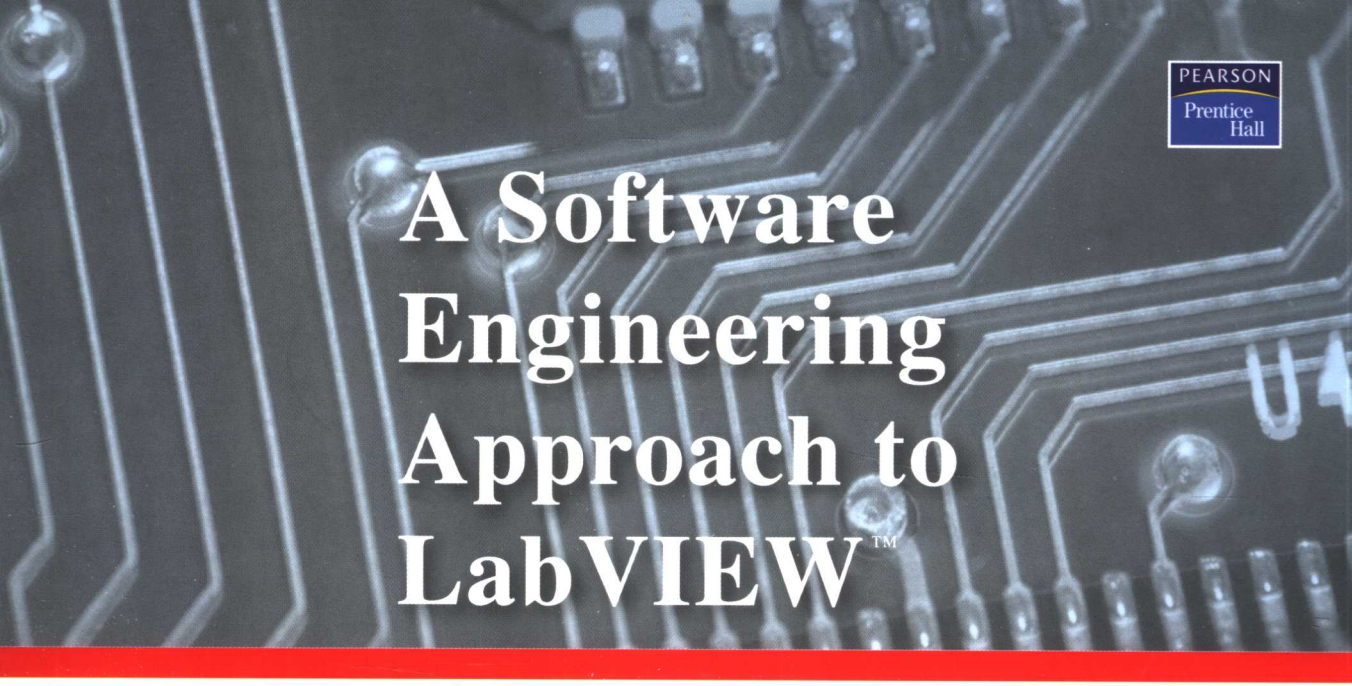

软件工程方法在LabVIEW™中的应用

Jon Conway , Steve Watts 著
罗霄 周毅 等译

清华大学出版社

A Software Engineering Approach to LabVIEW™

软件工程方法在LabVIEW™中的应用

Jon Conway , Steve Watts 著
罗霄 周毅 等译

清华大学出版社
北京

EISBN: 0-13-009365-3

图书在版编目(CIP)数据

软件工程方法在 LabVIEW™ 中的应用/(美)康威(Conway, J.),(美)瓦特(Watts, S.)著；罗霄，周毅等译. —北京：清华大学出版社，2006.4

书名原文：A Software Engineering Approach to LabVIEW™

ISBN 7-302-12579-1

Ⅰ. 软…　Ⅱ. ①康…②瓦…③罗…④周…　Ⅲ. 软件工具，LabVIEW－程序设计　Ⅳ. TP311.56

中国版本图书馆 CIP 数据核字(2006)第 012715 号

出 版 者：清华大学出版社
http://www.tup.com.cn
社 总 机：010-62770175

地　　址：北京清华大学学研大厦
邮　　编：100084
客户服务：010-62776969

责任编辑：李晔
印 装 者：北京国马印刷厂
发 行 者：新华书店总店北京发行所
开　　本：185×230　**印张**：11.25　**字数**：249 千字
版　　次：2006 年 4 月第 1 版　2006 年 4 月第 1 次印刷
书　　号：ISBN 7-302-12579-1/TP・8043
印　　数：1～3000
定　　价：25.00 元

