



何时、为何以及如何进行单元测试

Adam Kolawa 博士访谈

Parasoft delivers quality as a continuous process

本文是对 Parasoft CEO 也即《自动缺陷预防：软件管理中的最优实践》(Wiley-IEEE,2007) 一书的共同作者 Adam Kolawa 系列采访的一部分，在这个部分中，Adam 论述了何时，为何以及如何引入基础软件验证。同时，这个部分还谈到了静态代码分析、代码走查、内存错误检测、消息/协议测试、功能测试以及负荷测试等。

在本访谈中，Kolawa 驳斥了单元测试即将淘汰的说法，解释了单元测试是如何为用户在“早期测试”中实现价值的，并且指出代码覆盖率并不是评估测试套件完整性的最佳方法。通读本文能帮助读者明确如何迅速可靠地修改代码，以及如何方便地开始。

**目前，单元测试在理论界是一个热门，而在实际使用中却是少有问题。请问你对此有什么看法呢？**

单元测试需要设置实际且有效的测试套件，这是相当复杂的。开发者往往对应用程序的外部表现很清楚，比如应用程序将怎样响应用户的操作等，但是对应用程序内部本身的运作情况却知之甚少。而这对编写单元测试套件而言确实是相当关键的。

当使用单元测试时，开发者需要创建相应的初始化条件，这能使得应用程序像一个完整的系统那样工作。如果初始化条件设置不正确，那么单元测试的结果将不能反映程序在实际中的运行情况，这样的结果对开发者而言也不会非常有用。除非能模拟程序运行的真实情况，否则你不能确保某些模块正确无误。这一点是相当关键的。比如，某个模块可能要求支持很多条件...但当你创建它的时候，你就需要知道其相应代码是否能够维持该要求。除非你能在相应的环境中进行验证，否则你不能确保这些模块的正确性。

创建初始化条件是相当困难的，但又是绝对必须的。我认为试图严格地创建初始化条件的开发者可能会为其所需的工作量而感到气馁，相反，其他的开发者可能会为单元测试报告的无价值的结果而感到失望。

**现在关于单元测试是否有存在的必要性有很多的争论。你怎么看呢？**

我认为单元测试在现在比以往任何时候都更加有用。但我认为我们应该有较人们普遍关注的单元测试所不同的测试，也即早期测试。下面解释一下。

今天我们所构建的系统，如 SOA 及 RIA 等，无论从大小还是复杂度上都是以往所不能同日而语的，再加上较以前更短的开发周期。一般而言，我们对哪些功能需要测试都有一个清楚的了解，因为在需求中对测试用例已经有了很好的定义。尽管如此，由于其复杂性，例如很多系统往往与外部世界相连，分期系统难以建立和维护，每次测试均需要大量的设置等等诸多困难，所以作为整体测试的时候仍然是很困难的。也许对每个用例测试一次是可行的，但是你必须得进行日常的测试以保证破坏性的修改能够被即时发现。

**这也是单元测试的产生原因吧？**

是的。单元测试在帮助开发者测试模块的同时省去了处理复杂系统的烦恼。使用类似 Parasoft Tracer 的技术，开发者可以创建单元测试用例来捕捉其中的功能。使用应用程序的图形用户界面 (GUI)，SOA 或者网络测试客户端，开发者只需要在打开 Tracer 功能的时候运行测试用例。在测试用例执行的同时，Tracer 能够监视所有被创建的对象以及输入输

出数据，随后为开发者创建一个能模拟这些操作的单元测试套件甚至设置好相应初始化条件。

这些生成的单元测试套件可以在其它机器上独立使用。这就意味着开发者可以只使用一台机器，甚至是一个标准桌面开发环境，以在验证过程中复制一个复杂系统的行为。将这些测试套件加入到回归测试套件中，并且持续地运行能使开发者迅速地发现代码的修改是否影响了功能。

