这可能是因为提出 XP 的肯特·贝克（Kent Beck）自己就是程序员，他知道像他那样的程序员在什么样的环境中能有最好的表现。这种思想有点极端，但是凸显了作为脑力工作的软件开发的本质一面。

11.8 快速编写优秀软件的敏捷宣言

受 XP 的刺激，之后又出现了许多具有类似思想的轻量级迭代式开发流程。另外，还出现了将它们统一命名为**敏捷开发方法**进行推进的活动。由于这种开发方法重视省去无用的作业，快速编写优秀的软件，所以这样命名。

2001 年，这些开发方法的提倡者和推进者一起总结了**敏捷软件开发宣言**（Agile Manifesto）（图 11-4）。

敏捷软件开发宣言
(Manifesto for Agile Software Development)

我们一直在实践中探寻更好的软件开发方法，
在身体力行的同时也帮助他人。由此我们建立了如下价值观：

个体和互动 高于 流程和工具
工作的软件 高于 详尽的文档
客户合作 高于 合同谈判
响应变化 高于 遵循计划

也就是说，尽管右项有其自身的价值，
但我们更重视左项的价值。

Kent Beck
Alistair Cockburn
James Grenning
Ron Jeffries
Robert C. Martin
Jeff Sutherland
Mike Beedle
Ward Cunningham
Jim Highsmith
Jon Kern
Steve Mellor
Dave Thomas
Arie van Bennekum
Martin Fowler
Andrew Hunt
Brian Marick
Ken Schwaber

图 11-4 敏捷软件开发宣言
（引自 http://agilemanifesto.org/iso/zhchs/manifesto.html）

现在，“敏捷开发”一词作为以 XP 为代表的轻量级迭代式开发方法的总称使用。

11.9　支持敏捷开发的实践

在采用敏捷开发方法的情况下，从工程的初始阶段就开始编写最终成果的代码。另外，在工程期间，由于会对代码进行功能扩展和规格修改，所以需要一直确保代码质量。为了顺利推进敏捷开发，相关技术窍门被整理为了**实践**（实践方法）。

接下来，我们将介绍三种具有代表性的实践——测试驱动开发、重构和持续集成。它们的目的都是编写、改善并维持最终成果的代码。现在，这些实践已被广泛使用，并不局限于敏捷开发。

11.10　先编写测试代码，一边运行一边开发的测试驱动开发

测试驱动开发译自 Test Driven Development，简称为 TDD。

以前，提到编码，就是指编写代码进行编译。常见的做法是，在编写代码之前进行详细的设计，在编译之后考虑测试用例，进行单元测试。

TDD 中将详细设计、编码和单元测试汇总在一起，采用与以往相反的顺序进行推进，具体步骤如下所示。

<TDD 的作业步骤 >

① 编写测试代码。

② 编译通过。

③ 进行测试，确认失败情况。

④ 编写代码，使测试成功。

⑤ 去除重复的代码。

最开始是编写测试代码（①）。这一步骤将要开发的代码的规格和期待的动作形成为具体的形式。然后，为了编译通过（②），编写所需的最低限度的逻辑，并确认最开始编写的测试的失败情况（③）。到这里为止都是准备工作。④相当于传统的编码，这里编写代码主体，使测试成功。在代码正确运行之后，删除冗余的逻辑，改善代码，使其变整洁（⑤相当于下一节将要介绍的重构）。

在开发某个方法或类时，通常会以几分钟的周期，多次循环执行①到⑤的步骤，逐渐加入功能。

习惯传统编码方式的人可能会认为每次编写测试代码是一项冗余作业。不过，TDD 有很多好处。

最大的好处就是能够切实编写实际运行的代码。这是因为，为了完成步骤④，不仅要使编译通过，还需要使测试用例运行成功。反馈快也是一大好处。传统的做法是依次进行详细设计、编码和单元测试，如果详细设计时造成了缺陷，那么直到单元测试时发现，通常要经过很多天。而在 TDD 中，步骤①到⑤是以非常短的周期推进的，因此，设计和编码时造成的缺陷在几分钟之后就可以被发现并修改。TDD 这一开发方法也可以看作将以较小的迭代单位进行开发的敏捷开发思想引入到编码作业中。