前　言

设计和实现系统可以有许多种方法。我们并不要求你马上用本书介绍的技巧替代你目前在设计和编写软件时所使用的方法。特别地，我们所介绍的只是在实际编写应用程序时所使用的软件设计和实现方法，并希望读者能够有自己的体会。

重要的是，应当注意到本书的作者都是有实际工作经验的工程师，他们通过编写软件来偿还抵押贷款，而不是通过写书。

测试工程师的看法

Steve Watts写道：

作为一名接受过正规训练的测试工程师，我已经编写了许多年的测试系统，并且使用过许多种不同的编程语言（如 HPBasic、UCLA Pascal、Turbo Pascal、Visual Basic 以及 QuickBasic）。在编写许多复杂的系统时我都有过相同的经历。

如果在开始编写代码之前没怎么做设计工作，那么在工作进行到50%的阶段可以正常进展，到90%的阶段仍然可以完成，在我开始沾沾自喜的时候，事情就发生了！

我现在要用“复杂度爆炸”来描述这种现象：软件中很小的变化都可能引起整个系统的问题。用户可能额外提出“计划外”的要求。我不再能够在头脑中清晰描述这个系统。项目最后10%的工作将会花费另外90%的时间。

我知道有些事情出了问题，但却描述不清情况、原因或程度。最后，我只能将它放到一边，感到软件真是麻烦。

几年前Jon加盟公司时，推荐了一种名为LabVIEW的语言。LabVIEW成为了公司的标准，于是，我开始学习使用它。我用它编写的第一个应用程序是一个小型温度记录仪（必须声明，我很不喜欢那种编程方式）。问题仍然会出现，但对我来说却变得清楚了。确实，较之于我那时正在使用的Pascal和Visual Basic语言，G大大提高了生产力，但“复杂度爆炸”现象仍然存在。于是，我去找Jon，并跟他讨论这个问题，Jon向我介绍了LCOD。我以前从未想到还有这样一门叫做软件工程的学科（我想我确实是一名软件工程师，因为我编写软件），我也没有听说过耦合度、结合度或者信息隐藏的说法，OOD、OOA以及结构化软件设计，我以前也都忽略了。

我是那种要从本质上，而不是只从文字上理解一种方法的人。在我们还在讨论相当

抽象的概念时，我就开始试图真正理解。我学习了软件工程和面向对象编程的研究生课程，并在我的项目中对结构化软件设计、CASE工具和OOA进行了实践。从理论到实践的固有难度以及让软件同行们接受的难度（可能超出你的想象！），让我相信整个过程会比我想象的困难。但事实却相反！

我开始看到，通过应用这些技术，程序变得可管理了，接近项目尾声的时候，程序的复杂度不会更高，我还可以进行后期的修改，却不会降低程序的健壮性，维护工作也变得简便、快速，用户很满意并且对此印象深刻，压力小了，世界上也不再有疾病、瘟疫，邻居友爱，事事太平。

别以为我在胡说，这些技术并不会使复杂问题变简单，但应用这些技术至少不会使问题变得更复杂。

作为软件工程师，我们要以下面的原则共勉：

- 应提交计划要提交的内容。
- 应按时提交。
- 应确保可按预定方法操作。
- 应确保修订和纠错工作不会破坏程序的稳定性或影响其实现。
- 应当注意复杂度管理工作。

充满玄机的软件 = 坏

简单的软件 = 好

我们的一个客户曾写下了下面的意见（我们可没有付费给他！）：“LCOD让一个复杂的测试系统更简单、更灵活，时间将证明这一点。”

让我们用一次旅行来比喻（全书我们都在使用这个比喻），我们认为已经采取了足够多的措施以保证我们能够返回，而且做了一些标记，正如希望的一样，这些标记会在旅行中帮助你。

