

十年磨剑——CSDN原软件工程大版主心血凝聚之作
新版再现——理论与实践的密切结合

软件工程 之全程建模实现

青 润 著

- 延续实用风格，扩展最新研究成果
- 直击建模过程中的用户问题，有的放矢
- 建模工具rose、EA和trufun组合出击，突出重点
- 配有讲座视频，指导性强
- 随书附赠丰富学习资源



机械工业出版社
China Machine Press



软件工程之全程建模实现

本书主要介绍如何使用UML建模方法来实现软件工程的主要过程，包括需求、分析、设计、代码导出、设计模型维护等，对协作开发团队开发所要求的必备知识也进行了详细的解释。作者在书中采用国内实际软件工程中的大量截图，通过图形和示例描述工程中UML的实际应用，深入解析了软件开发中的各种难题。

书中对UML进行了深入分析和探讨，可以帮助急需学习和了解UML语言如何应用于实践的软件开发人员。

书中也针对软件项目管理者 and 软件企业的负责人进行了与软件项目市场相关内容的分析和软件开发心理学内容的补充，深入剖析了问题根源，答疑解惑。

本书适合有UML基础知识，并且实际参加过一些工程项目开发又对UML实际应用或者全程建模实现感兴趣的人员阅读，也适合作为软件工程培训机构的辅导材料。



本书附带光盘内容：

- 书中示例——便于读者亲手操作
- 全程建模培训视频——视频指导，建模无忧
- 学习指南——实现过程指导，指点迷津
- 讲座PPT——汇聚精华，拓展思维

客服热线：(010) 88378991, 88361066
购书热线：(010) 68326294, 88379649, 68995259
投稿热线：(010) 88379604
读者信箱：hzsj@hzbook.com

华章网站 <http://www.hzbook.com>



网上购书：www.china-pub.com

定价：49.00元（附光盘）

软件工程 之全程建模实现

青 润 著



机械工业出版社
China Machine Press



本书采用 UML 建模实现了软件工程的主要过程：需求、分析、设计、代码导出、设计模型维护等。本书采用了大量国内实际工程软件过程中的截图，通过图形和示例来描述工程实际中的问题和过程。这在国内原创的软件工程类书籍中是很少见的。本书对软件工程领域进行了重新划分，其中涉及软件工程方法、软件工程过程，以及软件工程理论的发展和现状，还包括全新的软件组织方法论：交换编程、可度量绩效管理模型、实用知识库体系构建，同时对软件开发中心态对抗问题进行了初步研究。

本书适合于对 UML 基础知识有一定了解的、同时参加过一些实际工程项目开发的人员阅读，也适合对全程建模过程实现和人性化软件开发感兴趣的读者参考。

封底无防伪标均为盗版

版权所有，侵权必究

本书法律顾问 北京市展达律师事务所

图书在版编目 (CIP) 数据

软件工程之全程建模实现/青润著. —北京：机械工业出版社，2010.8

ISBN 978-7-111-30790-7

I. 软… II. 青… III. 软件工程—系统建模 IV. TP311.5

中国版本图书馆 CIP 数据核字 (2010) 第 128901 号

机械工业出版社 (北京市西城区百万庄大街 22 号 邮政编码 100037)

责任编辑：秦健

北京市荣盛彩色印刷有限公司印刷

2010 年 10 月第 1 版第 1 次印刷

186mm × 240mm · 22.25 印张

标准书号：ISBN 978-7-111-30790-7

ISBN 978-7-89451-533-9 (光盘)

定价：49.00 元 (附光盘)

凡购本书，如有缺页、倒页、脱页，由本社发行部调换

客服热线：(010) 88378991；88361066

购书热线：(010) 68326294；88379649；68995259

投稿热线：(010) 88379604

读者信箱：hzjsj@hzbook.com



Preface 前言

写作经过

本书的撰写过程整整耗费了8年多的时间。回想起来,初次产生写本书的想法是在2002年的4月,当时笔者正在上海做一个企业ERP项目。这个项目完成后,笔者就觉得实验没有完全成功,因为笔者只是亲自验证了需求部分的实践,而没有时间来完整地验证分析、设计和编码的后续过程,与其他人配合开发的过程,以及将一个模型任务分配给其他程序员的实现过程。直到2002年的7月,笔者有机会到南京参加了一个大型信息系统实用性原型的开发,从而顺利地完成了分析、设计和编码过程的实践。同时,还完成了一本针对高级用户进行培训的开发过程教材。

经过笔者一段时间对教程资料的整理,以及在2001年中完成的几个项目的经历,笔者最终完成了本书现有的第3章(分析设计)和第4章(代码模型一致性)内容的第一次成稿。当时命名为《基于Rose的全程建模实例》,并将前几部分发表于《程序员》杂志2002年第10期~第12期。后来因为全文较长,CSDN^①就将全文直接合发于《程序员》杂志2002年合订本。

该文发表后,笔者来到北京进行一个大型项目的需求调研工作。期间,笔者与Wonder、Ozzzzzz和Lynx几位同仁从上午10点开始一直到晚上7点多进行了一次长谈。除了讨论软件工程过程的相关内容外,笔者还和Lynx专门针对全程建模过程中从设计阶段到编码阶段可能遇到的问题进行了相当长时间的讨论,我们的观点略有不同。笔者认为:设计可以有效地转换成代码,实现软件工程的模式,实现设计与编码的分离;Lynx兄则认为:在目前条件下,设计与编码不可能分开。不过,他认

① 中国软件开发网,英文全称为China Software Developer Net(CSDN)。国内一个著名的开发者论坛网站。

可了在设计人员水平足够高的情况下,是可以实现设计与编码分离的。但是,实现这样的分离对设计人员的要求可能会过高,而在国内这种高水平的设计人员目前还十分稀缺。

《基于 Rose 的全程建模实例》全文发表之后,笔者开始撰写《基于 Rose 的全程建模之二——Rose 中的协作开发》一文。当时,只是打算把 Rose 中提供的协作开发方式和配置管理工具的配合使用方式全部写出来,但是,随着公司的软件在全国范围内的铺开,笔者发现自己无法将三种主要的配置管理工具的环境都搭建起来,不得不放弃了对配置管理工具部分的撰写,最终就只剩下本书第 5 章(协作开发)部分的内容了。

在撰写《基于 Rose 的全程建模之二——Rose 中的协作开发》的同时,笔者开始规划《基于 Rose 的全程建模之三——如何进行需求分析》部分的撰写。笔者于 2002 年 10 月开始调研中国电信集团公司的业务需求,并将调研过程中总结的方法和内容写入这篇文章中。这部分内容撰写的时间最长。在笔者第一次到拉萨出差时,借着期间等待合同小签的空闲时间完成了这部分的主要内容,回北京后完成了全部内容。最后形成了本书的第 2 章(需求工程)部分的内容。

由于笔者第一篇文章的内容获得不错的反响,而第二篇文章的内容较多,CSDN 的孟迎霞认为发表在杂志上不是很合适,于是建议笔者将这些内容全部写出来然后出一本书。因此,笔者在 2003 年 3 月决定将所有的文章合并起来作为一本书来考虑。

2003 年 7 月下旬,在完成了需求阶段的内容后,笔者开始整理全书的基础目录部分,于 7 月底成功地完成了全书的第一次整合。

凭着笔者在电信集团公司不错的人缘关系,在项目合同基本全部签订之后,笔者开始在集团公司负责实施、处理 5 个开发商之间的系统接口和与集团系统的上报下发关系,同时开始规划系统第二个版本(BI 版本)的功能。笔者在电信集团公司停留的前两个月,在空闲时间不断地对全书的内容进行修订,经过对书稿的 9 次打印稿的纸面修改,最终在 9 月初完成了对全书的一次整体修订,并向出版社交付了全书初稿。

本书在 2004 年 5 月的第 1 版印刷后,笔者从电信四川设计院辞职,此后分别在一家外企和中科院自动化所工作了几年,期间独立对外提供技术咨询和培训服务。

其中 2006 年元旦的培训中，很多朋友提供了他们的建议和考虑，尤其是来自山东的朋友满桂平自从 2004 年 7 月看过我的书后在后续的几年中不断给笔者提供了他的实践信息和补充材料。

第 2 版中融合了 2004 年~2010 年初笔者所总结的除了已经采集了数据还没有进行文章撰写以外的几乎全部方法。其中包括：软件开发心理学的 5 章的内容，其中 2 章是首次对外发布；软件组织方法论的 4 章的内容，其中实用知识库体系构建一章也是首次发布；在技巧和相关文档一章中，删除了已经过时的对 Rational 在 IBM 中的顾虑、关于 Rose 与 Together 的 2 章内容，补充了项目计划如何做、评审如何做、配置管理工具如何使用以及 Use Case 阐述模型化问题及延伸的内容，同时给读者提供了一个最简化开发过程模型的建议——因为在培训和咨询中发现很多企业和院校都有人提到过，不知道什么样的过程是最简化的，哪些是软件开发中可以省略的，哪些是必不可少的。

同时，在这个版本的撰写过程中，还单独增加了软件工程概论一章，以期从整个软件工程的整体层面进行细分，阐述在新技术和环境下软件工程自身形态与过去的区别和变化过程，给读者一个关于软件工程的整体印象，同时对现有的传统软件工程体系进行了一系列的反思并建议了一个初步的针对整个软件工程体系的划分方式。

针对第 1 版中已经存在的需求工程、分析设计、代码模型一致性和协作开发 4 个核心章都进行了修订，尤其是对需求工程进行了较大篇幅的修订，将需求代码间的映射关系进行了着重的补充，这也是这几年针对整个方法论中的最重要的补充之一。

本书第 2 版的积累长达 6 年多的时间，实际整合和撰写时间长达一年半（可以参看本书修订历史记录信息），但是其中肯定还存在很多不足之处，也希望这些年从事实践全程建模方法论的朋友们能够不吝赐教，以便于后续的补充和修订，笔者希望下次本书修订再版的时候，您也会出现在感谢名单中或者出现在作者一栏中。

本书特色

本书中所有的示例都来自 2001 年初~2009 年间笔者经历的、采用 UML 进行建模支持的 3 个大中型工程软件项目和近 10 个中小型工程软件项目。通过十多年来实际工程项目的经验总结，本书采用 UML 建模方式进行软件的开发，覆盖了软件工程从需求开始，经过分析（概要设计）、设计（详细设计）、编码的导出，以及

设计模型的维护、协作开发等所有主要阶段的内容，本书还包括最近4~5年内参加培训和咨询的学员们提供的各种改进建议和方法。

本书中的所有例子都源于国内实际的软件工程项目，因此具有很高的现实意义和参考价值。书中的示例和操作技巧，大部分都是第一次公开，其中小部分内容曾经在CSDN的软件工程版块出现过，但是在CSDN上公开的部分都无法张贴示例图，所以，无法形象地描述清楚，本书的撰写也为此提供了一次完整解释的机会。

本书中关于软件开发方法论以外的大部分内容都曾以对话或者单独成文的方式分别发表于《程序员》杂志、IT168/ITPUB技术社区，最后都曾整理发布于青润的两个主要blog空间，分别是：

青润心情 <http://blog.csdn.net/qingrun>

青润风度 <http://space.itpub.net/257598/>

青润的其他blog空间都不是主要空间，内容更新会有所不同，而上述两个空间将同步更新所有最新内容，2008年4月以前的内容基本上都在青润心情发布。

由于篇幅所限，本书中没有对UML的基础知识和相关符号进行更多的解释和说明。关于这部分内容，希望读者能主动学习一些UML的基本知识后再来学习本书中的内容。

本书的组织结构

第1章（软件工程概论）从软件工程的总体上讨论软件工程知识的细分问题，其中涉及软件工程方法、软件工程过程，以及软件工程理论的发展和现状。

第2章（需求工程）详细地阐述实现目标所需要进行的前期需求工程的准备，以及需求调研过程中经常遇到和需要注意的一些问题。需求工程的内容可以协助开发人员顺利完成需求调研工作，让用户和开发人员都能得到一个满意的结果。

第3章（分析设计）和第4章（代码模型一致性），详细阐述笔者所持的关于设计过程与编码实施过程可以实现完全分离开发的观点。

第5章（协作开发）介绍了一些关于团队开发过程中涉及的内容。如协作开发步骤、多人协作开发的方式和文档模型结合等第1版中已经发布过的内容。

第6章（软件组织方法论）是本书的一个尝试性的全新扩展，包括了自2005年起发布过的交换编程和可度量绩效管理模型等内容，还有从未发布过的实用性知

识库体系构建方面的知识，这些都是软件团队组织形态方面的内容，为整个软件工程理论提供了支撑。

第7章（软件开发中的心态对抗问题研究），如果提升到一定层面那就是软件开发心理学问题。这一章可以看做是软件开发心理学部分的雏形篇章。由于作者本人的经历和精力有限，在这个方向上的研究还比较薄弱，目前只能完成这仅有的5个方面的分析，还有至少两类角色的心态研究内容正处于撰写中，已经完成了市场人员心态和游戏玩家心态的数据积累，但是还不能完成这方面全面的文字阐述。

第8章（技巧和相关文档）是为本书中实现团队开发所需的一些必备资料或技巧。这些文档和资料将协助你和团队实现更快速更高效的建模开发过程。

致谢

本书撰写过程中，尤其是在第1版的撰写过程中，很多朋友对笔者提供了无私的帮助和支持。书中的很多内容也来源于和这些朋友的对话、讨论、争执，甚至部分内容就是笔者偷师学艺所得。本不敢藏私，但在第1版出版的过程中由于过度仓促，没有能够在书中对这些朋友表示感谢，在这里一并对下面的朋友表示感谢：

感谢在托普工作期间，与笔者一同为了项目而拼命工作的成都中央研究院和上海研究院的朋友们。

感谢在中国电信集团网络发展部工作的朋友和网发部的各位领导，以及曾经在2002年~2004年与笔者一同借调到中国电信集团网络发展部工作的各省电信公司以及几大电信设计院的朋友们。这里就不一一提名致谢了。

感谢2001年~2003年在CSDN软工/管理板块上与笔者讨论的朋友们，他们是：吴晓军（Wonder）、马仲男（Mach）、刘寅琥（Lynx）、杨威（Ltp3）、刘新生（Ozzzzzz）、周永丽（Rose）、周海峰等。还要感谢孟岩、蒋涛、孟迎霞等CSDN的老朋友们。如果不是孟岩兄2004年初推心置腹的一番话，估计青润这辈子都不会写书了。

另外，还要感谢2005年后和青润合作的曾在中科院自动化所一起工作和学习的朋友们。

最后，感谢我的妻子朱丹女士在青润没有什么收入的几年中给我的工作和理想的默默支持和奉献。

目 录 Contents

前言

第1章 软件工程概论/1

1.1 概述/2

1.1.1 全程建模下的软件工程分类/2

1.1.2 传统软件工程分类/3

1.1.3 SWBOOK2004 中软件工程体系划分/5

1.2 软件工程过程/7

1.2.1 软件工程管理过程/7

1.2.2 软件工程实施过程/8

1.3 软件工程方法/8

1.3.1 软件开发方法论/9

1.3.2 软件组织方法论/14

第2章 需求工程/16

2.1 需求的定义/17

2.2 需求调研/18

2.2.1 调研中存在的问题/18

2.2.2 过去和现在的对比/21

2.2.3 新现象的分析/22

2.2.4 软件外包与编码/23

2.3 如何和用户交流/23



- 2.3.1 交流四要点/23
- 2.3.2 什么是用户/27
- 2.3.3 参与人员建议/30
- 2.3.4 输入输出/30
- 2.3.5 UML 元素/30
- 2.3.6 常见问题/31
- 2.4 业务建模/31
 - 2.4.1 目的/31
 - 2.4.2 基本操作步骤/33
 - 2.4.3 建议/34
 - 2.4.4 参与人员建议/35
 - 2.4.5 输入输出/35
 - 2.4.6 UML 元素/36
 - 2.4.7 常见问题/36
 - 2.4.8 示例说明/38
- 2.5 Use Case 模型/39
 - 2.5.1 Use Case 图的绘制/39
 - 2.5.2 如何查找 Use Case 和 Actor/43
 - 2.5.3 Use Case 的处理/49
 - 2.5.4 如何分包/52
 - 2.5.5 参与人员建议/59
 - 2.5.6 输入输出/59
 - 2.5.7 UML 元素/60
 - 2.5.8 常见问题/60
 - 2.5.9 楚凡 UML 工具模型示例/67
- 2.6 Use Case 阐述/72
 - 2.6.1 目的/72
 - 2.6.2 要求/73
 - 2.6.3 Use Case 阐述的格式/73
 - 2.6.4 参与人员建议/75



- 2.6.5 输入输出/75
- 2.6.6 UML 元素/76
- 2.6.7 例子/76
- 2.6.8 常见问题/78
- 2.7 交互建模/82
 - 2.7.1 目的/82
 - 2.7.2 要求/82
 - 2.7.3 例子/83
 - 2.7.4 参与人员建议/86
 - 2.7.5 输入输出/86
 - 2.7.6 UML 元素/86
 - 2.7.7 常见问题/87
- 2.8 界面设计/88
 - 2.8.1 什么是界面设计/88
 - 2.8.2 界面设计的原则/89
 - 2.8.3 RUP 中的相关描述/89
 - 2.8.4 界面设计的基本要求/90
 - 2.8.5 界面设计的步骤/91
 - 2.8.6 界面设计的定位/92
 - 2.8.7 界面设计的时间特点/92
 - 2.8.8 界面设计与原型法/93
 - 2.8.9 需要注意的问题/93
 - 2.8.10 常见问题分析/95
 - 2.8.11 参与人员建议/95
 - 2.8.12 输入输出/96
 - 2.8.13 UML 元素/96

第3章 分析设计/97

- 3.1 概述/98
 - 3.1.1 原因/98



- 3.1.2 两个项目简介/98
- 3.1.3 初步结论/99
- 3.2 分析设计的定义/99
 - 3.2.1 什么是分析/99
 - 3.2.2 什么是设计/100
- 3.3 分析模型/101
 - 3.3.1 模式的选择与应用/101
 - 3.3.2 构建分析类/102
 - 3.3.3 分析模型时序图/110
 - 3.3.4 参与人员建议/113
 - 3.3.5 输入输出/113
 - 3.3.6 UML 元素/114
 - 3.3.7 常见问题/114
 - 3.3.8 楚凡 UML 工具模型示例/117
- 3.4 设计模型/119
 - 3.4.1 设计模式的选择与应用/120
 - 3.4.2 设计类的构建/122
 - 3.4.3 设计模型时序图/136
 - 3.4.4 一个示例/139
 - 3.4.5 参与人员建议/142
 - 3.4.6 输入输出/142
 - 3.4.7 UML 元素/142
 - 3.4.8 楚凡 UML 工具模型示例/142

第4章 代码模型一致性/147

- 4.1 概述/148
- 4.2 生成框架代码/148
 - 4.2.1 类的语法检查/148
 - 4.2.2 ClassPath 的设置/151



- 4.2.3 导出代码/154
- 4.3 维护设计模型/159
 - 4.3.1 目的/159
 - 4.3.2 维护方式/160
 - 4.3.3 维护步骤/160
- 4.4 反工/161
 - 4.4.1 作用/161
 - 4.4.2 操作步骤/162
- 4.5 参与人员建议/166
 - 4.5.1 输入输出/167
 - 4.5.2 UML元素/167
- 4.6 附加说明/167
 - 4.6.1 需求变化时的策略/167
 - 4.6.2 需要注意的问题/167

第5章 协作开发/169

- 5.1 协作开发过程/170
 - 5.1.1 准备工作/170
 - 5.1.2 分包/172
 - 5.1.3 合并分包/174
 - 5.1.4 分包特殊操作/175
- 5.2 多人协作开发/179
 - 5.2.1 第一种方法/179
 - 5.2.2 第二种方法/179
 - 5.2.3 一个特殊情况/179
- 5.3 文档模型结合/180
 - 5.3.1 概述/180
 - 5.3.2 文档模型的结合/180



第6章 软件组织方法论/183

- 6.1 结对编程/184
 - 6.1.1 产生/184
 - 6.1.2 结对编程形成/184
 - 6.1.3 结对编程的变形模式/185
- 6.2 交换编程/188
 - 6.2.1 引言/188
 - 6.2.2 相关工作/189
 - 6.2.3 方法定义/191
 - 6.2.4 优劣分析/193
 - 6.2.5 试验结果/195
 - 6.2.6 小结/196
- 6.3 可度量绩效管理模型/197
 - 6.3.1 引言/197
 - 6.3.2 团队组成与管理划分/197
 - 6.3.3 绩效管理办法基础/199
 - 6.3.4 绩效管理实现模型/200
 - 6.3.5 薪资收入组成模型/204
 - 6.3.6 绩效管理算法模型/206
 - 6.3.7 小结/206
- 6.4 实用性知识库体系构建/206
 - 6.4.1 基本原则/207
 - 6.4.2 内容分类/207
 - 6.4.3 企业基本制度和新员工学习材料库/207
 - 6.4.4 技术分类文档和学习材料库/208
 - 6.4.5 企业代码库/209
 - 6.4.6 行业客户资料和业务资料库/210
 - 6.4.7 企业知识库管理创建规则/210
 - 6.4.8 小结/211

第7章 软件开发中的心态对抗问题研究/212

7.1 项目管理者心态问题研究/214

7.1.1 引言/214

7.1.2 角色定位/215

7.1.3 心态对抗解析/217

7.1.4 小结/220

7.2 高层管理者心态问题研究/220

7.2.1 引言/220

7.2.2 角色定位/221

7.2.3 心态对抗解析/223

7.2.4 小结/230

7.3 管理中的权力使用分析/230

7.3.1 管理中的放权与分权/230

7.3.2 管理中的权力限制/234

7.3.3 项目管理者如何与组员合作/236

7.3.4 一些常见问题/237

7.4 项目经理分类/239

7.4.1 项目管理之传统/240

7.4.2 强势项目经理/241

7.4.3 弱势项目经理/242

7.4.4 题外话/244

7.4.5 小结/244

7.5 论本本主义/244

7.5.1 2009年的几件事情/244

7.5.2 引发的思考/245

7.5.3 产品、技术、标准/247

7.5.4 软件开发本本主义/251

7.5.5 最后的分析/252

新学网
PDG

第8章 技巧和相关文档/253

8.1 cat 包的修改/254

8.1.1 示例环境/254

8.1.2 操作步骤/257

8.1.3 结果/261

8.2 ClearCase 安装指南/261

8.2.1 概述/261

8.2.2 安装前的准备/262

8.2.3 配置安装/264

8.2.4 关于 License 的申请安装/267

8.3 关于 CMM 的一些观点/267

8.3.1 SQA 和 SQC/267

8.3.2 CMM, 评估、认证/269

8.3.3 只对组织中的某一个部门进行评估/269

8.3.4 评估是反复不断的/269

8.3.5 关于 CMM 过级时间的个人观点/269

8.4 什么是系统分析/271

8.4.1 定义/271

8.4.2 现实状况/271

8.4.3 一些建议/271

8.5 项目计划如何做/272

8.5.1 什么是计划/272

8.5.2 评价你们的计划状态/273

8.5.3 项目中的计划/273

8.5.4 计划的生命周期/274

8.5.5 计划中的常见错误/279

8.5.6 小结/280

8.6 评审如何做/280

8.6.1 什么是评审/280



- 8.6.2 对评审状态的评价/280
- 8.6.3 评审过程/281
- 8.6.4 评审常见错误/282
- 8.6.5 小结/283
- 8.7 配置管理工具如何用/283
 - 8.7.1 配置管理工具的使用方法/284
 - 8.7.2 小结/287
- 8.8 软件人员的分化/287
 - 8.8.1 人员分化原因/287
 - 8.8.2 需求人员的分离/288
 - 8.8.3 设计人员的分离/289
 - 8.8.4 编码人员的分离/290
 - 8.8.5 关于测试人员/291
 - 8.8.6 小结/295
- 8.9 Use Case 阐述模型化问题及延伸/295
- 8.10 最简化开发过程/297

附录 A 全程建模方法论/299

附录 B 幼儿教育系统/307

附录 C 全程建模培训介绍/312

附录 D 作者随笔一/316

附录 E 作者随笔二——毕业十年记/322

附录 F 文档信息和文档修订历史/327

后记/336

参考文献/337



第1章

软件工程概论

本章为读者提供了一个关于软件工程的整体印象，它将帮助读者更深入地理解本书的主体内容，同时对现有的传统软件 engineering 体系进行了一系列的反思，并建设性地提出一个初步的针对整个软件 engineering 体系的划分方式。

本章内容包括：

- 软件工程概述；
- 软件工程过程；
- 软件工程方法。

第1项主要描述了传统软件 engineering 体系和新的软件 engineering 体系的分类。第2项和第3项分别介绍了软件工程过程的划分和软件工程方法论。



1.1 概述

在本书的第1版中,关于软件工程定义部分没有做更多的描述,这是因为本书的第1版定位于方法论的讨论,而本版将在继续丰富完整方法的基础上扩展更多的内容,以便于展现一套更完善的软件工程系。

在第1版中关于软件工程的定义是:软件工程是指导计算机软件开发和维护的工程学科,是采用工程的概念、原理、技术和方法来开发与维护软件,把经过时间考验而证明正确的管理技术和当前能够得到的最合适的技术方法结合起来的实现过程。

另外,还有一些定义如下(这部分定义引自8848.com的seconcept.pdf,该文档的总结做得相当不错):

软件工程是一类求解软件的工程。它应用计算机科学、数学及管理科学等原理,借鉴传统工程的原则、方法,创建软件以达到提高质量、降低成本的目的。其中,计算机科学、数学用于构造模型与算法;工程科学用于制定规范、设计范型、评估成本及确定权衡;管理科学用于计划、资源、质量、成本等管理。软件工程是一门指导计算机软件开发和维护的工程学科,也是一门交叉学科。

Boehm:运用现代科学技术知识来设计并构造计算机程序和一些为开发、运行及维护这些程序所必需的相关文件资料。

IEEE:软件工程是开发、运行、维护和修复软件的系统方法。

Fritz Bauer:建立并使用完善的工程化原则,以较经济的手段获得能在实际机器上有效运行的可靠软件的一系列方法。

这些都是以传统软件工程为基础而产生的定义。在这些定义的背后带来的就是如何对软件工程体系的细节划分问题。本节的目的不是改变软件工程的定义和概念,而是详细阐述基于这样的概念和定义,软件工程在应用UML语言情况下具体实用的分类形式。因为,只有较好的分类理论才能更有效地引导具体问题的解决。

由于本书侧重的方法论表述方式是基于UML语言的模型化表述形式,因此和传统的文字描述方式有着较大的差异。其差异的核心就在于表述形式改变了,即开发人员对开发各阶段工件的表现形式的不同,而这种区别使得开发人员在面对一个软件项目时其思维方式也有了较大的转变。传统模式下的语言文字描述了图形化表述的改变,使得人们对软件工程全部过程的想法及其具体作用都产生了影响。

1.1.1 全程建模下的软件工程分类

在2006年12月21日的一篇题为“[软件工程]我对软件工程领域划分的认识之一”(http://blog.csdn.net/qingrun/archive/2006/12/21/1451598.aspx)文章的评论中,笔者与国

内某高校教授软件类学科的老师进行了关于划分的细节讨论,讨论中我们可以看到传统软件工程细分和当前软件工程细分之间的差异。

本书定义的软件工程基础体系划分模型如图 1-1 所示。

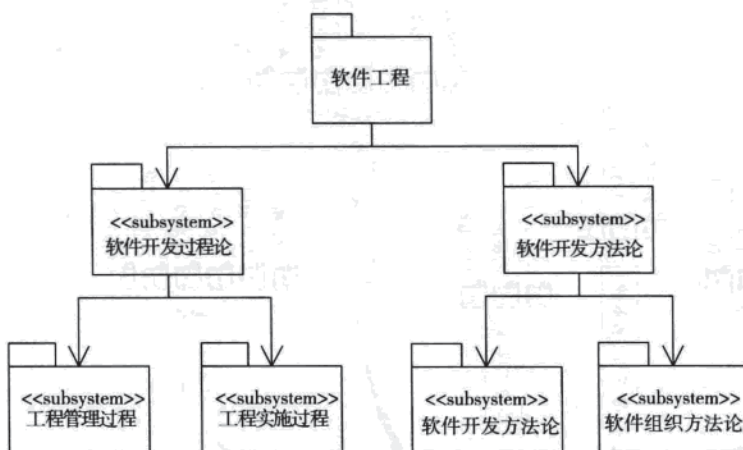


图 1-1 软件工程体系分类

图中 4 个子项分类将在后面的几节中分别进行阐述。

这种划分方式与传统软件工程定义中的内容有较大的差异。从工程项目到产品的角度来看,本书建议的方式会更贴近软件开发的实际,而目前许多教科书中传统软件工程定义划分(参看 1.1.2 小节)是基于 20 世纪软件行业中常规的开发方法实践积累形成的,这部分开发方法论形成于 20 世纪 90 年代中期以前。它们大部分已经不能适应新时代软件项目和产品开发中对新技术的要求与用户对产品的定义,所以,客观地看待各个阶段的定义和划分是十分必要的,也有利于我们开发出真正适合目前阶段的软件成品(包括产品和工程软件系统)。

从前面内容中,我们可以看到新的软件工程体系与传统软件工程体系之间的差异,同时也能看到它们之间的传承关系。事物的发展都不是孤立的,它们都与过去和将来有着十分紧密的关联。如果割裂时代对某一个时期进行孤立的分析,就不可能得到客观而实际的分析结果,凝聚其中的僵化与错误将会对实际软件工程活动的有效开展产生阻碍。

1.1.2 传统软件工程分类

传统软件工程知识体系全景如图 1-2 所示。^①

图中的覆盖范围非常大,不仅囊括了传统软件工程知识体系的分类,还包括软件系统和相关学科的融合关联关系的表达,甚至将基础知识及许多行业也都容纳其中。另外,这张图

^① 此图源自 <http://blog.csdn.net/kongdong/>, 原文参考《中国软件工程学科教程》(清华大学出版社)。

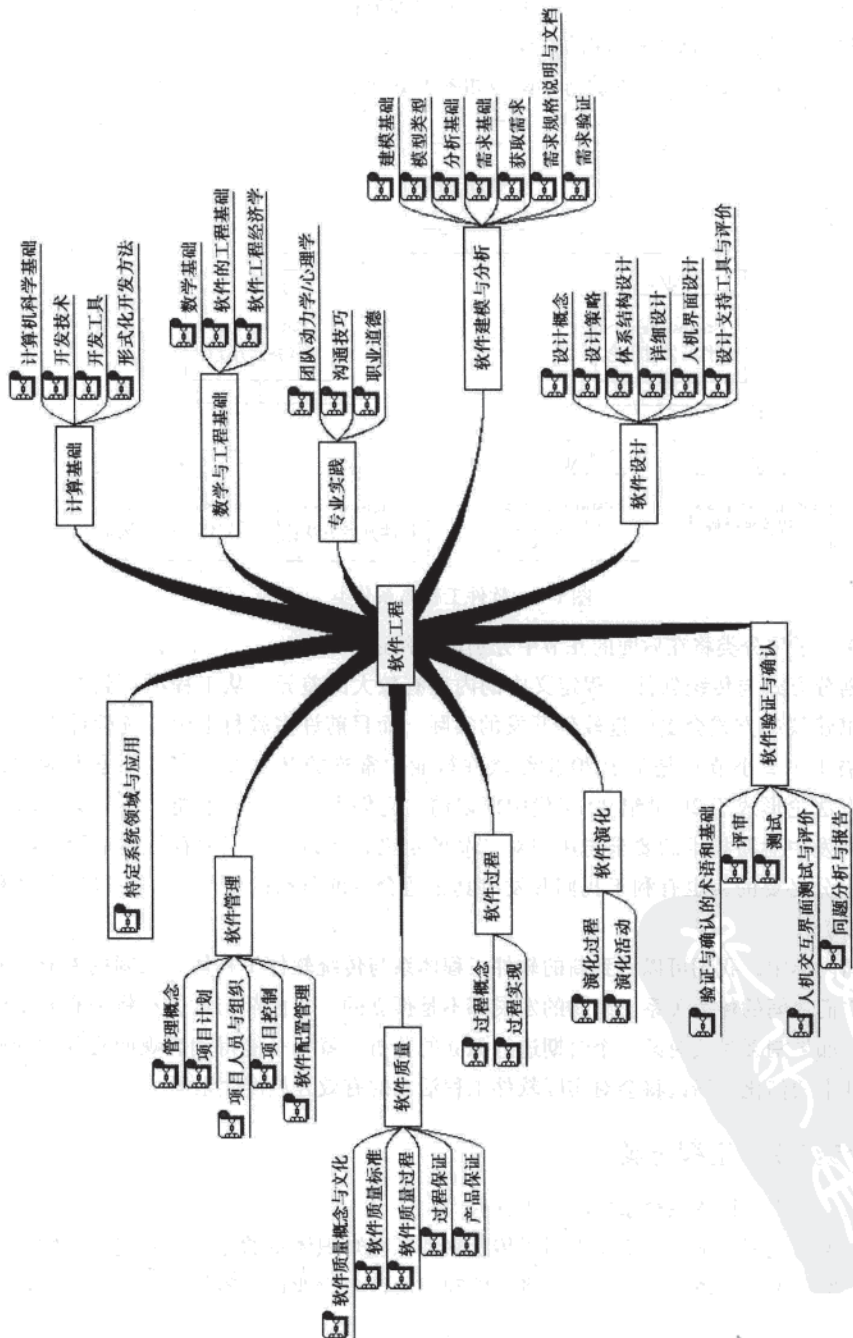


图1-2 传统软件工程知识体系

可以说在一定程度上模糊了软件工程与应用/行业软件之间的区别。

诚然，在不同的行业，软件的表现形式会有所差异，但是，在一个抽象的理论层面，软件工程应该是可以包容所有行业的，并不是一个基础应用行业软件的方法就可以抽象成为软件工程理论。比如，深圳某金融行业公司的负责人提出的树形软件工程理论，其实是其本人在带队完成深交所相关行业软件的过程中总结出来的方法的汇总，具有行业现实意义，但是却不能与作为抽象和具有高度概括性的软件工程理论相提并论。而软件工程理论是可以从其总结的树形软件工程方法中进行借鉴，取其精华去其糟粕。^①

