XP系列丛书

探索极限编程

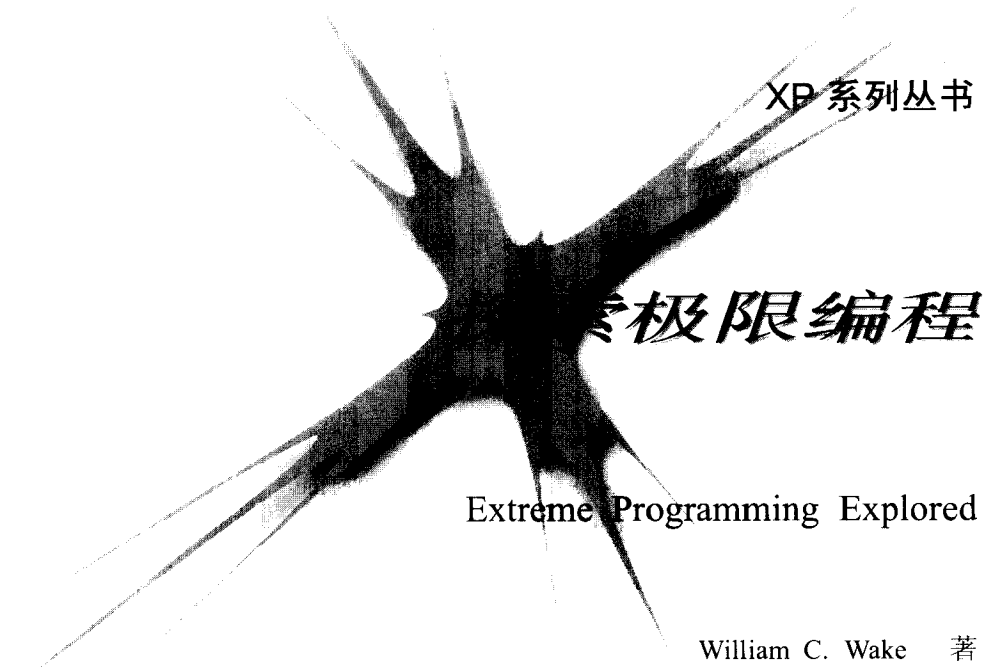
Extreme Programming Explored

William C. Wake 著
郑荣林 译
北京 SPIN 审校

Foreword by Dave Thomas

Pearson Education

XP 系列丛书



极限编程

Extreme Programming Explored

William C. Wake 著

郑荣林 译

北京 SPIN 审校

人民邮电出版社 Addison Wesley Pearson Education 出版集团

图书在版编目（CIP）数据

探索极限编程 /（美）韦克（Wake,W.C.），编；郑荣林译.
—北京：人民邮电出版社，2002.6
（XP 系列丛书）
ISBN 7-115-10383-6

Ⅰ. 探… Ⅱ. ①韦…②郑… Ⅲ. 软件开发—方法 Ⅳ. TP311.52

中国版本图书馆 CIP 数据核字（2002）第 043172 号

XP 系列丛书

探索极限编程

◆ 著 William C.Wake
译 郑荣林
审 校 北 京 SPIN
责任编辑 俞 彬

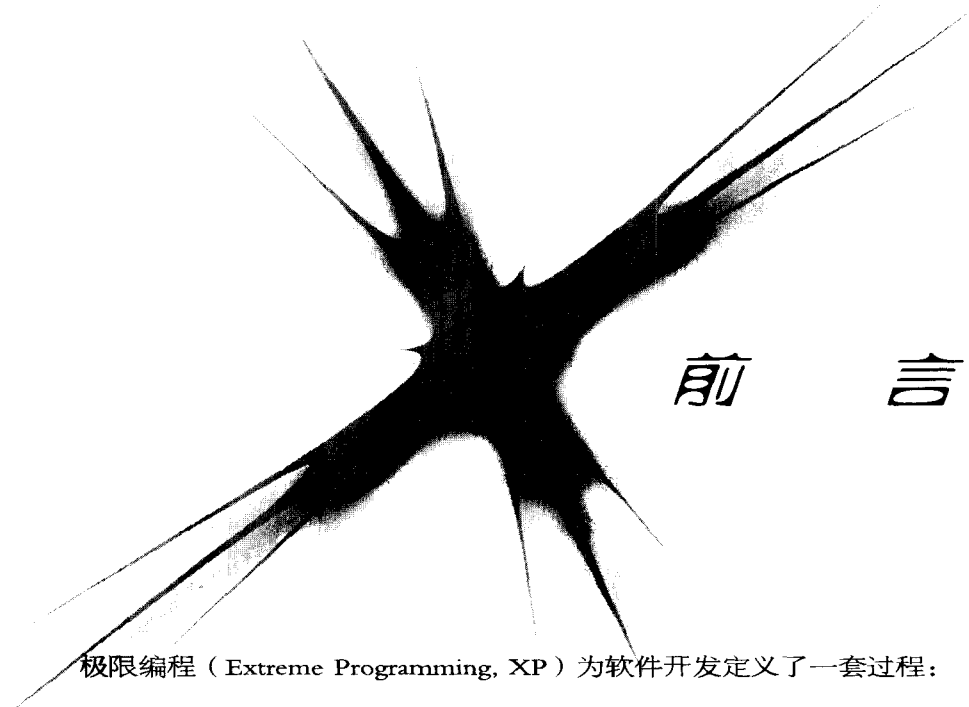
◆ 人民邮电出版社出版发行 北京市崇文区夕照寺街 14 号
邮编 100061 电子函件 315@ptpress.com.cn
网址 http://www. ptpress.com.cn
读者热线 010-67180876
北京汉魂图文设计有限公司制作
北京顺义振华印刷厂印刷
新华书店总店北京发行所经销

◆ 开本：800×1000 1/16
印张：11.25
字数：180 千字 2002 年 6 月第 1 版
印数：1-5 000 册 2002 年 6 月北京第 1 次印刷

著作权合同登记 图字：01 - 2001 - 4828 号

ISBN 7-115-10383-6/TP • 2934

定价：26.00 元



前　言

极限编程（Extreme Programming, XP）为软件开发定义了一套过程：它解决了从早期探索论证到多版本发布过程中的种种问题。