我从来都不会后悔增加了软件的灵活性，但总是后悔没有注意灵活性问题。

本书介绍的技术非常易于理解。如果学会，那么我们认为每个人都能够成功运用其中之一二。我们的目的是通过易于理解和可应用的方式向大家介绍和解释使用LabVIEW进行软件设计的概念。许多技术和方法学都拘泥于计算机科学理论，从而忘记了设计的初衷，我们却一直关注设计并希望能够说明某些计算机科学理论。

致谢

感谢Bernard Goodwin（Prentice Hall）和Tiffany Kwehn（Carlisle 出版服务机构）的帮助。还要感谢本书审稿人的有益的和有建设性的意见，特别是John Compton-Smith。

感谢我的家人和朋友，感谢Yvette、Jasmine和Poppy女士，没有她们，我会更快地完成本书，但她们的“干扰”点亮了我每天的生活。还要感谢“星期二夜总会”和Red House（Whitchurch,Hampshire, UK）耐心的工作人员，感谢他们的啤酒、食物和讨论。最后，感谢所有使用这些“高科技玩具”让我挣到钞票的人们。

——Steve Watts

目　录

第1章 简 介

欢迎阅读本书。希望本书介绍的方法和示例能够给你的工作带来帮助，或者至少对你目前使用 Laboratory Virtual Instrument Engineering Workbench（以下简称 LabVIEW）的方法带来一些启示。本书介绍的技术会使你的生活更加轻松，使复杂的项目变得更加可行。本书的作者是每天应用这些技术的实践者，所以我们希望本书能够更加实用。

在每种工程规范中，好的设计都是通向成功的最大推动力；相反，不好的设计也是导致失败的主要原因。希望在读完本书后，你能记住这个事实。

下面要介绍我们的软件之旅中最重要的部分，关于 LabVIEW 的重要性。本书并没有什么“秘笈”，只有一些设计策略以及实践证明非常有效的软件实现技巧。

祝你旅途顺利。

1.1 LabVIEW 的缺点

J. Conway 写道，“LabVIEW Sucks!”是的，我被告知使用的这种愚蠢的“图形化”语言绝对是垃圾。Tom 是我的同事，那时我正在进行一个大型的“自动测试设备”（ATE）项目。那是 1993 年，我现在相信，管理层选择 LabVIEW，而不是那时可用的其他语言是明智之举。我以前的经验都是基于课本的，而不是基于 PC 的。我绝对憎恨这种语言，还有 Windows 3.1。

我的遭遇和许多其他憎恨 LabVIEW 的软件工程师一样（憎恨是一个表达强烈感情的词，但它好像确实能够概括他们的所有感觉）。可是，为什么和许多工程师一样，我也经历了那样的一段艰苦时间呢？我想答案非常简单，在 LabVIEW 中编程实在太糟糕了，因为你不得不使用数据流程图（data flow diagram）。一般地，这表明 LabVIEW 的所有用户都必须以某种特定方式编程。对许多程序员来说，这会让人发怒，但现在我认为，他们以及以前的我，都已经忘记了这段历史。

别以为我在胡说，LabVIEW 并不能让一个实习程序员写出超级工程化的代码，事实正好相反（由于本书中的某个基本观点）。LabVIEW 的作用只是为如何构建软件打下基础。例如，在 C++的世界里，对软件工程师来说，最流行的语言也可以有许多种编码方法。如果你是 C++的用户，那么可能会对这种说法不以为然，但我确实这样认为。许多人认为 C++是一种面向对象（以下简称 OO）语言，因此就认为所编写的 C++代码也是面

向对象的。其实不然，用 C++编写软件，很可能和面向对象的程序设计完全不搭界，因为工程师们并不一定要使用面向对象的技术。

LabVIEW 的不同之处在于：除了用数据流规则编写软件外，你别无选择。事实上，如果你确实努力尝试过，就会了解这一点，但这确实也让人烦躁到无法工作。它消除了误入歧途或者发明自己的系统的可能，有些人可能会说这是遏制创造性；我却觉得他们说的纯属无稽之谈。任何软件都只是表达某种设计的工具，就好像楼房表现的是建筑师的设计一样。重要的是软件是否结实，就好像让楼房矗立的内部连接件。想象一下这样的混乱场面吧：建筑工人可以随意使用自己喜欢的任何一种材料，并以某种方式将其连接在一起，最后散乱成一堆。我想你可能已经明白了。看看那些失败、超时或者不能满足要求的软件项目，你就会开始明白，遵守规则对工作是多么有益。

