结构化语言的设计中基本上是听不到的。

使用 OOP 的类结构来汇总变量和子程序，能够提高软件构件的独立性，笔者认为这非常接近于担任某种职务的“某某主管”等的情形。

10.8　进行了职责分配的软件创建的奇妙世界

不过，现实世界和软件结构存在很大不同，因此，进行了职责分配的软件世界是脱离现实的奇妙世界。

我们来看一个例子。以银行存款的取款处理为例，我们将软件进行职责分配的情形套用在现实世界中，如图 10-7 所示。银行账户相当于钱包，所以请求“账户”对象进行取款，就相当于请求钱包“给我 3 万日元”。除此之外，将集中处理日期的“日历”类、打印交易结果的“存折”类等放在现实世界中来考虑，就相当于无生命的钱包和存折回答问题并执行被请求的工作一样。真是一个奇妙的世界！

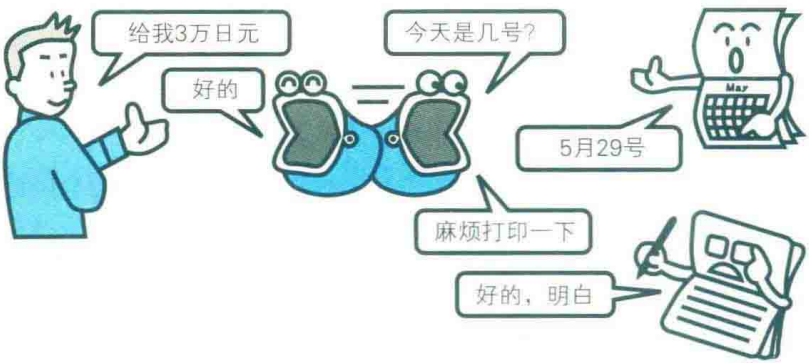

图 10-7　软件进行职责分配的奇妙世界

不过，在使用面向对象编写软件的情况下，无生命的事物（object）会承担职责，互相发送消息，从而完成整个工作[1]。通过这样进行设计，可以提高类和包的独立性，以及整个软件的可维护性和可重用性。

① 在实际的应用程序设计中，除了与现实世界中存在的事物相应的对象之外，还会定义很多软件专用的对象，来承担具体的职责，从而实现整体的功能。

本书在前面的章节中主张面向对象具有编程技术和归纳整理法两方面，我们应该将它们作为不同的内容进行考虑。而在设计阶段，归纳整理法和编程技术这两方面都必不可少。因此，可以说同时考虑这两方面正是面向对象设计的“感觉”。

另外，以笔者的经验，在采用 OOP 进行设计和编程的情况下，尽管头脑中清楚现实世界和软件结构是完全不同的，但仍会有它们是一样的错觉。

笔者认为其原因是，不管是现实世界中的人和组织，还是作为逻辑集合的软件，进行职责分配的大脑的用法都是一样的。

这或许也是大家接受“面向对象是将现实世界直接表示为软件的技术”这一说明的一个原因吧。

深入学习的参考书籍

[1] Robert. C. Martin. 敏捷软件开发：原则、模式与实践 [M]. 孙鸣，邓辉，译. 北京：人民邮电出版社，2008.

☆☆

虽然书名中带有“敏捷”一词，但是该书的主题是面向对象设计。该书基于丰富的示例代码和案例分析，详细介绍了为了恰当地设计类而应该遵循的各种设计原则（SRP、OCP 和 LSP 等），以及 GoF 设计模式、XP 实践等相关内容。虽然这是一本超过 670 页的大部头著作，但是对于想具体学习面向对象设计的技术窍门的人来说，该书是非常值得阅读的。

[2] Alan Shalloway，James R. Trott. 设计模式解析 [M]. 徐言声，译. 北京：人民邮电出版社，2006.

☆☆

该书结合 GoF 设计模式的应用示例，具体介绍了面向对象设计的相关内容，包括类的责任分配（职责分配）、内聚度和耦合度、接口与实现的分离，以及可变部分和共性部分的区分等。该书并非仅仅停留在抽象地介绍面向对象设计的“感觉”，而是对其进行了具体的讲解。

[3] Martin Fowler. 重构：改善既有代码的设计 [M]. 熊节，译. 北京：人民邮电出版社，2010.