根本地讲，这能从两个方面解决今天的复杂系统中包含的问题：

1. 帮助开发者设置耗时且繁琐的实际测试条件。
2. 帮助开发者创建能独立运行的持续性全自动测试用例。

**你的意思是，单元测试是在应用程序可以实际运行后创建的。那么为什么说单元测试是早期测试呢？**

你所指的单元测试根本不是早期测试。它是一种完全不同的单元测试，我认为它甚至比早期测试更有价值。但是，早期测试仍然是有价值的。

举个例子吧，比如你新建了一个应用程序。当你在实现某些功能的时候，为什么不顺便创建一个能捕捉其功能运行情况的单元测试呢？这时，你对实现的需求以及代码运行的一些细节都非常明确...所以这是正确设置初始化条件以及编写验证需求的测试用例的最佳时机。当你在着手进行这些工作的时候，务必明确该测试用例所实现的功能。这样做能保证测试用例和需求的一致性。换句话说，也就是该测试失败能告诉开发者具体哪些需求受到了影响。

当你每完成一个测试用例时，将其加入回归测试套件中像其它单元测试那样持续执行。

你可以选择在代码编写完毕后编写相应测试套件，甚至在代码编写前就创建测试套件，以实践 TDD (测试驱动开发)。

**这种基于需求的测试开发过程都是手动进行的吗？**

编写验证需求的测试套件是个创造性的工作。开发者必须真切的明确代码的细节以及如何有效地对其进行测试。但也有一些工具能够为开发者提供一些帮助。类似 Parasoft Jtest 的自动化单元测试工具能帮助减少开发者用在设置实际有效的测试用例上的时间。

例如，存储在对象库里的初始化对象对设置实际的初始化条件而言是非常有用的。桩函数库可以帮助屏蔽外部引用情况以使单元测试能够独立运行。手动设置桩函数和对象不是一件轻松的工作，所以自动化工具能为开发者节省很多时间。同时，测试用例参数化设置 (Test Case Parameterization) 能用来将诸如数据源输入或使用边界条件 (corner case conditions) 自动生成的输入数据流等额外输入加入测试用例。

这些工具并不能帮助开发者判断哪些测试需要进行...但是它们却能帮助开发者高效地实现测试。

**关于单元测试的另一种普遍认识是单元测试的代码覆盖率越高就越有效。你同意吗？**

事实上，我认为在绝大多数情况下代码覆盖率都不是必需的。人们希望高的代码覆盖率...但是如果要达到 80%以上或更高的代码覆盖率，你就不得不毫无头绪地执行所有的代码。这就如要求一个钢琴师要触及所有的琴键而不是演奏美妙的音乐。当他在演奏音乐时，他的琴键覆盖率是取决于乐章本身的。

关于代码覆盖率还有一个问题就是当代码覆盖率超过某个程度（约 70%以上）时，其测试套件就变得越来越难以维护了。当一个测试套件要实现的代码覆盖率更高时，这通常是一些函数被混乱执行的信号。一个典型的例子就是假设你有合适粒度级别的断言，并且这些断言对代码变更很敏感。结果，每天所报告的急需处理的断言失败就会很多，但是这些断言失败基本上都不会告知开发者任何代码变更。

很多人都问我是否能够实现 100%的代码覆盖率。答案是能...但是除了实现 100%的代码覆盖率之外，对开发者而言并没有什么特别的好处。

**那么你怎么评估测试套件的完整性呢？**

如同钢琴协奏曲只有在所有音符都被演奏后才算结束一样，测试套件也只有把所有的用例都被覆盖后才算测试完毕。

每个用例都应该至少含有一个测试用例。这可以是单元测试、其它的测试或者是所有这些测试的组合。当我将所有的用例都用作测试用例时，我就可以日常地运行我的回归测试套件并且得知是否当日所修改的代码破坏或对用例中所涉及到的功能性造成了负面的影响。这就为经常需要升级的应用程序提供了一张根本性的安全网。

**代码覆盖率能告诉我们一些什么信息吗？**

是的，事实上它会告诉开发者一些很有趣的信息。假设你对所有的用例都只编写了一个测试用例，那么这大概会带来 40%的代码覆盖率。这就意味着 60%的代码将不会被你的用例所关联到，在实际中，这是很可怕的。开发者可以据此问出下列的一些问题，你的代码中是否有很多是无效的呢？应用程序的需求明确了吗？你的用例定义中是否有很多冗余呢？可能你丢失了一些需求的用例。或者你的需求本身就不完整。这些问题是很值得推敲的。

另一方面，如果有人在不使用很多测试用例的情况下还是实现了很高的代码覆盖率呢？这就说明其代码编写很严谨且高效，并且使用了很多可重用的设计。

**当你已经创建好了一组代码覆盖率很高的测试用例时，接下来该做什么呢？**

你需要确定这些测试用例是经维护过的。单元测试并不是指创建单元测试用例。而是维护这些单元测试用例。如果你没有保证这些测试用例和应用程序的同步，那么它们很快就将失效。