使用 LabVIEW 还有许多其他的好处，这在后面的章节中将会讨论。在向前继续之前，掌握工具的使用方法的概念十分重要。软件领域的另一个重要问题是语言叙述方法。LabVIEW 通常被认为是使软件编写工作更加容易的一种软件包。

软件并不简单，LabVIEW 的作用是使简单的事情做起来也简单，但对于所有现代编程语言来说，一旦超出了任务的最简单部分，软件结构难度的降低就成了成功的前提。如果只是打算编写最简单的应用程序，那么你可能不需要本书。但是，如果打算参与到大型项目中，那么，这里介绍的基本原则和技巧就可能给你提供获得成功的特别机会。本书并未介绍什么新内容，它只是从另一个角度介绍了一些非常重要的内容。本书介绍如何将那些已经明确定义的软件工程原则应用于 LabVIEW。

我们并不希望通过速成手册、秘笈或者过时或超前版本的书籍学习 LabVIEW 编程；而是希望工程师们能够遵照基本的软件工程原则，使构建 LabVIEW 代码的工作变得简单。本书也不是为那些从不购买最重要的和实用的软件工程与设计方面的书籍的人而写。我想总结的并不是一种新的做事方法，而是正确的方法。在接下来的章节中，我们将介绍应用 LabVIEW 所完成项目中的一些实践经验。对我而言，这是一些成功的经验，我认为将要介绍的这些内容会对你有很大帮助。

本书的根本目的不是介绍某些抽象问题的提示或技巧，而是介绍在正确设计的基础上，构建 LabVIEW 代码的方法。如果你不喜欢这些，那么请放下本书走开，因为后面还有许多内容。正如标题所说，本书是一本使用了我自己惯于使用和思考的术语的手册，并希望能够总结出一套解决问题的方法。但是，我并没有自大到（有些人可能不同意）说明这是惟一的方法，或者是在任何情况下对做任何事都是最好的方法。我希望本书成为一项持续性的工作，能够搭建起那些对 LabVIEW 方面的软件工程发展有兴趣的人的沟通桥梁。我希望本书得到尽可能多的建议或意见，并以此推动 LabVIEW 软件构建方法的持续发展。

1.2　什么情况下不要购买本书

有时你会买一本列出了强烈推荐理由和用途的书。我们想反其道而行之，下面将列出一些理由，在其中任何情况出现时，你都不应该购买本书。

- 如果你从不阅读软件工程方面的书籍。
- 如果你对新的设计思想不感兴趣。
- 如果你没有任何软件工程方面的书籍。
- 即使很必要，你也不喜欢以团队方式工作。你将自己的代码藏起来，并宣称它们是你的私有财产。
- 你宣称自己有 8 年的编程经验，而且从来没人能够挑出你的毛病（事实上，你可能只有总共 2 年、分成 4 段的重复性工作经验）。
- 你认为所有其他有关 LabVIEW 的书籍都要比这本书好（这简直是笑话）。

上面的话虽然有一些调侃的意味，但我们的意思是说，如果我们到目前为止所说的你都不能认同的话，那么本书可能确实不适合你。

1.3　附 加 说 明

编写 LabVIEW 代码有哪些错误的方式？为什么没有多少资料来介绍如何将正确的软件工程方法应用于 LabVIEW？我们说，答案在于如何定位 LabVIEW。我们经常抱怨：有些人只乐意听到 LabVIEW 的问世是一个巨大的里程碑这样的说法。我们认为，LabVIEW 是一种功能非常丰富的、成熟的和健壮的软件开发环境。令人遗憾的是，LabVIEW 往往被看作是给那些几乎没有软件工程和编程经验的人使用的简单工具。

确实，使用 LabVIEW，简单的工作很容易完成，特别是与其他类似的语言，如 Visual Basic 相比，更是如此。但在快速数据获取（DAQ）应用程序以外，问题就出现了。为什么呢？因为大部分 LabVIEW 用户都没有多少软件工程或编程经验，结果导致了充其量只能算还凑合的应用程序。为什么有人会认为没有经验的人也能创建应用程序，这个问题并不在我们的讨论范围内。在我们看来，这和给那些完全没有建筑经验的人最先进有效的工具，然后让他们建房子的效果是一样的。利用那些工具当然会更容易一些，难道不是吗？不，当然不是！