根据 TDD 的步骤，我们也可以同时得到用于验证代码的测试代码。通过运行该测试代码，即使在以后的开发中修改规格或添加功能，也可以随时确认既有代码是否运行正确。为了简洁地编写测试代码，还需要提高模块的独立性，因此，这也有助于提高设计质量。只要测试代码写得通俗易懂，就还可以用作描述设计和外部规格的文档。由于 TDD 对提高编码作业的质量和生产率做出了很大贡献，所以得到了广泛使用，而不只是局限于敏捷开发。

11.11　在程序完成后改善运行代码的重构

重构（refactoring）是指不改变已完成的程序的外部规格，安全地改善其内部结构[①]。重构不只是一个概念，更是具体的技术窍门集。原版于 1999 年出版的《重构：改善既有代码的设计》[②]一书以模式的形式对重构的动机、具体步骤和示例代码进行了整理，可以说是该领域的经典书。例如，我们来看一下具有代表性的重构——提炼方法。

< 提炼方法的步骤 >

① 创建一个名称合适的新方法。
② 将提炼的逻辑复制到新方法中。
③ 为了使提炼的逻辑中用到的局部变量适应新方法，根据需要修改为参数或返回值。
④ 进行编译。
⑤ 修改原来的方法，来调用新方法。
⑥ 编译并测试。

像这样，作业内容写得非常详细。书中记载了 68 个改善项目（各个改善项目也被称为重构），其中记述了更加详细的步骤和示例代码。

在进行重构的情况下，为了保证代码的外部规格不发生改变，需要准备测试代码，在重构之前和之后运行一下。另外，即使有大规模的修改，也不要一下子全修改，而是逐步循环进行小的修改，这是窍门。这样就可以将发生错误时的回退控制在最小限度。通过持续进行重构，就可以将程序维持在容易理解的状态，即使不断添加功能或修改规格，代码质量也不会变差。

① 常用的字典中并没有收录“重构”（refactoring）一词。据说这个名称的来源是，将事后对混乱的代码进行整理使其变整洁的作业比作数学的因式分解（factoring），所以命名为 re-factoring。

② 具体请参考第 10 章章末的参考书籍。

在以前，“设计在编码之前进行，之后就不应该修改了”“不碰正确运行的代码”等都是常识。重构则与这些思想相反，但是人们逐渐意识到，为了在经过很长一段时间后对完成的代码进行修改以便继续使用，这是非常有用的实践。特别是，为了实践分阶段编写软件并发布的敏捷开发方法，这是必需的实践。

11.12 经常进行系统整合的持续集成

持续集成译自 Continuous Integration，简称为 CI。

在敏捷开发中，由于会以两周或一个月的较短单位频繁进行发布，所以代码需要保持随时可以交付的状态。另外，由于在整个工程期间会一直对代码进行修改，所以在之后混入缺陷的情况下，也需要能够立刻发现问题。

持续集成就是一种在将代码保持在随时能够交付的状态的同时维持代码质量的结构。准备一个名为 CI 服务器的专用机器环境，使用工具每隔几小时就自动执行一次从编译、构建到单元测试这一连串作业。

在不执行持续集成的情况下，模块的整合经常会成为大问题。即使各个成员分别负责的模块在自己的开发环境下能够正确运行，一旦整合到一起，也会因为接口不一致或动作环境不同而出现构建失败或异常结束等问题。最终，有时甚至要花费几天或者几周的时间才能够开始进行集成测试和系统测试。

持续集成能够防止发生这种情况。通过经常进行构建和测试，即使开发小组的某个成员造成了缺陷，我们也能够立刻发现问题并进行处理。

另外，该实践还能够提高成员的质量意识。如果提交错误的代码，导致构建和测试失败，就会给整个团队造成麻烦，因此，各个成员都会努力使构建和测试可以通过。以短的周期进行构建和测试，也有助于养成 TDD 或重构推荐的以较小的步骤进行开发和改善代码的习惯。

11.13　敏捷开发和 TDD 源于面向对象

读到这里，可能有的读者会有疑问："迭代式开发流程和敏捷开发实践与面向对象是什么关系呢?"