我们将全面深入探索 XP。

首先，XP 是一套编程规则。让我们先了解一下创新的实质："测试先行"是如何改变常规编程过程的？我们同时会讨论重构——XP 程序员改进他们代码的方法。

其次，XP 是创建高效团队的行之有效的团队规则。我们会把 XP 和其他的实践活动相比较，看看 XP 的团队是如何实践的。

最后，XP 是一套与客户一起工作的规则。XP 是一个针对计划和日常活动的详细而明确的过程。我们会看到一个团队是如何规划一个版本或迭代的进度，以及团队每天的具体活动。

一、您为什么选择阅读本书？

如果听说过关于 XP 的一些情况，您可能会怀疑 XP 各方面的机制或目的。本书将尽量捕捉我曾遇到过的问题，以及我所发现的解决办法。

我对 XP 的几个方面特别惊奇，特别是紧凑的测试先行编程（test-first

programming）周期（仅有几分钟），隐喻的使用，以及客户和程序员之间的明确分工。我们会关注这些方面以及其他的相关话题。

您同样会从本书中发现几个您感兴趣的方面：

- ✧ 比如Java和面向对象编程。本书的第一部分使用Java编程语言作为示例讨论了测试先行的编程和重构。程序员可能认为有关团队的实践活动也很有用，特别是关于隐喻和简单设计的理念。
- ✧ 从程序员、客户和管理者的不同视角探讨的极限编程。我们除了探讨有关 XP 的理论体系外，会从比其他更深的层次或不同的角度来探讨几个方面，特别是面向团队的实践活动、隐喻、计划过程和日常活动。
- ✧ 常规的软件过程。XP是最近几年提出的被称作敏捷、简单、自动适应的软件过程中的一种。通过对 XP 过程的深层次研究，我们可以更清晰地比较出 XP 与其他这些过程的异同。

二、作者是谁？为什么要写这本书？

本人是一位拥有 15 年编程经验的程序员，一半时间开发编译器，其他时间做图书馆、电信和金融领域的应用软件的开发和服务。

我第一次参加 XP 专业课程是在 1999 年 12 月。虽然我已经拜读过《解析极限编程——拥抱变化》和网上的大量有关 XP 资料，我还是怀疑测试先行的编程是否能够真正发挥作用（比我期望的更短的开发周期）。