另外，从图 1-2 中可以看出已将需求工程划分为需求基础、获取需求、需求规格说明书、需求验证 4 个部分，并把这 4 个部分归结到软件建模与分析子类别中。这样做是不合适的，需求工程是整个软件工程活动中非常独立的一个阶段，其重要性完全可以与软件管理、软件建模、软件质量、软件验证与确认等比肩。它的地位和作用决定了其不应该作为这样一个三级子项存在，而应该作为一个独立的二级类别存在，这一点将在第 2 章中做出更详细的阐述。

图 1-2 很明确地忽视了 DSL 类建模语言和模型化分析表达式的存在，而把文档作为一种被唯一明确提出的模型/信息保存方式，在笔者查阅了该书的内容后，确认上面提到的建模内容并不是基于建模语言等表达方式的图形化模型的建模含义，因此可以认定这里的基础仍然属于传统软件工程知识体系。

1.1.3 SWBOOK2004 中软件工程体系划分

SWBOOK2004 版对软件工程体系的划分如图 1-3 所示，内容描述如表 1-1 所示。

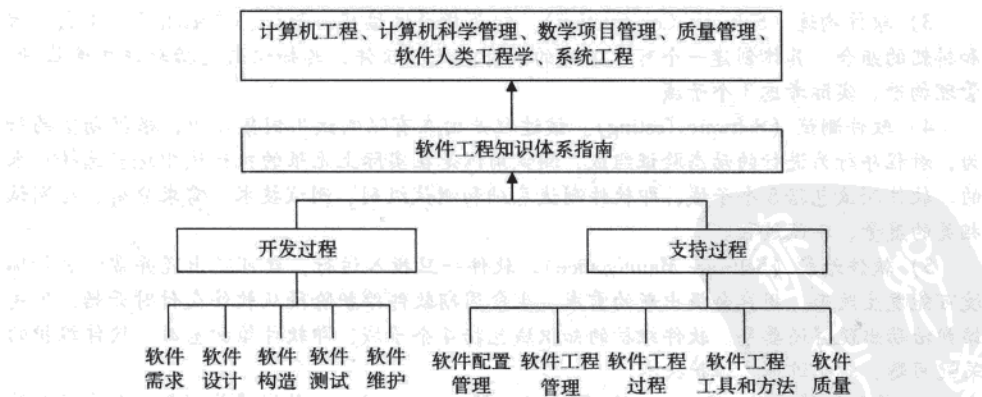


图 1-3 软件工程知识体系指南类图

① 这里的糟粕不一定是说不好的东西，而是说一些个性的不具有通用性的方法都应该斟酌进行去除或者在恰当的区域使用。

表 1-1 软件工程知识体系指南 2004 版

知 识 域	内 容
软件需求	软件需求基础、需求过程、需求获取、需求分析、需求规格说明、需求确认和实践考虑
软件设计	软件设计基础、软件设计关键问题、软件结构与体系结构、软件设计质量的分析与评价、软件设计符号、软件设计的策略与方法
软件构造	软件构造基础、管理构造、实际考虑
软件测试	软件测试基础和测试级别、测试技术、需求分析、与测试是相关的度量、测试过程
软件维护	软件维护基础、软件维护的关键问题、维护过程、维护技术
软件配置管理	软件配置管理过程管理、软件配置标志、软件配置控制、软件配置状态统计、软件配置审核、软件发行管理和交付
软件工程管理	启动和范围定义、软件项目计划、软件项目实施、评审与评价、关闭、软件工程度量
软件工程过程	过程实施与改变、过程定义、过程评定、过程和产品度量
软件工程工具和方法	软件工具：软件需求工具、软件设计工具、软件构造工具、软件测试工具、软件维护工具、软件配置管理工具、软件工程过程工具、软件质量工具和其他工具 软件工程方法：启发式方法、形式化方法、原型方法
软件质量	软件质量基础、软件质量过程、实践考虑

1) 软件需求 (Software Requirements)。真实世界问题而必须展示的特性。软件需求的知识域包括 7 个子域, 即软件需求基础、需求过程、需求获取、需求分析、需求规格说明、需求确认和实践考虑。

2) 软件设计 (Software Design)。根据 IEEE [IEEE 610.12-90], 设计既是“定义一个系统或组件的体系结构、组件、接口和其他特征的过程”, 又是“这个过程的结果”。软件设计知识域包括 6 个子域, 即软件设计基础、软件设计关键问题、软件结构与体系结构、软件设计质量的分析与评价、软件设计符号、软件设计的策略与方法。

3) 软件构造 (Software Construction)。它是指通过编码、验证、单元测试、集成测试和排错的组合。具体创建一个可以工作的、有意义的软件, 其知识域包括软件构造基础、管理构造、实际考虑 3 个子域。

4) 软件测试 (Software Testing)。该过程是由在有限测试用例集合中, 根据期望的行为, 对程序行为进行的动态验证组成, 测试用例是在实际上无限的执行域中适当选择出来的。软件测试包括 5 个子域, 即软件测试基础和测试级别、测试技术、需求分析、与测试相关的度量、测试过程。

5) 软件维护 (Software Maintenance)。软件一旦投入运行, 就可能出现异常, 运行环境可能发生改变, 用户会提出新的需求。生命周期软件维护阶段从软件交付时开始, 但是维护活动出现得还要早。软件维护的知识域包括 4 个子域, 即软件维护基础、软件维护的关键问题、维护过程、维护技术。

6) 软件配置管理 (Software Configuration Management)。软件配置管理是一个为了系统地控制配置的变更与维护在整个系统生命周期中的完整性和可追踪性, 而标志软件在时间上不同点的配置的学科。软件配置管理包括 6 个子域, 即软件配置管理过程管理、软件配置标志、软件配置控制、软件配置状态统计、软件配置审核、软件发行管理和交付。

7) 软件工程管理 (Software Engineering Management)。处理软件工程的管理与度量。

虽然度量是所有知识域的一个重要方面，但是这里涉及的是度量程序的专题。软件工程管理包括6个子域，即启动和范围定义、软件项目计划、软件项目实施、评审与评价、关闭、软件工程度量。前5个子域覆盖了软件过程工程管理，第6个子域描述软件度量的程序。

8) 软件工程过程 (Software Engineering Process)。它涉及软件工程过程本身的定义、实现、评定、度量、管理、变更和改进。软件工程过程包括4个子域，即过程实施与改变、过程定义、过程评定、过程和产品度量。

9) 软件工程工具和方法 (Software Engineering Tool and Method)。它包括软件工程工具、软件工程方法两个子域。

10) 软件质量 (Software Quality)。处理跨越软件生命周期过程的软件质量的考虑。由于软件质量在软件工程中无处不在，其他知识域也涉及质量问题。软件质量包括3个子域，即软件质量基础、软件质量过程、实践考虑。

仔细阅读上面的内容后，会感觉 SWBOOK2004 的分类也存在一些问题。比如，配置管理与通常定义的配置管理并没有什么区别，但是，这里将配置管理和软件工程管理并列是不太妥当的，软件工程管理中包含了通常的项目管理的内容，而实际上配置管理也是项目管理的一个子集，而且是一个不可或缺的子集，或者说软件工程管理包含配置管理；而同时在软件工程管理里面包含了软件工程度量的内容，实际上软件工程度量是不能脱离软件工程过程而单独描述的，不同的过程模型和开发方法下软件工程度量会产生不同的实际数据，这些数据的存在都是现实的并且有其必然性的；另外，如果按照这样的方式定义，那么需求管理必须在软件需求中明确指出，而这里软件需求的定义中却没有需求管理的内容。同样的，软件开发中定义的风险管理、合同/子合同管理等也应该单独列出来。

1.2 软件工程过程

软件工程过程论分为软件工程管理过程和软件工程施工过程两个部分。

1.2.1 软件工程管理过程

软件工程管理过程包括：团队的管理思想、领导者的思路、团队的组织形式、市场定义、客户行为分析和定位等。或者说，这个过程是对外的过程；对外，指的是开发团队之外的组织或个人，如管理层、客户、市场，等等。软件工程管理过程的分类如图1-4所示。

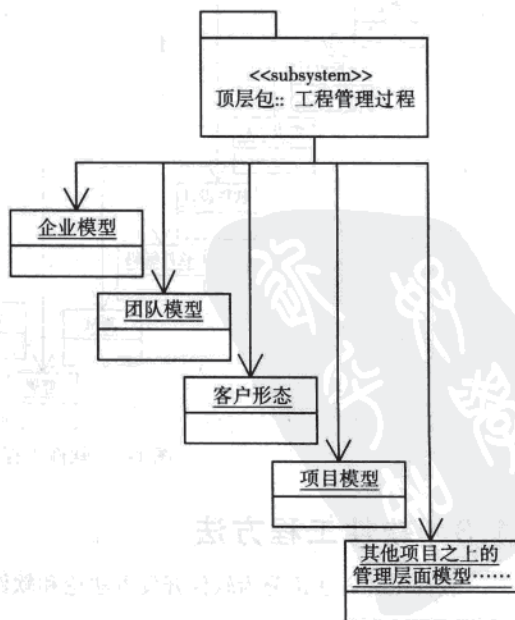


图 1-4 软件工程管理过程的分类

1.2.2 软件工程实施过程

软件工程实施过程包括工程阶段划分模型和工程过程模型两个子集。前者是人为定义或者按照工程实施阶段划分定义出来的各个子项，后者表达的是前者各子项的组合形态，也就是通常所说的软件工程过程模型，如瀑布式、迭代式、循环式、螺旋式、喷泉式等，如图 1-5 所示。

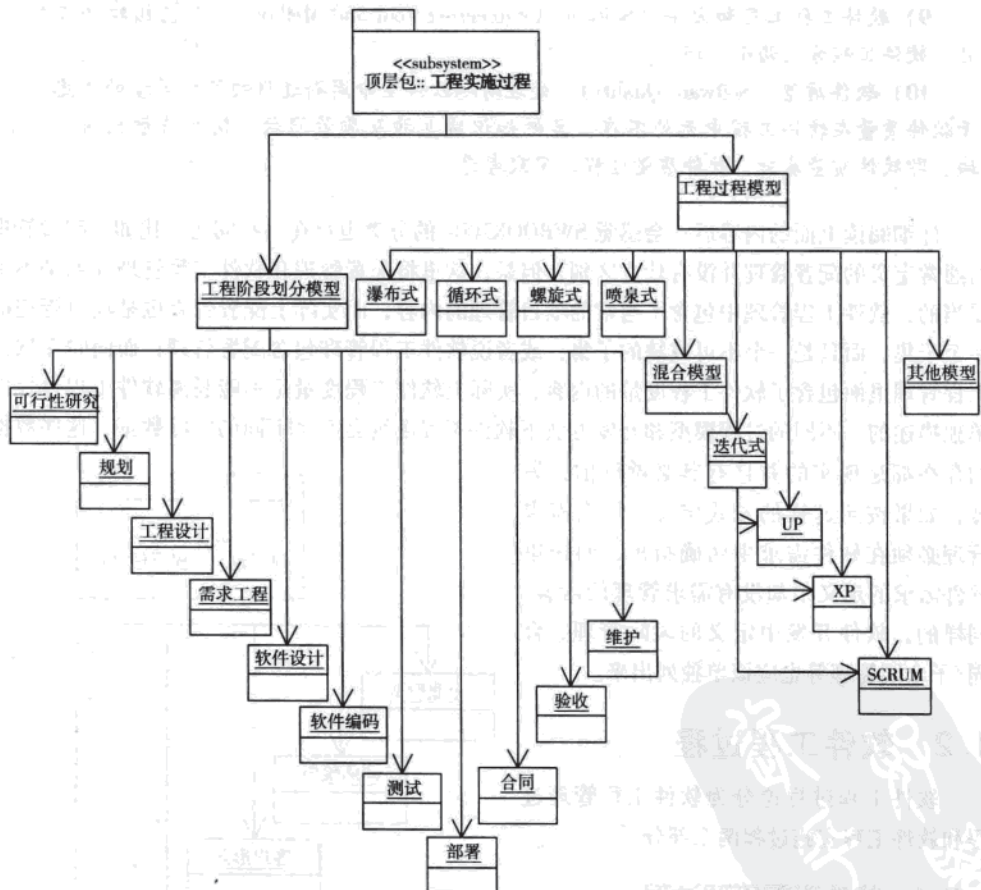


图 1-5 软件工程实施过程的分类

1.3 软件工程方法

软件工程方法论分为软件开发方法论和软件组织方法论两个部分。

1.3.1 软件开发方法论

1. 什么是软件开发方法论

方法论的英文为 Methodology, 编程的方法论是指软件开发的一整套方法、过程、规则、实践和技术。我们通常提到的方法论都侧重于项目、过程和人员的管理。

Agile Software Development 的作者 Alistair Cockburn 提出方法论具有以下要素: 角色 (Roles)、个性 (Personality)、技能 (Skills)、团队 (Teams)、技术 (Techniques)、活动 (Activities)、过程 (Process)、产品 (Products)、里程碑 (Milestones)、标准 (Standards)、质量 (Quality)、工具 (Tools)、团队价值 (Team Values), 它们之间的关系如图 1-6 所示。

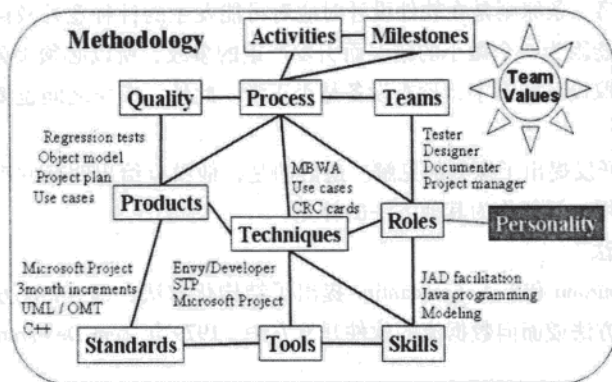


图 1-6 Alistair Cockburn 提出的方法论^②

图中其实已包含了我们认为的软件开发方法论和软件组织方法论两部分内容。

作者本人认为, 软件开发方法论是软件工程活动中需要用到的一整套各种具体方法的综合, 当这些具体方法由于某种相似性和相互关联而统一在一起使用来解决问题的时候, 我们就可以称之为是一套软件开发方法。

2. 方法论的起源和分类

20 世纪 60 年代中期爆发了众所周知的软件危机。为了克服这一危机, 1968 年和 1969 年连续召开的两次著名的 NATO 会议提出了软件工程这一术语, 并在以后不断发展与完善。与此同时, 国外大的软件公司和机构的软件研究人员也在不断探索新的软件开发方法。迄今为

② Alistair Cockburn 提出的方法论部分内容翻译 (此处的翻译与机械工业出版社出版的《敏捷软件开发 (原书第 2 版)》130 页上图 4-1 中有些不同, 这里是基于笔者的理解):

- Regression tests (回归测试)、Object model (对象模型)、Project plan (项目计划)、Use cases
- MBWA (走动管理)、Use cases、CRC cards (CRC 卡)
- Tester (测试员)、Designer (设计人员)、Documenter (文档编写员)、Project Manager (项目管理者)
- Microsoft Project (微软 Project 工具)、3month increments (3 个月的演进开发)、UML/OMT、C++
- Envoy/Developer (令人羡慕的开发组合)、STP (软件测试过程)、Microsoft Project
- JAD facilitation (联合应用开发)、Java programming (Java 编程)、Modeling (模型化)

止已经提出了很多实际的开发方法, 诸如 Parnas 方法、结构化法、生命周期法、原型法、面向对象法等。下面介绍几种比较流行的开发方法。

(1) Parnas 方法

最早的软件开发方法是由 D. Parnas 在 1972 年提出的。由于当时软件在可维护性和可靠性方面存在严重问题, 因此 Parnas 提出的方法是针对这两个问题的。首先, Parnas 提出了信息隐蔽原则: 在概要设计时列出将来可能发生变化的因素, 并在模块划分时将这些因素放到个别模块的内部。这样, 在将来由于这些因素变化而需修改软件时, 只需修改这些个别的模块, 其他模块不受影响。信息隐蔽技术不仅提高了软件的可维护性, 而且也避免了错误的蔓延, 改善了软件的可靠性。现在信息隐蔽原则已成为软件工程学中的一条重要原则。

Parnas 提出的第二条原则是在软件设计时应对可能发生的种种意外故障采取措施。软件是很脆弱的, 很可能因为一个微小的错误而引发严重的事故, 所以必须加强防范。如在分配使用设备前, 应该取设备状态字, 检查设备是否正常。此外, 模块之间也要加强检查, 防止错误蔓延。

Parnas 对软件开发提出了深刻的见解。遗憾的是, 他没有给出明确的工作流程。所以这一方法不能独立使用, 只能作为其他方法的补充。

(2) 结构化方法^①

1978 年, E. Yourdon 和 L. L. Constantine 提出了结构化方法, 即 SASD 方法, 也可称为面向功能的软件开发方法或面向数据流的软件开发方法。1979 年 Tom DeMarco 对此方法作了进一步的完善。

Yourdon 方法是 20 世纪 80 年代使用最广泛的软件开发方法。它首先用结构化分析 (SA) 方法对软件进行需求分析, 然后用结构化设计 (SD) 方法进行总体设计, 最后是结构化编程 (SP)。这一方法不仅开发步骤明确, SA、SD、SP 相辅相成, 一气呵成, 而且给出了两类典型的软件结构 (变换型和事务型), 便于参照, 使软件开发的成功率大大提高, 因而深受软件开发人员的青睐。

(3) 面向数据结构的软件开发方法

Jackson 方法是最典型的面向数据结构的软件开发方法, 1975 年, M. A. Jackson 提出了一类至今仍广泛使用的软件开发方法。这一方法从目标系统的输入、输出数据结构入手, 导出程序框架结构, 再补充其他细节, 就可得到完整的程序结构图。这一方法对输入、输出数据结构明确的中小型系统特别有效, 如商业应用中的文件表格处理。该方法也可与其他方法结合, 用于模块的详细设计。

Jackson 方法把问题分解为可由 3 种基本结构形式表示的各部分的层次结构。3 种基本结构形式就是顺序、选择和重复。3 种数据结构可以进行组合, 形成复杂的结构体系。

1974 年, J. D. Warnier 提出的软件开发方法与 Jackson 方法类似。

① 参见《软件开发方法评述》和胡朝晖博士早年撰写的《关于软件工程理论的认识与运用误区》一文。

差别有3点：一是它们使用的图形工具不同，分别使用 Warnier 图和 Jackson 图；另一个差别是使用的伪码不同；第三也是最主要的差别是在构造程序框架时，Warnier 方法仅考虑输入数据结构，而 Jackson 方法不仅考虑输入数据结构，而且还考虑输出的数据结构。

(4) 面向问题的分析法

PAM (Problem Analysis Method) 是20世纪80年代末由日立公司提出的一种软件开发方法。它的基本思想是考虑输入、输出数据结构，指导系统的分解，在系统分析指导下逐步综合。

这一方法的具体步骤是：从输入、输出数据结构导出基本处理框；分析这些处理框之间的先后关系；按先后关系逐步综合处理框，直到画出整个系统的 PAD 图。

这一方法本质上是综合的自底向上的方法，但在逐步综合之前已进行了有目的的分解，该目的就是充分考虑系统的输入、输出数据结构。PAM 方法的另一个优点是使用 PAD 图。这是一种二维树形结构图，是目前为止最好的详细设计表示方法之一。当然，由于在输入、输出数据结构与整个系统之间存在着鸿沟，这一方法仍只适用于中小型问题。

(5) 原型化方法

原型化方法 (Prototyping) 是20世纪80年代随着计算机软件技术的发展，特别是在关系数据库系统 (Relational Data Base System, RDBS)、第四代程序生成语言 (4th Generation Language, 4GL) 和各种系统开发生成环境产生的基础上提出的一种从设计思想、工具、手段都全新的系统开发方法。它扬弃了那种一步步周密细致地调查分析，然后逐步整理出文字档案，最后才能让用户看到结果的繁琐作法。

产生原型化方法的原因很多，主要随着人们系统开发经验的增多，发现并非所有的需求都能预先定义，而且反复修改是不可避免的。能够采用原型化方法是因为开发工具的快速发展，比如用 VB、Delphi 等工具就可以迅速地开发出一个能让用户看得见、摸得着的系统框架，这样，对于计算机不是很熟悉的用户可以根据这种模式提出自己的需求。

开发原型化系统一般有以下几个阶段：

- 确定用户需求；
- 开发原始模型；
- 征求用户对初始原型的改进意见；
- 修改原型。

原型化开发比较适合于用户需求不清、业务理论不确定、需求经常变化的情况。当系统规模不是很大也不太复杂时采用该方法是比较好的。

原型法随着时代的变迁不断被丰富，目前常用的原型法又分为：快速原型法、实用性原型法、抛弃性原型、演进原型法、实验原型法等。

快速原型法就是最初的原型法，通过原型引导用户需求的方法。

实用性原型法是每次重新构建原型后将用户认可的部分记录汇总形成最终用户系统的方法。

抛弃性原型法是每次构建原型后着重记录用户不认可的信息，经过多次反复后将用户不认可的部分全部排除而形成最终用户系统的方法。

演进原型法又称为增长原型法，通常是先开发运行一个子系统或分子系统，再扩充其功能，或者开发另一个相关的子系统或分子系统，并归并集成，这样自底向上地逐步得到一个较完整的系统。

实验原型法 (Experiment Prototyping, EP)，它是通过建立真实系统的模型，由局部模型不断实验改进，最后得到整个系统的模型。

实用性原型法和抛弃性原型法属于快速需求获取的原型法，其产生的结果是用户的需求信息和不需要的信息，却不是直接形成可交付系统本身。而演进原型法和实验原型法产生的结果是最终可交付系统的一部分，需要用户方有人能够全程跟进或者参与评审提出意见。

(6) 面向对象的软件开发方法

当前计算机业界最流行的几个术语就是分布式、并行和面向对象。由此可见，面向对象这个概念在当前计算机业界的地位。比如当前流行的两大面向对象技术 DCOM 和 CORBA 就是例子。当然我们实际用到的还是面向对象的编程语言，比如 C++。不可否认，面向对象技术是软件技术的一次革命，在软件开发史上具有里程碑的意义。

随着 OOP（面向对象编程）向 OOD（面向对象设计）和 OOA（面向对象分析）的发展，最终形成了面向对象的软件开发方法 OMT（Object Modeling Technique）。这是一种自底向上和自顶向下相结合的方法，而且它以对象建模为基础，从而不仅考虑了输入、输出数据结构，实际上也包含了所有对象的数据结构。OMT 彻底实现了 PAM 没有完全实现的目标。不仅如此，OO 技术在需求分析、可维护性和可靠性这三个软件开发的关键环节和质量指标上有了实质性的突破，基本上解决了在这些方面存在的严重问题。

综上所述，面向对象系统采用了自底向上归纳和自顶向下分解的方法，它通过对对象模型的建立，能够真正建立基于用户的需求，而且系统的可维护性大大改善。当前业界关于面向对象建模的标准是 UML（Unified Modeling Language）。

在此我们需要介绍一下微软的 MSF（Microsoft Solutions Framework）框架，它简单地把系统设计分成三个阶段：概念设计、逻辑设计和物理设计。概念设计阶段是指从用户的角度出发可以得到多少个对象，并且以对象为主体，画出业务框架。逻辑设计阶段是对概念设计阶段的对象进行再分析、细分、整合与删除，并建立各个对象的方法属性以及对象之间的关系。而物理设计阶段实际上就是要确定我们实际需要的组件、服务和采用的框架结构及具体的编程语言等。MSF 整个结构比较清楚，是基于对象开发的一个比较好的可操作框架系统。

本书中所指的开发方法论是 OO 方法论在 UML 语言表述下的具体操作方法。

我们认为软件开发方法论的定义是软件工程实现过程中的表达方式，比如采用某种开发语言、某种设计模式，建立什么架构，细节的具体实现方法等，如图 1-7 所示。

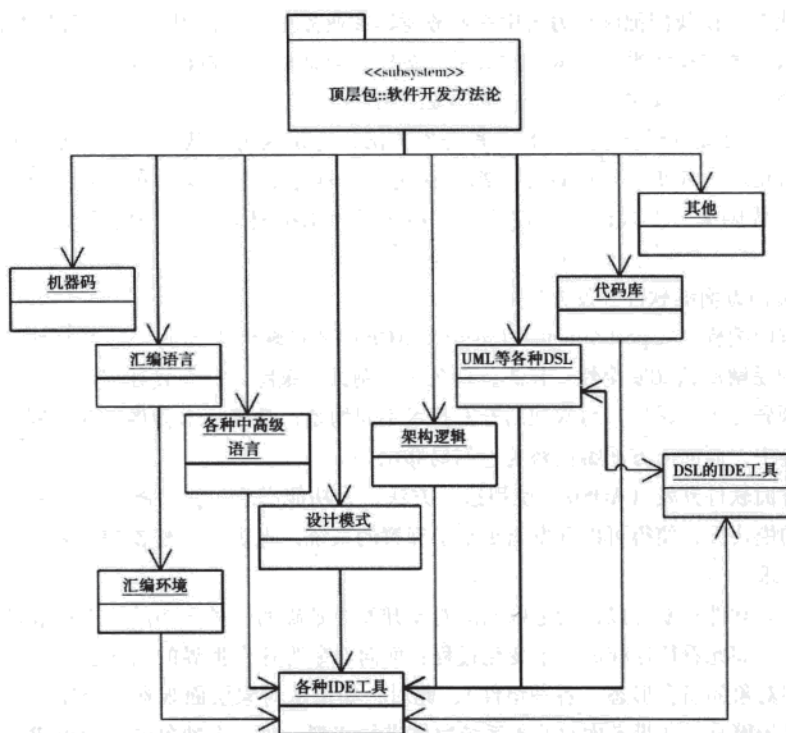


图 1-7 软件开发方法论图

由于各种 IDE 工具是一种把语言的描述工具、编译工具、调试工具，甚至设计模式、架构等实现方法融合起来的支持性工具，因此不能算作软件工程理论的内容，只能算是软件工程自身对自身制造的一种产品。因为，即使没有 IDE 工具，采用最基础的编译环境也一样可以完成整个软件，只是增加工作量而已。所以，在上述划分形式中就将其定义为对语言和其他的支持性工具，而不单独和软件工程体系本身发生直接关系。

(7) 可视化开发方法

其实可视化开发并不能单独地作为一种开发方法，更加贴切地说，它是一种辅助工具。比如，用过 SYBASE 的 S-Design 的人都知道，用这个工具可以进行可视化的图形化数据库模式的建立，并可以导入到不同的数据库中。便用过 S-Design 的人也许并不多，但用过 VB、Delphi、C++ Builder 等开发工具的人一定不少，你也许就是其中一员。也就是说，所有的 IDE 工具都是可视化开发方法的表现工具。

当然，不可否认的是，你只是在编程这个环节上用到了可视化方法，而不是在系统分析和系统设计这个高层次上用了可视化的方法。实际上，建立系统分析和系统设计的可视化工具是一个很好的卖点，国外有很多工具都致力于这方面产品的设计。比如 Business Object 就是一个非常好的数据库可视化分析工具。

可视化开发使我们把注意力集中在业务逻辑和业务流程上,用户界面可以用可视化工具方便地构成。通过操作界面元素,如菜单、按钮、对话框、编辑框、单选框、复选框、列表框和滚动条等,由可视化开发工具自动生成应用软件。

最近有人质疑可视化开发不是一种开发方法,这是因为自从可视化开发工具,也就是 IDE 工具出现后,几乎所有的程序员都是在这些工具之上进行软件开发的,他们没有经历过传统的行代码编写方式的开发模式,习惯中默认 IDE 环境下的开发就是最基本的开发模式。

(8) 面向方面的软件开发方法

面向方面编程 (Aspect-Oriented Process, AOP) 是对软件工程的一种革新性思考。引入 AOP 的目的是解决诸如安全性、日志、持久化、调试、跟踪、分布式处理、性能监控以及更有效地处理异常等问题的。与常规的开发技术不同的是,常规技术会将这些不同的关注点实现于多个类中,而面向方面编程将使它们局部化。

面向方面软件开发 (AOSD) 使用这一方法,为功能需求、非功能需求、平台特性等创造了更好的模块性,使得可以开发出更易于理解的系统,也更易于配置和扩展,以满足和解决涉众的需求。

面向方面软件开发可以认为是以面向对象开发为基础的一种新型的、看待业务系统的思考方法。可以如此看待这样的一个变化过程:面向对象为这个世界的描述提供了大量的基础对象和基础对象的组合形态(各种组件),此时,如果从对象层面观察一个庞大的业务系统就会显得过于繁复,如果考虑对业务系统整体进行切割,每一个抽象的切面形成一个完整的业务系统的层面,则相关的分析与研究就基于这样的层面考虑,然后下面细化为对象和组件,上面组合形成业务系统。

所以,我们认为面向方面的软件开发方法只是基于 OO 方法的一个较高抽象层面的方法,并不是一个完全崭新的开发方法论。

1.3.2 软件组织方法论

软件组织方法论指的是团队内成员间的合作方法以及相互关系,诸如结对编程,单人编程,还有笔者于 2006 年提出的交换编程(见 6.2 节)等,如图 1-8 所示,另外,资源的分配等也属于这个方面。

图 1-8 中其他组队模型部分不属于本书重点关注的内容,这里不做扩展分析。因为本书的重点放在开发方法论的完善上,除了对第 1 版中方法论的补充完善外,还将扩充部分软件开发心理学的内容,也将加入结对编程和交换编程的章节作为对团队组织模型的介绍。