从结论来说，两者之间并无直接关系。开发流程和敏捷开发实践与表示集合论和职责分配的归纳整理法并无特别关系，也不是以类、多态和继承等编程语言的结构为前提的技术。

尽管如此，RUP 和 XP、重构和 TDD 通常都被认为是面向对象的一部分。这有两个原因：一是技术十分吻合，二是涉及的人员相同。下面我们来依次介绍一下。

第一个原因是迭代式开发流程和敏捷开发实践与面向对象十分吻合。

在开发环境比较落后的时代，软件的修改成本很高，因此，为了按期完成系统，在早期阶段就确定需求是十分重要的。而现在，随着以面向对象为代表的开发技术的发展、开发工具的完善和机器性能的提高等，编码和测试的生产率比以往有了很大提高。迭代式开发流程，尤其是以编码为中心的 XP 等可以说让这些开发技术和硬件的进步成为可能。

特别是 TDD 和重构都深受 OOP 的恩惠。TDD 中使用的 xUnit 单元测试框架[1]就是运用 OOP 结构编写的，重构的许多技术也都灵活运用了方法、实例变量和继承等 OOP 结构和设计模式。

第二个原因是提倡并推进迭代式开发流程和实践的人基本上都与面向对象有很深的关系。

正如前面介绍的那样，提出 RUP 的美国 Rational Software 公司（后被 IBM 公司收购）主营面向对象的咨询和软件开发业务，对 UML 的标准化也起到了主导性的作用。提倡并推进 XP、TDD 和重构等的许多技术人员都是 Smalltalk、Java 的先驱程序员，对设计模式和建模也做出了很大贡

① xUnit 是各种语言的单元测试框架的总称，Java 中是 Junit，.NET 中是 Nunit，Ruby 中是 RubyUnit。

献。参加敏捷软件开发宣言的许多技术人员也都从事面向对象的研究和实践。

虽然迭代式开发流程和敏捷开发实践与面向对象并无直接关系，但实际上有着很深的关联。敏捷开发和 TDD 可以说都源于面向对象。

11.14 不存在最好的开发流程

本章从传统的瀑布式开发流程介绍到近年来备受关注的敏捷开发方法，但是很遗憾，最好的开发流程并不存在。

前面我们介绍过，瀑布式开发流程存在在早期阶段确定需求比较困难，以及技术风险被发现得比较晚的问题。

迭代式开发流程的好处是在开发中途允许修改，比较灵活，但是工程管理非常困难。特别是在开始阶段基本确定了整体的开发范围的情况下，中途允许修改的灵活性是一把双刃剑。

RUP 提供方针和技术窍门，但由于以客户化为前提，所以它并不是仅按步骤进行推进就能完成系统的作业手册。

如果恰当应用 XP，就能提高成员干劲，快速编写出可运行的软件。但由于 XP 不是在最开始就设立整体目标，所以本质上并不适合软件开发的一般合同形式——一揽子合同。

像这样，在任何情况下都适用的万能开发流程是不存在的。因此，在实际的软件开发中，我们需要参考这些开发流程，考虑并实践适合具体工程情况的推进方法。说到底，开发流程只不过是为软件开发提供一些启发而已。

本质上，软件开发是人类共同进行的脑力工作。

虽然确定这一脑力工作的推进方式和成果形式很重要，但是并不是仅此就会一切顺利。在实际工程中，负责人如何带动成员，成员如何以整个小组的成功为目标而行动等，这些开发流程中无法表示的人的一面也非常重要。

今后，随着技术的进步，为了熟练运用该技术，也会有相应的开发流程被提出。另外，简单的工作会不断交给计算机来做。尽管如此，模糊的、无法形成具体步骤的脑力工作最后应该还是会留下来。正因为如此，软件开发才是一项有趣的工作。

深入学习的参考书籍

[1] Craig Larman. 敏捷迭代开发：管理者指南 [M]. 张晓坤，译. 北京：中国电力出版社，2004.

☆☆

该书采用条理清晰、简洁明快的讲解方法，为读者介绍了 Scrum、XP、UP（RUP 的子集）和 Evo 等 4 个迭代式开发方法的概要、实践、成果、推进方法及注意事项等。书中保持观点中立，因此读者可以充分地了解各个方法的特征。如果读者想了解敏捷开发和迭代式开发的概要，这本书再合适不过了。

[2] James Shore，Shane Warden. 敏捷开发的艺术 [M]. 王江平等，译. 北京：机械工业出版社，2009.