☆☆☆

该书总结了对已完成的程序设计进行修改并改善其结构的技术窍门，对提高可维护性和可重用性，以及掌握面向对象设计的感觉也非常有用。对从事 OOP 设计和编程相关工作的技术人员来说，这可以说是一本必读的书。

[4] Joshua Kerievsky. 重构与模式 [M]. 杨光，刘基诚，译. 北京：人民邮电出版社，2006.

☆☆

该书主要讲解了灵活应用设计模式来改善既有代码设计的重构技术。书中没有一味地推荐应用设计模式，而是针对设计模式，按照“接近”“采用”和“避开”三种类型对各个重构技巧进行分类。该书能加深读者对设计模式的目的和适用情形的理解。

[5] Michael C. Feathers. 修改代码的艺术 [M]. 侯伯薇，译. 北京：机械工业出版社，2014.

☆☆

该书以质量较差的既有代码为对象，介绍了通过编写测试代码来改善设计的手法。通过阅读该书，读者可以了解到追求测试的便捷性对设计优秀的软件有很大帮助。“无测试的代码就是遗留代码”这一定义曾在业界掀起轩然大波，该书深入最基础的编程现场，为大家介绍极具实践性的技术窍门。

[6] G. J. Myers. Composite/Structured Design[M]. New York：Van Nostrand Reinhold Company，1978.

☆

在面向对象成为主流之前的 20 世纪 70 年代，为了便于维护和功能扩展，该书主张强化内聚度（该书中称为强度），降低耦合度。书中定义了内聚度评判标准的 7 个等级，以及耦合度评判标准的 6 个等级。评判内聚度的最优秀的标准——“信息强度”就是现在的面向对象的类结构，由此开始，面向对象逐渐成了主流的软件开发技术（另外，评判耦合度的最优秀的标准——“数据耦合”相当于函数式语言中的函数）。虽然书中的编程语言采用的是 PL/1，时代背景比较古老，但其深刻的洞察力直到现在也不逊色。

第11章 衍生：敏捷开发和TDD

本章的关键词

瀑布式开发、迭代式开发、迭代、RUP、XP、敏捷开发、TDD、重构、持续集成

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。

问题

关于“敏捷软件开发宣言”强调的四个价值，下面哪一项描述不正确？

A. 个体和互动高于流程和工具

B. 工作的软件高于详尽的文档

C. 客户合作高于合同谈判

D. 成员的动力高于遵循计划

D. 成员的动力高于遵循计划

解 析

敏捷开发是与传统的瀑布式开发相对的一个概念，是指通过不断进行较小的发布，阶段性地开发软件的软件开发方法。传统的开发流程着重于通过确定作业顺序和成果的形式，避免对特定人的依赖，按计划推进软件开发。而敏捷开发则着重于极力排除中间成果，重视顾客和成员的互动，灵活应对变化。

“敏捷软件开发宣言”由各种敏捷开发方法的提倡者和推进者总结而成，表明了敏捷开发的基本价值。

A~D 中最后一项是不正确的，正确内容是“响应变化高于遵循计划”。

本章重点

本章将介绍迭代式开发流程及其子集敏捷开发方法的概要和支持敏捷开发的具有代表性的实践（实践方法）。

首先介绍开发流程的两种类型，分别是过去广泛使用的瀑布式开发流程和近年来迅速普及的迭代式开发流程。然后介绍后一种流程中具有代表性的重量级流程 RUP（Rational Unified Process，统一软件开发过程）和轻量级流程 XP（eXtreme Programming，极限编程）。

通过表示重视人、工作的代码、客户合作和响应变化这一理念的“敏捷软件开发宣言”，以 XP 为首的轻量级迭代式开发流程作为敏捷开发方法开始被广为人知。另外，本章还将介绍用于顺利推进敏捷开发的具有代表性的三种实践（实践方法），即测试驱动开发（TDD）、重构和持续集成（CI）。

11.1 仅靠技术和技术窍门，软件开发并不会成功

到目前为止，我们分别介绍了面向对象中的各种技术。那么，要想成功地进行软件开发，仅靠这些技术就够了吗？

当然，并一定如此。

像企业的基础系统那样大规模的软件，要几十、几百人组成团队，耗费几个月甚至几年来开发。从工期、预算和投入的人数来看，这样大规模的软件开发是一项大工程，可以匹敌大厦的建设。即使没有这么大的规模，想要顺利推进软件开发工程并最终取得成功，也并不容易。