测试用户界面的问题在课堂上被提出来；Kent Beck 说他通常不会使用测试先行的方法开发用户界面，但是他反过来问听课者，“你会采用吗？”这就为我撰写本话题的一篇文章带来了灵感。

我边学习边写，当我研究 XP 的不同话题时，我就写成了被称作

"Xplorations"的系列文章，并把它们放在网上。在同事的鼓励下，我精选这些文章中的一部分编成这本书，以便清楚连贯地解释有关 XP 的一些观点。

三、这本书的哲学思想是什么？

具体。使用实际的（或至少是实际的）实例。采用 Java 编写代码。

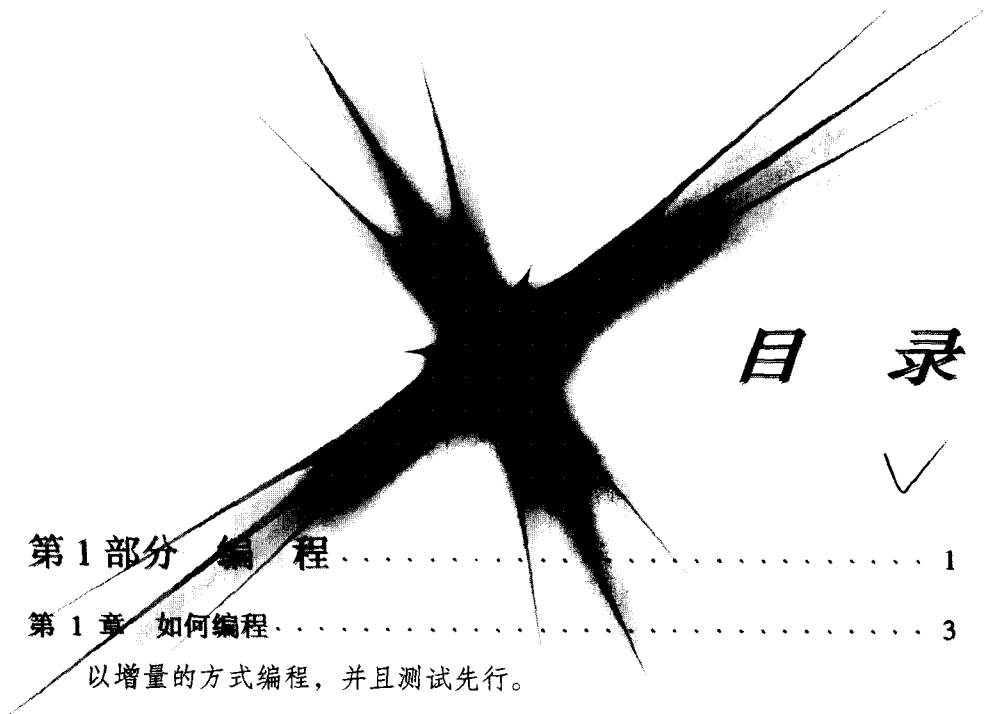
回答问题。由于本书的大部分章节是我在学习或教别人的时候写成的心得短文，所以每章都以一个问题提出和简短的回答开始。有几章还包含常见问题与解答）。

专注。每一章集中讨论一个话题，并尽可能与其他章节相联系。

准确而不拘泥。本书大量使用"我"、"我们"和"你"。在大部分中，"你"是指一个程序员，但是在某些章节，是指管理人员或客户。

经验共享。我把这些资料与宝贵经验进行了完美的融合。

William C. Wake



目 录

第1部分 编 程

第 1 章 如何编程

以增量的方式编程，并且测试先行。

采用增量方式编程的原因是，因为我们的编程周期非常紧张，甚至一次都不能编写一个完整的类，而只能是几行代码或某个方法。

采用测试先行（test-first）的原因是，因为我们先编写可重复使用的自动执行的单元测试，然后再编写使这些测试用例可以运行的代码。

这种方法有以下几点好处：

- ✧ 代码是可测的：它生来就是如此。
- ✧ 测试是可测的：如果还没有支持测试的代码，我们就知道测试准会失败；如果添写了要求的代码，测试就会通过。
- ✧ 测试是可重复的：它们以代码的形式出现。
- ✧ 测试有助于为代码编写文档：任何人想要察看对象如何工作或者要进行更改，都可以察看和运行测试。
- ✧ 设计最小化：我们用“正好够用的设计”来支持所编写的测试。