为了保证这个过程地进行，你需要一个支持架构和一个工作流程。一个理想的解决方案就是让这个架构在夜间日常地运行：

1. 从源码控制系统中同步最新修改的代码。
2. 运行整个回归测试套件。
3. 判断代码修改造成了哪些断言失败。
4. 确定造成断言失败的开发团队成员。
5. 将相应信息分发到相应的开发者处。

这些任务是自动进行的，不需要额外的开发团队的人力资源。

然后，当开发者开始每一天的工作的时候，首先将结果导入到其桌面或 IDE 中，检查其代码造成的断言失败，并对其进行相应的处理。这种处理可能是对代码中功能性缺陷的处理或者更新测试套件以使其正确反应代码的行为。

通过这种日常进程，开发者在每天工作开始前只需进行一点额外的工作就可以提升测试套件的价值以及生命周期，同时这种对未预见性影响的早期发现能够帮助开发者将其消灭在萌芽状态。

如果没有这种支持流程以及让这种流程自动化运行的框架，单元测试将会事倍功半。

### 有什么技巧可以让这个过程变得尽量简单吗？

上面我所提到的自动化断言失败检测与分发机制是关键性的。Parasoft 的开发团队已经发现了这一点。很多年前，当我们开始审视我们的单元测试流程时，我们发现开发者在代码编写完毕后会编写相应的功能性单元测试套件。尽管如此，当应用程序经修改后造成的测试断言失败却并没有被迅速地进行处理。

在任何情况下，开发者都需要检查这些断言失败并且确定引起这些失败的相应行为修改是有意还是无意的。我们的开源单元测试执行工具并不能告诉我们哪个部分引发了哪些测试失败，原因以及时间。取之则是我们日常测试报告的类似 3000 多个测试中有 150 个失败这样的信息。除非挨个查看了所有的失败，并确定了是谁的代码导致了这些失败，否则没人能够确定其测试是否失败。这么看来，我们的系统是不够负责的。

起先，我们试图要求所有的开发者都检查这些测试失败，但是那需要耗费大量的时间并且不可能成为一个日常的开发习惯。后来，我们尝试在开发团队中制定专人来检查这些测试失败并且进行分发。尽管如此，没有人愿意做这种工作，因为这是一件相当繁琐的事情，并且分发的任务很难被团队成员所接受。

最终，我们在单元测试产品中内建了自动错误分发机制，并且在内部试用。现在，断言失败通过源码控制系统可以被自动分发到其相应的开发者处。如果某个开发者造成了一个或更多的断言失败，自动错误分发机制可以通过 email 通知他，然后在新一天的工作开始前，他可以导入这些相应的结果到其 IDE 中将这些问题解决。这就为开发团队节省了大量的时间。

我想我明白你所谓的保持该过程进行下去的意思了。但是应该从哪里着手呢...特别是对于没有用大量测试用例测试过的既有应用程序而言？

在只进行过很少量测试或者根本没有进行过测试的代码库基础上工作无异于如履薄冰：你的任何动作都有可能是破坏性的...但对软件而言，这些破坏性的结果并不是立即就表现出来的。在你进行下一步工作前，先创建一个作为修改检测安全网的单元测试套件。这样你就能确保能及时得知修改是否影响了应用程序的功能。

如果你知道既有功能的相应用例有哪些，你就可以在执行这些用例时来追踪单元测试用例。这样，你就能获得一组能够日常运行的测试用例来确保你对代码的修改不会破坏或改变软件的核心功能。

甚至即使你没有任何用例，你也可以通过工具自动生成我所谓的行为级回归测试套件：即一个能捕捉代码当前功能的基准单元测试套件。正如我前面提到的，测试套件的可维护性随着代码覆盖率高于某个层度后会显著降低，所以如果你的工具需要你选择是创建一个可维护性的测试套件还是一个高覆盖率的测试套件时，我们推荐你选择可维护性的测试套件。为了检测对基准版本的测试套件的修改，确保代码库经修改后会自动地被相应的测试套件定期地测试，最好将这种测试定为日常的。这样，你就能得知基准测试套件的功能性是否遭到了影响，并且当你评估这些影响时，你将获得更多的既有功能的细节。