作为软件开发成果的程序是无形的，从某种程度上来说，只要未完成，就无法运行并确认结果。要编写出完美的程序来控制不会随机应变的计算机是一件很不容易的事，这在本书中已经提到过多次。另外，对于用二维图形来表示需求规格和源代码的 UML，如果不是专业人员，也无法判断其质量。

软件开发工作一般不会像事务性工作那么简单。与客户和成员开碰头会，将想法形成文档和程序，这就涵盖了沟通工作和脑力工作。可能稍微有些夸张，但如何顺利推进这种由人类共同进行的脑力工作，或许可以说是一个永恒的课题。

11.2　系统地汇总了作业步骤和成果的开发流程

为了顺利推进软件开发，对作业项目和步骤、成果的形式和开发成员的职责等进行系统的定义，从而形成**开发流程**。过去，开发流程一般都是计算机厂商和开发厂商制定的自己公司的标准。而近年来，以各个技术人员和社区为中心，将整个业界的优秀思想进行共享的活动变得活跃起来。

下面，我们将首先介绍一下开发流程的两种基本类型，即瀑布式开发流程和迭代式开发流程。然后介绍最具代表性的两种迭代式开发流程 RUP 和 XP，这两种迭代式开发流程可以形成鲜明对比。

11.3　限制修改的瀑布式开发流程

我们先来介绍一下瀑布式开发流程。

瀑布式开发流程是过去广泛使用的一种最具代表性的开发流程。正如“瀑布”一词所表示的那样，这种开发流程的目标是避免返工，像水流一样不断向前推进，因此才如此命名。

如图 11-1 所示，瀑布式开发流程中依次实施需求定义、设计、编码和测试等作业[1]。首先制定整体计划，然后在规定期限内实施各个作业，并对作业成果进行评审和认定，再进入下一个阶段。

① 第 9 章中介绍的业务分析由于是在软件开发之前实施的，所以通常不包含在软件开发流程中。

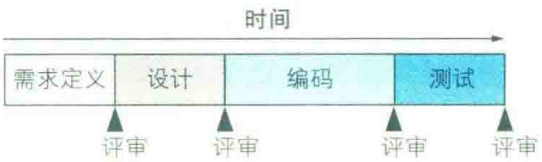

图 11-1 瀑布式开发流程

该瀑布式开发流程的基本前提是“软件的修改会产生很大的成本”。

软件本质上是很精细的。哪怕只有一行命令错误或者一个字符拼写错误，也会导致程序出错并结束运行。特别是在开发环境远不如现在的时代，软件的修改成本要比现在高很多。因此，即使是简单的规格变化，修改和测试程序的成本也很高，修改时引发新 bug 的风险也不可忽视，更不要说在开发进行过程中进行规格的大幅修改了。

因此，主张在一开始就确定需求，并将规格的修改控制在最小限度的瀑布式开发流程便应运而生。

另外，瀑布式开发流程中会编写大量的文档，这些文档是需求定义和设计的成果，其前提是“编写文档的成本要比编写软件低得多”。

虽然最近有不少人开始批判这种开发流程会编写大量没用的文档，无法跟上需求的变化，是一种过时的官僚主义的开发流程，但是瀑布式开发流程作为开发流程的一种基本形式，笔者还是希望大家能够好好理解一下。

11.4 瀑布式开发流程的极限

不过，瀑布式开发流程是存在极限的，这大致可以分为需求问题和技术问题两种。我们来分别介绍一下。

首先是需求问题。在瀑布式开发流程的最开始阶段要定义系统要求的所有功能，但这不一定能实现。

原因有很多。首先，人可能考虑得不够全面。另外，确定需求规格的人并不一定完全理解整个业务，有时意见不一致或者想法不同会造成规格说明存在矛盾。实际上，在运行已经完成的系统时，还会有新的想法，这

种情况并不少见。另外，系统的前提条件和目标也可能会受到外部因素的影响而发生变化。在企业系统中，如果业务目标和环境发生变化，也会影响对系统的需求。

因此，无论在最开始阶段多么充分地定义需求，之后往往也会不断地修改规格说明。

另一个是技术问题。在计算机领域，技术革新非常快，采用新技术和尚未被广泛使用的产品来构建系统的情况也不少见。一般来说，只有在实际编写程序并运行时，才能发现这些新技术和新产品的恰当用法和性能方面的问题。而在瀑布式开发流程中，编码和测试处于开发的后半阶段，因此，发现这些问题时也已经为时已晚了。如果存在大问题，那么开发工程的整个计划都会受到很大影响。

