road, Huangpu District 200023 Shanghai, P.R. China
www.parasoft.com

能在不同的级别上报告编码规则的遵守程度： 许多编码规则检查器支持文件级别的报告。然而，出于管理目的，文件包级和项目级的报告成为管理者们所希望有的。比如，要项目经理经常希望按文件包或项目来浏览编码规则违规以编码规则违规的趋势和对编码规则违规校正的优先级别，特别是在项目工期快要到了的时候。

能与开发环境相集成：许多规则违规能被很容易地校正。比如，像“使用 TAB 而不是空格进行缩进”之类的违规可以通过简单地用 TAB 替换空格就可以校正。在这样的情形中，有一个可以直接访问违规源代码（通过与开发环境紧密集成）的编码规则检查器，就可以非常有效地减少校正所需要的时间。

另外，当一个工程向前推进的时候，它将所含更多的文件和更的"include/directive"设定。如果检查器不在 IDE 内运行，那么检查器就要通过导入或同步 IDE 项目文件（如 MAKEFILES，DSP/DSW 文件等）来创建工程文件。

能为 C/C++创建统一的规则：我们的主要的编程语言，C 和 C++，有一个相似的结构（除了 C++具有更多的基于对象的和类属性编程）。在两种语言上为相同的项维护两种不同的规则将需要额外的资源。

能够识别语言变量：相比 C，C++的历史比较短。编译器提供商们在 ISO C/C++发布以前产出了他们自己人的 C++编译器，并且研究显示，许多 C++的实际应用并不能很好地支持 C++ ISO 标准(http://www.ddj.com/184405483)。因为编码规则检查器经常分析源代码，所以能识别语言变量就显得非常重要了。

能检查未经预编译的头文件：一些编码规则检查器缺少对头文件的直接检查，取而代之的是，在头文件中的代码违规，是通过检查在预编译执行的文件中的头文件进行间接地报告。在这种情况下，一些代码违规被忽略了，如在头文件中与预处理程序指令和注释相关代码违规，这里通常包含一些被其他开发人员使用的重要信息。所以，直接对头文件的检查是一个我们所希望的功能。

使用检查器的经验

应用一个编码规则检查器可以减少在应用中的编码标准违规。然而，侦测和消除的违规数量取决于目标工程的特点和编码规则的质量。我们发现每个工程都有其相似的整体倾向，但同时又有影响编码规则检查的不同细节。因些，得出来的结果应该被看成是一个趋势而不是一个确定的样式。

上海市打浦路 1 号金玉兰广场西楼 902 室
Rm. 902, Golden Magnolia Plaza, W. 1 Dapu Road, Huangpu District 200023 Shanghai, P.R. China
www.parasoft.com

直接的利益是指减少违规数量是如何提高代码质量的。

非直接利益是指其他不曾预料到的带给开发人员的好处。

图 2 显示了编码违规数量的总体趋势。为消除规则不断发展带来的影响，我们使用最新的规则来进行所有的检查。在一个从 10 月 4 日到 1 月 5 日间相对稳定的违规数量之后，在 2 月 5 日有一个大约 1/8 的违规减少。从 2 月 5 日到 3 月 5 日间，有一个不希望有的违规增加，但这是可以接受的，因为目标项目仍在继续向前推进。

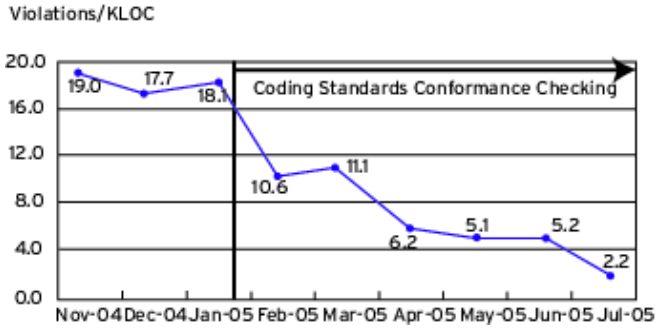

图 2：总的违规数量/每千行代码

图 3 显示了违规校正对只对当前模块有影响的违规数量。从采用检查器之日起，违规数量减少了 6.6 个百分比。这个数量没有包括注释规则--这通常不会有大的变化影响但确实是需要重要的进行变更的工作。

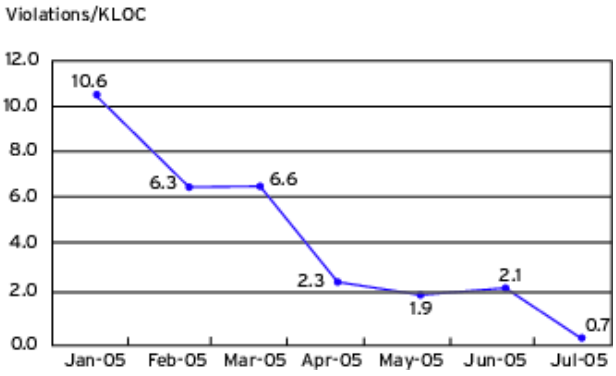

图 3：小变化违规数量/每千行代码

图 4 显示了校正不只影响当前模块的违规数量。从引入检查器以来，违规减少了 19.6 个百分点。

上海市打浦路 1 号金玉兰广场西楼 902 室
Rm. 902, Golden Magnolia Plaza, W. 1 Dapu Road, Huangpu District 200023 Shanghai, P.R. China
www.parasoft.com