### 能给我们下一些结论吗？

当今的应用程序已经变得越来越复杂了，所以开发团队在提高软件质量以及回归测试套件上面所做的努力也相应地水涨船高。从单元测试开始就是一个很好的开端...当然，这并不是圣经。只通过单元测试是不能提高软件质量的，也不能检测处所有的缺陷以及代码修改造成的影响，所以，单元测试也并不是万灵丹。

如果你确实想提升软件的质量并且想知道代码修改对程序功能带来的影响的一切细节，那么你需要一个从静态代码分析到同行代码走查、单元测试、消息/协议层测试以及负荷测试的持续性质量改善流程。建立这些组件并且让它们作为一个整体工作需要耗费团队的一些资源，但是相对于在团队开发效率以及应用程序安全性上所得到的回报却是相当划算的。

## 关于 Adam Kolawa

Adam Kolawa 是 Parasoft 的共同建立者兼 CEO, 他被认为是软件开发领域的权威以及促进软件质量的持续性改善方法的领先创新者。2007 年, 他入选 eWeek 的 100 位 IT 界最有影响力的人物。

Kolawa 是《自动缺陷预防: 软件管理中的最优实践》(Wiley-IEEE, 2007) 和《安全网络应用》(Wiley, 2001) 这两本书的共同作者, 并撰写了 O'Reilly 的《完美的代码》中的一章。他还在《华尔街日报》、《CIO》、《Computerworld》以及《Dr. Dobb's Journal》等报刊中发表了数百篇点评以及技术文章, 并在物理学和并行处理方面著有大量的学术论文。

Kolawa 在加州理工大学取得了理论物理博士学位。并享有软件技术方面的 15 项专利。

## 关于 Parasoft 的单元测试

Parasoft 于 1997 年在市场上首先推出了自动化单元测试解决方案, 一直至今。

Parasoft 的质量解决方案提供一个完整的架构来创建、管理单元测试, 并帮助用户通过单元测试获得更高的价值。我们的专利技术能自动生成单元测试套件, 这些测试套件不仅能在多个层级上验证代码的正确性和可靠性, 并且能捕捉代码行为来为回归测试套件建立一个可维护的基准套件。在基准套件建立好后, 开发者能从下述三个方面来使其变得更加智能化:

- 打开 Parasoft Tracer 功能并执行用例的相应情景。在与应用程序交互的过程中, Parasoft Tracer 能对其进行监视, 然后在真实的功能性单元测试用例中捕捉这些动作。
- 人工扩展自动生成的 xUnit (JUnit, Cactus, CppUnit, NUnit 等) 测试用例, 或者使用类似鲁棒性对象库、桩函数库、测试用例参数化设置等功能来验证具体的需求。
- 集成既有 xUnit 测试用例 (或者新建), 这能使用户集中管理测试执行、报告以及代码覆盖率分析并且让用户能得益于我们的自动错误分发机制。

总的来讲, 这些测试用例为担心代码修改会影响程序功能的团队构建了一张安全网。

为了确保使用单元测试获得长期成功, 实现一个质量改善流程是很关键的, 这个流程应不仅能提高企业化解决方案自动化程度, 并且还应该是可持续且可调整的。Parasoft 能帮助用户实现这些任务。将单元测试作为可持续性质量改善流程的一部分, 并且显著减少软件缺陷提升开发效率, 这这个领域, Parasoft 拥有 10 年的经验。

如需了解更多信息, 请以下述方式联系 Parasoft, 或者访问:  
<http://www.parasoft.com/solutions>。

## 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

### 美国

101 E.Huntington Drive, 2<sup>nd</sup> Floor  
Monrovia, CA 91016  
Toll Free: (888) 305-0041  
Tel: (626) 305-0041  
Fax: (626) 305-3036  
Email: [info@parasoft.com](mailto:info@parasoft.com)  
URL: [www.parasoft.com](http://www.parasoft.com)

### 欧洲

法国： Tel: +33 (1) 64 89 26 00  
英国： Tel: +44 (0) 1923 858005  
德国： Tel: +49 89 4613323-0  
Email: [info-europe@parasoft.com](mailto:info-europe@parasoft.com)

### 亚洲

Tel: +886 2 6636-8090  
Email: [info-psa@parasoft.com](mailto:info-psa@parasoft.com)  
其它地区  
请浏览 <http://www.paraosft.com/jsp/pr/contacts>