那么，猜猜接下来是什么？编写软件并不轻松。我们想用一个给浴室墙面贴瓷砖的例子来比喻。如果要在墙上贴一块瓷砖，这项工作好像相当简单。好吧，如果要继续在

墙上贴瓷砖，并且要加上足够的粘合剂使其固定在墙上，这些工作好像全都很简单。但是，如果要贴 5 块瓷砖，就会明白这需要一些技巧。一个小小的错误都会影响到其他 4 块；所以必须考虑每块砖的沟缝；还需要确保每块砖和相邻的砖都很契合；如果墙面不平，还要用粘合剂补。如果仍然是 5 块瓷砖，这些问题还不会太明显。但是，如果墙面由 500 块瓷砖组成，那么要是第 1 块砖有一点偏差，那么第 500 块的偏差就可能很大。从这个例子可以看出，越庞杂的工作，需要考虑的因素就越多。

这个例子说明了为什么小型工作比大型工作更简单，因为开始犯的任何错误如果不能马上被制止就会被放大。比如，一个 ATE 计划通过通用接口总线（GPIB）连接示波器。但它没有使用常规接口，而是将发送给示波器的所有命令散发给应用程序的所有部分，任何时候，示波器哪怕只需要与 G 程序的某个小部分通信都是发散性的。现在，已经接近了开发工作的尾声，渐渐地，发现了频率测量功能的基本缺陷。频率将从 100~1 000Hz 改变到 10~2 000Hz 的新范围，如果这个范围数据在某一个组件中，并通过一套常规接口使用，那么修改工作就会很简单。但是，我们这里所说的软件并不是按照这种思想编写的，现在需要找出并修改所有涉及到这个范围数据的代码。一个小应用程序也许只有几处需要修改，但在大型应用程序中可能会有成千上万处需要修改。

这个例子非常简单，但它确实在本书的作者之一（Jon）所参与的项目中发生了。在经过几个月的修改工作后，我们发现还需要采取一些措施来修改这些数据，因为客户又打电话来要求另外修改。想想看，如果要一次又一次地修改会怎么样。如果整个 ATE 没有注意按照好的设计标准来写，那么很明显，每一次修改本身都会引起更多的错误最后导致混乱局面。听起来很熟悉吧？在许多项目中，这种情况或多或少都会出现，其实这是导致系统瓦解或难以维护的根本原因。

那么，有什么解决方法吗？奇怪的是，这个问题的答案非常简单。完全了解问题域、完善的设计以及结构良好的软件，能够保证应用程序健壮、易于更新和维护。所需要考虑的全部问题，就是能够按时提交程序，并且最好能够超出用户或客户的期望。

1.4 关 于 本 书

感谢 John Compton-Smith 的说明。

在两个在线 LabVIEW 社区中，都经常有成员寻求一些软件工程方面的资料，以便帮助他们改进工作。

一般的 LabVIEW 用户都没有接受过软件工程技术方面的训练。除非必须，他们一般也不会看软件工程或者质量控制方面的书籍。我们想向这些人着重介绍将主要的软件工程原则应用于实际工作的简单方法学。使用 LabVIEW 的面向组件设计的方法，是理解贯

穿本书的软件工程原则的基础。

本书并不像通常的软件工程书籍那样难懂。因为我们试图从分析细节问题的层面概括观点，这些都是软件用户实际需要的；除此以外本书没有更多内容。我们希望针对本主题提供实用的，而非学术性的方法。

有许多关于软件设计和软件工程的大部头书籍，这些书籍大多有助于改进你的软件制作模式，但对那些希望借此解决以后遇到的问题的工程师却不太合适。同样地，本书也没有针对初学者安排入门级章节，因为随 LabVIEW 软件提供的相关文档非常好，在此没有必要复述。