11.5　灵活响应变化的迭代式开发流程

为了应对上述问题，**迭代式开发流程**应运而生。在迭代式开发流程中，从需求定义到设计、编码和测试的一连串作业称为迭代（iteration）。迭代有“循环”“反复”之意。一次软件开发中会执行多次迭代（图 11-2）。

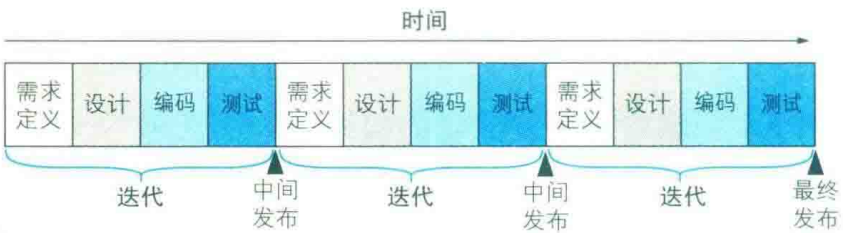

图 11-2　迭代式开发流程

采用这种开发方法，前面介绍的需求问题和技术问题就可以解决了。

首先是需求问题。迭代式开发流程不在一开始就确定所有的需求规格，而是阶段性地编写软件，进行多次中间发布，并每次都获取用户的反馈。通过像这样进行推进，就容易接受需求的变化。

另外，为了尽早发现技术问题，从工程的早期阶段就会编写并运行一部分实际的应用程序。即使发生严重问题，如果能够在早期发现，由于时间比较充裕，我们也容易提出合适的对策。

这种迭代式开发流程存在各种各样的方法，其中最具代表性的是 RUP 和 XP。虽然它们都是迭代式开发流程，但是基本思想却大不相同。下面，我们来分别介绍一下。

11.6 RUP 按时间分解和管理开发

RUP 是 Rational Unified Process 的缩写，译为“统一软件开发过程”。Rational 是“理性的”“合理的”的意思，而这里并不是一般的用语，而是指公司名称。该美国 Rational Software 公司[1] 最早提出 UML，并对其标准化做出了很大贡献。该公司开发、销售的商用开发流程就是 RUP。

RUP 详细定义了将整个计划划分为若干阶段的方针、各成员的职责、作业内容、成果形式和作业指南等，总结成了几百页的超文本。

RUP 的最大特征是以迭代式开发为前提，提供了将整个工程计划划分为 4 个阶段的方针。RUP 对这 4 个阶段分别进行了命名，并对其定位进行了说明。

- **初始 (inception) 阶段**
  定义系统的全貌。
- **细化 (elaboration) 阶段**
  对采用的技术进行评价，创建应用程序的基本结构。为此要提出一部分极其重要的功能，并实际编写应用程序。
- **构造 (construction) 阶段**
  开发应用程序的所有功能。建议按照应用程序功能的优先级顺序进行开发。

① 该公司在 2003 年被 IBM 公司收购，但之后仍作为产品名称使用。

- 交付（transition）阶段

  为了实际运行而进行必要的工作（综合测试、性能改善、操作培训等）。

这 4 个阶段是通过将整个工程按时间进行分割来管理进度的大框架。在实际工程中，需要根据应用程序需求、技术风险程度和开发成员的经验等来定义这 4 个阶段的具体工作。RUP 的 4 个阶段的应用示例如图 11-3 所示。