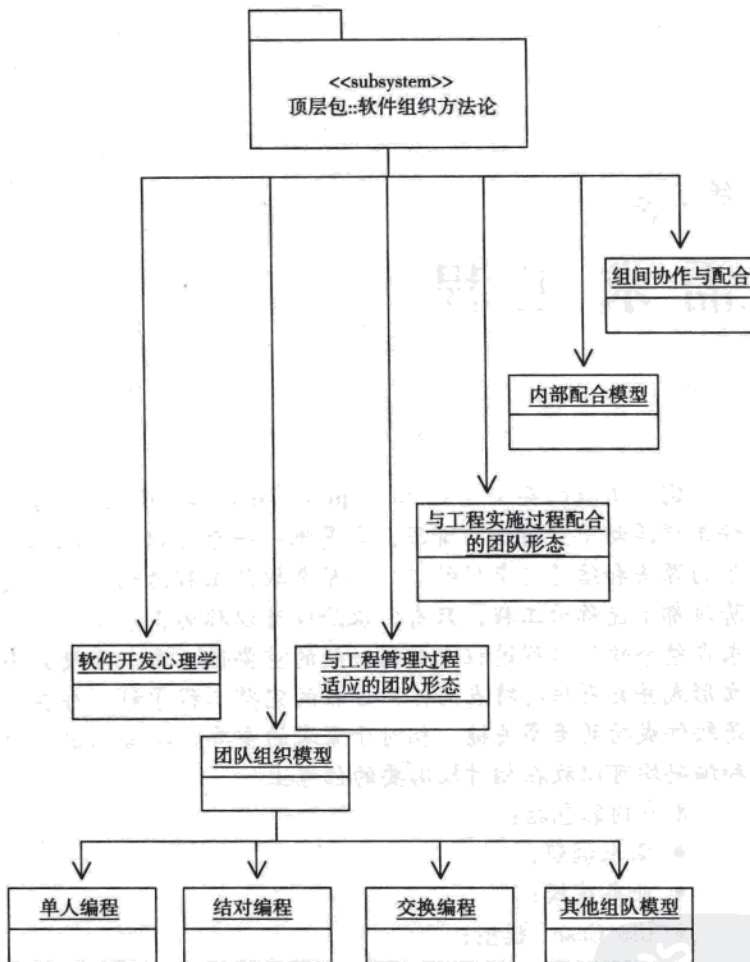


图 1-8 软件组织方法论图

第2章

需求工程

需求工程的英文全称为 Requirements Engineering。需求是整个软件工程活动中最特殊的阶段，它是唯一一个可以完全独立出来，有自己的源头和结尾可交付的工作。整个软件工程活动中，任何一个其他阶段都不能称为工程，只有需求阶段可以称为需求工程。这体现了需求在整个软件工程过程中非同一般的重要性。需求阶段完全可以以独立形式并具有独特特点的体系过程的完整工程学科而存在。需求工程是软件成功的至高关键。相对于需求的重要性而言来说，分析、设计和编码均可以放在相对较次要的位置上。

本章内容包括：

- 需求调研；
- 业务建模；
- Use Case[⊖] 模型；
- Use Case 阐述；
- 交互建模；
- 界面设计。

什么是需求？在需求工程的整个过程中要做哪些事情？如何采用 Rational Rose 进行用户需求的建模？如何处理建模得到的结果？需要经过哪些具体的操作步骤？

以上这些问题都是在本章中需要阐述的。

⊖ Use Case 一词在国内有很多种翻译，诸如：用例、用况、实例、使用案例等，而且为它的命名问题很多人还争得面红耳赤，但笔者常用的是用例这个译词。为了保持全书的一致性，笔者最终决定直接使用原型 Use Case。对于一些引用文字中的翻译，笔者将仍然尊重原文的译法不做修改。

2.1 需求的定义

关于需求的定义,本书在这里给出的是一种较为抽象的解释:

需求,是指对用户需要解决的问题的整体描述。

需求是软件实现之源。

下面看一下国际上较为通用的关于需求的定义。

在 IEEE^① 软件工程^② 标准词汇表 (1997 年) 中定义需求为:

- 1) 用户解决问题或达到目标所需的条件或权能 (Capability)。
- 2) 系统或系统部件要满足合同、标准、规范或其他正式规定文档所需具有的条件或权能。
- 3) 一种反映上面 1) 或 2) 所描述的条件或权能的文档说明。

从这个标准定义中可以看出需求在软件工程中的重大作用。

实际上它表达出了一个隐含的意义:

用户方与开发方双方的意愿性需求^③驱动一切。

在这个意愿性需求的驱使之下,双方的人员开始接触,先是由用户方给开发方提供他们需要解决的问题和要求,经过开发方一段时间的努力,开发方将给用户方交付针对用户需求的软件实现产品,同时用户根据工期再把符合条款要求的约定款项分批分期支付给开发方,作为开发方的酬劳。软件工程过程就是这样以利益驱动下的需求调研开始,以货币的转移交付结束。

从商业的角度来看,软件工程的全部过程实际上就是实现一次交易的过程,只不过,这个交易过程有着软件产业的特点,与传统的商品交易有着较大的物质形态差异,它有着介质附带可以产生目标劳动价值的(在这里目标劳动价值是最终用户所需的劳动价值,而这个劳动价值必须在程序员的劳动价值的积累和凝聚的基础上才能完成)劳动价值的积累和凝聚。

① IEEE: 电气电子工程师学会, 英文全称为 The Institute of Electric & Electronics Engineers。IEEE 创建于 1884 年, 是一个跨国性学术组织, 现有会员 334 000 人, 其中在美国的会员有 224 000 人, 另外的 110 000 人分布在全世界的 150 个国家和地区。[2003]

② 软件工程是指导计算机软件开发和维护的工程学科, 是采用工程的概念、原理、技术和方法来开发与维护软件, 把经过时间考验而证明正确的管理技术和当前能够得到的最合适的技术方法结合起来的实现过程。

③ 注意: 驱动这个意愿性需求的是双方的共同利益。这个利益可能是金钱, 也可能是政绩, 或者职位提升, 或者其他形式的收益。

2.2 需求调研

如何进行需求调研^①？这个问题在软件工程的实现过程中是一个至关重要的问题，因为需求是软件之源，没有了需求，软件也就无从谈起。

但是，在我国目前的软件项目中，最基础的需求调研工作也仍然处于一个非常不规范的行业状态下，由此可见中国软件业工程化的道路还会有多远？

这里笔者把需求调研称为“基础的需求调研工作”，在软件工程中处于最底层的就是需求调研，基于其上的是业务建模，然后是 Use Case 模型、Use Case 阐述、交互建模、界面设计等，这些构成了整个需求工程的两个大的阶段：需求收集阶段和需求分析阶段。此后就进入分析模型、设计模型、代码导出、设计模型维护、代码编写、测试活动、部署实施等后续活动阶段中，其关联关系如图 2-1 所示。

2.2.1 调研中存在的问题

下面列举了一些在实际工程的需求调研过程中经常遇到的情况。这里做一个大致的归纳，希望能在读者今后的工作中起到抛砖引玉的启示作用。

1. 小型项目

对于比较小的项目，一般是指客户方的投入在几万元^②到三四十万元之间的工程项目。很多时候，双方只要有一个意向性的要求，软件开发商就开始了需求调研的工作，没有签署任何的合同或协议，而仅仅只有一个表面的意向。

因此，在开发过程中，用户就会不断提出各种各样的问题和要求，以至于需求工作无法正常进行。当然，有些时候由于开发人员对用户的业务不熟悉，也会造成双方的隔阂，影响工作进度，结果自然不言而喻。

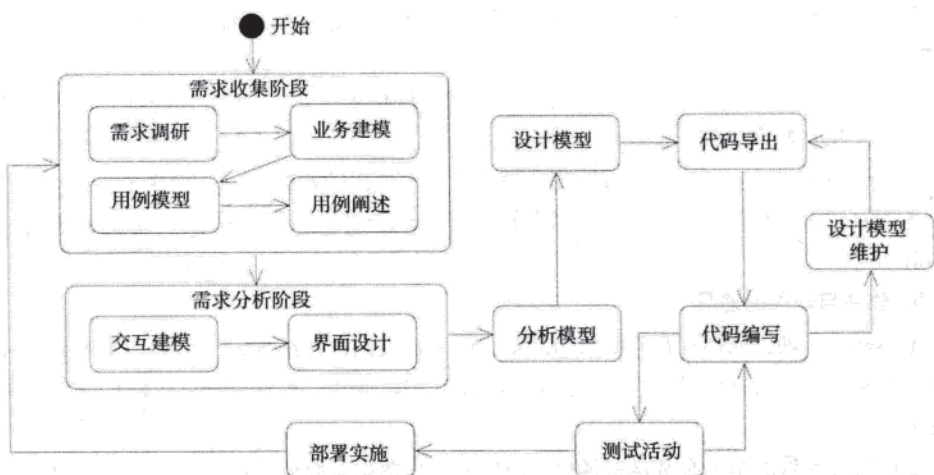
这类问题的解决方法一般是通过商务活动，争取获得一些用户预付款的方式，要求市场部在签订合同之前，由有经验的技术人员配合对项目边界也就是项目的需求部分进行需求范围的确认。

2. 较大的项目

对于比较大的项目，一般是指客户方的投入在三四十万到几百或者几千万元的工程项目，软件开发商急切想得到这个项目，于是就迫使开发人员尽快投入开发，必须不惜一切代价去拿下订单。这样就直接造成在没有签订合同和正式协议之前，开发人员已不得进入用户的工作区进行需求调研工作。此时，如果从商务上最终没能谈妥，就会使得开发人员的工

① 需求调研：需求调研实际上包括了需求的获取和初步研究。需求的获取是为了进行需求的研究，而对需求的初步研究也是为了能够获取更为完整更为真实的需求信息。通过调研的方式可以获得完整的业务需求和功能性需求。

② 这里，对于开发投资额度为几千到一两万元以内的项目而言就一般不会作为项目考虑了。

图 2-1 软件工程的 UML^①图形表达

作和努力付诸东流。

一般来说，国内的公司是不会考虑开发人员在这个没有洽谈好的项目的不确定阶段所付出的劳动和辛苦的，而且往往还会以开发成本过高、工期过长、技术难度过大、技术人员不够配合商务活动等理由将责任推到开发人员身上。

对于这类项目的前期配合工作是不可避免的，于是，尽可能选择公司已有的类似产品和已有产品原型的系统才是解决这个问题的较好办法。否则，轻易地进入用户场地进行开发，只要关系不够好合同未签订，技术人员的出面往往只是给了市场人员一个可推卸责任的无风险选择。

3. 正在成熟的用户

2001 年中期以来，用户也不像几年前那样了（参见 2.2.2 小节中对 2001 年以前的详细论述），用户会要求软件厂商先拿出相应的产品或者软件，然后才会考虑和软件厂商签订进一步的合同。用户甚至希望，先试用软件（其实是使用软件），然后签订合同。一旦合同签订完成，软件就可全部交付使用。

但是，由于国内企业特殊的具体情况，使得任何一个现成的工程软件产品都不可能不做任何修改地交给第二家客户使用，结果就造成了现场定制开发工作的存在，这自然和用户的需求存在着激烈的矛盾。

4. 用户过于繁忙

在一些调研工作中，由于用户较忙，很少有人能抽出时间来与需求调研人员一起讨论有关需求的详细问题。由于项目工期较紧，如果调研工作到期不能完成，则后续工作将无法开

① UML：统一建模语言，英文全称为 Unified Modeling Language。

展,最后可能导致项目失败。因此,需求调研人员还应做好与用户的协调工作,来保证调研工作顺利完成。

但需要掌握的一个原则是:不能逼迫用户(如催促等行为),不要以与项目相关的任何理由来迫使用户配合你的工作。

其实只需注意一个问题,就可以使上面的矛盾得到比较好的解决:这些无比忙碌的、系统将来的直接用户在他们的朋友意外到来的时候,却总能抽出一些时间来。即你要努力成为他们的朋友。

5. 软件与硬件的差异

虽然软件与硬件同样存在着贬值的问题,但是,人们的传统地认为,软件本身是一种摸不到、看不见、无法直接感知的产品,无法估量其实际价值。相对于软件,硬件就比较容易接受了。

这种情况导致的结果常常是:客户对硬件配置和费用一般很少提出异议,而且愿意很快地付钱,而一谈到软件费用,就会非常犹豫,不愿付钱或不断地找各种理由来压低软件费用。因此,软件与硬件的这种差异无形中造成了需求调研工作乃至整个软件工程活动过程中的一些意外的困难。

6. 对于已成型的产品

对于已经成型的产品,用户也会提出各种不同的需求来强调自己的与众不同,这些要求也许并不合理,但他们也会以相当强硬的态度要求开发人员必须实现。

由于行业软件的特性,几乎没有任何成型的软件产品可随时供其他用户直接使用,因此,二次开发就成了必需的过程。

7. 不同的软件不同的结果

对于那些能让客户直接见到利益的软件系统,客户愿意为此付款,而且也能配合开发人员的工作。但是这样的项目大部分都隶属于增值业务的范围之内,如电信行业中的短消息相关服务、WAP、定位、VOD、代理付费、网络游戏等。

还有相当数量的软件产品是不可能让客户直接体会到收益的,此时应该注意将隐藏在软件产品后面的收益尽量呈现出来,让客户充分体会到其效能,并愉快地配合开发人员的需求调研工作。

8. 高层的关注

高层领导非常关注的大型项目且直接影响到收益问题的系统项目,如电信行业的计费、网管系统,税务的税收、缴费系统,电力的生产、调度、计费、电检系统等,一般比较容易推广。因为有高层领导的关注,具体的办事人员会比较积极地配合。因此,在某些情况下需要一些特殊的方式来引起高层的关注。需要注意的是,不要因为有高层的关注就放松了对具体办事人员的关注,更不能因此而引起具体办事人员的反感,这样会使项目进展面临异乎寻常的困境。

9. 热点项目的竞争

一些用户比较愿意付款的项目同时也就成了各软件厂商相互竞争的热点,因为大家都看到了利益。这种利益的竞争,同时也引发了一系列的负面效应,比如不正当的竞争引入。

对于收益较大的项目,竞争中不仅要获得用户对产品的认可,还要体现出自己产品与竞争对手产品的差异性。差异体现独有价值的存在。

10. 结论

以上9个方面是针对正常的项目而言。也就是说,客户不只是为了做一个形象工程,而是真正想通过项目的开发及使用开发出来的产品而获利,否则,你只要让用户觉得高兴,基本上也就可以完成项目了。

总之,各种手段的协调应用,调动用户的积极性,是我们解决问题的最好办法。

2.2.2 过去和现在的对比

1. 过去的情况

几年前,至少在2001年以前,用户还处于下面这种状态(这种状态在2010年以前也一样存在于信息化进展较慢的部分行业内,而大部分用户已经有过几次失败的教训,则不会如此容易地签订合同了):

只要软件公司开发过一些系统并且在软件行业也有一些名气,派出几名技术人员和人员到用户那里许下承诺,做一个大的而且非常有诱惑力的规划方案^①,用户就会很友好地和软件公司签订协议或者合同,从而开始软件的开发工作。即使用户要求的开发时间很短,实现的功能要求也有很多不合理的地方,但是,开发人员仍然可以按照这样的条件进行开发,最后往往会发生一些纷争。其结果是通过关系的协调来解决这些纷争——其实这种现象至今仍然存在。

这是一个矛盾,也就是用户的需求与要求之间的矛盾,用户对软件产品开发过程认识的深度将决定这个矛盾的激化程度。当然,这也许是一个无法调和的矛盾,只能遇到具体问题进行分析、具体研究、再具体解决。

现在这种现象已经越来越少了,客户也越来越成熟了,简单的几句话和一个美好的框架

① 规划方案:

这个规划往往是指项目方案,方案中一般要包括下面这些内容:

项目背景分析——为什么要实施这个项目;

项目前景预期——实施项目之后会有哪些好处:收益率、投入产出比、响应速度等;

技术部分——系统将采用什么样的技术架构来实施;

资源部分——软件公司投入的人力、物力、财力和开发周期;

服务部分——软件公司将为该软件提供的相关售后服务和培训等的计划;

软硬件环境规划——该软件部署的软硬件环境的初步规划;

成本预算——实施这个系统投入的资金数额的预算;

等主要的7个部分。

已经不能说明什么具体的问题，最多只能算是给了客户一个方向性的概念而已。所以，开发人员需要做更多的工作。

2. 现在的情形

大概从2001年中期开始，用户明显成熟了很多，尤其是在一些大型行业，如电信、金融、电力、税务等行业。对于一些大型系统的开发，用户已经有了一些成型的考虑。也许是因为多年的经历折磨（建立项目后，不成功，重建，再次失败的循环往复），用户已开始形成自己的专家组，同时招聘一些相关的技术人员进入其行业内部参与这种信息化工作中，或者说是其e化流程的工作中。

用户方面的技术组往往会提出一些更高层次的理解和要求，同时进行多方的对比，比如，一些行业出现了软件系统的工程过程“分阶段签约、分阶段实施”的现象。如果经费允许的话，用户会找很多厂商同时做需求计划，看看谁做得更好，更适合自己的需要，然后，再进行第二阶段的招投标选择，最后决定采用哪一家开发商。

这只是全新用户现象中的一种，还有很多同类的现象，这里就不再一一赘述。总之，这些新现象都代表着用户成熟度的提高，对软件认识的深入。但是，用户毕竟不是软件行业人员，因此，一定还会提出很多不合理的要求，仍然需要开发人员格外注意那些即将出现的新问题、新要求。

2.2.3 新现象的分析

出现新问题、新要求其实是一种好现象，一方面表明用户开始考虑软件实现过程中的具体问题，也认识到必须提高自己对软件的认识程度和理解意识。用户开始招聘一些熟悉软件开发的人员，并使他们成为自己企业内的计算机专家。总结一句话就是：在其他行业内开始出现十分熟悉软件业的从业人员。

另一方面从另一个角度来分析，这些新现象也在要求软件公司或软件开发人员能够真正进行工程化的开发和管理（现在绝大多数软件公司还做不到真正的工程化的开发和管理），而不是像过去的完全作坊式的个人开发。现在大多数客户还是要求软件开发商提供软件开发过程文档，包括需求、分析、设计、编码、测试及相关项目计划等文档资料，来证明该开发商有着持续不断的开发维护能力，以及对该软件的持续发展有着长远的考虑。

用户也同样要求开发人员的分工更为明确。人员分工明确可以表现出一个软件开发公司的实力和管理水平。其实在开发人员中，同样有人提出了一些很好的理念（笔者也十分赞同），是否可以成立“系统分析师事务所”，正如律师有他们的律师事务所，医生有他们的诊所一样。对于需求分析这种难度较大，对个人经验和素质要求相对较高的工作，开发人员也同样可以考虑成立属于自己的专业性工作室——系统分析师事务所。依此类推，后面的分工就是一部分做高端分析设计的软件公司，以及一些从事低层次编码工作的软件工厂。

目前国外上已经存在类似的公司，相信这样的事务所在不久的将来同样会在我国诞生！

2.2.4 软件外包与编码

1. 软件外包为什么势在必行

软件产业的工程化是大势所趋！国际化的合作和分工曾经是大工业机器时代的特征，软件业迟早要走上这样的道路。同样的，软件的外包和国家间协作开发也是势在必行，各国之间的行业融和也会从其他行业进入到软件行业内。

例如，如果我们的设计水平达到了印度人认为可以编码实现的程度，那么当我国需要对外进行外包时，只要有利可图，印度的公司一定会接收该订单！——大多数人都会赞同笔者的这个看法，不是吗？

总有一天，我们也可以把软件外包给印度的公司，而我们只需要做分析和设计的工作。所以，不要认为印度的模式如何，美国的方式又怎样，有很多时候，这也许就是世界上已经为我们准备好的一些环境，一些可以让我们直接利用的环境。

2. 我们还需要编码工作吗

当然，我们也需要编码，我们需要进行的是创造性的编码，撰写功能实现的部分微代码，撰写高难度算法实现的微代码。这部分代码是一般软件蓝领工人无法编写的，这部分代码需要较高深的数学知识和计算机理论以及对行业业务的深入理解作为基础才可能编写出来的，这部分代码也需要代码的编写者对语言有着独到的理解和相当深入的认识和经验。

从上面提出的观点来看，作者认为：一般大学生毕业后至少需要从事某一语言方向两三年的实际编码工作，不过，不一定要完全去参与最低层次的编码工作，可以在有一定的编写低层次代码的基础之后主要从事编写微代码的工作——这也许就是将来区别开发人员中白领与蓝领的关键。

2.3 如何和用户交流

由于与用户交流的重要性，本书在这里用一个小节来进行讲述。

与用户进行正常的交流是非常重要的。交流所获得的内容将影响整个软件的最终形态，而交流方式则将决定是否可以获得完整适当的内容。

2.3.1 交流四要点^①

下面将针对交流中应该注意的4个要点进行详细的介绍。

1. 要点一：交流中的心态定位问题

首先，我们要让用户意识到：我们是为他们工作的。

同时，还要让他们意识到，我们的工作不会造成他或她被裁掉，我们只是为了简化他们的工作，减轻他们的工作负担，让他们可以更舒服，更轻松地工作——即使，最终结果可能

^① 本节中的内容可以参看第6章软件开发中的心态对抗问题研究获得更深入的扩展分析。

完全相反。

这是一个交流中的心态问题，也是一个重要的系统定位问题。

2. 要点二：我们要为用户考虑

要让用户意识到，我们所说的每一句话都是在为他们考虑，为了让他获取更多的利益，不会因为这个系统的构建而受到任何伤害。

对于不同的用户，考虑层面不同。也就是说，一定要站在用户的立场上进行考虑：

对领导，我们要说，通过这个软件可以让他更多地了解其下属的工作情况和业绩，更轻松地掌握职务内的事情，轻松获得所需要的信息，使其领导地位更稳。

而对于一般职员，我们可能要说，通过使用这个软件，使你本来很繁重的工作，或者自己完成不了的工作可以很轻松地完成，不再需要找其他人帮忙。这个软件系统将提高你的工作效率，增加你的工作业绩，领导也将因此对你刮目相看。

总之，要用巧妙的方式来引导用户进入你的构想中，让他们能够积极地配合你的工作，这样才能真正达到自己的目的。

注意：

为用户考虑一定要真正全身心地投入，而不是只把这个词挂在口头上！虚情假意只能蒙蔽一时，而且很容易被看穿，要长久地得到用户的信任，就应该全身心地投入。只有长久地得到用户的信任，才有可能不断地获取信息，也更有可能从用户那里得到更多的项目或更多的合作机会，这就是可持续性发展战略与策略的完美统一。要知道可持续发展不只在于人与自然的关系上，任何地方都有它的生存之地。

3. 要点三：采用适当的交流语言

要用方便、易懂、适当的语言来和用户交流。所谓方便、易懂、适当的语言，是指尽量采用用户行业内的语言或者用户可以理解的语言和方式。具体这些方式有哪些，下面笔者就做些抛砖引玉的介绍，大家以此为基础，也许能找到更合适有效的方法。

(1) 思路清晰的流程图

通常交流的对象对其本行业应相当了解，而且对流程图也有一定的了解，从浅显易懂的流程图，用户很容易看明白你所要表达的意思。

所以，使用表述清晰的流程图，可以得到比任何语言文字性的描述都更准确、更好的效果。不需要做更多的解释，只需要将画好的图让用户看，然后对每个操作/环节做出相应的简单而详尽的解释/注释，让用户明白这些图的确已经完整地表达了其工作过程。

最常用的方式是：

用户边描述，需求人员边绘图，然后把绘好的图拿给他们看，让他们根据图来指出问题，然后，再修改，再看，如此反复，直到用户满意为止。

如图 2-2 所示，描绘的是某一行业一项业务的流程图。

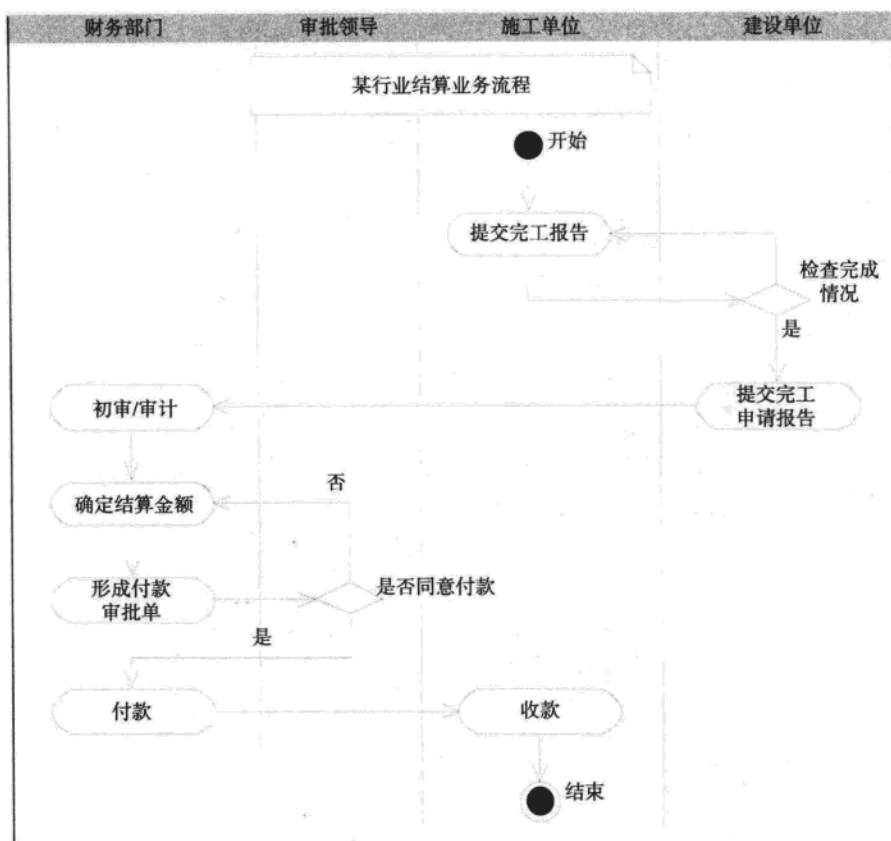


图 2-2 某行业业务流程示意图

(2) 清晰明了的表格

我们使用流程图来描述工作流程，通过表格可以描述工作的结果，或者阶段性成果。用户可用这些表格来表述他们的业绩，提升他们在公司的地位。这样，他们就会非常乐意和你进行详细的讨论，直到表格达到他们的要求为止。例如，本书后面所使用的表 2-1 中，可以清晰地说明要阐明的的问题。在这里，读者不妨先去看看那个表，是不是可以很美观很简洁地把问题说清楚呢？

(3) 形象美观的图形

除了流程图外，可能还需要绘制一些其他的图，如设备部署图、网络结构图、资源分布图等，这些图形一定要绘制得美观、精致、准确，使用户看到后会感到很舒服，很真实，如图 2-3 所示。

(4) 必不可少的文字

对于任何工作而言，文字叙述都是必不可少的，因为只有文字才可以最直接、最简单地

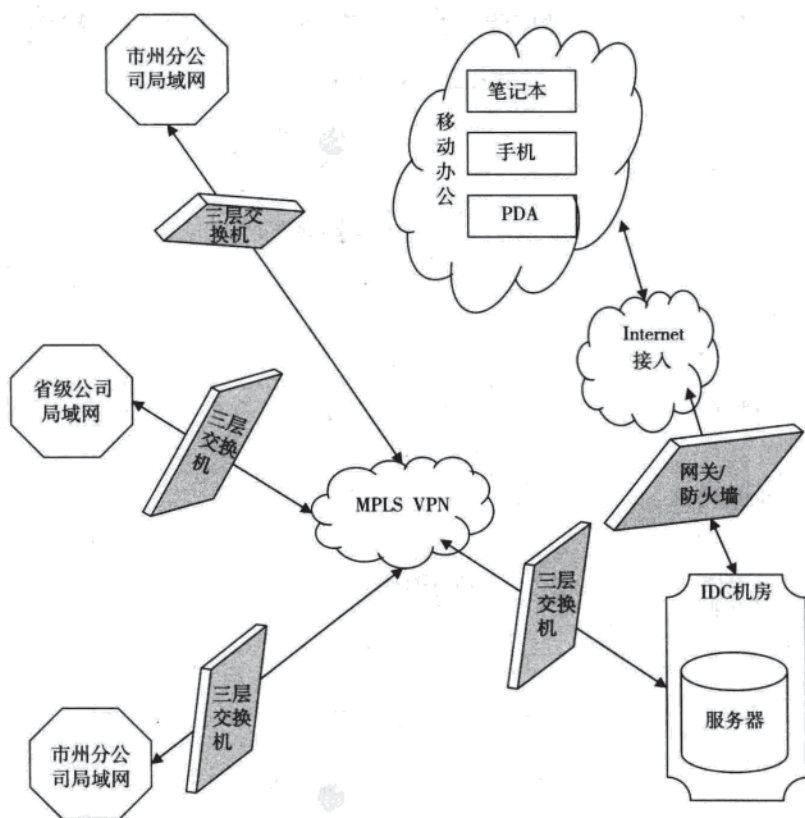


图 2-3 某行业业务实现网络拓扑示意图

完成对需求的描述。这也是最容易实现，信手拈来的——也是过去传统的需求调研工作的表达方式。

图形给用户一种直观明了的整体感觉，而用文字做一些细节性的特性介绍。因此，图形要尽量表达完整清晰，文字要尽量简洁明了。

需要强调的是：文字越少越好，描述性文字一定要精炼、短小，切忌长篇大论，避免用户对此生厌。没有人用很长时间去看大段的文字，因为用户工作都很忙的。如果这样的事情发生一次，用户就会对交流而产生抵触心理。

4. 要点四：交流的记录和整理

最后，在每一次交流过程中都要进行文字记录，在交流结束后，要尽快（两个小时内）整理所有的交流会议记录。

交流过程中，在获得用户的同意后尽可能用录音笔进行录音，其目的是为了防止笔头记录和大脑记忆的缺失（确认交流内容的完整性），防止多个参与交流的人员之间发生争议而无法确定正误（个人理解的片面性）。

2.3.2 什么是用户

用户与客户这两个词被很多人很多文章误用为同义词，而实际上他们只是近义词而已，下面我们来看看这两者的根本区别。

在英文中，简单说来，用户是 User，而客户是 Customer；在字典中，用户是 a user (of utilities, for example) 或者 subscriber to a service、a consumer，客户是 [Commerce] a client、a customer、a connection、a correspondent。

1. 什么才是真正的用户

前面已多次提到过用户这个词，那么什么才是真正的用户呢？下面我们来详细进行界定。

什么是用户？用户包括哪些人？哪些人在什么时候可以成为用户？每个阶段我们的用户包括哪些人？如何有重点地抓住用户的主体，让我们的工作能顺利地进行下去呢？

说到用户，每一个阶段用户对象的主体范畴是有差别的，每一种项目中的角色所对应的用户也是有区别的，这一点不能孤立不变地看待。

1) 在签订意向合同或者协议的时候，签字方就是用户的主体。

这里其实在概念上我们应该称之为客户。

2) 在进行需求调研的时候，提供需求的人也就是将来直接使用系统的人就组成了用户的主体。

签字方和付款方当然也是用户的一个组成部分，但在这个阶段，我们的对象用户不能以他们为主体。确切地说，签字方和付款方应该是客户的一个组成部分，用户也是客户的一部分。

3) 开发过程中，用户的主体还会有一些变化。

比如开发人员的用户可以由需求人员和使用方的相关人员共同组成的。而需求人员的用户，主要是系统购买方的直接使用者。

4) 市场人员的用户将是系统购买方的高层管理人员到使用者，尤其是高层管理人员，也就是有权付款的人，这些人就是市场人员所面对的用户主体。

需要注意的是，用户和客户这两个词其含意并不完全一样：

对于软件开发人员来说用户应该是软件系统的直接使用者。

而客户，对于软件开发人员来说应该是软件系统使用方的全部人员，其中包括用户，也包括很多与系统使用根本无关的角色。

所以，提到用户，应该针对不同的阶段进行分析。

2. 用户组成情况

如表 2-1 所示为对应阶段每一种参与角色与用户组成情况。

表 2-1 阶段角色与用户范围

阶段名称	阶段描述	参与角色	用户主体	用户范围	附加说明
意向方案阶段	项目达成意向前的阶段，这段时间包括招投标过程等	售前技术支持人员	招标方技术人员、起决定作用的人	招标方的全部人员	用户范畴中还包括一些需要调用的关系，比如朋友、亲戚等
		市场人员	招标方商务人员、起决定作用的人		
		公司高层管理人员	招标方起决定作用的人		
需求调研阶段	项目签订后，就需要进入这个阶段。在这个阶段，需要进行需求的收集和最初的整理	行业专家（如果没有行业专家，则省略此项）	用户群中使用系统的人员、投资者	系统使用方和投资方的全部人员	
		系统分析人员（主要进行需求调研整理工作）	用户群中使用系统的人员、投资者		
		市场人员	投资者		
分析阶段	完成初步的需求收集整理后，对需求进行分析的过程	行业专家	用户群中使用系统的人员	能够对系统提供使用信息的人员	这个阶段是为了向设计转化的阶段 注意： 这里的行业专家、系统分析员可能是代表用户存在的。 以下同
		系统分析员	用户群中使用系统的人员		
		分析设计人员	行业专家、系统分析员		
设计阶段	对分析结果进行设计建模的实现过程	分析设计人员	行业专家、系统分析员以及部分用户中的高级技术人员	能够对系统提供使用信息的人员	这个阶段是为了向编码实施转化的阶段
实施阶段	对设计结果进行实现的过程	分析设计人员	行业专家、系统分析员以及部分用户中的高级技术人员	能够对系统提供使用信息的人员	
		编码实施人员	用户群中使用系统的人员（系统的直接使用用户）		
测试阶段	对实现的成果进行测试的过程	测试人员	用户群中使用系统的人员（系统的直接使用用户）	能够对系统提供使用信息的人员	