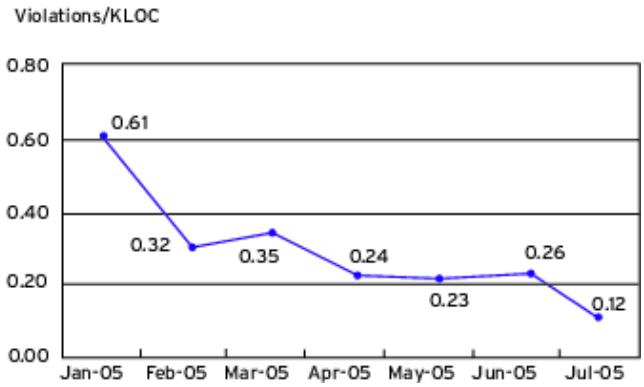

图 4：有大的改变影响的违规的数量/每千行代码

代码规则检查给我们带来的一个意外收获，就是它对开发人员的教育和知识能力提升。新的开发人员，甚至是一些有经验的开发人员，在开始的时候不大理解一些编码规则项的意义和重要性。比如，编码规则项“最好进行初始化而不是分配”是被认为是一个有效的方法来初始化一个构造函数中的成员变量（可参考：《Effective C++》，作者：Scott Meyers，Addison-Wesley，1992）。一些开发人员不理解为什么在一个构造函数的列表中对成员进行初始化要比在构造函数体中对成员分配初始化值要更好。他们在查看检查器的检查结果时讨论这些问题。这就最终帮助开发人员完全理解了 C++的特点，并且也展示了编码规则检查是如何帮助开发人员，使他们从犯错中进行学习成长的。

另一个间接带来的好处就是去除了那些不切实际的编码规则项。当我们应用编码规则一项目 MOBILE 中去的时候，我们发现大多数开发人员并不遵守这条规则即“每行不要 80 格”，这条规则的产生是因为有些老的开发环境是 80 格显示。我们检查了我们的开发环境并得出结论：在我们的条件下，这种有限制的开发环境很少使用，我们建议一行使用更长的格。”所有我们将一行的格数限制从 80 格改到 150 格。这种编码规则修改也可以提高开发人员对遵守编码规则的接受程度。

学到了什么

直到最近以前，我们的检查还只是 QA 部门在工程级别上检查是否遵守编码规则。这对于追踪缺陷倾向和维护一套规则是有效果的。但是会有一些缺点。

首先，报告的违规来源并不总是开发环境中的那段代码。开发人员通常并不在一个集中的源码库中操作，他们在自己的区域进行书写和修改代码，然后复制或 check-in 源码到集中的代码库中。如果开发源

上海市打浦路 1 号金玉兰广场西楼 902 室
Rm. 902, Golden Magnolia Plaza, W. 1 Dapu Road, Huangpu District 200023 Shanghai, P.R. China
www.parasoft.com

码和 QA 测试的代码不一样，那么开发人员要识别和修改报告违规的来源就比较困难了。

再者，开发人员希望立即能够验证到对于违规的校正已经消除了存在的问题。

基于这些原因，我们决定要让开发人员和 QA 一样可以运行编码规则检查工具。

那些不认真遵守规则的开发人员会产生很多有违规的代码，并且也通常不会去校正这些代码违规。这是尤其会产生与识别或设计相关的违规问题，这样会使得在以后的开发阶段来校正这些错误非常困难，因为那时进行这样的校正会有一个较大的，整个工程范围的影响。所以我们推荐将编码规则检查从早期的编码阶段就开始，这样违规代码就能在传出模块之前就得到校正。

为什么我们以前的工具没能在编码规则检查上发挥效应的也是因为在整个组织级别上缺乏对规则的维护。

在一套初始规则建立之后，规则应该得到不断的修改定义，因为某些规则也许不适用于特定的项目，开发风格会根据开发的领域不同而会有改变。一套规则应在项目不断往前推进的过程中根据项目的具体操作而进行不断的维护更新。

自动的编码规则检查是对代码走查的一个补充。在我们的开发过程中，代码走查是强制的，因为这样可以有效地找出开发人员的逻辑错误和失误。然而，开发人员也会因为工期太紧的压力跳过代码走查。根据我们的经验和研究显示，一些开发人员很容易被一些编码风格问题弄得很头痛，并在代码走查的过程中花很多工夫到这个上面(参见 :D. Kelly 和 T. Shepard 编写的"Qualitative Observations from Software Code Inspection Experiments"一书; CASCON, 2002)，从而，一些开发人员将代码走查看作是一件耗时却收效甚微的工作，并不跳过这一步。这种情形可以通过在代码走查之前使用自动化的代码规则检查来消除代码违规的方法来进行避免。这样的话，在代码走查的时候我们就能将精力集中在发现逻辑错误和严重错误上来。而且，自动化的检查只能涉及到我们编码规则项中的大约 50%(一些规则项用自动化工具具体执行起来非常复杂)，所以代码走查需要用来检查剩余的规则项。

小 结

我们应用 C++TEST 到了我们几个项目中并在每个项目中都取得了很好代码质量提高效果。我们准备将其应用到更多的项目中去，用其分析代码的质量。

上海市打浦路 1 号金玉兰广场西楼 902 室
Rm. 902, Golden Magnolia Plaza, W. 1 Dapu Road, Huangpu District 200023 Shanghai, P.R. China
www.parasoft.com