☆☆

该书详细讲解了敏捷开发的方法和实践（实践方法）。除了 TDD 和重构，还详细讲解了制定计划、进行敏捷开发的心理准备、与领域专家进行对话的方法、团队的组织战略及工作场所的搭建等课题。这是一本超过 400 页的大部头，但对于将要从事敏捷开发的人和想要掌握敏捷开发习惯的程序员来说，这是一本必备的手边书。

[3] Kent Beck. 测试驱动开发 [M]. 孙平平，张小龙，赵辉等，译. 北京：中国电力出版社，2004.

☆☆

该书讲解了测试驱动开发（TDD）的实际推进方法。该书以 Java 的 Money 对象和 Python 的 xUnit 为例，具体介绍了 TDD 的步骤和思想。

[4] Paul M. Duvall，Steve Matyas，Andrew Glover. 持续集成：软件质量改进和风险降低之道 [M]. 王海鹏，贾立群，译. 北京：机械工业出版社，2008.

☆☆

该书全面介绍了持续集成（CI）。除了 CI 的价值、CI 的原则和实践、构建和测试之外，还详细讲解了 CI 中应该包含的元素（数据库、各种测试、代码的质量评价及部署等）。

专栏

编程往事

过去不被允许的 XP

如果 XP（极限编程）出现在笔者年轻的时候，一定会被认为荒诞无稽，无人使用。

这是因为当时的开发环境太落后了，根本无法实践 XP。说到当时的开发环境，通常都是几十、几百名开发人员共用一台大型机，进行编译和测试作业。

因此，采用 COBOL 或汇编语言编写的一个程序编译起来通常都要花费几十秒到几分钟，花一晚上来构建整个应用程序也是常有的事。

也就是说，每当程序员按下执行编译的“发送”按钮，就必须在终端前静待几十秒。当开发到深夜时，在编译自己写的程序的空隙，都可以打个盹儿。

* * *

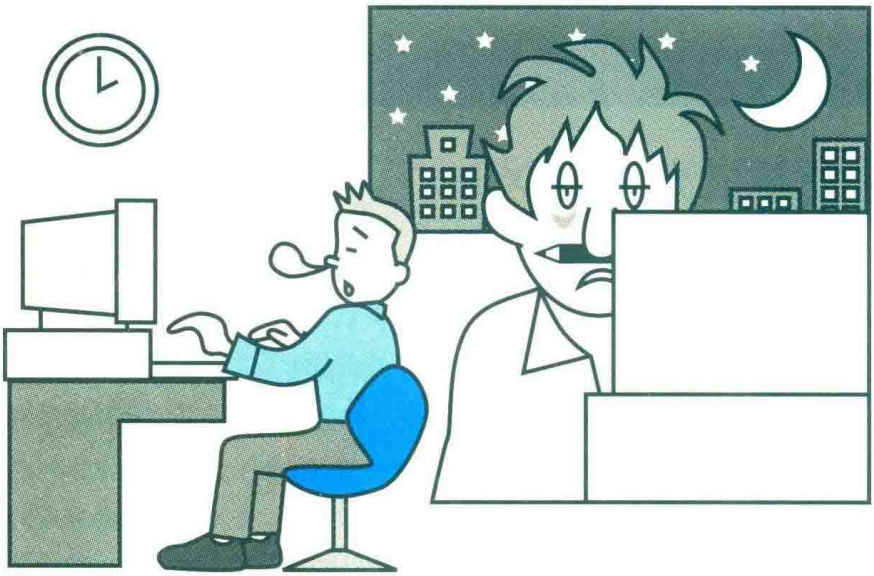
对比当时的情况，可以实践 XP 的现在真是轻松多了。换个角度来说，正是因为出现了使用方便的编译器和调试器等高级功能，具备了人手一台能够轻松运行这些功能的高性能计算机的开发环境，XP 等开发流程才成为现实。

作为在当时的开发环境中没有可行性的 XP 实践，大家首先想到的应该是重构吧。重构是对已经完成的程序的结构进行改善的技术。即使是简单的修改，为了防止出错，也要一点一点地修改代码，并每次都进行编译和单元测试。