(续)

阶段名称	阶段描述	参与角色	用户主体	用户范围	附加说明
β 版测试阶段	进行外部公开测试的过程	测试人员、编码实施人员、分析设计人员	参加测试的用户 行业专家	所有对用户业务熟悉的人	
		市场人员	用户群中参与测试的高级管理人员		
验收测试阶段	用户参与的系统最终验收过程	项目开发人员、行业专家、技术支持人员、市场人员、公司中高层管理人员	用户群中使用系统的人员 用户中的决策人员(投资者)	系统使用方的全部人员	
后期维护阶段	系统开发结束后的系统运行日常维护过程	技术支持人员 项目开发人员	用户群中使用系统的人员	系统使用方的全部人员	
版本升级阶段	版本升级的过程	技术支持人员 项目开发人员	用户群中使用系统的人员、用户中的高层管理人员、投资者	系统使用方和投资方的全部人员	涉及后期的升级维护和付费等问题

注：α 版测试为开发组内部测试，一般不需要用户方的参与，所以上面没有这个阶段。

3. 常见错误和习惯

通过上表中的描述，可以看到在不同的阶段用户的主体会不同，这与开发方的角色有着密切的关系。

前一个阶段都是在为后一个阶段做准备，前一个阶段的用户范围会直接影响后一个阶段的用户范围。

用户和客户这两个词的确很难区分，即使在 RUP2002 中文版中也仍然存在同样的问题。举个例子。

在 RUP2002 中文版的“业务用例指南”中有下面这样一段话：

确定合适的主角时，首先指定至少 2~3 名能担任所讨论主角的人，然后严格评估每个人需要什么样的支持。例如，假定你最初确定了一个名为“客户”的主角，而后，随着你仔细考查每个客户所需的支持，可能发现三种不同的客户：产品的一般“用户”、“采购员”和“评估员”，他们能够将产品与其他同类产品进行比较。每种客户都需要一个单独的业务用例，因为它们代表了作用于业务的不同角色。

上述的描述其实缺少了一个环境范围的限制。现在，可以假设一个环境范围，只有这样，我们才可能明确地区分出用户和客户的关系。

这里，我们假设的环境范围是：该软件是一个产品销售管理信息系统。

在上面这个假设的前提下，上文中所提到的“客户”就应该是指通常意义下的用户，因

为它不包括用户群中的高层管理人员（诸如董事长、董事、总经理、部门主管等管理角色）、市场人员等与系统使用无直接关系的人员，它只代表直接使用系统的相关人员。

然后，RUP2002 表达了这样的观点：这个“客户”分为三类，用户（其实是指用户的用户，也就是购买商品的人，而不是指购买/使用软件系统的人，这些用户的用户可能会直接操作本系统，也可能根本不知道本系统的存在）、采购员（用户之一）、评估员（用户之二）。

由此可见，很多英文在翻译成中文的时候，大都存在着类似的问题，而笔者过去也曾经犯过类似的错误。

通过上面这个例子，相信读者应该可以区分客户和用户的差别了。

2.3.3 参与人员建议

1. 主要开发人员

行业专家、系统分析员（需求调研人员）、美工（至少一位，最好是需求调研人员中具有美工基本知识的人）。

2. 主要审核人员

用户代表（用户）、行业专家、项目经理。^①

2.3.4 输入输出

1. 输入工件

需求调研工作没有具体工件的输入，因为它是整个工程活动的第一步。

2. 输出工件

用户方相关业务规范文档、业务操作手册、用户方岗位职位说明书、调研记录和用户确认信息。

2.3.5 UML 元素

1. 相关 UML 元素

Use Case、Use Case 图、Use Case 模型、状态/活动图。

2. 用处

这些 UML 元素全部用于绘制图形与用户进行直接交流时使用，并将绘制后的图形贴于相关的调研记录和文档内。

^① 注意：在本书中各章节的参与人员建议中都有项目经理这个角色的存在，这里的项目经理的主要任务是审核的协调工作，让评审会/审核工作能够更有效地开展起来。这也同时要求项目经理在本项目中最好不要从事任何一项具体的开发工作，最好是作为一个全职的管理人员而存在。如果做不到这一点，那么就应该准备一个人员可以顶替该项目经理在这项活动中的角色。

2.3.6 常见问题

1. 问题一：响应变更要求

在用户确认他们的说法之后，可能还会有变更，所以，在合同没有正式签订前，不要让用户感到存在任何的变更困难。当然签订合同后，对于合理的变更要求也要给予考虑。

对于这种变更要求的接受必须有一个度的考虑，那就是：如何拒绝用户的不合理要求。

2. 问题二：拒绝不合理要求

当用户提出一些不合理的要求时，要通过正当的、合理的方式来进行拒绝。

首先，要通过引导，让用户心里明白这个要求的不合理成分，或者这个要求暂时是无法实现的，或者是超越了当前阶段的实现范围。让他自己主动地放弃这个要求，只要他今后不再重新提出这个要求或者在你私下里问到的时候，他主动告诉你说不再提这个问题，你就成功了！另外，在适当的时候用户提出的合理变更要求也需要拒绝，这些被拒绝的变更要求可以在你拒绝不合理变更要求的时候作为谈判和缓和气氛的砝码来使用。

记住一点：

切忌逼着用户认输认错，逼着用户当场说：自己的要求不合理，自己的要求错了。那样的话就是你错了！——得罪了用户，你后面的日子是绝对不会好过的。

虽然做到这一点很难，但是，这是必须要做到的！否则，你的项目就注定要失败！但是，在进行这种拒绝的时候，一定要考虑清楚，千万不要没有考虑清楚就下结论，如果让用户感到你的无能，那也是你末日来临的时刻……

2.4 业务建模

完成需求调研工作后，将开始对需求进行分类和整理。然后，通过 UML 绘制的模型来完整、适当地对需求进行描述。

本书到这里才真正开始了建模的内容，也是软件工程过程中建模工具和语言第一次介入到工程过程中的阶段。

2.4.1 目的

这里借用一下 RUP 中的有关描述来做引子，RUP 中关于业务建模目的描述如下：

- 1) 了解目标组织（将要在其中部署系统的组织）的结构及机制。
- 2) 了解目标组织中当前存在的问题并确定改进的可能性。
- 3) 确保客户、最终用户和开发人员就目标组织达成共识。
- 4) 导出支持目标组织所需的系统需求。

为实现这些目标，业务建模工作流程说明了如何拟定新目标组织的前景，并基于该前景来确定该组织在业务用例模型和业务对象模型中的流程、角色以及职责。

下面对上面的内容逐条进行解释说明：

1) 了解目标组织（将要在其中部署系统的组织）的结构及机制。

如果需求调研人员希望在用户方很好地开展需求调研工作，以完成业务建模的目标，首先，需求调研人员必须要了解用户方本身的结构和机制。即理论上用户相互之间的层级关系、运作机制及其内部的实际关系组成。也许，某位负责人的亲属或关系很好的朋友就在所在调研的部门工作，通过他的影响，需求调研人员也许可以使得整个项目自上而下地推动起来，而如果忽视了这个人的存在，就可能受到意想不到的打击和阻碍。

除此之外，需求调研人员还需要清楚，这次工作需要和哪些部门进行衔接，他们的负责人和执行者是谁。

不要奢望这些负责人来配合你的工作，一般来说他们会指定由某个人来配合你的工作，也就是执行者。

2) 了解目标组织中当前存在的问题并确定改进的可能性。

这个过程是要在调研过程中进行总结和发现问题。一般来说，用户经常提到的关于工作中比较繁琐的事情，或者一些经常遇到的问题，就是需求调研人员需要关注的要点，对这些问题的关注会让用户感到很欣慰，他们会认为你的确是一位很好的听众，也就更愿意将一些烦恼的事情讲给你听。

这样的倾听可以拉近你和用户的关系，除了了解到用户正常的工作规程之外，还可以了解到用户习惯的一些变通的工作规程，这些变通的工作规程往往是业务系统开发中不可或缺的关注点，也就是这几年软件开发中的热点之一——用户体验的来源。

这个时候，需要通过流程图来表现用户的工作流程。

对这些问题不要妄自揣测，一定要让用户进行确认。同时，注意不要任意对问题扩大化，这会增加你的工作量，还不会得到用户的认可。尤其是后者发生的时候，你会遇到很尴尬的场面，如果处理不好，就会和用户产生意见分歧，最后难以解决。

3) 确保客户、最终用户和开发人员就目标组织达成共识。

在需求调研人员进行业务建模的最后，就应该要求用户对需求调研人员的工作结果进行确认，即达成共识。让所有的人对他们曾经提出和想到的信息做最后的确认，这也是减少需求变更造成系统开发延误的一种前期准备手段。

这句话还表达了如下几种含义：

第一，客户和最终用户是不同的。

第二，管理者和使用者要同时与开发人员就目标组织达成共识。

第三，需求必须是明确的。

第四，需求必须是稳定的。

注意：

这是一项非常关键的操作步骤，其结果将对后续工作有着很大的影响，一定要小心处理。

4) 导出支持目标组织所需的系统需求。

这是业务建模的最终结果,业务建模完成后,需求调研人员应根据业务建模时收集并形成的信息完成系统需求。到此,开发方才获得了需求的第一次整体性描述。

2.4.2 基本操作步骤[⊖]

业务建模的实现过程是一项相对较为具体的工作,它不像需求调研工作那样抽象。具体实现过程分以下几步完成:

1) 对用户的业务进行详细的了解,并绘制流程图。

流程图一定要根据用户提出的工作流程进行绘制。流程图中要把用户提出的主要工作内容表示清楚,并加上一些针对用户问题的注释和说明性文字。

在回到公司或开发场所后需要对流程图进行整理,要把用户的业务流程进行一定的规范化,提取重要的数据信息和用户期望的内容,以便使流程更加清晰。

注意:

流程图将会影响界面原型的开发,尤其是在业务流程实现中的界面关联关系上。

要特别注意的是,最后在离开之前,此次调研的所有工作记录一定要让用户进行确认。需求研讨过程最好有录音记录,以便核查。

2) 通过流程图与用户进行交流,记录用户的意见,并修改流程图,由此反复2~3次。

再次与用户交流时,让用户对已绘制的流程图重新进行确认和补充。

当有用户对流程图提出看法和意见时,要将他们的看法和意见记录下来(并记录该修改意见由谁提出),然后对流程图再次进行修改,直到符合用户要求。

注意:

此时,用户可能会推翻上一次提出的业务流程图,提出一个新的过程,发生这样的情况,一点都不奇怪。原因请看2.4.3小节中的那个关于寄信例子的描述。

这些反复过程中的方式和方法与第一步中的相似。但是,对业务流程进行第二次以上交流必须考虑方式方法,不能以用户重新讲述的方式进行,需要以提问和不理解等应答方式与用户进行交流,以免给用户留下不专业或者记录疏漏的印象。

3) 由用户对业务流程图进行确认。

除了每一次讨论都要进行确认外,在调研的最后一定要进行一次确认,这次需要用户进行签字确认。这时候,如果用户的地位太高,那么往往开发人员可能拿不到这样的确认,这就会给后来的需求变动造成早期的隐患,但是地位太低,又可能造成说出的话和签下的字没有效力,所以一定要小心!

最好的办法是,想尽一切可能利用的手段,拿到一个比较合适有效的确认。

[⊖] 这里的实现步骤是笔者建议的操作步骤,与RUP没有什么关系,大家要根据情况来选择使用。

4) 对业务流程图中的每一个环节进行详细的描述,并提炼出相应的操作步骤和相关的岗位信息。

这个过程是在对业务需求操作进行整理的过程中完成的。整理过程将会影响系统权限的设计。

对于每一个环节的操作都将受到权限的控制,而这些环节又将与地域、部门、专业以及其他类别的关系相关联形成一个完整的受控权限管理机制。

5) 对操作和岗位的设置与描述同用户进行交流,并反复修改。

该步骤与界面原型设计^①相关联,和用户的习惯性操作方式有很密切的联系。岗位设置是需要用户进行确认的,要与用户目前的职位相关联,但是,又不完全相同。这是因为用户中的每一个职位可能会包括几个岗位的职责。

6) 由用户对操作和岗位的描述进行确认。

这个步骤的作用与第3步的作用是完全相同的,这里就不再赘述。

2.4.3 建议

参与业务建模的角色应该包括:行业专家^②、系统分析员和用户。

这里的角色是按照国内目前的分布情况进行划分的,而不是按照 RUP 中的建议。

基于目前的现状(也就是基于 MIS 系统开发中的业务建模问题),我们对业务建模有如下的建议:

1) 尽可能使用流程图绘制用户的业务流程,抓住业务环节。

在绘制业务流程图时可以回想一下处理一些模式过程的详细步骤。

例如日常生活中,我们为了给外地的朋友寄一封信可能采用下面几种方式之一:

第一,我们先写好信,然后到邮局,购买信封和邮票,贴上后放到邮局的信箱中。

第二,已准备信封,只需要购买邮票,然后将信投入附近的信箱中即可。

第三,或者已有信封和邮票,不需要去邮局了,直接将信投入附近的信箱中即可。

第四,又或者,你也可以将信交给一个朋友,让他帮你带到邮局寄出去。

这么简单的一件事情,就有至少4种解决方式,也许还有人能提出更多的方式。

同样的,用户在进行某项工作的时候也会有很多种解决方法和实现过程。到这里,相信

① 界面原型设计:界面原型设计与第1.8节的界面设计不一定相同。界面原型设计会产生多种界面方案,这些界面方案可能会有一种被选定为最终的界面设计雏形。但是,界面原型设计同时需要实现部分后台的功能,而界面设计仅仅是界面层的设计,不包含任何后台功能的实现。

② 在中国,目前最缺的就是行业专家。而在目前的实际操作过程中,为了弥补这个角色的缺失,往往是由系统分析员和用户配合起来共同承担起这个角色应有的作用的。

不过,中国软件业企业已经有将近15年的发展历史了,而行业软件的开发到现在,大都还仍然处于 MIS 系统开发的阶段,也就不必说什么专家系统或者辅助决策支持系统了。笔者也希望真的能够在国内看到真正的更高级别的系统。笔者目前也在为达到这个目标而努力。

你可以明白为什么用户有时对他们工作流程的描述会有那么多的不同之处或者用户的说法为什么会一天一变,甚至说不清楚他们自己的实际操作流程。——这也就是为什么经常有一些开发人员认为需求变化大的一个很重要的因素。

在这里,开发人员要做的就是用流程图一步一步地引导用户把他们认为最完整的流程,尤其是流程中的每一个不可再分的操作步骤(这里可以称之为“业务环节”)都提出来。这样开发人员通过对这些业务环节的一次次重组就可以实现整个业务流程的过程描述,从而满足用户各种不同的实现流程。

2) 关注流程的描述,但不沉没于流程的实现中。

MIS 系统的重点在于业务环节的实现,也就是业务中数据信息的记录和表现。

要给用户一个自主重整流程的机会。不要想一次就将用户的每一个业务流程都整理清楚,因为那是不可能的(从前面也可以看出来)。其实在 MIS 系统中,完全可以通过一张码表将所有的环节串起来,实现重组全新的业务流程。

掌握以上两点原则,开发人员就基本上可以跳出业务流程频繁变化无法控制的怪圈。

2.4.4 参与人员建议

1. 主要开发人员

行业专家、需求调研人员、业务建模人员、美工(1位,最好是需求调研人员中具有美工基本知识的人)。

注意:

一般来说需求调研人员和业务建模人员是同一组人。

2. 主要审核人员

用户代表(用户)、行业专家、项目经理、Use Case 阐述人员。

注意:

一般来说 Use Case 阐述人员和业务建模人员是同一组人。如果是,那么评审人员中的这一角色应该由有相应能力的开发人员来担任。

2.4.5 输入输出

1. 输入工件

用户方相关业务规范文档、业务操作手册、用户方岗位职位说明书、调研记录 and 用户确认信息。

2. 输出工件

用户交流记录、前景文档(也就是 RUP 中的 vision)、需求规格说明书、补充规约(功能和功能性要求)、词汇表(可有可无)。

2.4.6 UML 元素

1. 相关 UML 元素

Use Case、Use Case 图、Use Case 模型、状态/活动图。

2. 用处

与上一小节相似，这些 UML 元素全部用于绘制图形或与用户进行直接交流时使用，并将绘制后的图形贴于相关的调研记录和文档内，主要是需求规格说明书、补充规约，初步完成 Use Case 模型，用状态/活动图清晰地描述用户实际业务操作流程。

2.4.7 常见问题

1. 问题一：Rose 的分析结果给谁看，用户看什么

问：当客户看不懂用 Rational rose 做的需求分析时，是否只能由专业人员看呢？

答：在本书第 2 章的 2.3 节和第 3 章的 3.1 节中针对这个问题进行过相关的描述。

这里需要提醒注意的是：第一次无论如何不要将 Use Case 交给用户看。一般来说，用户是不会主动去学习和理解 Use Case 的。

那么，开发人员能直接拿给用户看的是什么呢？开发人员可以将由用户所描述的业务过程绘制出来的表格和流程图等内容拿给用户看，这些才是用户真正必须看到的，也是用户完全可以看懂的。他们习惯的是流程，所以开发人员一定要清楚地描述用户的流程，让他们看到自己的工作过程是如何通过计算机展现出来的。就像在 CSDN 上的一位开发者在一个帖子上曾经说道：“第一次看到用 UML 绘制出来的自己熟悉的 C++ 类图，居然激动得哭了起来。”

在经过了一段时间的交流后，开发人员与用户之间也较为熟悉了。这时，开发人员可以尝试引导他们来看一些 Use Case。在用户看 Use Case 的时候，最好能加上一些界面原型等辅助性材料，并积极对用户进行讲解。这样用户才会愿意尝试来看 Use Case，才可能看得比较明白，并逐渐接受这种表达方式。

2. 问题二：Use Case 模型中是否需要添加协作^①图^②？

问：曾经有人问笔者：在使用 UML 进行建模的过程中，我参照的是《统一软件开发过程》这本书，书中指出在最开始的阶段建立 Use Case 模型，那么一旦建立 Use Case 模型以后，是否需要绘制一个协作图来对整个系统进行一次描述呢？

答：这里首先声明：笔者没有看过《统一软件开发过程》这本书。不过，笔者比较仔细

① 协作：Collaboration。描述了一组对象如何进行交互来完成某个特定的功能，一个协作有一个上下文和一个交互。上下文显示了在该协作中设计的对象以及它们之间的链接，交互显示了在该协作中对象所进行的通信。

② 协作图：Collaboration Diagram。描述了对象是如何交互，侧重于空间的协作，意即明确地给出对象之间的关系（关联）。在 Rational Rose 工具中，协作图和时序图（Sequence Diagram）是一一对应，可以直接转化的。

地阅读过 RUP 的全文，并在项目中进行过相关的实施。

因此认为，最开始的阶段不应该是建立 Use Case 模型，而应该是对用户业务过程的调研和整理，有可能的话，开发人员需要充当行业专家的角色对用户的业务过程提出一些改进性的建议。其实，开发人员在任何项目的初始阶段都是这样的，要先来通过对用户的工作过程和操作进行了解来了解用户的业务状况。要做到了解甚至熟悉用户的业务过程，有些时候会要求开发人员比用户还要熟悉他们的业务过程（当然，这并不代表开发人员可以代替用户来进行这项工作），这需要在了解的过程中必须进行反复地交流。

问：那么，交流的方式是什么呢？

答：建议的做法是：

通过 Rose 的状态/活动图^①进行描述，也就是流程图。通过这种方式对用户的需求进行建模，然后进一步从流程中提取相应的 Use Case——这应该是 Use Case 的来源了。详细情况请参看本书 2.5.2 小节中的内容。

3. 问题三：需求面向谁，如何衔接

问：需求分析人员既要面向最终用户，又要面向设计人员。但是需求文档编写过程中，文档的内容应该是侧重于用户，还是侧重于设计人员阅读呢？或者是否要写两份文档（两份文档之间的衔接关系又该如何处理）？

答：从项目的全过程或者从需求工作的全过程来看，需求工作实际上应该分为两个阶段：第一个阶段是需求获取阶段，第二个阶段是需求分析阶段。

任何事物的发展过程都可以看做是从一无所有起步的。对于软件项目来说，第一步就是获取需求，在 RUP 中对应的就是业务建模过程，该过程要考虑用户的实际工作流程，对它们进行收集和整理，这些内容是用户可以看懂的，应该说，是完全侧重于用户的阶段。这也就是需求获取阶段的主要任务。

在获取业务需求后（不一定需要完整的业务需求才可以进行下一步的工作，但最后一定要获得完整的业务需求），开发人员就可以开始对需求进行了，分析的过程其实就是需求深化理解的阶段，这个阶段的最终成果就是我们在第 3 章的 3.3 节中提到的分析模型，当然，还包括阶段中对 Use Case 阐述的交互建模部分。这个阶段的内容是侧重于技术人员的，也应该是高级用户能够理解的部分。这就是需求分析阶段的主要任务。

关于两者的衔接问题，请看下文中的描述。到这里为止，我们仅仅完成了需求获取和业务建模两个阶段的工作，需求分析阶段的工作还没有开始。

① 状态/活动图：Statechart/Activity Diagram。这里面包含了两个概念：状态图和活动图。状态图：显示了对象的生命周期（也是子系统和系统生命周期）。状态图显示了事件（消息、时间、差错以及状态改变）是如何随着时间影像对象的状态的。活动图：以正在完成的工作为焦点显示系统的交互信息。活动图展示了在完成改工作而进行的一系列动作，包括交替的执行和对象。

本书认为：状态/活动图来源于传统的流程图的概念。

2.4.8 示例说明

如图 2-4 所示是笔者曾经完成的在需求获取过程中调研得到的一项业务的流程图，为了保密起见隐去了部分关键性业务词汇。

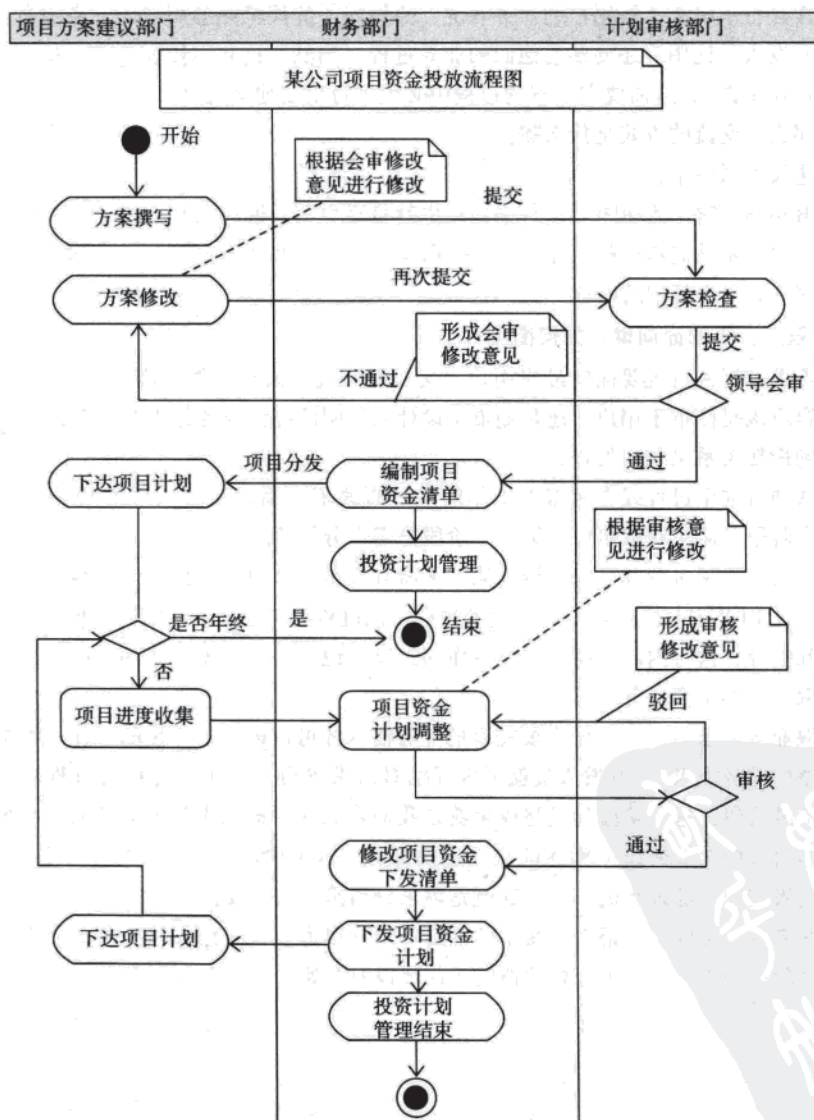


图 2-4 某公司项目资金投放流程图

当然同样的图还可以使用 Word 的绘图功能或者 Visio 等绘图工具来绘制，不过为了能够

使工具统一,使得尽可能多的图形在开发的全过程中都有效,而且在开发过程中打开多个工具来完成一个任务是非常让人感到厌烦的,因此除了极个别的图形(一般是网络拓扑图等与硬件关联而与软件开发关系不十分紧密的图形)外,本书是全部采用的 Rational Rose 工具绘制。

完成了业务建模后,就可以进入需求工作的第三个阶段——绘制 Use Case 模型。

2.5 Use Case 模型

提到 Use Case 模型的绘制,首先面临的的就是 Use Case 图的问题,而 Use Case 图必须有组成 Use Case 图的 Actor(操作者)和 Use Case 这两种基本的元素。所以,下面还是按照正常的操作顺序,开始描述 Use Case 图绘制的问题。

2.5.1 Use Case 图的绘制

1. Use Case 图的基本元素

首先,开发人员需要查找 Actor 和 Use Case。实际上在前面整理流程的时候,开发人员就应该可以整理出最细化的操作者和 Use Case,也就是细化到了极点的 Actor 和 Use Case。目前需要做的就是合并它们,并提取真正的 Actor 和 Use Case,请参看 2.5.2 小节中的阐述。

注意:

在 RUP 中,Use Case 图的绘制仍然属于业务建模阶段,但是在这里,我们对此进行了划分,这种划分方式是基于笔者的工作经验和一些习惯进行的,仅供参考。

2. Use Case 间的关系

Use Case 间的关系通常采用单向关联箭头(unidirectional associate)进行表述。或者采用依赖示例箭头(Dependency or instantiates)和范化箭头(Generalization,或者称为一般化)表述。

- 1) 单向关联箭头表述的关系有:include、extend、communicate、realize 和 subscribe。
- 2) 依赖示例箭头表述的关系有:derive、extend、include 和 refine。
- 3) 范化箭头表述的就是一般化的关系,在 Rose 没有定义更多的扩展。

下面主要描述使用最多的单向关联箭头的作用和功能,其他两种表述方式读者可以参考相关的 UML 学习类书籍,这里就不再赘述。

单向关联箭头是最常用而且用处最多的一种关系表达方式。它可以表示 Actor 和 Use Case 之间的关系、Use Case 与 Use Case 之间的关系,这种关系在所开发的系统中是实际存在的,也只有这种关系可显示在 Use Case 树形图上,如图 2-5 所示。

从图 2-5 中还可以看到,在右侧的树形图中显示出来的只有网上申报 Use Case 与 NewUseCase2 之间的关联关系。

下面简单介绍一下单向关联箭头所表示的几种关系的使用方法:

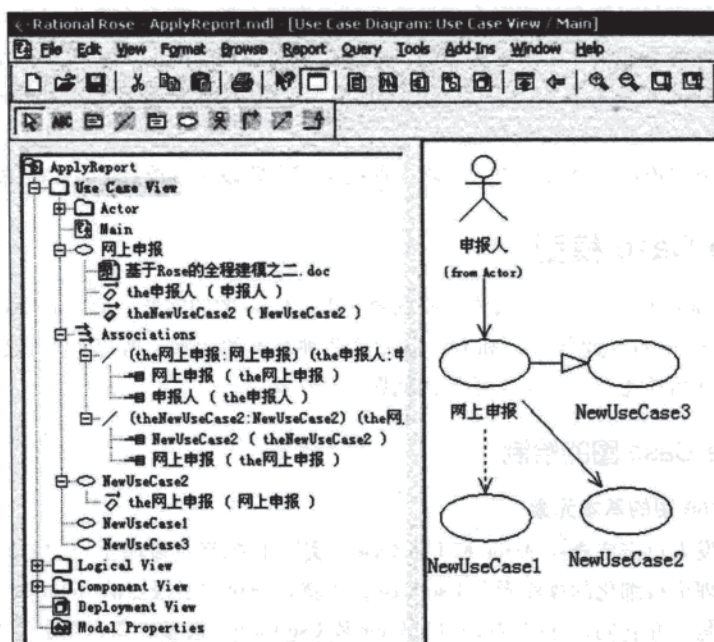


图 2-5 显示的单向关联关系

(1) include

包含关系，最常用的关系之一。

表示两个 Use Case 之间有包含关系，后者是前者的一部分，如图 2-6 所示。

当 NewUseCase1 每次被调用时，系统也会调用 NewUseCase2。NewUseCase2 其实是 NewUseCase1 的一个附属 Use Case。当然，NewUseCase2 也可能是另外一个 Use Case 的附属 Use Case，如图 2-7 所示。

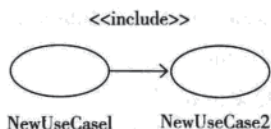


图 2-6 include 关系



图 2-7 两个 include 关系

(2) extend

扩展关系，最常用的关系之一。

extend 表示两个 Use Case 之间有扩展关系，后者是前者信息或者业务功能的扩展，如图 2-8 所示。

当 NewUseCase1 每次被调用时，系统也可能会调用 NewUseCase2，但并不是每次调用 NewUseCase1 的时候 NewUseCase2 都会被执行。NewUseCase2 只是 NewUseCase1 的一个扩展

Use Case, 关系不像 include 那么紧密。

(3) communicate

通信关系, 比较常用的关系。

表示两个 Use Case 之间有信息沟通, 但没有互相调用的关系, 如图 2-9 所示。

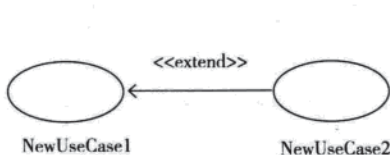


图 2-8 extend 关系

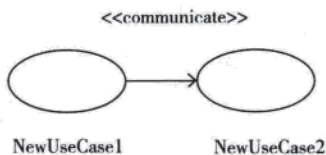


图 2-9 communicate 关系

(4) realize

表示两个 Use Case 之间有实现关系。一般用于业务 Use Case (business use case) 和 Use Case (use case, 包括 business use-case realization 和 use-case realization) 之间, 如图 2-10 所示。

(5) subscribe

订阅关系, 使用最少的一个关系。

它在 RUP 中没有详细地描述, 只是在“构造型参考”中有一条对应点。它是在实时建模时才会用到的关联关系, 这里建议用户只在需要的时候再去查找一些相关的资料。

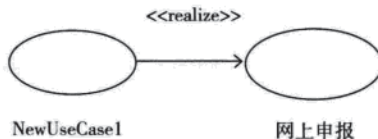


图 2-10 realize 关系

通过这些关系将 Use Case 之间连接起来就可以形成系统的整个 Use Case 图。下面就是一个 Use Case 图的示例。

3. Use Case 图示例与分析

这是一张取自实际工程项目的 Use Case 图, 如图 2-11 所示 (注: 此图在原图的基础上经过大量地删减修改, 改动了很多 Use Case 的名称和关系)。