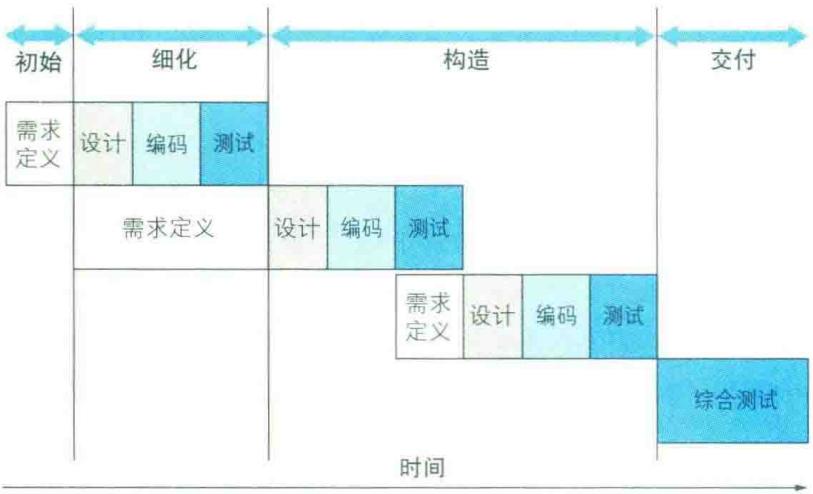

图 11-3　RUP 的 4 个阶段的时间分割

RUP 的另一个特征是以客户化为前提。也就是说，RUP 定义的作业项目和成果都是可选的，可以根据工程的性质和情况来选择所需的内容进行使用。

这种高自由度可以说是 RUP 的优势。这种灵活的思想在传统开发流程中是没有的。传统开发流程中定义的作业内容和成果大多是强制性的，其中不乏让工程管理者和开发成员感到死板、难用而敬而远之的东西。

RUP 开发流程在迭代式开发中加入了管理的观点，便于灵活运用。完整版 RUP 是一个包含大量文档的大型开发流程，可以说是过去优秀实践的

集大成者。而正因为其庞大，如果使用不熟练，就可能会淹没在其中定义的作业项目和成果的大海中。不过，如果是能够灵活运用软件开发流程的有经验的管理者，那么一定可以从中获得很多启发吧。

11.7 打破诸多限制的 XP

XP 是 eXtreme Programming 的缩写，extreme 是“极限的”“极端的”的意思，所以 XP 译为“极限编程”。XP 曾在技术人员之间引起了诸多关注。

XP 与 RUP 一样，也是一种迭代式开发流程[1]，但与 RUP 不同的是，它基本上没有定义形式上的作业步骤和文档形式的成果，而是定义了“4 个价值”的基本理念和实践 XP 的“12 个实践（实践方法）”。

<XP 的 4 个价值 >

- 沟通
- 反馈
- 简单
- 勇气

从这 4 个价值中可以看出，XP 是一种很有特色的开发流程。听到“沟通”“简单”“反馈”“勇气”这些词语，肯定很多人都想不到这是关于软件开发流程的话题。这 4 个价值的具体含义分别如下所示。

- **沟通（communication）**
  重视与小组成员和客户的沟通。
- **简单（simplicity）**
  不拘泥于设计，鼓励从最简单的解决方式入手并不断进行重构，以实现在需要修改时能够随时改善既有程序的内部结构。
- **反馈（feedback）**

① 是否将 XP 归属于开发流程还存在争论，但这里我们将其作为一种开发流程进行讲解。

立即运行编写的程序进行测试，并反馈测试结果进行改善。

- **勇气（courage）**

  需要时有勇气改变设计。

<XP 的 12 个实践 >

- 计划博弈
- 隐喻
- 测试
- 结对编程
- 持续集成
- 现场客户
- 小型发布
- 简单设计
- 重构
- 代码集体所有
- 每周 40 小时工作制
- 编码规范

“12 个实践”[1] 中的很多内容按照以往的常识来说都是禁忌。例如，两个人共享一台计算机进行编程的“结对编程”、之后重新改善已完成的程序的内部结构的“重构”、客户常驻开发小组一起工作的“现场客户”等。

在过去，提到开发流程，人们就会联想到形式主义，重视文档，而 XP 则打破了人们的这一固有印象。从这一点来看，XP 与包括 RUP 在内的传统开发流程形成了鲜明对比。另外，XP 受到了很多一线开发人员，尤其是程序员的支持。这种开发流程从“成员的观点”来看软件开发，如此受到一线开发人员大力支持的开发流程是未曾有过的。

而 XP 之前的开发流程是从“管理者的观点”来看软件开发的。传统开发流程的目标是将作业步骤和成果的形式标准化，排除对特定人的依赖，像生产工业产品一样编写软件，并将成员作为资源进行处理。

XP 的立场则完全不同。XP 重视成员的干劲和沟通。为了快速、有效地实现最终成果，并轻松对应变化，XP 省略了无用的作业和中间成果。“如果单元测试进展顺利，大家就去庆祝一下”“一边吃零食一边编码”等，与之前死板的开发流程有着 180 度的转变。

① 《解析极限编程：拥抱变化》一书中的定义。XP 的实践后来被多次修订。