而如果在当时的开发环境中这么做，现在 10 分钟就能结束的工作都要花费 1 小时还要多。

另外，不碰已经测试结束并发布的程序是当时的铁则。这不仅是因为编译和测试比较耗时，还是因为对于使用汇编语言编写的程序来说，稍微一点修改错误都会导致程序失控。因此，如果因内部结构不完美等而修改正确运行的程序，就会受到良心上的苛责。

结对编程也是不现实的。由于当



时大型机的终端非常昂贵，通常都由多人共享一台开发终端，所以，在重要的终端资源前坐太长时间是不被允许的。

正是因为这样的开发环境，所以尽量不使用计算机进行调试是当时比较推崇的做法。我们也经常被教导说，在 A3 大小的连续的纸张上打印出来的程序清单上标注红色，在大脑中跟踪逻辑，查找缺陷，将使用机器的时间控制在最短，这是优秀的 SE 应该做的事情。

在当时，如果两个人一整天都占着终端，一边吃零食一边编码，或者敲锣打鼓地大肆庆祝，这种如今人们已经习以为常的 XP 实践在当时一定会引起轩然大波。当事人一定会被辞退，再也别想回来了。

第12章 熟练掌握面向对象

热身问答

在阅读正文之前，请挑战一下下面的问题来热热身吧。

下面哪一项是对面向切面编程（Aspect Oriented Programming，AOP）的正确描述？

A. 以自顶向下的观点，将整个系统的功能按阶段进行分割、细化，最终导出程序结构的开发方法

B. 业务应用程序的规格反映在数据结构上，以数据结构的分析和设计为中心来构建系统的开发方法

C. 通过对分散在程序各个部分上的横切功能进行汇总来提高软件灵活性的方法

D. 作为以用户代理为中心自律地活动的智能集合来编写软件的方法

答案

C. 通过对分散在程序各个部分上的横切功能进行汇总来提高软件灵活性的方法

解析

面向切面作为面向对象的下一个趋势，曾受到诸多关注。

诸如日志记录和事务控制，有一些共同的处理对应用程序各个部分都起作用，在面向切面中，这种处理被称为横切关注点（crosscutting concerns）。

在使用 OOP 等如今广泛普及的编程语言的情况下，相当于这种横切关注点的逻辑分散在程序各处。面向切面编程通过将这种处理作为切面（aspect）独立进行描述，从而提高软件灵活性。

另外，其他选项所对应的技术如下所示。

A. 结构化分析与设计方法

B. 数据中心解决方案

D. 面向代理

本章重点

本章将进行一个简短的总结。

我们将再次复习一下面向对象的全貌，确认这是让软件开发变轻松的综合技术。面向对象是在软件开发技术的改良和研究过程中必然出现的，不会仅流行一时。即使今后作为下一代技术而受到关注的面向切面和面向代理等普及了，它们也都是面向对象的延伸。

面向对象除了能在实际工作中发挥作用之外，还会刺激人们的求知欲，是一门非常有趣的技术。因此，大家一定要熟练掌握面向对象，充分享受软件开发的乐趣。

12.1 面向对象这一强大概念是原动力

到目前为止，我们已经介绍了从 OOP 到可重用构件群、设计模式、UML、建模、设计和开发流程等面向对象结构中包含的各个技术。这些技术都很深奥，覆盖了软件开发的重要部分，关于这一点，大家应该都感受到了。

同时，我们还提到了面向对象并非替换了之前的开发技术，而是之前的优秀技术的延伸。OOP 是结构化语言的发展形式，UML 的用例图、活动图和状态机图采用了过去就在使用的图形表示。设计中的内聚度和耦合度也是在面向对象出现之前就已经被提出的旧思想，作为归纳整理法的面向对象就是数学中的集合论。另外，虽然本书中没有详细介绍，但是第 9 章介绍的建模与面向对象之前的结构化分析与设计方法、数据中心解决方案相比，本质部分并没有什么变化。

像这样，面向对象的一个特征就是吸收各种技术并不断扩展。

其原动力就是“面向对象”这一强大概念。通过将意为“面向物”“以物为中心”的“面向对象”概念赋予优秀的编程语言，其思想得到了大幅扩展，并被应用到软件开发的各个领域。另外，在被应用到开发现场时，