另一个重要观点是，尽管本书主要针对测试系统进行案例分析，但其中的方法学适用于用 LabVIEW 编写的任何软件。

1.5 参 考 站 点

如需获取完整的源代码或本书的更新内容，请访问参考站点：
http://authors.phptr.com/watts。

第 2 章　LabVIEW 的优点

2.1　LabVIEW 优点详述

又要引起争论了。我们认为，LabVIEW 是目前市场上最好的通用语言。

LabVIEW 在市场上的定位并不是如上面所说，而是为非专业程序员提供的测试和度量工具。即使最中庸的专家也会说 LabVIEW 对于某些工作来说是最好的工具。在我们的经验中，对于我们所做过的项目中的大部分工作来说，LabVIEW 都是最好的编程环境，反对这种观点的人相当少。其中一种意见是 LabVIEW 的鼠标定位控制不是很方便。当然，这很难成为阻碍项目进展的因素；我们已经找到了确实关心窗口准确性和装饰效果的客户。National Instruments（国家仪器公司，NI）正在通过现实的努力推动 LabVIEW 的应用。总有一天，LabVIEW 会打破其专业市场的界限，成为主流。

LabVIEW 为什么会这样好，这是个有争议的问题。我们花费了许多时间进行各种研究，并得出了一个结论，就是我们应当有自己的观点。很显然，我们对 LabVIEW 有浓厚的兴趣，那么，对那些对某种语言特别忠诚的程序员来说，又会怎样呢？我们的兴趣纯粹是关于生产率的。我们尝试过许多语言和方法，单就生产率而言，没有能够与 LabVIEW 媲美的。那么，LabVIEW 具备哪些其他语言没有的优点，我们所说的 LabVIEW 在生产率方面的优势又意味着什么？让我们首先来讨论生产率的问题。下面是一种语言的重要特性列表。如果某种语言的得分很高，那么就有理由认为这种语言是高生产率的。

可读性

在一个计算机屏幕上可以看到的问题范围是多大？在读懂源代码前需要了解多少知识？这种语言的原语的复杂度如何？（原语是基本的构建模块，如果基本结构和数据类型数量太多或者定义不清晰，那么这种语言就不好用。）程序的结构是否清晰？是否支持信息隐藏？

可写性

从设计到实现的转换过程能够有多快？是否可以将用户界面的设计与底层源代码分开？只修改很少的代码是否可以达到改变程序的目的？

是否可以将源代码分成许多块，以降低复杂度？这些代码的接口定义是否清晰？是

否可以将这些代码分出层次以降低复杂度？

可编辑性

编辑代码比重新创建要简单和快速得多。设计时面对一张空白的表格是最糟糕的事情。这种语言在模板的表单中提供了哪些内容？修改已有的代码是否困难？

可重用性

有多少工作需要从头开始？是否可以构建一个开发环境，这样就可以方便地将你自己编写的代码作为语言内嵌的功能模块重复使用？

可理解性

在将问题图形化方面这种语言提供了什么？如果客户想参与其中，那么他们能否理解大致的过程？

那么，LabVIEW 得分如何，我们的列表又是否完整呢？

1．可读性

LabVIEW 是分层结构，因此只要在一个屏幕显示其完整的顶层虚拟仪器即可。其基本原语和数据结构的数量很少（图形化特性有助于功能定位，帮助文档对学习也很有帮助）。纠错工具对程序运行也非常有效。

基本的数据结构（序列、条件、While 和 For 循环）让代码的结构清晰，这使你能够定义代码结构。如果你曾试图在无数没有缩进的 Pascal、C 或者 BASIC 程序中找出一个循环终点的话，你就会喜欢这个特性。

数据流程图也很易于理解。只要有常识经验的人就能读懂即使最难理解的程序。对 LabVIEW 的一个批评是认为在某些方面数据流程图具有某些限制。但这些限制有助于读懂程序。像 Visual Basic 这样的面向对象（OO）语言对问题的定义有太多的个性化解释，缺少约束规则意味着一个问题可能有 1 001 种解决办法。对这些语言来说，问题的解决办法不只是要找到答案，还要对其进行剪贴。

2．可写性

在与 LabVIEW 开发人员谈论他们喜欢 LabVIEW 的原因时，经常会提到的就是用 LabVIEW 编程非常有趣。这好像表明了 LabVIEW 在可写性方面得分很高。如果用 LabVIEW 来实现你的想法非常困难，你就不会觉得有趣。如果人们不能认同你的工作成绩，你也不会认为 LabVIEW 有趣。那么，这种感觉是不是源于能够将图标拖放到一个框图上？希望不是。我们希望这种感觉来自 LabVIEW 的图形设计特性。你可以创建