可以看到图 2-11 十分复杂, 而且表现得十分混乱, 这样的图拿出来不仅用户看不懂, 开发人员同样也无法看明白, 而且还会造成很多连带的问题, 诸如:

- 某些应有的关系可能被忘记, 但却无法找到;
- 某些 Use Case 之间的关系绘制可能存在错误, 但却无法检查;
- 无法清晰地找到某一个 Use Case 与其他 Use Case、Actor 之间的所有关系;
- 无法清晰的体现系统的层级关系, 造成系统无法进行更深入地分析和设计;
- 混乱的表现造成后续工作无法开展。

但是, 这的确是 2002 年底一家公司实际工程 Use Case 图的最初展示相信国内还有很多类似这样的工程 Use Case 图存在。

因为上述的这些原因, 开发人员不能直接使用该 Use Case 图, 因此需要对这张 Use Case 图进行整理, 进行分包细化。关于 Use Case 图的整理与分包细化等问题将在 2.5.4 小节中进行阐述。

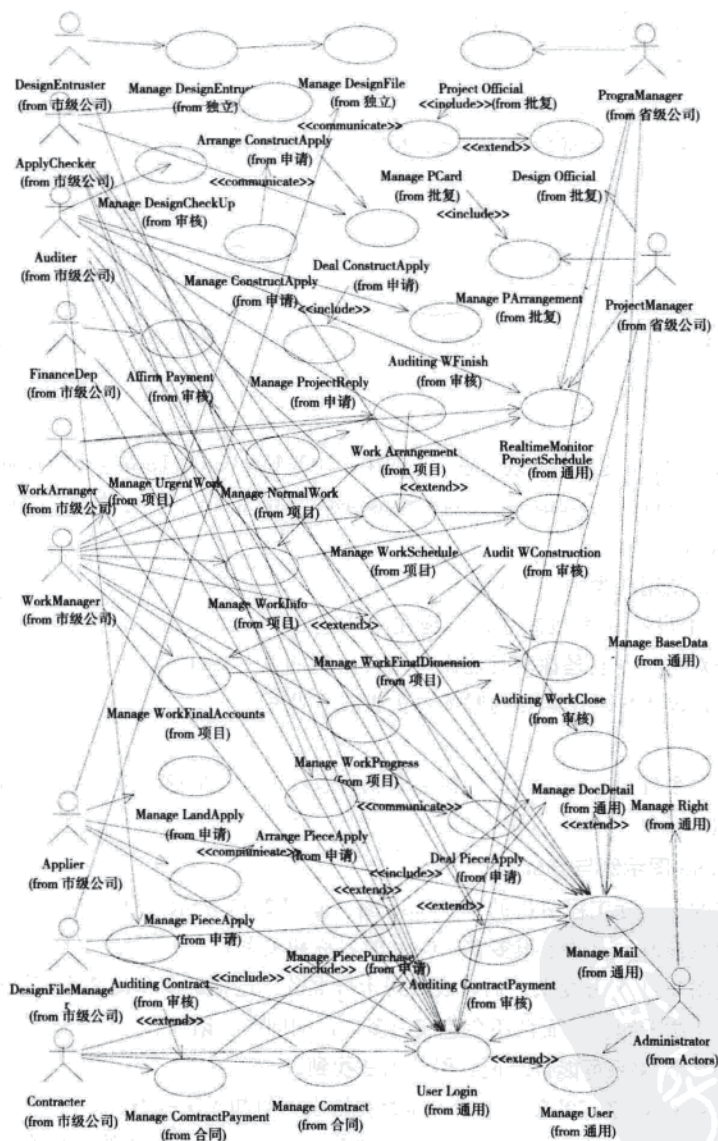


图 2-11 一个实际工程的最初 Use Case 图

- ⊖ 这张图中的 extend 关系就全部标注反了，但是为了忠实于实际工程，所以本书对这张图不做 extend 关系标注的修改。以此来提醒大家，extend 关系箭头是从后向前绘制的，正好和 include 相反。而以下各分包图则全部做了 extend 关系的修改。

2.5.2 如何查找 Use Case 和 Actor

1. RUP 的建议

这里笔者查找 Use Case 和 Actor 的方法仍然在很大程度上要依赖 RUP 中的描述, 这里笔者截取了其中非常重要的一些描述并加入了笔者的一些修改和看法:

在 RUP 中对于本节所定义的名称是: “活动: 查找业务主角和用例”。

下面是这个活动的相关描述, 笔者将按照惯例在其中穿插讲述笔者的理解和建议。

(1) 目的

- 1) 概述业务中的各个流程。
- 2) 为待建模的业务定义边界。
- 3) 定义将与业务交互的对象 (人或事物)。
- 4) 制作业务用例模型图。
- 5) 撰写业务用例模型调查。

下面将逐条进行解释^①:

1) 概述业务中的各个流程。

在这里需要进行业务流程性的描述, 其实在后文中所提到的 Use Case 阐述也就是对这项工作的一次细化。

2) 为待建模的业务定义边界。

定义边界, 也就是划分业务边界。通过边界的定义, 区分哪些应该是手工完成的, 哪些应该是由计算机自动完成的, 哪些是外部提供接口的系统完成的, 哪些是为完成这些业务功能所提供的辅助功能……通过对这些边界的确定, 也就确定了 Use Case 阐述中业务流程的内容。

上面提到的“由计算机自动进行的”, 也就是在软件系统中应该进行处理和实现的内容。

上面第 1) 和第 2) 项任务已经在 2.4 节中完成了, 也就是业务建模的主要目的。

3) 定义将与业务交互的对象 (人或事物)。

目的主要是为了选择 Actor, 也就是查找业务 Actor, 请参考下面步骤中关于查找业务 Actor 的内容。

4) 制作业务用例模型图。

绘制 Use Case 图, 这就是本节后面提到的 Use Case 的处理、如何分包的问题。完成 Use Case 模型的绘制工作。这部分内容将在第 2.5.3 小节和 2.5.4 小节中进行详细地描述。

5) 撰写业务用例模型调查。

主要是进行业务 Use Case 的实现过程调研, 然后撰写业务 Use Case 的详细阐述。这部分

^① 声明: 这些解释是基于笔者目前的理解, 肯定会存在很多的不足和缺陷, 欢迎与大家批评、指正。

内容将在 2.6 节中进行详细地介绍。

(2) 步骤

- 1) 查找业务主角。
- 2) 查找业务用例。
- 3) 确定业务用例的优先级。
- 4) 编写业务用例工作流程的概述。
- 5) 描述业务主角与用例交互的方式。
- 6) 将业务用例与主角打包。
- 7) 在用例图中表示业务用例模型。
- 8) 撰写业务用例模型调查。
- 9) 评估结果。

下面将对上述步骤进行逐条解释：

1) 查找业务主角。

查找业务 Use Case 的 Actor, Actor 一般就是对 Use Case 产生作用的外部接口或者行为者。Actor 对于这个系统具有主动性, 就是通过一些行为对系统中的 Use Case 产生影响, 使得系统发生一系列的变化。

关于查找 Actor 的详细描述请参看接下来提到的“查找业务 Actor”和“如何查找 Actor”的内容。

2) 查找业务用例。

查找业务 Use Case 是非常重要的一个步骤, 这部分内容将成为系统的主体。相对而言, 功能性 Use Case 应该说是系统的附体, 是为主体服务, 使主体可以更灵活地表现出其特性。

业务 Use Case 是对业务需求的第一步细化, 因此起着至关重要的作用。

关于查找 Use Case 的详细描述请参看接下来提到的“查找业务 Use Case”和“如何查找 Use Case”的内容。

3) 确定业务用例的优先级。

查找 Use Case 之后, 需要对 Use Case 进行难度系数的估算和优先级的排序, 这也是对需求进行优先级别的排序。这样就可以在需要的情况下根据优先级别和难度系数来分配工作时间, 项目经理可根据所选择的工程过程模型来安排具体工作任务。

由于对 Use Case 优先级和难度系数的排序计算方式和过程比较简单, 而且在 RUP 中都有很详细的内容, 所以, 本书中不做更多的描述。

4) 编写业务用例工作流程的概述。

对业务 Use Case 工作流程的概述, 也就是 Use Case 阐述的内容, 这部分内容将在 2.6 节中做详细地介绍。

5) 描述业务主角与用例交互的方式。

这是描述业务 Actor 与 Use Case 交互方式的过程，我们称之为交互建模，这部分内容将在 2.7 节中进行介绍。

6) 将业务用例与主角打包。

将业务 Use Case 和 Actor 打包的过程可参见 2.5.3 小节和 2.5.4 小节中的描述。

7) 在用例图中表示业务用例模型。

在 Use Case 图中表示业务 Use Case 模型，其实就是绘制 Use Case 模型的过程，详细内容请参见 2.5.1 小节中的描述。

8) 撰写业务用例模型调查。

这部分的内容在本书中没有很明确的对应点，不过，即使在 RUP 中这一点也不是十分重要，它所表达的内容已经在 Use Case 图中进行了表述，只是通过文档报告的方式重新描写了一遍，它所关联的关键性文档也就是整个 Use Case 模型图。

9) 评估结果。

评估只是一种审核的手段，能够起到一种监督的作用，表示开发人员每次提交的工件都是经过审核的，而不能奢望以此来提高员工的素质或者技能水平。所以，这个步骤和类似的审核步骤在本书中都没有很明确的对应内容。

因为本书的重点在于软件工程过程的几个关键点的建模问题，而不是模型的审核检查问题，所以，这里就不对评估过程做过多的描述，但是对于如何进行评审和评估可以参考本书 8.6 节的内容。

下面关于查找业务 Actor 和查找业务 Use Case 的内容是笔者针对 RUP 内容的一个概述，详细内容请用户参见 RUP 中关于查找业务 Actor 和查找业务 Use Case 的建议方式。

(3) 查找业务 Actor

业务主角备选者可以是与业务进行交互的任何个人、小组、组织、公司或计算机。你可以从以下方面查找业务 Actor：

- 客户；
- 合作伙伴；
- 供应商；
- 委员会（法律、法规等）；
- 子公司；
- 拥有者与投资者。确定是将董事会作为业务的一部分，还是将其作为主角进行建模；
- 业务之外的信息系统。

如果即将建模的业务是大公司的一部分，还可能会包括业务 Actor。

- 公司的其他部分；
- 其他部门内的各个角色。

为各业务主角命名, 用该名称来代表其在业务中的角色。撰写可定义各业务主角的简要说明, 其中应包括各业务主角的职责及其与业务交互的原因。

另请参见指南: 业务 Actor。

如何查找业务 Actor, 笔者也有一些相关的建议, 详细内容请参见接下来介绍的“如何查找 Actor”。

(4) 查找业务 Use Case

1) 为了找到主要的业务用例, 请考虑各业务主角可以从业务中获取的价值。请从主要且最重要的业务主角(即客户)开始, 考虑以下问题:

业务为客户提供的主要服务有哪些? 客户与业务的首次接触是什么? 客户经历了业务的哪些阶段或状态?

一种有效的方法是研究客户的生命周期。

2) 具有更多业务支持特征的流程(包括不直接使客户受益的活动)也可以用业务用例来表示。查找以下类型的活动:

人员的发展与维护、业务内部 IT 的开发与维护、办公室的设立与维护、安全性、法律活动。

3) 具有管理特征的流程可以用业务用例来表示, 尽管这些流程通常对于信息系统不具有重要的意义。要查找这些流程, 可以先查找与业务的整体管理相关的活动。这种流程通常会与拥有者主角进行交互。考虑拥有者主角将从业务中得到什么。查找以下类型的活动:

为拥有者与投资者开发并提供有关业务的信息、设定长期的预算目标、协调业务中的其他用例并确定其优先级、在业务中创建新的流程、监测业务中的流程。

这种流程的生命周期往往会持续一个财政年度。

4) 查找业务用例的另一种方法是请领域专家说明现有业务中的每项活动, 然后将这些活动分组, 组成业务用例。

给用例命名, 并对其进行简要说明。

另请参见指南: 业务用例模型和指南: 业务用例。

5) 确定业务用例的优先级。

业务主角与业务用例一旦确定, 就必须确定业务用例的优先级, 看哪些业务用例值得进行详细说明。其中包括以下内容:

如果执行业务工程的目的是为了发现对信息系统的需求, 则应确定哪些业务用例对预期的系统具有价值。这些业务用例需要进行详细说明(请参见活动: 详细说明业务用例)。

对于有些业务用例, 如果不能确定它们与信息系统之间是否存在关系, 那么在决定是否包括这些业务用例之前需编写分步说明。

6) 编写业务用例工作流程的概述。

为了解业务用例的目的,通常需要使用工作流程的概述。这种概述可采用分步说明的形式。相关人员(即使是同一个人)在以后具体说明业务用例时,需要使用该分步说明。

示例:

对于个人登记这一业务用例,其分步工作流程说明的初稿可能包括如下内容。

乘客在登记处排队、乘客将机票交给登记代理、登记代理验票、登记代理登记行李、登记代理为乘客预订座位、打印登机证、登记代理将登机证交给乘客、乘客离开登记处。

请注意:这只是一个初稿,它很可能会缺少一些将在稍后发现的活动的。在此草稿中也可包括备选流。

将精力集中于最重要的业务用例,即那些具有最大改进潜力的业务用例。它的规模是否会明显增加,以致最初由客户完成的工作(或没有人来完成的工作)现在由目标组织来完成?或者,它的规模是否会明显减小,以致以前由目标组织完成的工作现在由客户来完成?如果业务用例为客户提供了更好的服务,则表明业务用例得到了改进,这意味着它变得更简单、生产的产品更好、交付周期更短等。

为各业务用例设定可评测目标,以利用它们来核实自己是否成功。当在以后建立新的目标组织时,可以利用这些目标来连续地评测业务用例的运作情况与改进情况。

另请参见指南:业务用例。

7) 描述业务主角与用例交互的方式。

要确定哪些业务主角与业务用例进行交互,可通过定义它们之间的通信关联关系来进行。如果必须要表示通信的发起方,则应向该关联关系中添加导向性。

另请参见指南:业务用例模型中的通信关联关系。

8) 将业务用例与主角打包。

如果你拥有很多业务用例,就可以将它们划分成用例包,使文档更易于理解。

9) 在用例图中表示业务用例模型。

用例图用于综合说明业务主角、业务用例以及它们之间的关系。用例图中可以包括以下任何内容:

- 业务主角和所有与他进行交互的业务用例;
- 与相同业务主角进行交互的业务用例;
- 通常按顺序执行的业务用例;
- 属于同一个用例包的业务用例。

最重要的业务用例图可作为整个业务用例模型的概览图,它有助于对该模型进行复审。

另请参见指南:业务用例模型中的用例图。

10) 撰写业务用例模型调查。

业务用例的调查说明应表述以下信息：

- 所述业务的目的；
- 业务用例的典型部署顺序；
- 业务用例模型中未包括的业务部分。

11) 评估结果

在本阶段应检查业务用例模型，以核实你的工作是否按计划进行，但不必对该模型进行详细复审。在检查该模型时，还应当考虑业务用例模型的检查点，需要对下面几个方面进行判断：

- 是否确定了所有必要的业务用例；
- 是否确定了任何不必要的业务用例；
- 是否按正确顺序说明了各业务用例的行为；
- 各业务用例的工作流程是否达到了现阶段可能具备的完整程度；
- 业务用例模型的调查说明是否明白易懂。

有关复审的详细信息，请参见检查点：业务用例模型、检查点→业务用例和检查点→补充业务规约。

关于如何查找业务 Use Case，笔者也有一些相关建议，详情请查看接下来介绍的“如何查找 Use Case”的内容。

2. 建议

(1) 如何查找 Use Case

在前面的流程整理过程中，开发人员整理出来的环节，每一个针对环节所存在的操作，都可以看做是一个细化到极点的 Use Case。

在 MIS (Manage Informations System) 系统中最重要的就是信息 (Informations)。开发人员要尽可能地将所有可用的信息收集起来，然后将目前所需的信息展现出来，其他有用信息可在今后进行专家系统或更高级的决策支持系统的开发中使用，这也是对初期管理信息系统一个扩展性功能的支持。因此，所有的业务 Use Case 都和信息的收集有关。另外，还包括一些系统 Use Case，诸如用户管理、权限管理、日志管理等。

(2) 如何查找 Actor

Actor 的查找和 Use Case 相似。每一个操作都会对应于一个操作者，这样开发人员就可以获取最基础的最细化的 Actor 了。

到了这里，就完成了最基本的 Use Case 和 Actor 的查找。而实际上这样查找出来的 Actor 太多了，因此需要进行一些合并操作后才能获得真正有用的 Actor。

但是因为 Actor 将不会形成系统内的任何部分，因此它的合并相对比较简单，只需要将同类的用户或者进行同样操作的角色进行合并就可以了，不需要做过多的考虑，但是也不能合并得过多，要根据具体的情况进行考虑合并。

但是,如果有一些不应该存在的 Actor 存在,那么就会影响到系统的边界划分问题,因此需要特别注意。

(3) 查找步骤

1) 提取环节。

在 MIS 中,开发人员可以从绘制的流程图中提取出开发所需要的所有重要环节,这些将组成 MIS 系统的基本原型。

注意:

这些环节都是在业务流程中不可或缺的,是在 MIS 系统中不可不存在的。

2) 理解目标。

由于系统管理的是信息,进行基础数据的收集与整理,对这些基础数据进行处理是本阶段要实现的目标。

3) 划分系统边界。

划分系统边界时注意系统需要实现的部分、被调用的内容和需要从外部调用的内容。通过这三者的划分就可以获得完整的系统边界,从而有效地提取出业务 Use Case 和 Actor。

4) 提取业务 Use Case 和 Actor。

从这些环节进行考虑,然后提取出 Use Case 和 Actor。例如,Use Case 有:电子邮件撰写、邮件查看、合同录入、合同修改、合同删除、合同查看;Actor 有:邮件用户、合同管理员等。

5) 提取系统 Use Case 和 Actor。

用于系统实现所必需的管理性 Use Case,这些 Use Case 不直接存在于业务中,不对业务过程产生直接的影响,或者是由业务以外的信息获取到的。例如,基础 Use Case 有:用户添加、用户修改、用户删除、权限授予、权限收回等;从业务 Use Case 中扩展出来的有:合同查询、邮件查询等;相关的 Actor 有:系统管理员、区域管理员、一般用户等。

2.5.3 Use Case 的处理

Use Case 的处理主要是描述如何规划 Use Case,即对 Use Case 的大小进行界定,并进行 Use Case 的拆分和合并,这时就会涉及一个很重要的问题:Use Case 的粒度。

下面对 Use Case 的粒度、合并与拆分进行详细介绍。

1. Use Case 的粒度

到底将 Use Case 划分到什么程度才合适呢?

于是,有些开发人员就产生了这样的想法:“关于 Use Case 的粒度问题。前两天提了一个问题就是在对复杂的系统进行建模的时候,在最初的阶段,只能用一些较为粗略的 Use Case 来表示系统的功能,随着进一步细化,是否要将原来的较为粗略的 Use Case 细化成很多个更加详细的 Use Case。但是这样做觉得走向了功能分解的路上去了。”结果,越来越混乱,

最后就无法继续分析，开发无法进行下去。

关于这个问题的解决，首先来看看粒度细化的标准。

1) 粒度细化的标准。

对于 Use Case 的粒度问题，并不是很好判断的，因为 Use Case 的描述可以详细也可以粗略，可多可少。笔者的判断标准也仅仅是依赖于过去做类似系统的经验和教训而已。下面是笔者认为的后期判断标准：

在进行 Use Case 阐述时，如果发现该 Use Case 阐述达到了第四级，也就是子流的分支流，仍然无法描述清楚，那么就需要考虑这个 Use Case 的拆分问题。否则，只要符合一般的 Use Case 划分要求，就不存在问题。在和 Rational 公司的人接触的过程中，他们对于如何确定 Use Case 的粒度到底应该细化到什么程度也没有一个很明确的说法，言辞方面也是十分的含糊。所以，大家也不用太担心自己的 Use Case 是否划分得不符合规范，什么是规范？符合要求的就是规范，能解决问题的就是规范！

注意：

同一个部门至少在同一个项目中，所有的划分标准和方式应该是相同的，否则，会给人一种混乱的感觉，这会在客户那里对开发团队造成一些负面的影响。

不要为了牵强附会或者保持与书本上内容的统一而进行不必要的疑惑，当然，书本上的很多东西还是很有用的，很有指导意义^①的。

2) 一些顾虑。

也许有人会问，如何知道在这个阶段我们对 Use Case 的划分是否可以解决问题？

的确，现在是看不出来最终的结果。

但是要注意，开发人员现在要解决的是进行 Use Case 的分析，也就是需求的深入分析阶段，既然看不出最终的结果，那也就不必过多地去考虑，只要认为目前的分析结果是最合理的，是自己能够做到的目前认为最好的或者相当好的也就足够了。没有人是神仙，有的只是基于经验的分析，所以这个阶段的成果很大程度上依赖于开发人员的经验，对开发过程经验越丰富的人划分出来的 Use Case 就会相对越准确，有效性越高。

3) 附加说明。

在 RUP 的实现中，每一次迭代，都是为了更进一步的细化需求，应该说是：细化每一个阶段的成果。所以，开发人员既要在意当前的成果，也不要太在乎当前的成果。说起来好像很抽象。就像这样一句名言：战略上藐视敌人，战术上重视敌人！其实在这里也就是这样的一种考虑方式。

① 说它有指导意义，同时也就是说：它不一定具有现实意义！它只是给我们指出了方向，而这个方向上到底有什么，必须我们自己来摸索，而没有一个成文的定量的阐述！

所以，大家也不用因为某些书上的描述和说明而怀疑自己的划分，只要你符合了一般性的原则，其他的完全可以由自己来掌握——注意：关键是能够解决问题！这才是我们最需要掌握和控制的。

更确切地说,那就是:要详细分析当前的成果,分析得到这种结果的原因,但,却不要让当前的成果影响了团队的士气和进度;要综合分析目前的状态,全力为下一步的工作做好充分的准备。

2. Use Case 的合并

Use Case 的合并是将一些同类或相似的 Use Case 合并起来的过程,这样可以使系统看上去更有层次感、更为明确,也更易于管理。

我们还使用前面所说的:电子邮件撰写、邮件查看、合同录入、合同修改、合同删除、合同查看等例子。

这里我们首先要考虑的就是合并的方式,一般情况下,合并的方式有以下两种:

1) 功能性合并。

该合并过程就是按照相同的功能对业务 Use Case 进行合并,然后提取公共的名称名词和功能性描述词汇来形成 Use Case 名称。

例如:可以将电子邮件撰写和合同录入合并为文档录入,邮件查看和合同查看合并为文档查看。

2) 业务性合并。

该合并过程就是按照业务相关性,将同一业务相关的操作过程进行合并,提取业务名称加上“管理”两个字来形成 Use Case 名称。

例如:把电子邮件撰写、邮件查看等邮件相关的内容合并为邮件管理,把合同录入、合同修改、合同删除、合同查看等合同管理合并为合同管理。

在一般情况下,建议进行业务性合并,毕竟所开发的是业务系统,用户在习惯上会更倾向于将同一类业务相关的操作放在一起来管理和操作。

从长远系统开发角度来看,业务性合并更有利于专家系统的开发,因为专家系统是针对某一个业务进行综合管理与控制的过程性系统。如果前期按照功能进行了划分,在专家系统的开发中就不可能进行更好的、基于业务的分割了,也就不可能让系统具有更强的扩展性。

当然,功能性合并可以提高开发效率,减少开发时间,从公司的利益考虑,如果公司不打算做该系统的后续扩展开发,也不打算做长期的开发,那么就可以考虑功能性合并方式。

3. Use Case 的拆分

前面说完了 Use Case 的合并,这里就需要介绍一下 Use Case 的拆分。

1) 拆分的原因。

因为在查找 Use Case 的过程中很可能找到的是一些比较大也比较复杂的 Use Case,这些 Use Case 可以细化成一些比较小的独立 Use Case,如果开始进行工作时对这个 Use Case 的复杂性认识不足,后续的开发过程中会发现对这个 Use Case 地深入分析无从下手,或者不知道

该如何进行分析,这就是由于这个 Use Case 涉及的点过多造成的。

此时就需要考虑 Use Case 拆分的问题了。

2) 示例说明。

在某个项目中,开始时定义了这样一个 Use Case: Manage User。

其概要描述是这样的:

所有与系统的使用者相关的操作。这些操作包括:添加用户、修改用户信息、删除用户、查找用户、修改用户口令、变更用户级别等功能。

开始时,为了方便或对系统认识不足而将它们全部放置在一个 Use Case 中来处理,处理过程中发现该 Use Case 太大了,根本无法清楚阐述其内容,在这个时候你就应该考虑拆分它的问题了。

笔者建议按 Manage UserInfo、Manage User、Manage UserRight 3 个部分划分。

- Manage UserInfo 包括修改用户信息、查看用户信息、修改用户口令 3 个部分;
- Manage User 包括添加用户、删除用户两个部分;
- Manage UserRight 包括变更用户级别、给用户授权、收回授权等权限相关的部分。

通过这样的划分,关于这个大功能模块的开发实现思路是不是清晰了许多?这种拆分的方式是按照业务相关性实现的。

2.5.4 如何分包

完成了 Use Case 的合并和拆分后,开发人员会发现 Use Case 虽然找出来了,但是 Use Case 图上到处都是交叉的线,整个图面显得很乱,很难清楚地看明白 Use Case 之间的关系和 Actor 与 Use Case 间的关系,就像图 2-11 所示那样。

于是就要考虑对这些 Use Case 进行分包管理了。通过分包,开发人员可以标示出 Use Case 之间的相互关系,使所有人都可以更容易地看清楚这些 Use Case 的本来面目和 Use Case 间的直接关系。

1. 分包示例

图 2-12 ~ 图 2-18 是图 2-11 所示的实际工程中分包后的几幅图。

(1) 系统总体一级视图

如图 2-12 所示,对于各个业务相关的模块都进行了一定的细化分类,从而在第二层上将展现出这些子包的内部关联信息。

(2) 申请子包视图

申请子包中所有的 Use Case 都是处理用户业务过程中需要向上级领导或者上级部门提交申请等待批复的工作内容的业务 Use Case,如图 2-13 所示。

(3) 批复子包视图

批复子包视图如图 2-14 所示。

(4) 审核子包视图

在审核子包中存放的都是一些与检查审核类工作相关联的业务 Use Case,如图 2-15 所示。

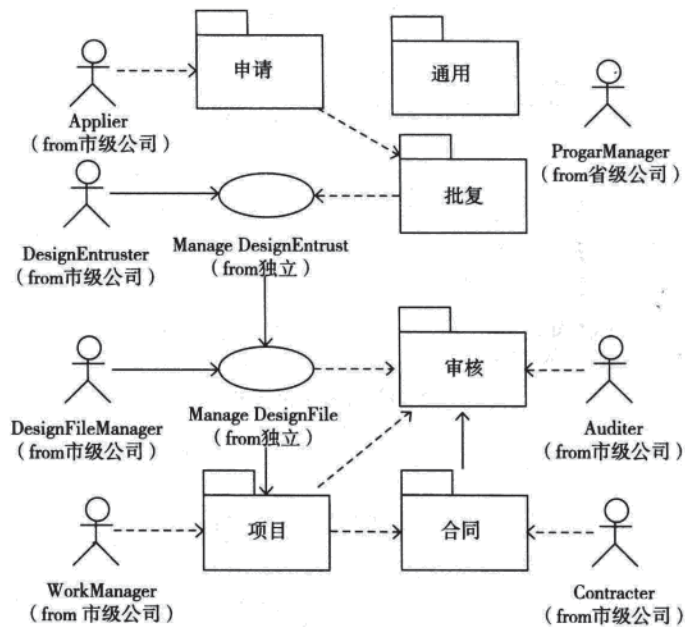


图 2-12 系统总体视图

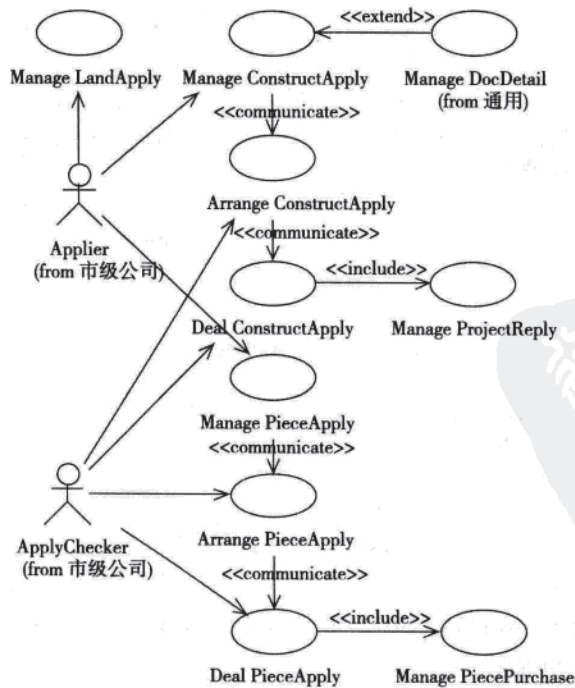


图 2-13 申请子包视图

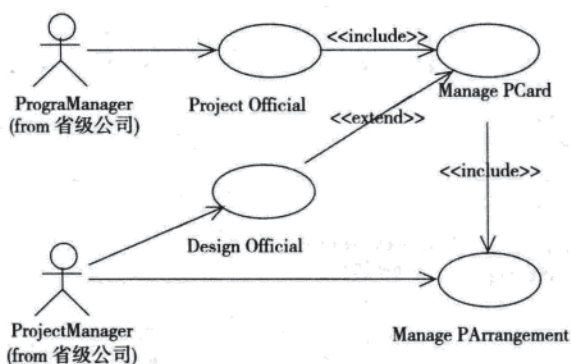


图 2-14 批复子包视图

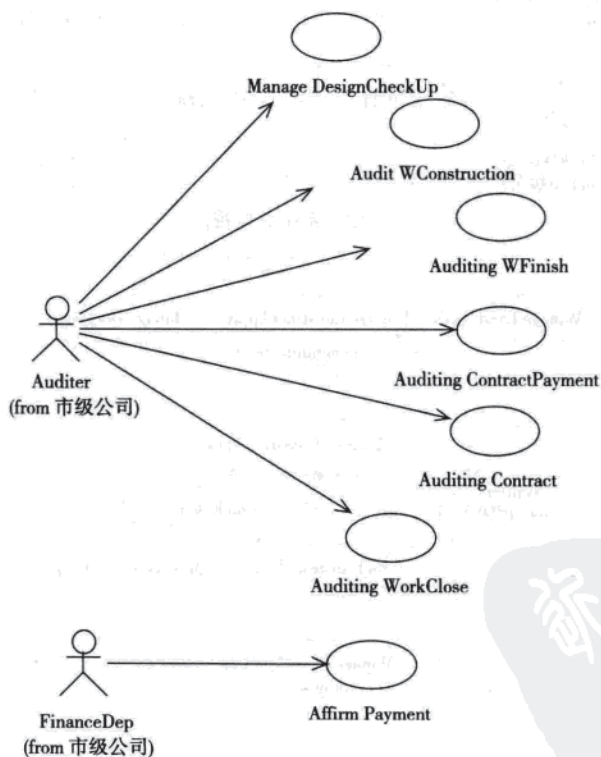


图 2-15 审核子包视图

(5) 合同子包视图

与法律事务和商务过程相关联的业务 Use Case 所形成的合同子包视图如图 2-16 所示。

(6) 工程实施子包视图

完成了相关的附加工作，最关键的还是工程实施过程所涉及的所有业务，这是系统中相

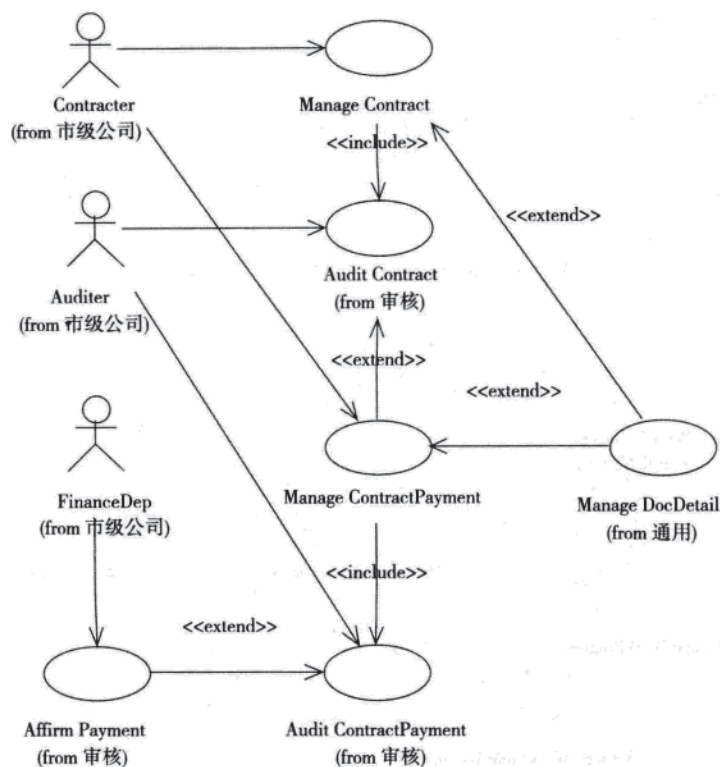


图 2-16 合同管理子包视图

对而言最复杂，功能要求最多的业务子包。工程实施子包视图如图 2-17 所示。

(7) 通用子包视图

业务 Use Case 的分包结束后，应该将所有功能性 Use Case 划分到一个包内，定名为通用子包。其实在更为复杂的系统中，这个子包应该做进一步的细化，也就是说需要分出多个功能性子包才能较完整地展现功能性需求的形态。通用子包视图如图 2-18 所示。