为了说明这一方法，我们将用测试先行的方法开发一个小型的书目管

理系统。我们会一起进行单元测试和简单设计。一小步一小步地进行编码过程，用正好够用的新代码使每个测试都能运行。按照一定的节奏，就像钟表的钟摆一样：测试一点，编写一点，再测试一点，再编写一点。

1.1 单元测试和 JUnit

单元测试是测试先行编程方法的关键环节。我们将使用 Kent Beck 和 Erich Gamma 开发的 JUnit 框架（适用于 Java），可以从 www.junit.org 网站获得，其他不同语言的测试框架可从 http://www.xprogramming.com 网站上获得。

下面是一个典型的单元测试模式：

- ✧ 创建几个对象。
- ✧ 调用一些影响这些对象的方法。
- ✧ 对结果状态进行一些声明。

通过 Junit，我们将测试嵌在一个实体型类中。下面是 Vector 的几个方法的测试示例：

```
import junit.framework.*;
public class TestVector extends TestCase {
  public TestVector(String name) {super(name);}

  protected void setUp() {
      // stuff to do before each test case
  }

  public void testAddElement() {
```

```
    //setup
  Vector v = new Vector();

    //call methods
    v.addElement("Some string");
    v.addElement("Another string");

      // asserts
    assertEquals(2, v.size());
  }

  public void testSomethingElse(){
      // another test
  }
}
```

TestRunner 类将调用 setUp()来运行这个测试，然后启动一个测试方法；TestRunner 类将再次调用 setUp()，然后是其他的测试方法。（每个测试可以按任何先后顺序进行。）如果声明的测试失败了，则将错误记入日志。

JUnit 还有其他几个特性；若需要更多信息，请参阅其他更多的文档资料。

1.2 设计

假如我们有包括作者、书名和出版年份的书目数据。我们的目标是编写一个系统，该系统可以查找我们指定的内容。我们设想的界面如图 1.1 所示。

我们将系统分成两部分：模块和用户界面。本章将演示如何在模块开发中采用测试先行的编程方式。如果想进一步了解如何在用户界面开发中

采用测试先行的方法，请参阅在本章结尾处的补充说明。

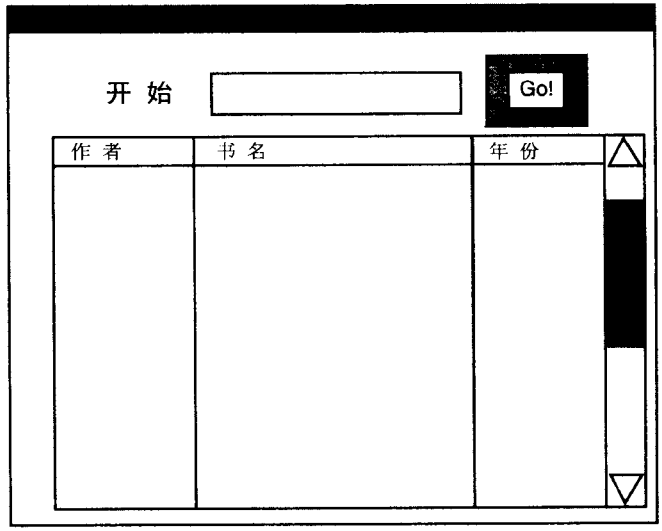


图 1.1 是查询系统的用户界面

我们先从一个简短的设计会议开始。首先，我们知道我们有一个 Documents 集合。Documents 包含它们的属性（作者、书名和出版年份）。

Searcher 类知道如何查找 Documents：对于每一个 Query，它将返回一个 Result（与 Documents 集合相匹配），如图 1.2 所示。

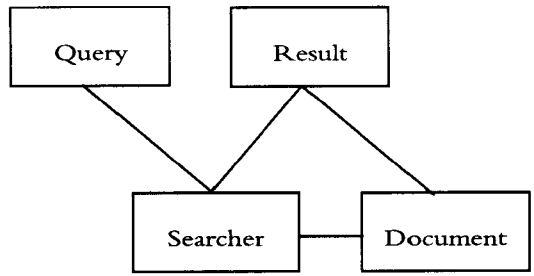


图 1.2 是四个主要对象

请注意，我们将设计放在几张卡片上。由于我们将在本章讨论隐喻，

所以我们的设计使用了“自然隐喻”：对象是基于域对象的。

在这个设计之后，我们将自下而上地创建单元测试类：Document、Result、Query 和 Searcher。（“自下至上”不是测试先行所必需的；它只是用来简化设计。）