从本节这组图中可以看到，经过分包后，整体系统看起来清晰多了，也更容易表现各个不同的 Use Case 之间的关系和相互之间的约束条件，开发人员也更容易理解该系统各个不同部分之间的联系，同时也有利于进行后期工作的划分。

2. 分包的目的

分包的第一个目的是为了系统的 Use Case 图能够更为清晰地表现出系统的业务逻辑关系和层次，让用户能够通过开发人员的描述看到他们自己的业务结构，并对此认可。

分包的第二个目的是为了对系统进行模块的分割，这种分割将影响系统今后的开发，以及系统的最终表现形式。对于程序员今后工作的影响将非常大，因为他们会在开发过程中自主地考虑一些通用和公共的问题。

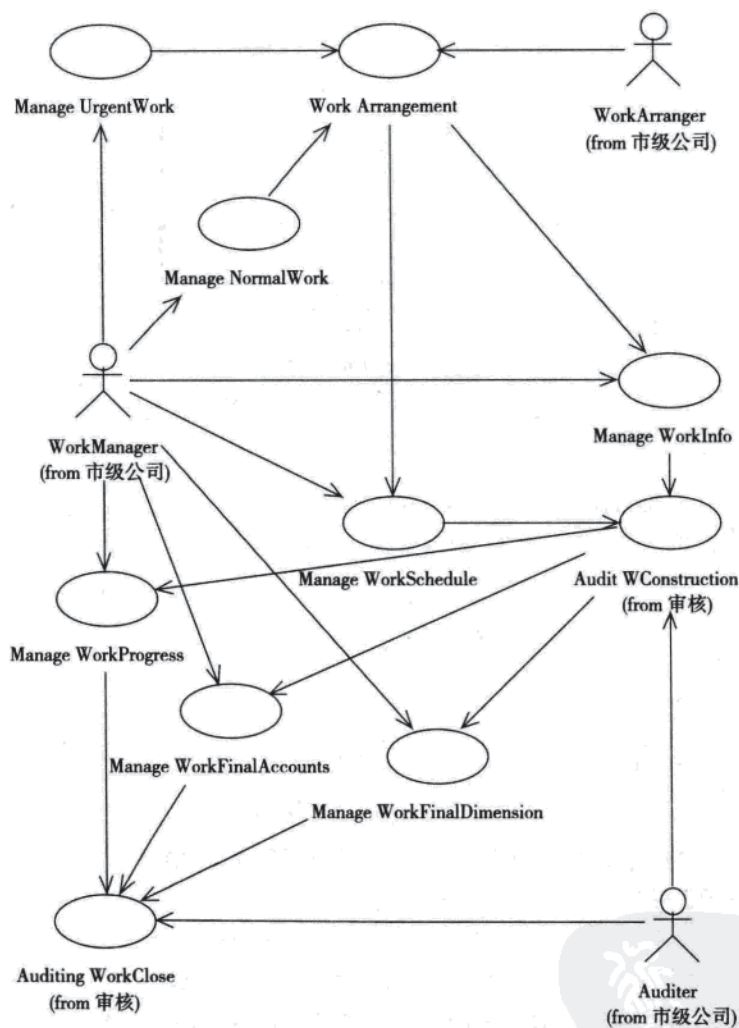


图 2-17 工程实施子包视图

3. 分包的基本原则

Use Case 分包有如下两个原则：业务相关性和功能相似性。

(1) 业务相关性

根据业务的所属关系对 Use Case 进行划分，以形成系统的业务包和子系统。

比如，可以把合同相关的 Use Case 划分到同一个合同管理包中，把系统相关的 Use Case 划分到系统管理包中等。最简单的形式就是开发人员可以按照前面整理的流程图对 Use Case 进行分包管理。把一个流程图所涉及的所有环节串起来形成一个独立的包或者成为子系统。

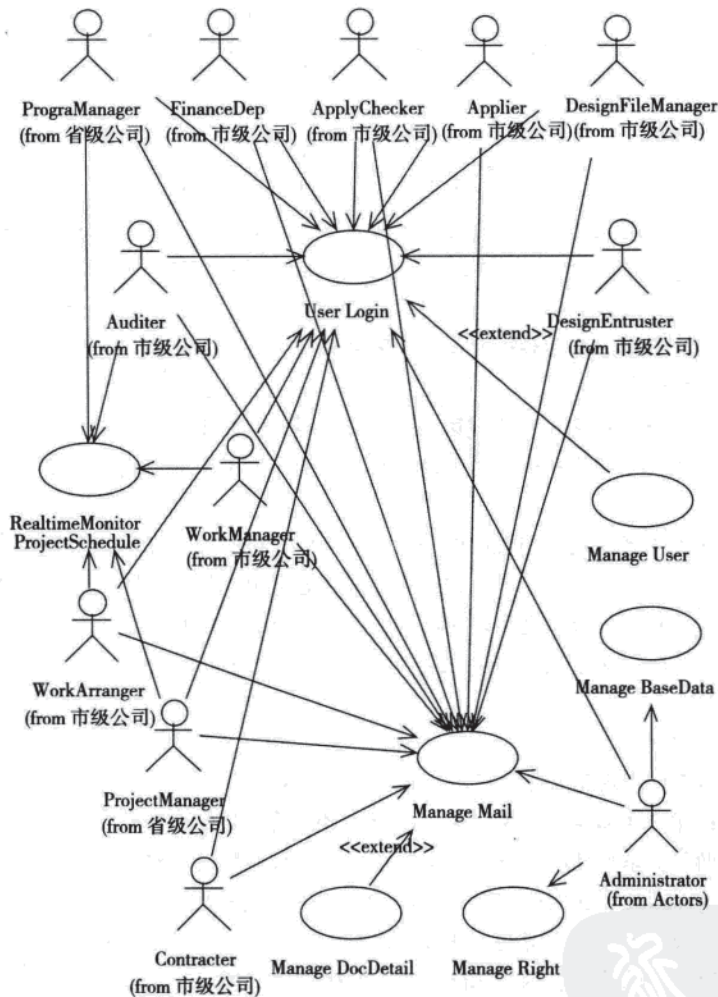


图 2-18 通用功能管理子包视图

本章图 2-12 到图 2-18 的分包形式就是业务相关性的划分结果。

(2) 功能相似性

根据 Use Case 的功能关联关系，将功能实现相似的 Use Case 组合在一起，形成系统的功能包和子系统的做法就是功能相似性。

业务相关性划分后仍然比较复杂的子 Use Case 图可以通过这个原则进行简化，划分出第二层子包，或者成为子系统。

一般来说，通过上面两次划分就可以完全避免线条的交叉和混乱的图样了。可以达到清晰表现 Use Case 间以及 Use Case 与 Actor 间的关系的目的了。

注意:

这些分包会影响到将来整个系统实现上的分包和子系统划分问题,所以,一定要格外小心。

4. 分包的过程

很多人会问:如何使用这两个原则?怎样才能更合理清晰地分包?

分包的主要目的是为了让系统能够更清晰地表现出来,让开发人员更容易的抓住系统的要点,完成最终系统的实现(更详细的描述参看前面介绍的“分包示例”)。

(1) 步骤

分包第一步:在第一次分包的时候,一般来说先考虑第一个原则,也就是业务相关性原则,进行第一层的分包。

对于一般的 MIS 系统来说,完成第一步的分包后,基本上也就足够了,不需要进行第二层的分包考虑。也就是说,在很多时候,开发人员不需要考虑采用功能相似性原则进行分包处理。

关于为什么不先考虑功能相似性原则,将在下面分包的理由部分做相关解释。

分包第二步:只有对较大的系统,才需要再考虑利用功能相似性原则进行第二层分包。

在完成了业务相关性分包之后,在进行同一业务内的功能相似性划分可以提高系统某一模块的内聚性,同时,对于较大的系统还可以划分出比较有效的子系统,提高开发效率和成果。

(2) 分包的理由

上述分包步骤的考虑是基于系统的长远发展,如果该系统只开发这一次,不会再有第二个版本(这里的第二个版本是指基于第一个版本之上的更高层次系统的研发,例如在 MIS 系统之上开发辅助分析系统和后续的决策支持系统等,而不是说那种修改一两个方法或流程就转手卖给另外一个用户形成的所谓第二版),那么,也许你只需采用功能相关性原则对系统进行划分就可以极大地提高系统的开发效率,而且可以大量地采用公共类和方法,以减少工作量,降低系统出错的几率(因为相同的方法和操作只出现一次,只要设计合理,就可以达到这个效果)。

如果你的公司或者团队还在考虑系统的长远发展,考虑将来在 MIS 系统上开发辅助分析系统乃至更高层次的系统,那么笔者建议,按照本书推荐的分包过程进行系统的分包,同时还要降低系统各个模块之间的耦合度^①,提高软件的内聚性,减少将来可能扩展的与业务相关方法的相互调用和公共方法的使用。

(3) 关于耦合度

耦合度低的软件可以很容易被重用、维护和扩充。而高耦合度的系统是很难维护的。对某一部分的修改将引起另一部分甚至更多部分的变动和修改。

① 耦合度:系统内部各模块之间相互调用以及公共部分提取的程度。

可以通过以下方法降低程序的耦合度：

- 隐藏实现细节；
- 强制构件接口定义；
- 不使用公用数据结构；
- 不让应用程序直接操作数据库。

耦合度分为输入耦合度和输出耦合度两个部分。

一个模块在软件中重用的次数越多，那么整个软件就越依赖于它。模块的输入依赖性越强（较多的其他模块依赖于该模块），则它的稳定性也就越高（即难以修改）。

换句话说，其他模块越依赖于该模块，那么要想修改该模块时而不影响其他模块也就越困难。

模块的输入依赖性和输出依赖性评测模块稳定性和不稳定性的重要因素。模块的输出依赖性强，而输入依赖性弱，则稳定性就越低（即容易修改）。这是因为修改或变动不会影响到其他方面。另一方面，模块的输入量越多，它就越难以修改，则越稳定。

关于耦合度，本章结尾有一个相关问题，根据该问题的解析过程应该可以让读者更好地了解耦合与内聚的联系与区别。

2.5.5 参与人员建议

1. 主要开发人员

行业专家、业务建模人员、Use Case 阐述人员、美工（1位，最好是需求调研人员中具有美工基本知识的人）。

注意：

一般来说业务建模人员和 Use Case 阐述人员是同一组人。

2. 主要审核人员

用户代表、用户、行业专家、项目经理、设计实现人员、系统分析员（需求调研人员）。

注意：

一般来说 Use Case 阐述人员和设计实现人员可能是同一组人，如果是，那么评审人员中的这部分角色应该由有相应能力的开发人员来担任。

2.5.6 输入输出

1. 输入工件

用户交流记录、前景文档（也就是 RUP 中的 vision）、需求规格说明书、补充规约（功能和功能性要求）、词汇表（可有可无）。

2. 输出工件

Use Case 模型、Use Case 讨论交流记录。

2.5.7 UML 元素

1. 相关 UML 元素

Use Case、Use Case 图、Use Case 模型、状态/活动图。

2. 用处

主要用于完成和细化 Use Case 模型，完善 Use Case 之间的关系和分层分包，细化状态/活动图，更清晰地描述用户实际业务内容的抽象提升。

2.5.8 常见问题

1. Use Case 的命名问题

问：Use Case 图中，Use Case 的命名有什么要求或者规范吗？是否可以随自己的需求而定？

在概要设计阶段，要从 Use Case 图中抽象出类，而此类名应该从 Use Case 中提取，所以 Use Case 图中的 Use Case 名是否就应该用英文名（不用中文名），以方便对应类图中相应的类？

但是，Use Case 图是用来做需求分析的，应该不考虑后期的软件实现，这一点又要求用中文来命名 Use Case，以方便用户检查需求在系统中是否完全包含。

答：下面是 RUP 中关于 Use Case 命名的阐述。

每个用例都应有一个名称，以表明它与主角进行交互的结果。名称应该是几个便于理解的单词。用例名称不应重复。

例如，回收机示例中的“回收储藏物品”用例可以具有以下的不同名称：

接收储藏物品、正在接收储藏物品、返回储藏物品、储藏物品、储藏物品回收。

笔者通过自己的开发实践经验认为像下面这样命名会比较合适：

Use Case 的名称采用主谓结构表述；亦即前面是名词做主语，后面是动词做谓语。

当然，在一些很难用主谓结构描述的情况下也可以少量地采用动宾结构，但是，应该尽可能减少这种情况的发生。

由于 Use Case 模型最初是供普通用户看的，但是考虑到后面开发过程中与其他相关工作^①中的命名一致性，因此建议采用如下命名规则：Use Case 命名采用英文。

这是因为 Use Case 的这个名称在后续的开发中要用于分析模型和设计模型中，命名的缩

① 工件：是项目过程中生成并使用的最终或中间产物。工件用于获取和传达项目信息。工件可以是以下的任何一种：

- 文档，如商业理由或软件构架文档。
- 模型和模型元素，如用例模型或设计模型，模型中的元素，如类或子系统。
- 代码，测试用例等。

写或者全文都可能作为子系统名或者包名出现，所以，最好还是采用英文命名方式，这样容易对系统的后续开发提供扩展性的关联支持。^⑤

而，Use Case 的注释可以使用中文描述。

因为我们的开发团队基本上都是中国人，使用通用的语言即可，当然，如果这个团队中所有的人都可以流畅地使用某种语言，那就可以使用该语言进行注释。

详细的命名示例请参看图 2-19 所示：

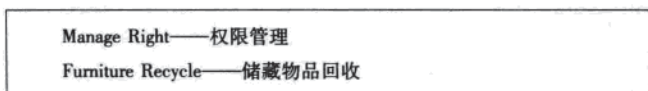


图 2-19 Use Case 名称示例

2. 在 Use Case 图中体现权限级别

问：怎样在 Use Case 图中体现角色的权限级别呢？

比如，角色分“普通用户”、“区域管理员”、“系统管理员”，如果将他们的操作分别写成 Use Case，必然造成冗余。那么怎样描述他们之间的关系呢？角色之间能否也继承呢？

答：采用“泛化”关联，如“区域管理员”、“系统管理员”可以泛化为“普通用户”，“系统管理员”又可以泛化为“区域管理员”等。这样能区分角色的级别，级别越高，它就越处于“泛化”关系中的最末端。就像类继承一样，父类级别低，子类级别高。如图 2-20 所示。

通常会针对用户角色进行类关系的建模（因为用户对象其实也是类对象），这样可以清晰地显示之间的关系，在诸如 Color UML 中会建议使用不同的颜色进行建模标示，颜色由深到浅表示其继承关系。

另外，还有一般性的用户关系关联，如普通用户可以分化为模块 1 用户、模块 2 用户等，如图 2-21 所示。



图 2-20 权限级别的体现

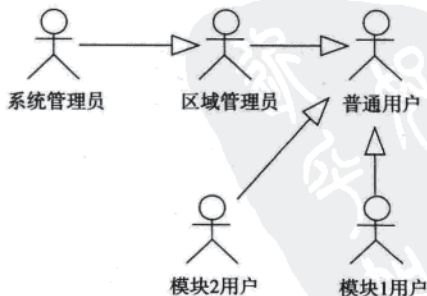


图 2-21 用户级别的体现

⑤ Use Case 名称的缩写基本上就是后续系统的包名，除非 Use Case 名称过长的情况下才会进行缩写，在分析模型时序图和设计模型时序图的命名中一般要采用 Use Case 名称加上 A 或者 D 来表示区分，详情参看第 3 章中的相关内容。

假设 A 与 B 是两个系统，两者之间互有信息传递，现在把 A 当做 Actor 来开发 B 系统。在这个时候，研究对象是 B，不是 A。开发人员不需要考虑 B 对 A 的作用以及作用产生的后果，开发人员需要考虑的仅仅是 B 以及 A 对 B 的所有作用和作用的后果。只有在将 B 作为 Actor 来开发 A 系统的时候，才需要考虑 B 对 A 的所有作用和作用的后果。

另外一种出现该现象的原因是，开发者没有清楚区分系统边界，可能是因为把用户和用户使用的工具混为一体，最后使得一些操作的结果直接发送给 Actor，而实际上应该是 Actor 使用的计算机、或者计算机界面、或者 pda、或者手机等终端和软件的表现层。

4. OO 原则中松耦合与高内聚的分析

在水木社区软工板块 2010 年初有人问了下面这样一个问题：

“A team of programmers is reviewing a proposed API for a new utility class. After some discussion, they realize that they can reduce the number of methods in the API without losing any functionality. If they implement the new design, which two OO principles will they be promoting?”

- A. Looser coupling
- D. Higher cohesion
- F. Stronger encapsulation”

提问者也不知道正确答案有几个，想不明白。

本来没打算回答这样的概念性问题，但是发现几个朋友的解答都存在一些错误，感觉到这是一个容易出现误解的概念，于是笔者仔细分析后做了如下回复：

“这个问题就是说原来有一个类，经过分析后发现，可以减少一些类中的方法，而类的设计功能不变。

而减少了方法后的类，必然是提高了类内部的聚合能力，所以 D 的可能性很大。

这里只提到了类的问题，没有提到与其他模块之间的连接关系如何实现，也没有提到是否因此需要在原来的类中多个方法调用来用一个方法完成，所以，无法确定是否一定实现了 A。因此 A 不确定。

同理，F 也有可能，但是，也不能确定是否一定有它，因为这段内容对类的实现没有说出具体的设计改变的方法。

因此结论是，F 的描述也不是很多，但是有可能，而 A 的可能性最小，D 的可能性最大。”

著名的钱五哥也给出了与笔者相同的答案：

“可能 D 和 F 都适用，减少 API 就意味着需要在接口中包含更多功能，则封装更多；同时对模块设计来说，内聚性更强。”

提问者继续问道：

“想象有两个场景，封装的都很好，一种是松耦合一种是紧耦合，显然紧耦合需要提供更多的 API 来支持耦合。是不是松耦合也有可能了？”

笔者续答道：

“OO 原则中提到的是松耦合，高内聚。

内聚性只需要看这个接口或者类的内部设计情况即可，而耦合是类间关系，没有直接的类间关系的描述，所以，这个问题中不能确定耦合度的变化问题。原题目内容较少，使得无法判断。”

到这里完全解答了提问者的疑惑。

关于 OO 的原则和分析方面的内容不是本书描述的重点，所以，在本书中没有做更多的介绍和说明。但是实际开发中却有着大量的 OO 原则的应用来解决实际问题，所以，一般在培训中，笔者会做较为全面的 OO 原则的阐述。

5. 需求是归纳出来的，不是分解出来的

问：2010 年 7 月 CSDN 上一个朋友提出了这样的问题：“在软件开发中，往往因为功能（任务）分解得不够细致而造成过程失控，可能因为功能的反复修改或遇到原来没有预见的技术难点而造成进度延迟，更有可能要推倒重来。在开发过程中，往往是公司交给一个项目组某个项目，只要求几个月完成；项目组分配到某个开发人员手里，只明确到某个人需要完成哪些模块，比如用户管理模块、邮件发送模块等。对公司来说，项目成了最小的分解单位，对开发人员来说，模块是最小分解单位。一个模块如果需要 1 000 行代码，7 个工作日，那么一个人在没有任何工具的支持下，一般不太可能知道具体的细节流程，更无法知道会遇到什么样的技术或流程风险。”这样的分解方式往往可能会因为某个模块中隐藏的复杂的业务炸弹而使得项目整体完全失去控制。

答：这段文字中的问题其实是因为需求没有做好的原因。关于最小分解单位可以对应于 Use Case 和 Use Story 的方式，在本书的定义中将之具体化为元 Use Case/Use Story 的概念，也就是最小的不可分的操作，对应于用户业务过程中的一个最小的动作，比如从办公室拿文件送给领导看就分为两个最小动作，拿过去和看！

如果认为从桌上拿起来也算是一个动作，但是要注意这个动作在软件系统中是不需要具体化的动作，软件系统中需要具体化的动作就是送到领导桌子上，然后是领导看，也就是领导点击打开，至于打开以后是否看了，谁也无法做主，只有领导本人知道，例如领导拿到你送上来的文件，眼睛盯着，心里却在想着三顿晚饭去吃哪一顿的问题，那你也没办法。

这个元 Use Case/Use Story 的基础原型就是本书前文提到的业务过程中的环节，这些环节的归纳组合形成了实际的 Use Case/Use Story。而不是在一个抽象的无法完全了解其内部业务逻辑的大型 Use Case/Use Story 上进行分解。

分解往往是无法做到公平与均衡的，而归纳却是可以控制规模和复杂度的。于是，在 2005 年初笔者首次提出了元 Use Case 的概念，同样可以扩展到元 Use Story 的概念上。

6. 元 Use Case

(1) 概念的起源

2004年6月,笔者在一次审核一个系统的业务和获取的业务用例时,遇到了一个概念上的麻烦。因为在UML的概念中对于每一个具体的用例事件流是没有具体的概念定义的。但是,从有些具体的业务中获取到的是没有一个业务流程的具体的操作环节,这些环节代表着用户业务实现的具体步骤,这里的每一个步骤也可能就是用例中对应的实际实现步骤,但也可能是一些用户的人工操作或者未来需要实现的功能点。

(2) 概念的定义

笔者在前期的工作中,已经让用户接受了用例的概念和介绍方式,如果这时候再增加一个业务流程的名词,用户也会感到很奇怪,要么就都采用用例这个名词来作解释,但是,这些业务流程合并起来才是一个实际的业务用例。概念上就发生了一次冲突。

为了解决这个问题,笔者在用例的基础上,提出了一个新的概念:元用例,或称为原用例也可以,英文为:meta Use Case。

这个概念作为对用例概念的一个补充,笔者的定义是:业务流程中对应的每一个细化到了极致的操作点。抽象一点的定义是:细化到了极致而不可再细分的用例。

这里的操作点可以是用户实际业务过程中针对某一项具体业务的一个具体的操作步骤和方式,也可以是与周围环境和系统之间的关联操作,也可能是一些业务当前或者将来才能扩展出来的后续支持性功能等。

在实际的后续开发过程中,我们不可能直接使用元用例来进行系统的分析,因为元用例过于细小,这将会带来很多不必要的麻烦。例如:让你的用例模型上的线条增加许多倍,由于线条过多,别人无法看清楚每一个元用例之间的关系。

(3) 元 Use Case 的用途

元 Use Case 是在业务建模过程中帮助你来理顺用户业务流程,对用户业务流程进行最细化的分解和分析的辅助概念。通过划分元 Use Case,你可以标识出哪些是本阶段需要实现的业务功能,哪些是需要后续阶段实现的,哪些是用户人工操作实现的,哪些是外部系统支持的等。

这样,系统分析员就可以把业务和实现做出细致的分离,不会因为用例的概念过大而对系统边界的描述无法确定,从而带来大量不必要的文字解释。

在开发过程中,通过对元 Use Case 的合并,就可以得到业务用例,同时标识出系统边界,也给用户一个系统分阶段开发和系统的远景规划。

(4) 业务 Use Case

元 Use Case 合并成业务用例的过程也要遵循用例合并的原则,用例合并的基本方式参考2.5.3节中 Use Case 的合并中的建议分为两种:一是功能性合并;二是业务性合并。

针对元 Use Case 的合并方式和上面建议的操作方式相同,这里,本书仍然建议以业务性

合并为主,功能性合并为辅助。

也就是说,在一般情况下,建议进行业务性合并,毕竟所开发的是业务系统,用户在习惯上会更倾向于将同一类业务相关的操作放在一起管理和使用。同时这种划分方式,也有利于对任务的分工,不会出现某一个方向的业务调研和开发需要多个同一角色的人来参与。

这里建议业务 Use Case 的合并一般应该不超过六个元 Use Case 的合并,否则,会使得实际开发的业务 Use Case 过度复杂而无法在 Use Case 阐述中表述清楚。

这就形成了一个新的 Use Case 规模的前期评定标准,配合本书建议的后期评定标准就可以很完善地进行 Use Case 规模的评定了。

(5) 系统边界

系统边界的划分往往是一个难点,因为用户往往希望系统具有完整性,而实际的开发过程则必须是循序渐进的过程,用户对系统的理解和使用也是循序渐进的,如果贸然地在项目的第一期就上马过于全面的系统功能实现,很有可能带来灾难性的后果。

因此在很多项目的实施中,采用分阶段、分步骤、逐层推进的方式不仅可以降低开发方的技术风险和业务理解风险,也是对用户方投资的最有效的保障建议。

系统边界划分的操作步骤和方式建议如下:

第一步:对元 Use Case 归类。

充分分析投资方对项目的预期,项目特点,开发方的实力,开发方对项目的预期等四个指标,对元 Use Case 进行归类划分。本文建议可以将元 Use Case 划分成:系统实现元 Use Case、接口实现元 Use Case、外部支持元 Use Case、人工操作元 Use Case 这四个部分。

第二步:系统实现元 Use Case 分类。

将系统实现元 Use Case 按照系统实现的阶段划分成:第一阶段实现元 Use Case、第二阶段实现元 Use Case……第 n 阶段实现元 Use Case。

第三步:元 Use Case 转换。

将已经分类的系统实现元 Use Case 进行转换,转换的规则如下:

第一阶段实现元 Use Case 不变。

第二阶段实现元 Use Case 转换成接口实现元 Use Case 和系统实现元 Use Case,这部分系统实现元 Use Case 与第一阶段实现元 Use Case 结合形成第二阶段开发工作进行所需要的对第一阶段业务和功能的扩充和修订。

此后的各阶段实现元 Use Case 作为前景预期,除非对第一阶段实现元 Use Case 有直接的关联影响,否则就可以先不作考虑。

第四步:边界确定。

这时候系统内所需要实现的元 Use Case 就包括:系统实现元 Use Case、接口实现元 Use Case 两个部分,外部关联关系包括外部支持元 Use Case、人工操作元 Use Case 这两种。

这里要注意的是：人工操作元 Use Case 有可能就是后续阶段（诸如专家系统等）需要软件实现的元 Use Case 的最初定义。

(6) 结论

于是，此后的系统需求的定义和划分都有了基础，用户新增的需求和想法也可以定义为新的元 Use Case 进行补充划分和归类。在与用户进行谈判的过程中，也就有了最细分需求的分类方式，可以清晰地定义出哪些需求是符合本阶段合同开发要求的，哪些属于超出合同范围的，不至于因为需求定义的规模较大而无法将当前阶段本不应该实现的功能也被迫放入系统中进行开发。

有了元 Use Case 的定义和用途，随之而来的基于元 Use Case 的度量模型，其中的概算度量法、精算度量法都随之而产生，6.3 节可度量绩效管理模型也将随着元 Use Case 的出现而获得新的理论和实践基础的支撑。

2.5.9 楚凡 UML 工具模型示例

Trufun 工具提供了对 Rose 模型的导入功能，但是导入后发现其导入的信息并没有获得完全的解析，为此笔者曾与西安楚凡公司进行了合作商谈，不过由于一些原因使得合作不得不暂时停止。关于 Trufun 工具的评价在后续章节中进行介绍。