1.3 Document

Document 需要知道其作者、书名和年份。我们将从创建“data bag”类开始，首先是它的测试。

```
public void testDocument(){
  Document d = new Document("a", "t", "y");
  assertEquals("a", d.getAuthor());
  assertEquals("t", d.getTitle());
  assertEquals("y", d.getYear());
}
```

我们还没有创建 Document 类，所以本测试不会编译，因此我们用 stubbed-out 方法创建类。

再次运行测试以确定测试确实失败。这看起来很可笑——难道我们不想通过测试吗？是的，我们是想通过测试。但是首先看到测试失败，我们才会对测试是否有效有一些把握。有时，测试出乎意料地通过了：“这太有趣了！”

最后，填上构造函数和方法使测试能够通过。

这一简化过程如图 1.3 所示。（我们将在下一章节中探讨重构。）

本过程将保证你既能看到测试失败也能看到测试通过，这就确保了该测试确实测试了某些东西，包括你进行更改所产生的效果，以及你所添加

的有价值的功能。

Kent Beck 和其他人会告诉你不用为设置器（setter）、获取器（getter）和构造函数专门编写测试：你只需要“测试可能失败的一切”（和那些不可能失败的）。Kent 说过：“编写过多的测试，就适得其反了。”

我在示例中使用了设置器和获取器，但我自己的风格却是偏离它们。我意识到采用大量的设置器和获取器可能是类没有尽职的一种迹象，并且可以更好地为类重新分配角色。

XP 中的测试/编码周期包括：

- ✧ **编写一个测试。**
- ✧ **编译该测试。应该不能通过编译，因为你还没有实现该测试所调用的代码。**
- ✧ **仅编写足以通过编译的代码。（如有必要，请首先进行重构。）**
- ✧ **运行该测试并看到它失败。**
- ✧ **增加足以保证测试通过的代码。**
- ✧ **运行该测试并看到测试通过。**
- ✧ **为清楚起见，通过重构来删除重复的代码。**
- ✧ **从第一步重复进行上述步骤。**

图 1.3　给出了 XP 的测试/编码周期

本周期要花多长时间？ 1 到 5 分钟，可能最多 10 分钟。

如果它花费的时间更长该怎么办？ 将测试变得更小一些。

5 分钟？是真的吗？ 是真的。

测试和代码之间的差别是不是有点太大了？不会的。例如，在 IBM 的 VisualAge for Java 中，你编写测试并点击“Save”保存。该编译环境给出

编译器错误的提示信息，因此需要你编写余下部分并点击“Save”保存。然后点击 JUnit （它正好处于运行状态）的“Run”按钮；该操作将重新加载新类并重新运行该测试，它们被显示为红色。然后编写真正的代码，“save”，并再次运行该测试直到他们被显示为绿色为止。

如果你计划任何时候都准备编写单元测试，那么你添加的错误代码将会成为负担，对这些代码进行测试将需要另花时间看到它们失败。即使你从命令行方式下工作，情况也不会如此糟糕，因为你可以一直打开编译器窗口。

*所有这些测试代码在以后的维护期间都会降低工作效率吗？如果业务发生了变化，就必须一起更新测试和代码。*不，测试实际上会在维护中提高效率，它们使你有信心面对业务的变化，因为你知道如果你某个地方做错了，测试会及时地给你警告。如果界面改变了，你必须同时改变测试，但这种改变不会很难。另外，你会发现测试先行的编程方法会使你的设计更可靠。

1.4 Result

Result 需要知道两件事情：它所包含的条目总数和 Document 列表。我们首先测试不包含任何条目的空结果。

```
public void testEmptyResult() {
  Result r = new Result();
  assertEquals(0, r.getCount());
}
```

创建 Result 类并建立 getCount()方法。在编码实现的时候，如果发现编译失败就添加“return 0”进行排除。请注意我们使用的最简单而又到位