本书所有章节中楚凡 UML 工具模型示例的例子文件在配套光盘中均可找到。

图 2-23 所示来自对本章对应 Use Case 模型的转换和修订，也就是用 Trufun 工具对本章实例内容的展示。

图 2-24 所示是图 2-11 的 Trufun 图示例。

图 2-25 所示是图 2-12 的 Trufun 图示例。

图 2-26 所示是图 2-13 的 Trufun 图示例。

图 2-27 所示是图 2-14 的 Trufun 图示例。

图 2-28 所示是图 2-15 的 Trufun 图示例。

图 2-29 所示是图 2-16 的 Trufun 图示例。

图 2-30 所示是图 2-17 的 Trufun 图示例。

图 2-31 所示是图 2-18 的 Trufun 图示例。

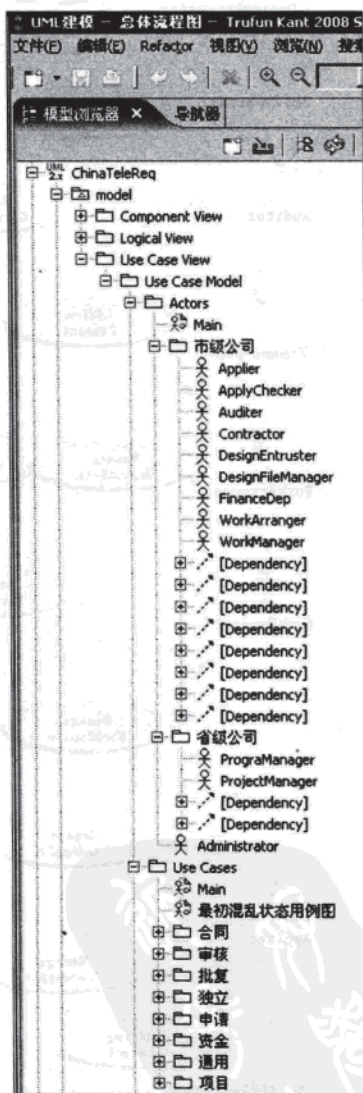


图 2-23 Trufun 工具系统树

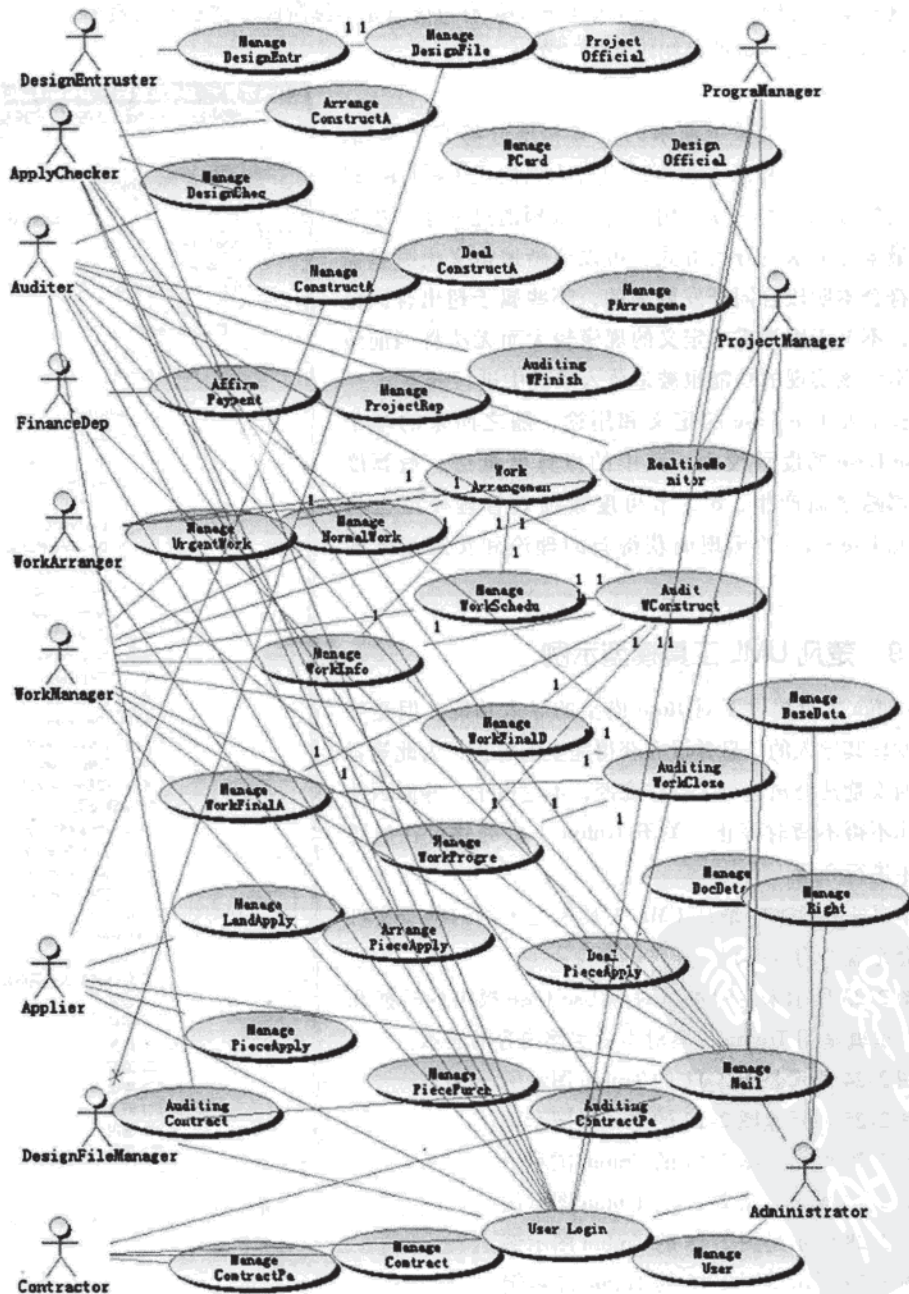


图 2-24 一个实际工程的最初 Use Case 图

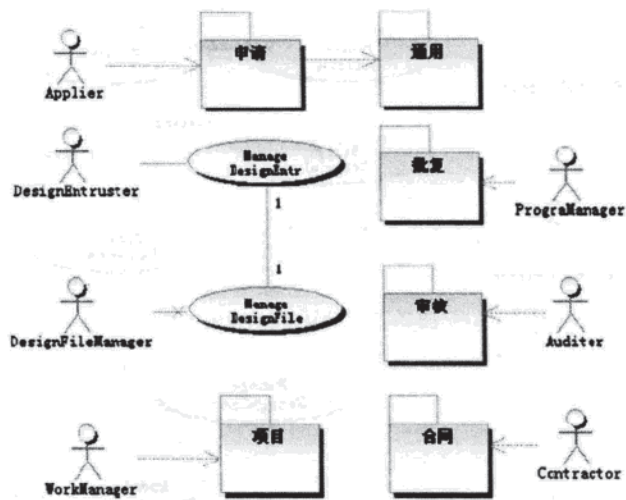


图 2-25 系统总体视图



图 2-26 申请子包视图

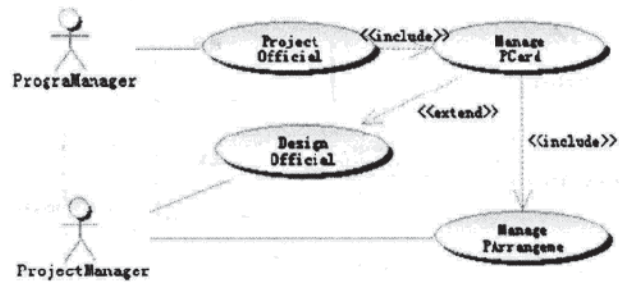


图 2-27 批复子包视图

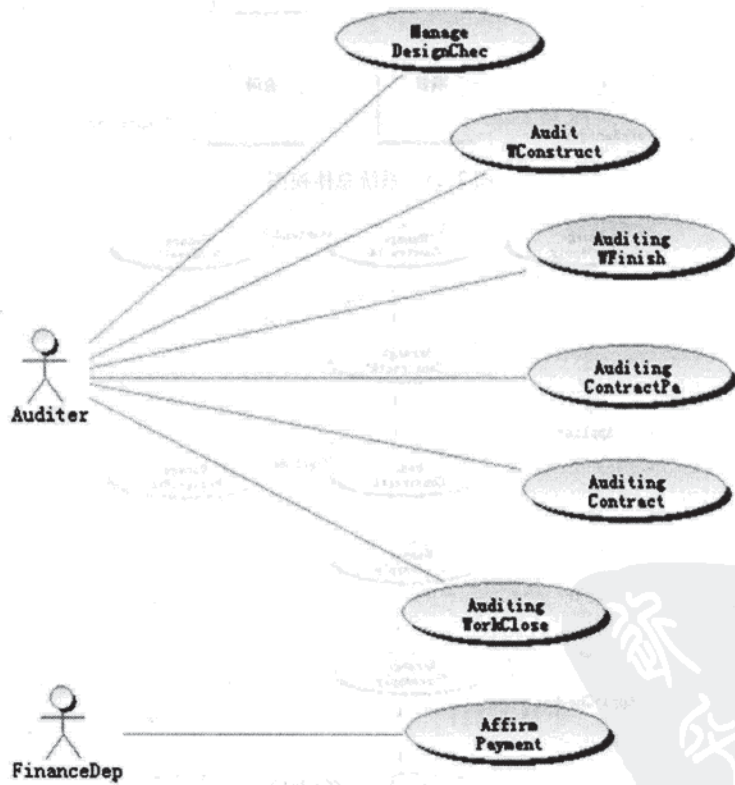


图 2-28 审核子包视图

数字资源
PDG

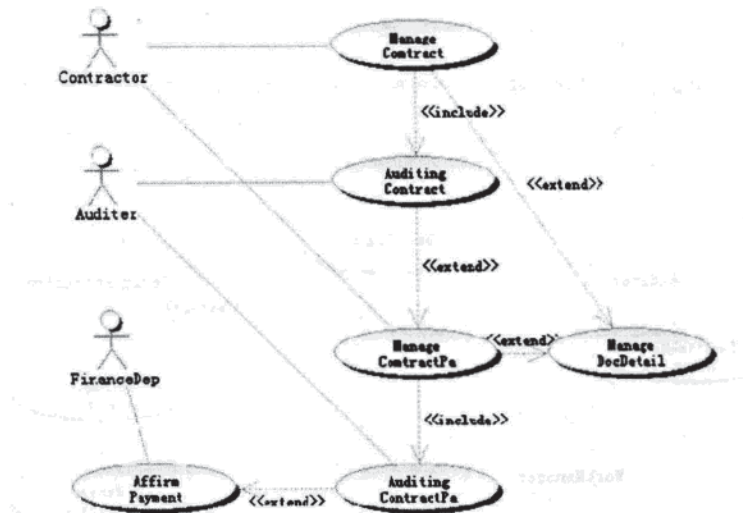


图 2-29 合同管理子包视图

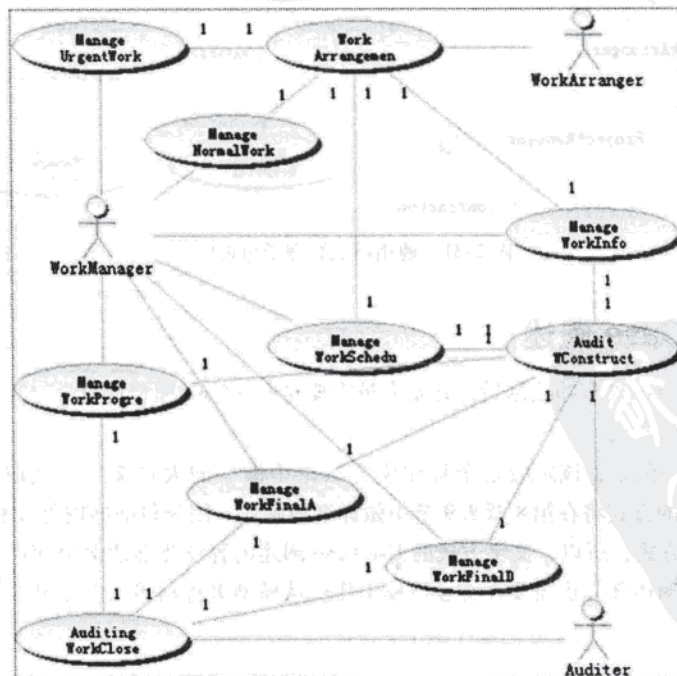


图 2-30 工程实施子包视图

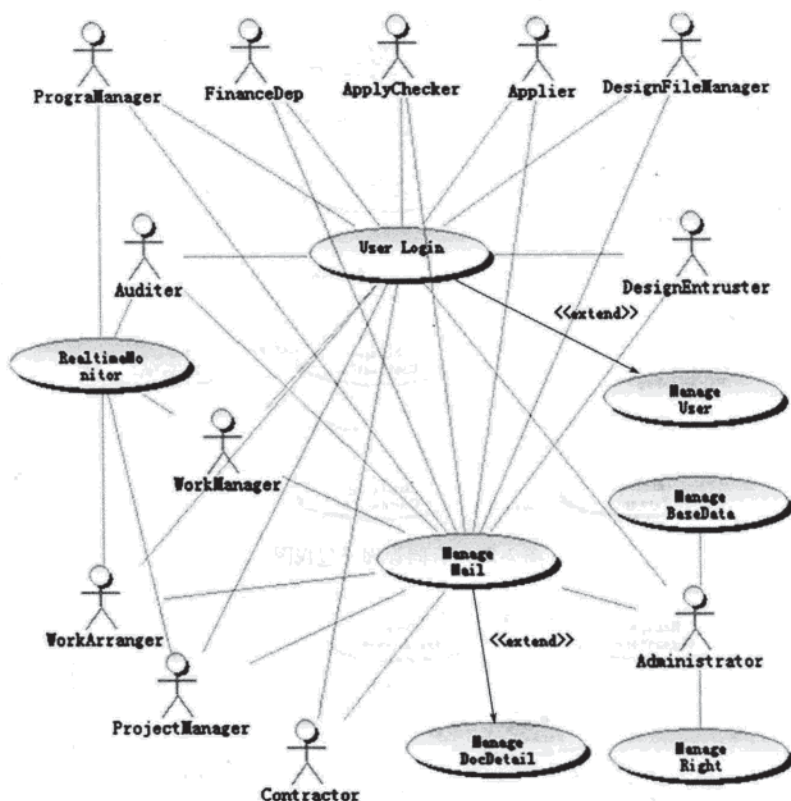


图 2-31 通用功能管理子包视图

2.6 Use Case 阐述

完成了 Use Case 模型的绘制后,开发人员需要对 Use Case 模型进行进一步的描述,以便不断地推动项目的发展。

需要特别说明的是,该阶段是全程建模方法论中惟一的大量文字描述阶段,关于 Use Case 阐述模型化的方式将在第 8 章 8.9 节中做详细地阐述。因为目前国内外没有 UML 工具可以支持这种实现方式,所以,文字方式的 Use Case 阐述内容还要继续保留使用,即使在模型化后的 Use Case 阐述部分也需要在交互建模工作前从模型里将阐述工件导出才能进行后续开发工作。

2.6.1 目的

Use Case 阐述是为了通过文字语言来描述用户的实际需求过程。这是对 Use Case 进行分析的第一步,也是进入需求分析的第一步。

完成 Use Case 阐述后的 Use Case, 就可以开始交互建模 (为界面设计做准备) 的工作, 其后是界面设计工作, 然后是分析模型的开发, 这样就逐渐从需求阶段进入分析设计阶段了。因此, Use Case 阐述的准确与否, 将直接影响到后续工作的开展, 因此 Use Case 阐述中的用词和语言都需要精雕细琢, 这也是整个开发过程中唯一进行大量文字描述的工作过程。

2.6.2 要求

Use Case 阐述的要求如下:

- (1) 简明扼要地描述该 Use Case 的具体工作流程。
- (2) 用词要准确, 不能有模糊性的词语。
- (3) 对于一些还没有弄清楚的过程, 可以通过注释的方式来表述, 以便于提醒自己或其他的开发人员, 这个地方还需要修改。
- (4) 对于提交审核的 Use Case 阐述不能继续保留注释的内容, 否则, 只能说明该 Use Case 阐述仍然没有开发完成。
- (5) Use Case 阐述的流程描述要这样进行: 一句描述操作者的行为, 一句描述系统的响应。例如:

- 输入用户名和密码, 并确认。
- 系统验证用户名和密码, 如果用户名和密码正确则执行 * * * 子流 (或步骤 n), 如果用户名或密码不正确则执行 * * * 子流 (或步骤 m)。

- (6) 每一个动作伴有一个触发性的操作, 并显示出给下一个操作的关联性界面或者提示。

2.6.3 Use Case 阐述的格式

本小节将对 Use Case 阐述中使用的格式进行诠释, 有利于读者对 Use Case 阐述中需要填写的内容的理解, 在 2.6.7 小节中将举一个实际工程的例子进行说明。

下面是笔者根据项目的特点, 对 RUP 中 Use Case 阐述模板进行修改后在实际项目中使用的模板, 这里仅作为参考。因为不同的项目会有不同的特点, 一个模板不可能用于所有的项目, 需要根据开发团队的特点和项目的特征进行修改, 否则, 就可能会出现问題。

如果在一个项目中使用 Use Case 仅仅是一个理解需求的过程, 对后面的工作没有太大的影响, 那么, 就可以使用这样的通用模板, 而不需进行细节的调整。比如说: 在项目设计过程中的需求变动, 这个项目只是直接修改设计或者编码, 而不必从需求文档开始。

1. Use Case 阐述的模板

下面是笔者根据 RUP 2002 中文版中的 Use Case 阐述模板修改后得到的 Use Case 阐述模板, 仅供读者参考。

Use Case Specification: < Use Case 名称 >

1. < Use Case 名称 >

中文名称: < 中文名称 >。

1.1 简要说明

1.2 事件流

1.2.1 基本流

1.2.1.1 子流 1

1.2.1.2 子流 1 第一分支流

1.2.1.3 子流 1 第二分支流

1.2.1.4 子流 2

1.2.1.5 子流 2 第一分支流

1.2.1.6 子流 2 第二分支流

1.2.2 备选流

1.2.2.1 备选流第一分支流

1.2.2.2 备选流第二分支流

1.3 特殊需求

1.3.1 第一特殊需求

1.3.2 第二特殊需求

1.4 前置条件

1.4.1 前置条件一

1.4.2 前置条件二

1.5 后置条件

1.5.1 后置条件一

1.5.2 后置条件二

1.6 扩展点

1.6.1 扩展点一

1.6.2 扩展点二

1.7 附加说明

1.7.1 附加说明一

1.7.2 附加说明二

注意：

这里的每一个子流和分支流的描述都将对应后续设计模型中的一些具体的业务处理类。

在本书第3章中所用的模式里，这里的子流对应于相应的助手类，子分支流则对应于助手类中的具体方法，相关具体内容请参见3.3和3.4节的内容。

2. 对模板的说明

上述模板的样式与内容和RUP中的建议模板基本相同，只增加了一个附加说明项，这是基于目前笔者所做项目提出的建议，这对我国目前大多数项目具有更强的适应性。有些系统还会对Use Case提出更多的附加条件，诸如外观的基本特征、操作习惯以及其他性能等方面的要求，对这些附加条件进行整理记录有利于系统的后期开发工作。

2.6.4 参与人员建议

1. 主要开发人员

行业专家、业务建模人员、Use Case 阐述人员、美工（1位，最好是需求调研人员中有具备美工基本知识的人）。

注意：

一般来说业务建模人员和Use Case 阐述人员是同一组人。

2. 主要审核人员

用户代表、用户、行业专家、项目经理、设计实现人员。

注意：

一般来说Use Case 阐述人员和设计实现人员可能是同一组人，如果是，那么评审人员中的该角色应该由具有相应能力的开发人员来担任。

2.6.5 输入输出

1. 输入工件

前景文档、需求规格说明书、补充规约、词汇表、Use Case 模型、Use Case 讨论交流记录。

2. 输出工件

Use Case 阐述工件集。

2.6.6 UML 元素

1. 相关 UML 元素

Use Case、Use Case 图、Use Case 模型、状态/活动图。

2. 用处

通过 Use Case 阐述的开发反馈相应的 Use Case 内容，将不确定的内容明确下来，通过这样的方式将 Use Case 模型最终完成。

2.6.7 例子

下面是笔者在 2002 年一个项目中编写的 Use Case 阐述的例子：

1. Manage ContractPayment

中文名称：合同付款管理。

合同付款管理包括 ** 合同、** 合同、** 合同、** 合同、** 合同、** 合同的付款管理。

1.1 简要说明

下面将对合同付款管理的流程进行统一的描述，这包括合同的创建、修改、查询和删除，然后将在附加说明中阐述各个合同的特殊性和需要说明的点。

1.2 事件流

1.2.1 基本流

操作者希望进行合同付款管理时启动此用例：

1. 操作者选择合同付款选项。
2. 系统显示操作者可以进行付款的合同列表。
3. 操作者选择要付款的合同。
4. 系统显示其基本信息及付款信息，并根据需要提示该合同的其他明细信息。
5. 根据用户的选择执行如下相应的操作：
 - (1) 用户选择付款操作，系统执行合同付款子流。
 - (2) 用户选择添加付款明细操作，系统执行合同付款明细添加子流。
 - (3) 用户选择修改付款明细操作，系统执行合同付款明细修改子流。
 - (4) 用户选择删除付款明细操作，系统执行合同付款明细删除子流。
6. 当用户选择其他操作时，系统结束此用例。

1.2.1.1 合同付款子流

1. 系统显示合同可付款明细和付款明细状态。
2. 根据合同的类型，用户可能进行下面两种操作：
 - (1) 非零星购置类合同，用户将输入付款信息，并确认操作。

(2) 零星购置类合同, 用户将在付款列表选中一条付款信息, 并确认付款。

(3) 用户不做选择, 则系统返回基本流步骤6。

3. 系统自动将本期合同付款金额数按照每条明细单项工程的应付金额在该合同的所有明细项目的金额总和中所占的比例进行分配。

4. 系统返回步骤1。

1.2.1.2 合同付款明细添加子流

1. 系统显示合同付款明细输入界面。

2. 用户可能进行下面两种操作:

(1) 用户选择返回, 则执行步骤4。

(2) 输入付款信息, 并确认操作。

3. 系统保存用户输入的信息, 并返回步骤1。

4. 系统返回基本流步骤6。

1.2.1.3 合同付款明细修改子流

1. 系统显示合同付款明细界面。

2. 用户可能进行下面两种操作:

(1) 用户选择返回, 则执行步骤6。

(2) 用户选择某条付款信息, 并确认操作。

3. 系统显示该付款信息修改界面。

4. 用户修改付款信息并确认保存。

5. 系统保存用户输入的信息, 并返回步骤1。

6. 系统返回基本流步骤6。

1.2.1.4 合同付款明细删除子流

1. 系统显示合同付款明细删除界面。

2. 用户可能进行下面两种操作:

(1) 用户选择返回, 则执行步骤4。

(2) 用户选择要删除的付款明细, 并确认操作。

3. 系统删除用户选择的付款信息, 并返回步骤1。

4. 系统返回基本流步骤6。

1.2.2 备选流

1.2.2.1 备选流一

1. 如果新建付款失败, 系统提示用户合同付款失败, 请用户与管理员联系。

2. 用户确认后系统退回到付款前的状态。

1.2.2.2 备选流二

1. 如果付款信息已经通过审核, 用户再试图编辑或者删除该付款信息时, 系统提示用户该付款信息已经通过审核, 用户不能进行修改或者删除操作。

2. 用户确认后退出用例流程。

1.3 特殊需求

1.3.1 第一特殊需求

付款金额只能输入数字和小数点，不允许输入其他字符。

1.4 前置条件

1.4.1 前置条件一

用户必须登录系统，通过身份验证后才能进行本用例操作。

1.4.2 前置条件二

该用户有权限进行付款操作。

1.4.3 前置条件三

经过合同审核流程后才可以进入合同付款管理流程。

1.5 后置条件

合同付款添加后必须进入合同付款审核流程。

1.6 扩展点

没有与此相关的内容。

1.7 附加说明

没有与此相关的内容。

注意：

在这里笔者删除了一些行业相关内容，删除的内容并不影响 Use Case 阐述的整体结构和内容表述。

阐述的每一步都将在分析模型中展现出来，要求描述准确，语言精练，层次结构明晰。

2.6.8 常见问题

下面将 Use Case 阐述中经常出现的问题，也是笔者在工作中经常解释的问题列出，供读者参考。

1. Use Case 阐述到什么程度才是足够的，如何判定

有些人在进行 Use Case 阐述时，会不断地进行分支流和子流的循环，例如：主流→子流→分支流→2级分支流→3级分支流→4级分支流→5级分支流→……

笔者建议：Use Case 阐述的层次不要超过4层。

如果出现超过4层的情况，那么你需要考虑两个问题。第一个问题是，Use Case 的阐述是否过于冗长，语言是否精炼，描述是否准确。

经过分析后，如果你认为自己的描述已经很准确，不存在冗余，那么需要考虑的第二个问题是，这个 Use Case 是否需要拆分。

所谓不超过4层也就是说，只能进行如下的描述：主流→子流→分支流→子分支流。

如果发现在子分支流下还需要继续划分下级分支，那就一定是出现了上面两个问题中的一种。

注意：

一般来说，这里的子流对应于设计类的一个具体方法或者一个具体实现的设计类。

如果应用了本书中建议的设计模式，那么子流对应于控制类中的一个控制方法和一个具体的实体类。

在本书第1版中有如下建议：这个说法目前只对 MIS 系统有效，笔者目前还没有在其他系统中进行过实践，所以这里不能对其他系统做任何结论。

但是，在后续的实践中发现了 Use Case 阐述与系统架构间的对应关系，于是可以认定此规则对于所有的业务系统都有效。

这是因为子流和分支流分别对应于设计类和类内的方法，如果超过了4层，意味着要把原来类内的方法独立成具体的类，这样才能对下一层的分支流进行实现。当超过第4层的时候，该系统的层级结构就会超过4层，系统的复杂度就会增加很多。同样的，如果达到第6层，那么原来类中的方法变成的类中的方法都必须再次独立成具体的类进行实现，系统就会超过5层调用关系，这样的系统一定存在严重的架构问题和性能问题。

2. Use Case 阐述的用词和语言

Use Case 阐述用词要准确，不能含糊其辞，语言要精练。一个 Use Case 阐述中描述保存失败的例子如图 2-32 所示。

1.2.2.1.1 保存失败

如果系统检测到必须填写的内容不完整，提示用户“数据录入不完整”，用户确认后，系统返回保存前的状态，用户重新输入正确的数据。

图 2-32 一个实际 Use Case 阐述的错误例子

这段描述存在问题，它没有明确地指明哪些内容是必填的，因为这些应该是在需求调研过程中已经获取的信息，如果没有获取到，在这里就应该让用户确认这些必要的信息。

另外，在进行 Use Case 阐述时，不要这样写：提示用户“数据录入不完整”——这样，会限制交互建模开发人员进行界面设计时的思路，应该写成：提示用户数据录入不完整。

按照惯例，被引号括起来的内容是属于固定不变的内容，而界面设计人员也许会认为：“你的数据信息录入不完整”，这样的描述会更贴切一些，而认为“数据录入不完整”过于生硬。当然，这里只是一个用词的问题。

对于上面的例子，笔者建议按图 2-33 所示方法描述更为合适。

1.2.2.1.1 保存失败

如果系统检测到该页面上必须填写的内容填写不完整,应该提示用户数据录入不完整,用户认可后,系统返回保存前的状态,用户可以重新输入/修改数据。

必须填写的内容包括:名称、类别、项目编号和起始日期。

图 2-33 对图 2-32 中内容修改后的描述

还应该注明类别有哪些,项目编号的格式和定义,起始日期的格式。因为这些可能是在所有的 Use Case 阐述中都需要引用的内容,所以,也可以将它们放入到词汇表中,对它们的解释进行统一管理。

对于类似:提示用户“数据录入不完整”这样的写法,应该在交互建模中进行实现。关于交互建模的内容,请参见 2.7 节的内容。

这里给出的例子中指出了经常会出错的情况,其流程过程非常简单,一句话即可说明所描述的流程顺序。当遇到比较复杂的容易出错的情况时,也要按照正常的结构(如同 Use Case 主流那样的结构)进行描述。

注意:

Use Case 的阐述有一个度的要求,如果写得过于详细,会影响界面设计人员的思路,使他们的设计受限于 Use Case 阐述中的描述,而无法形成更有竞争力、更美观实用的界面,详情请参见 2.8 节的内容。而过于简单,就会出现如图 2-32 中的现象,使得需求无法得到更准确的描述,使用户的一些关键性要求被忽略。

3. 如何对 Use Case 阐述进行描述

笔者建议:不要做 Use Case 阐述以外的工作。

Use Case 阐述中开发者经常容易犯两种错误:第一是 Use Case 阐述不够准确,不够细致;第二是 Use Case 阐述过于繁冗复杂。

第二种情况可能造成的直接结果就是把部分交互建模的工作提前完成了。一定有人认为这是一种很好的现象,任务提前完成了还不好吗?

不,这绝对不是什么好现象,而且在工程化流程中是一种严重违反开发过程进度步骤的事情。出现这种现象的原因一般是该 Use Case 阐述开发者对业务流程非常熟悉,或者已开发过该软件的早期版本,因此,会很自然地把以前的一些界面形式和内容用到当前系统的开发中,从而影响了后面进行设计时的系统表现形式。需求开发人员应该特别注意这一点。

如果我们仔细查看任何一个程序员做过的系统,可以发现都存在一定程度的相似性,不仅在代码结构和类定义方面有相似性,而且在界面形式以及控件的摆放上也有相似性,毕竟每个人在自己工作过程中的思路 and 开发方式都是持续不断的,而不是跳跃式的,不是吗?

下面还是以图 2-33 为例来进行说明。可能有些对业务十分熟悉的 Use Case 阐述开发人员

会将图 2-33 所示的内容写成如图 2-34 所示的样子。

1.2.2.1.1 保存失败

如果系统检测到该页面上必须填写的内容填写不完整,应该提示用户数据录入不完整,用户单击“确认”按钮确认后,系统返回保存前的状态,用户可以重新输入/修改数据。
必须填写的内容包括:名称、类别、项目编号和起始日期等输入框。

图 2-34 另一种常见的错误描述

类似图 2-34 中加粗部分的内容就属于提前进行了交互建模的开发。

由于很多公司的交互建模人员就是 Use Case 阐述人员,所以认为即使提前进行一些建模开发也没有关系。其实不然,因为他们在进行交互建模内容的开发时,无形中已经影响了后续的开发工作,其思考方式和开发内容将影响系统整体的表现形式,而且会分散自身进行 Use Case 阐述的精力,甚至为了考虑是采用确认按钮还是采用确认的链接而忽视对 Use Case 阐述其他内容的检查,从而造成不必要的错误。

就像在一个论坛的帖子中的 zhf_karen(zhf)说:

“有时候,需求、概设、详设之间,并没有什么跨不过的线,有时候,比较猛的需求人员,其实做了一部分概设的工作,嘿嘿。”

当时笔者的回答是:

“to zhf_karen(zhf),关于你的话中有一个观点我不敢赞同,虽然我个人认为这样的做法的确存在,但并不能表明这个程序员比较厉害或者水平比较高,而只能说明他不清楚需求的真实用处,这样会造成需求无法被用户正确理解,无法将需求调研清楚。

我记得我在另一个帖子中曾经描述过这样的观点:需求分为两个阶段,第一阶段是对需求的调研,这个阶段的内容要求必须让所有相关的用户全部理解;第二阶段是对需求的分析,这个阶段的内容只要求部分高级用户能够理解。

一旦进入了分析模型的开发,也就是概要设计阶段,这个阶段的产物是不能要求用户理解的。一个真正高水平的程序员如果把需求和设计混淆了,我真的很担心这个系统能否真实地表达用户的需求,因为它占用了了解用户需求的时间来进行后续的工作。

一般,软件开发过程中,为需求调研分配的时间本来就很少,如果再被占用,将会造成后续改动量的增大,这也是国内工程软件到中后期变动量极大的一个很重要的原因。

我个人是绝不赞成这种情况发生的,相反,从设计到编写代码的时间则是可以比较灵活地调整,或者有能力的活可先写出部分代码,但是从需求到设计的界限是十分大的。”

这里应该着重说明的是:因为这位“高手”占用需求调研的时间来搞设计,当进行正式设计时才发现有些需求没有调研清楚,因此不得不拿出时间来找客户确认一些问题。而这时如果客户很忙,他将没有机会与客户见面,一连串的恶性循环就这样造成了,最终结果是,项目经理出面揽下所有的责任,全组成员都会因此受到牵累,项目的结果如何,不

言而知。

经常有软件开发人员说：客户成熟度不够，开展调研工作时不能很好地配合，以至于需求调研不完整，最后造成项目的延误。是的，这种情况的确存在，但是，仔细思考一下，其中有一部分原因也是由开发人员自己造成的，比如类似上述这样的现象。

这种类似的事情还会持续不断地发生，不仅仅在 Use Case 阐述的时候存在，在交互建模以及开展其他阶段工作的时候也会同样存在，最终导致项目彻底失败。关于交互建模阶段的描述请查看 2.7.7 小节的第一个问题。

2.7 交互建模

完成 Use Case 阐述后，接着要进行的就交互建模工作，该阶段的工作将直接影响用户对系统使用过程中的感受。

2.7.1 目的

交互建模过程是为了进行界面设计，以便在进行需求分析的过程中最后一次对用户的需求进行确认。让用户更明确自己所要的产品是什么样子的，或者可以获取什么样的功能以及如何使用这些功能。

2.7.2 要求

1. 基本要求

交互建模的基本要求和 Use Case 阐述基本上相同，但交互建模有其特殊性要求，尤其是在格式和用词上。

交互建模的基本要求如下：

1) Use Case 阐述的内容主要是针对事件流进行交互建模，一般来说，其他内容不需要进行交互建模。

2) 用词要准确，不能有模糊性的词语（这一点和 Use Case 阐述的要求相同）。

3) 对于每一个 Use Case 中所包含元素的描述，^①都要详细说明有哪些文字（包括链接上的、按钮上的，以及一些说明性的文字）、按钮、输入框、查询条件、显示内容等。

对于描述操作者操作的内容，应该说明用户如何操作，是单击某个按钮/链接还是输入一些内容后单击按钮/链接，或者是选择一些后单击按钮/链接，也可能选择后不需要单击任何控件等，这些都要明确地描述出来。

对于描述系统响应内容的交互建模，一般来说内容会比较少（相对于描述操作者的交互

① 注意：不要描述这些元素的大小、摆放位置、颜色、形状、字体、字号、种类，因为这些是界面设计的内容。也就是说，Use Case 阐述的结果是形成元素的胚体，而这些胚体将来会以什么样的样式展现在用户面前将是界面设计人员的工作内容。

建模内容来说), 主要描述的内容有: 系统将要显示一个什么样的界面, 界面上有哪些元素, 元素的摆放位置, 以及哪些内容是可以单击获得的, 哪些内容是需要输入的等。

交互建模的内容应该与 Use Case 阐述的内容区分开, 通常可采用不同的颜色进行区分, 同时把交互建模内容^①用中括号“[]”括起来, 以便于区分。

2. 需要注意的问题

1) 对于保留注释内容的 Use Case 阐述应该退回开发组。

2) 对于一些还没有调研清楚的过程, 可以暂时通过注释的方式来表述的 Use Case 部分可暂不做交互建模, 以减少不必要的重复性工作。

3) 如果发现 Use Case 阐述中用词不清晰的内容应及早提出来, 并让开发组人员进行修改。

4) 一般来说, 交互建模的内容要比 Use Case 阐述的内容多出 3~4 倍。

注意:

这是一个很容易确定的定量指标, 如果达不到这个数字, 也就是说少于 3 倍, 很可能你的交互建模工作没有完成, 仍需补充。

2.7.3 例子

这里的例子是基于 2.6.7 小节的例子进行描述, 为了不浪费空间, 这里只对基本流的内容进行示例说明, 而……部分的内容表示省略:

1. Manage ContractPayment

中文名称: 合同付款管理。

合同付款管理包括 ** 合同、** 合同、** 合同、** 合同、** 合同、** 合同的付款管理。

1.1 简要说明

下面将对合同付款管理的流程进行统一的描述, 这包括合同的创建、修改、查询和删除, 然后将在附加说明中阐述各个合同的特殊性和需要说明的点。

1.2 事件流

1.2.1 基本流

操作者希望进行合同付款管理的时候启动此 Use Case:

1. 操作者选择 [通过单击“合同付款”链接] 合同付款选项 [合同付款选项包括合同名称、合同付款类型……]。

2. 系统显示操作者可以进行付款的合同列表 [页面中排列合同通用查询条件 (注:

① 交互建模内容: 笔者建议的是采用红色的中括号“[]”标注出来, 内容写在中括号内。

通用查询条件已经有专用文档进行了定义, 可以查看专门的查询条件说明文档, 该文档放在 Vss 中的“……”里面), 合同明细列表按照约定的明细列表方式进行排列, 合同明细信息包括合同名称、合同签订时间、合同种类……合同名称为一个超级链接, 将连接到合同基本信息和付款信息界面]。

3. 操作者选择要付款的合同 [用户用鼠标单击合同名称]。

4. 系统显示其基本信息 [合同的基本信息包括合同名称、合同相关项目名称、签订乙方……] 及付款信息 [付款信息包括……付款明细信息带有一个单选框, 选择该单选框, 然后选择付款系统将针对此付款信息进行付款, 显示付款操作界面; 如果选择添加, 则系统基于此付款信息添加新的付款信息的界面; 如果选择修改, 则系统将显示修改此付款信息的界面; 如果选择删除, 则系统将删除此付款信息], 并根据需要提示该合同的其他明细信息 [其他信息包括……] [系统界面上显示 4 个按钮: 付款、添加、修改、删除]。

5. 根据用户的选择执行如下相应的操作:

1) 用户选择付款操作 [用户用鼠标单击标识付款明细信息的单选框, 然后单击“付款”按钮], 系统执行合同付款子流。

2) 用户选择添加付款明细操作 [用户用鼠标单击标识付款明细信息的单选框, 然后单击“添加”按钮; 或者用户直接单击添加操作], 系统执行合同付款明细添加子流。

3) 用户选择修改付款明细操作 [用户用鼠标单击标识付款明细信息的单选框, 然后单击“修改”按钮], 系统执行合同付款明细修改子流。

4) 用户选择删除付款明细操作 [用户用鼠标单击标识付款明细信息的单选框, 然后单击“删除”按钮], 系统执行合同付款明细删除子流。

6. 当用户选择其他操作时, 系统结束此 Use Case [系统完成用户操作的响应内容; 用户直接单击该 Use Case 外的任何链接; 或者用户开始其他操作, 此 Use Case 结束]。

1.2.1.1 合同付款子流

1. 系统显示合同可付款明细 [] 和付款明细状态 []。

2. 根据合同的类型, 用户可能进行下面两种操作:

1) 非零星购置类合同 [], 用户将输入付款信息 [], 并确认操作 []。

2) 零星购置类合同 [], 用户将在付款列表中选中 [] 一条付款信息 [], 并确认付款 []。

3) 用户不做选择 [], 则系统返回基本流步骤 6 []。

3. 系统自动将本期合同付款金额数按照每条明细单项工程的应付金额在该合同的所有明细项目的金额总和中所占的比例进行分配 []。

4. 系统返回步骤 1。

1.2.1.2 合同付款明细添加子流

1. 系统显示合同付款明细输入界面 []。

2. 用户可能进行下面两种操作:

- 1) 用户选择返回 [], 则执行步骤4。
- 2) 输入付款信息 [], 并确认操作 []。
3. 系统保存用户输入的信息 [], 并返回步骤1。
4. 系统返回基本流步骤6。

1.2.1.3 合同付款明细修改子流

1. 系统显示合同付款明细界面 []。
2. 用户可能进行下面两种操作:
 - 1) 用户选择返回 [], 则执行步骤6。
 - 2) 用户选择某条付款信息 [], 并确认操作 []。
3. 系统显示该付款信息修改界面 []。
4. 用户修改付款信息 [] 并确认保存 []。
5. 系统保存用户输入的信息 [], 并返回步骤1。
6. 系统返回基本流步骤6。

1.2.1.4 合同付款明细删除子流

1. 系统显示合同付款明细删除界面 []。
2. 用户可能进行下面两种操作:
 - 1) 用户选择返回 [], 则执行步骤4。
 - 2) 用户选择要删除的付款明细 [], 并确认操作 []。
3. 系统删除用户选择的付款信息 [], 并返回步骤1。
4. 系统返回基本流步骤6。

1.2.2 备选流

1.2.2.1 备选流一

1. 如果新建付款失败 [], 系统提示用户合同付款失败, 请用户与管理员联系 []。
2. 用户确认 [] 后系统退回到付款前的状态 []。

1.2.2.2 备选流二

1. 如果付款信息已经通过审核 [], 用户再试图编辑或者删除该付款信息时, 系统提示用户该付款信息已经通过审核, 用户不能进行修改或者删除操作 []。

2. 用户确认 [] 后退出 Use Case 流程。

1.3 特殊需求

1.3.1 第一特殊需求

付款金额只能输入数字和小数点, 不允许输入其他字符。

1.4 前置条件

1.4.1 前置条件一

用户必须登录系统，通过身份验证后才能进行本 Use Case 操作。

1.4.2 前置条件二

该用户有权限进行付款操作。

1.4.3 前置条件三

经过合同审核流程后才可以进入合同付款管理流程。

1.5 后置条件

合同付款添加后必须进入合同付款审核流程。

1.6 扩展点

没有与此相关的内容。

1.7 附加说明

没有与此相关的内容。

说明：

考虑到内容和一些其他的问题，笔者在这个例子中只对基本流进行了交互建模，而且也省略了一些内容。对于子流、分支流、备选流的交互建模内容，笔者没有写在这里，但是笔者对于该进行交互建模阐述的项都增加了“[]”进行标识，相信读者可以理解。

2.7.4 参与人员建议

1. 参与的开发人员

行业专家、Use Case 阐述人员、部分美工（美工中主要的负责人员和开发人员，不需要全部的美工参与）。

2. 主要审核人员

用户代表、项目经理、行业专家（要和开发人员中的行业专家有所区别，也可能没有）、其他项目中的 Use Case 阐述人员、设计实现人员。

2.7.5 输入输出

1. 输入工件

前景文档、需求规格说明书、补充规约、词汇表、Use Case 模型、Use Case 讨论交流记录、Use Case 阐述工件集。

2. 输出工件

交互建模工件集。

2.7.6 UML 元素

基本上不涉及 UML 元素的开发，主要是阅读前面已经完成的 UML 模型和相关文件进行开发。

2.7.7 常见问题

1. 不要做交互建模以外的的工作

笔者建议：不要在交互建模开发的时间内做界面设计方面的工作。

类似的问题在 Use Case 阐述时也存在, 参见 2.5.8 小节的问题 3。

笔者再次强调：不要做交互建模以外的工作。

还是举个例子来说明，仍然来用 2.6.7 小节中的例子：

1. Manage ContractPayment

中文名称：合同付款管理。

合同付款管理包括**合同、**合同、**合同、**合同、**合同、**合同、**合同的付款管理。

1.1 简要说明

下面将对合同付款管理的流程进行统一的描述，这包括合同的创建、修改、查询和删除，然后将在附加说明中阐述各个合同的特殊性和需要说明的点。

1.2 事件流

1.2.1 基本流

操作者希望进行合同付款管理时启动此 Use Case:

1. 操作者选择 [通过单击“合同付款”链接] 合同付款选项 [合同付款选项包括合同名称、合同付款类型……]。

2. 系统显示操作者可以进行付款的合同列表 [页面中 (上方 3 列文字内) 排列合同通用查询条件 (注: 通用查询条件已经有专用文档进行了定义, 可以查看专门的查询条件说明文档, 该文档放在配置管理工具 Vss 中的……里面), 合同明细列表按照约定的明细列表方式进行排列 (在查询条件的下方), 合同明细信息包括合同名称、合同签订时间、合同种类……合同名称为一个超级链接, 将连接到合同基本信息和付款信息界面]。

3. 操作者选择要付款的合同 [用户用鼠标单击合同名称]。

4. 系统显示其基本信息 [合同的基本信息包括合同名称、合同相关项目名称、签订乙方…… (位置放在页面的最上方)] 及付款信息 [付款信息包括…… (位置在基本信息下方) 付款明细信息带有一个单选框, 选择该单选框, 然后选择付款系统将针对此付款信息进行付款, 显示付款操作界面; 如果选择添加, 则系统基于此付款信息添加新的付款信息的界面; 如果选择修改, 则系统将显示修改此付款信息的界面; 如果选择删除, 则系统将删除此付款信息], 并根据需要提示该合同的其他明细信息 [其他信息包括……] [系统界面上显示 4 个按钮: 付款、添加、修改、删除]。

5. 根据用户的选择执行如下相应的操作:

- 1) 用户选择付款操作 [用户用鼠标单击标识付款明细信息的单选框, 然后单击“付款”按钮], 系统执行合同付款子流。
- 2) 用户选择添加付款明细操作 [用户用鼠标单击标识付款明细信息的单选框, 然后单击“添加”按钮; 或者用户直接单击添加操作], 系统执行合同付款明细添加子流。
- 3) 用户选择修改付款明细操作 [用户用鼠标单击标识付款明细信息的单选框, 然后单击“修改”按钮], 系统执行合同付款明细修改子流。
- 4) 用户选择删除付款明细操作 [用户用鼠标单击标识付款明细信息的单选框, 然后单击“删除”按钮], 系统执行合同付款明细删除子流。
6. 当用户选择其他操作时, 系统结束此 Use Case [系统完成用户操作的响应内容; 用户直接点击该 Use Case 外的任何链接; 或者用户开始其他操作, 此 Use Case 结束]。

例子中小括号“()”内的内容就是多余的部分, 也就是界面设计中才应该涉及的问题。

2. 交互建模的用词问题

交互建模中的用词要准确, 不能含糊其辞, 所有的词语都要精确到位, 而且要精炼。

基本上和 Use Case 阐述中的要求相似, 这里就不多做描述了。

2.8 界面设计

完成交互建模后, 经过评审, 开发人员就可以启动界面设计过程了。界面设计过程可以是一项相对独立的工作, 一方面它可以独立地成为原型法, 另一方面也可以作为整个软件工程活动中的一个子项完成。

那么, 到底什么是界面设计, 界面设计有什么用呢? 下面将进行详细介绍。

2.8.1 什么是界面设计

界面设计是需求工作中非常重要的步骤, 但是, 它同时又属于设计工作的内容, 所以, 经常有人在争论: 界面设计应该在什么时间开始?

其实, 界面设计是一项比较特殊的工作, 严格来说界面设计应该分为两个阶段, 前期阶段是对客户的需求进行调研, 这包括客户的操作习惯、着装习惯、对颜色的敏感度、工作环境等, 这个阶段需要做出很多界面设计的版本, 也就是演示系统原型的版本 (也就是实用性原型), 然后由用户进行挑选与评判, 提出意见后开发人员继续进行修改; 而到了中后期, 界面设计就是纯粹的设计工作了, 这时要根据用户最后选定的演示性原型界面来开发设计出实际的系统界面, 以便于设计人员进行系统开发工作。

2.8.2 界面设计的原则

早在1987年,苹果公司首次发表了如下3项一般性界面设计的原则:

- 1) 在计算机环境中,若是能有现实的隐喻(Metaphor)将会带给使用者熟悉的预期。
- 2) 快速反馈(Feed back)将会带给使用者似乎直接操纵现实世界物体的感觉,也为使用者提供有价值的信息。
- 3) 必须看到点到的感觉,如此可以获得使用者的信任而相信自己的记忆。

(1999, 方裕民)

在设计原则中,有几项原则是值得进一步探讨的,如表现主题、接口熟悉度、设计一致性、接口设计回复性、界面设计的使用者引导、使用界面行为的认知性、亲和性、简单明了等。以下将分别阐述之:

- 表现主题:接口设计是否有固定之描述主题,若有描述必须确实一致清晰。
- 接口熟悉度:接口使用应该从使用者预期的使用类型中所得文字或观念进行设计。
- 设计的一致性:接口设计应该一致地在类似的动作上有相同的活动方式。避免所设计的接口让使用者不能接受,影响其他相关接口功能的操作使用。
- 接口设计的回复性:接口设计应该允许使用者因操作错误而恢复,具有回复的功能。
- 接口设计的使用者引导:接口设计应该整合某些形式的内容,适当配合相关的使用者引导与协助。
- 亲和性:接口设计必须能够亲切地邀请使用者进一步探索该系统。
- 简单明了:使用者能够迅速了解界面的架构,并开始使用它。务必避免选单太杂,或是使用难懂的设计隐喻或是太多之隐喻。
- 使用接口的认知性:接口的设计在这一部分是较难掌握的,取决于设计者的设计功力及整体的视觉感受、按键设计及色彩的运用等。

以上内容摘自我国台湾省的国家高速计算机中心助理研究员陈坤森的文章《计算机多媒体之使用者接口设计探讨》一文,笔者按照大陆的习惯做了少量的习惯性用语修改。

对上文进行整理,加上笔者个人的一些项目经历,界面设计的原则可以归纳为以下3条:

- 1) 界面美观大方,表现一致。
- 2) 操作过程符合用户习惯(或者说操作过程可以让用户很快习惯)。
- 3) 完整体现业务/功能要求。

2.8.3 RUP 中的相关描述

在RUP中涉及界面设计的内容除了用户界面建模和设计用户界面原型外,几乎找不到其

他任何直接关于界面设计的描述。

不过这两个部分的描述都十分简单，在 UMLChina 上的网友 roshui 说：“我曾问过瑞理（中国）的咨询师，他也认为 RUP 在支持界面设计方面确实有待改进。”

关于 RUP 中的要求，可以参考 RUP 中两个活动的相关内容：

(1) 活动：设计用户界面原型

这个活动主要是绘制用户操作界面的原型。是一个静态的过程，仅仅是描述业务各个不同阶段的操作性界面。

(2) 活动：用户界面建模

这个活动主要是描述用户界面原型是如何与操作和业务过程结合起来的。通过这个活动，将上面静态的界面串联起来，形成动态的可以表现业务操作过程的活动界面。

其他的内容与本节相去较远，只需看一下这两个部分即可。

2.8.4 界面设计的基本要求

通过对界面设计目的的讨论，基于界面设计原则，我们总结了界面设计的几项基本要求，现排列如下：

- 1) 界面设计要完整地体现用户需求的表现形式。
- 2) 界面设计要美观大方，要符合用户群的习惯性感官感觉。

例如：

在 C/S 的系统设计中，一般要求符合 Windows 操作界面习惯，这是因为用户通常情况下习惯使用的都是 Windows 操作系统。

而在 B/S 的系统设计中，由于目前国际上还没有统一的习惯性界面模式，所以，用户更需要的就是符合自己工作习惯和操作过程的浏览器互动方式。

这里需要注意的是：链接处的表现形式和界面上的提示。

因为 B/S 系统的自主性和个性化使得每个人的使用习惯有所不同，但是限于系统的开发时间和响应周期，又不可能为每个用户提供特定的操作方式和完全个性化的设置，因此，界面上的帮助信息和链接提示就显得异常重要了。

- 3) 界面设计中的交互操作过程要符合用户习惯性的工作过程。

由于在 MIS 系统中，用户通常在使用计算机系统以前，都是直接在报表上进行填写的，很多用户更希望能够在计算机上看到自己熟悉的填写格式，因此，他们就更希望能够看到类似报表的样式显示在计算机屏幕上。

但是，限于目前计算机的屏幕分辨率和可显示范围的要求，系统往往无法做到把所有的报表内容都填写在一个页面上让用户进行浏览和操作，因此，就需要开发人员与用户进行协调，把一些必要的信息进行显示，显示与操作录入进行适当的区分，尽量在前期贴近用户的习惯，逐渐对用户的习惯进行互动式的影响，让他们逐渐习惯于计算机化的定制和展现方

式,逐渐满足于系统提供的操作习惯并适应所开发的系统中可提供的操作方式。这些都是非常重要的过程和步骤,但是,在第一期系统的开发过程中,这些变化要尽量小,否则,用户就可能不会给您进一步变化的机会了。

开发人员应该遵循的一个原则是:先适应于用户,后改之。也就是说,开发人员要先适应于用户,然后再考虑让用户如何改变来适应开发人员,适应信息化的要求。这是一个循序渐进的过程,不可能一蹴而就。

2.8.5 界面设计的步骤

到这里,我们已经完成了界面设计的初步介绍。接下来的内容就是如何实施界面设计,如何获得最终的结果。

界面设计的一般步骤如下

(1) 收集各个阶段的业务操作展示界面

其实,这是界面设计的需求整理阶段,界面设计的需求调研/收集阶段与整个系统的需求调研/收集阶段是同时进行的。根据需求文档整理各个阶段或者业务过程中需要展示处理数据的界面表格。

这一步是在业务调研和 Use Case 图绘制/阐述的时候完成。

(2) 选择必要的界面元素

这是需求整理的第二阶段,并不是每个界面都需要把所有用户可能需要的信息全部展示出来,因此需要把这些信息整理,将必要的信息保留下来,这些被保留的信息就是界面元素。

这一步应该在交互建模时完成。

(3) 对界面进行统一化处理和修饰

这个过程主要是由美工人员来完成的。

美工人员需要设计出整个系统的统一化风格,该风格不仅会影响当前开发的系统,还将影响整个公司的所有产品。一个公司的产品风格必须是统一的、稳定的,这才是一个成熟公司所具有的特征。

例如,微软公司和 IBM 公司的产品都具有自己显著的特征,其产品风格的统一性,是众所周知的。

对于一些小的公司,他们也有自己的形象,也要有向大公司发展的考虑,因此,产品风格的统一性,是必须考虑。

(4) 进行必要的审核和管理

针对设计结果进行必要的评审和讨论,以便整个项目组、委托出资方和用户能够达到认识上的统一,促进后续工作的开展。如果不能达到认识上的统一,也许会造成同一个产品内不同的模块有着不同的展示风格和配色,这不但满足不了用户的要求,也使得开发团队可能会因此而承担不必要的责任。

(5) 不断修改系统的界面, 根据需求的差异进行相应的调整

这一项工作是配合后续开发工作进行的, 随着开发工作的继续, 会不断产生一些新的界面和业务模块, 这些都需要界面设计人员按照既定的风格进行统一化的处理, 在必要的时候进行相应的修正。

(6) 交付给开发人员进行后续的工作

开发人员拿到经过审核的设计界面后, 就可以开始后续的开发工作了。这部分内容将在第3章和第4章中进行阐述。

2.8.6 界面设计的定位

由于界面设计是需求阶段与分析设计阶段直接关联的一个相对独立的阶段, 因此它不会出现类似阐述 Use Case 到交互建模间的问题。实际上笔者更倾向于将它作为需求工作的最后一项工作来进行定位, 而不是把它完全独立出来。

但是, 从工作性质上来看, 界面设计的确是一种设计, 因为它的结果是直接完成了美工的全部工作。

2.8.7 界面设计的时间特点

这里首先介绍一下软件工程过程。

RUP 的过程是一个相对较好的过程, 从它的过程和相关内容中可以学到很多有用的东西。同样的, 也可以采用同样的思路和考虑方式来看待 XP、瀑布、迭代、螺旋等过程。

笔者认为: MIS 系统开发过程中的界面设计在软件工程过程中对应于需求获取的中后期和分析设计的中前期。这是因为界面设计的两个时间特点所决定的, 即界面设计的起始时间和完成时间。

界面设计的时间特点描述如下:

第一个特点是, 当开始界面设计的时候, 项目组应获取了用户的大部分需求, 并根据用户需求完成了基本的建模工作。

只有这样, 界面设计人员才可能从需求调研人员哪里获取用户所需要的一些业务操作习惯和习惯的计算机操作方式。

笔者建议: 界面设计人员在需求调研的中后期应该加入到需求工作中, 而不是简单地从需求调研人员那里获取需求。

第二个特点是, 对于某个业务模块的界面设计应该在该业务模块设计的过程中完成, 也就是说界面设计应该早于系统设计完成。

界面设计设定的操作步骤将与系统模块设计中的实现步骤一一对应。也就是说, 一个界面上的操作必将引起一连串的系统后台动作。如果它的完成时间过晚, 必然会造成系统模块设计的时间延迟, 否则, 界面设计将不能起到它应有的作用。

一般来说, 界面设计覆盖了需求工程的后期阶段和分析模型开发的前期阶段。图形表述

时间特征如图 2-35 所示。

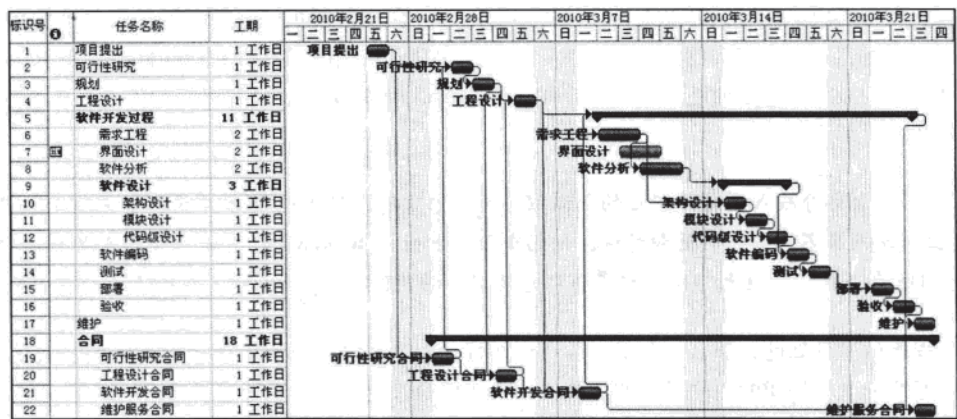


图 2-35 界面设计时间特点示意图

2.8.8 界面设计与原型法

界面设计本身也是塑造用户系统原型的过程，因此，针对于整个软件工程活动的整体时候，可以把界面设计工作看做原型法中的实用性原型法，详情请参见 1.3.1 小节的内容。

2.8.9 需要注意的问题

1. 习惯对人的影响

习惯对一个人的影响是非常大的，在日常饮食习惯的影响下，我们可以看到当一个人突然搬迁到另外一个地区生活的时候会出现水土不服的现象，都是习惯对人的作用和影响。而随意改变习惯，可能带来的不仅仅是身体的不适应，更多的可能来自内心和一些不为人知的区域。对于软件系统来说，我们需要注意如下的一些事情：

1) 用户经常使用哪些操作系统和软件，这些操作系统和软件的常规界面可以作为界面设计的第一个依据。

因为这符合用户的需要，按照用户常用的习惯来开发软件的操作过程，使用户一看到该软件就觉得软件可以使用，而不需要更多的培训。

这是非常重要的，可以降低该软件的后期维护工作和培训工作的负荷，提高用户使用软件的兴趣。

2) 用户工作环境等外围影响因素。要注意用户的工作环境，如果工作环境的颜色都是冷色调的，那么，这个软件采用非常鲜艳的颜色可能是一种错误，因为会给用户一种非常不和谐的感觉。

2. 人的个性

当然，在一些情况下应该考虑具体的原因和环境问题。例如，公司的老总如果喜欢标新

立异，那么这个时候，就应该考虑是否需要采用一些特殊的配色方案了。

在进行调研工作时，开发人员往往为多次地修改和变动所烦恼，一个软件产品从构想到需求到实现，需要很多次的修改，前期开发人员的投入和调研越充分，后期修改次数和改动量就会相对减少。一定要注意对用户个性需求的调研，以达到统一性需求。

举一个例子：

目前大部分投入到软件行业的初级程序员，很少有人使用过 DOS 系统，如果这时你开发出来的产品不能做到所见即所得，他们中除了少数非 Windows 应用软件开发人员可以接受 DOS 系统外，其他大部分人都不会愿意使用你的软件。

程序员尚且如此，那么，对于一般用户而言，如果你设计的系统是字符方式的，对于那些已习惯于微软创建的用户界面的计算机用户将很难接受这样的系统。

3. 关于 MIS 系统

仔细考虑一下国内目前所开发的系统都是些什么样的系统呢？90%，不，也许可以说是99%的国产软件都是 MIS 系统，记得《计算机爱好者》杂志在 1997/1998 年的时候有三家国内公司的广告，广告中宣称他们自己推出了所谓的 MIS 系统的开发平台，都自称是 MIS 系统的完整解决方案，但是，现在这三家公司基本上都没有什么音讯了。

由于实现技术的更新换代，用户要求的不断更新，各行业业务系统本身的专业性，用户习惯和环境的差异等，使得现在的 MIS 系统与几年前有了很大的不同，它不只是数据的录入和普通的展现，现在的系统要求的是能真正地继续扩展，能对将来的决策支持提供数据基础。

因为所开发的是 MIS 系统，所以，任何人都会非常重视界面设计，因为这直接影响到用户操作过程中的感受。

而对于其他的软件，则不一定会有界面设计的过程了。

比如：

在电信行业中的计费软件，他们要求的是准确度和及时性，不涉及界面问题，只要最终能将信息输出到其他业务管理系统，能将统计结果打印出来就可以达到目的了。

在金融行业中，银行的账务系统，也不存在界面问题，操作员看到的是完全的字符界面。那么，他们不需要更为美观的可视化界面吗？

答案是否定的。但是，目前他们使用的就是这样一套软件，他们不能也不敢去做变更。这并不是说目前没有更好的软件可以替代他们现在的系统，而是说没有已经过验证的更好的软件系统来替代他们现有的系统。

注意，这在很多行业都存在这种问题。我们并不是开发不出更好的系统，而是没有机会来验证所开发系统的可靠性。

但是，他们这种近乎原始的系统还要使用多久呢？也许5年，也许10年，也许更久。

在本书第1版出版后的2004年9月，一位曾经长期从事金融行业软件开发的朋友曾告诉笔者说，某家大型商业银行已经在尝试开发可视化界面软件了。

2.8.10 常见问题分析

1. 分析阶段与 demo

问：有人认为在分析阶段，就可将软件的 demo 做出来，然后再进入设计阶段，这样做是否正确？

答：在实际工程开发中这样做会影响后续开发工作的进度，也就是说开发进度的紧凑型不够。同时，界面设计需要参考原型法方面的内容，因为它也涉及调研用户需求和习惯的问题。

2. 迭代进行中的操作

问：一般开发时，我采用的是迭代方式，当清楚并分析一个流程后，就开始了界面设计工作。也就是说，在分析结束前，将软件开发范围所涉及的功能，都陆陆续续进行了界面设计，这样可以吗？

答：这样是可以的。每一个模块都是一个单独的工作流程线，每一个工作流程线都可单独做迭代考虑，这样才能最有效地利用时间，并提高开发效率。

3. 与建筑装修的对比

问：在建筑设计时，一般都是先出施工设计图，然后再出效果图。这里的施工“设计图”，就相当于软件的结构设计了；“效果图”，则相当于界面设计了。那么，装修房子的过程，是不是也先出“施工设计图”，再出“效果图”的？

答：这是有差别的，装修的效果图是阶段性的，不是最终的。

装修的效果图是可以在脑中先绘制出来的，而不是设计以后才出来的。

用户可以参照效果图进行选择，这并不代表该效果图就是用户所需的最终结果。

可以对效果图进行不断地更改调整，当效果图改好后，设计图也要做相应地调整。此时，你会觉得先出效果图比较好，然后再出设计图。

对于装修房子，笔者的做法是先在脑海中规划出效果图，然后，向设计人员提出设计要求，最后，施工人员实现设计要求。期间，施工人员根据他们的经验对设计要求提出了意见和建议，这就是评审。笔者根据评审的结果进行修订，重新告诉施工人员如何去做。这就是对应于有经验的开发人员对其前一阶段开发的经验性反作用。

2.8.11 参与人员建议

1. 主要开发人员

需求调研人员、界面设计人员（美工）。

重要说明：

界面设计前期的开发是界面设计人员配合需求调研人员，中后期的开发将是需求调研人员配合界面设计人员。

因为前期是为了让用户能尽快地看到界面的基本风格，确定界面的展示状态，而且，此时界面设计人员对于需求并不是很了解，所以，他们只能按要求去完成工作。

而到了中后期，界面设计人员对需求有了一定的了解，界面原型基本确定，他们可以较独立地进行各个模块的界面规划设计了，而这个时候，需求调研人员主要的工作在于与分析设计人员一同进行分析模型、设计模型的开发工作，不能而且也不需要全程跟着界面设计人员进行界面设计任务的开发。

2. 主要审核人员

用户代表（用户）、分析设计人员、开发人员、美术专业人士（至少1位）、项目经理。

2.8.12 输入输出

1. 输入工件

前景文档、需求规格说明书、补充规约、词汇表、交互建模工件集。

2. 输出工件

界面设计工件集。

2.8.13 UML 元素

基本上不涉及 UML 元素的开发，主要是参照前面已经完成的 UML 模型和相关文件进行开发。

但是在界面设计过程中，会根据用户的反馈和再次交流而确定一些业务流程，因此会涉及一些 UML 模型内容的最后修订工作。



第3章

分析设计

界面设计工作的结束，表示需求工作的全部完成，接下来的工作就是分析设计阶段。

本章内容包括：

- 分析设计的定义；
- 分析模型；
- 设计模型。



3.1 概述

分析设计工作是一个十分关键的阶段，这是一个把需求转化为代码实现的中间阶段，这个阶段在传统软件工程过程中被划分为概要设计和详细设计两个阶段，在本书中称之为分析模型和设计模型两个阶段。

3.1.1 原因

记得在 2001 年四五月份期间有人公开宣称用 UML 不能很好地支持全程建模，而且中国的用户不能很好通过 UML 所绘制的图形来理解需求。当时，一石激起千层浪，曾经引起了网上的一次关于 UML 的大讨论。

一方面为此，另一方面也为了更好地进行学习和实践，此后的四个多月内，笔者在所参与的两个较大规模的项目（项目组开发人员数量都超过 20 人）中进行了相对比较完整的试验。在项目的进行过程中，通过调动用户参与的积极性，收到了非常好的效果，用户对 UML 所绘制的图形表达出了很浓厚的兴趣和认同感。

这次的实践至少在工程项目方面证明了中国的客户仍然可以理解 UML 的图形所表达的需求含义，而且可以理解得很好！

在 csdn.net 的软件工程板块，经常有很多人希望能够得到全中文的 Rose 绘制的 UML 模型示例，想看看如何使用 UML 进行系统的分析和设计，但是，鉴于公司内部资料不能对外公开，笔者本人也一直忙于项目，因此也就无法给大家提供更多的有用资料。

经过一段时间的思考之后，笔者撰写了此文，文中的示例基本上都源于实际的工程项目，但全部都是笔者后来绘制并修改得到的，所以，也就不会牵涉公司机密泄露的问题。

上面这段开场白是笔者在 2002 年 8 月份完成《基于 Rose 的全程建模实例》中的一段话，现在想来，当时还真得有些冲动。在将这篇文章加入到本书的过程中，笔者也决定暂时不修改这段文字，也算作是对这段过去的一种纪念和回忆。

2003 年 10 月笔者在北航为软件工程硕士授课时，绘制了一个名为升级辅助工具软件的模式文件，该模型文件放在本书的光盘中，供读者参考。

3.1.2 两个项目简介

1. 某集团信息化项目

这是一个集团的信息化项目，该项目的参与人员近 30 人，包括开发人员和参与的管理者。

在项目前期，笔者带领 9 名刚刚接触 UML 和 Rose 平均不超过 1 个月的开发人员进行需求的分析和设计，绘制 Use Case 模型图，开发分析模型、设计模型以及部分代码的导出。

另外，该集团一名高级管理者（某上市公司的董事长）作为用户参与了开发工作。该管理者从来没有开发过软件，一直在从事企业的管理工作，只对办公软件比较熟悉，不懂任何

计算机语言,而且,在此项目开始之前,也没有看过 UML 方面的任何资料。

经过 12 天的需求活动后,该管理者参与了项目组的评审,每次评审前项目组成员都针对一些术语对他进行了简要的介绍。经过一段时间后,他对所绘制的 Use Case 图和分析模型的时序图能很好地理解了,并认为这种表达方式(指 UML 绘制的图形)并不是很复杂。

而且,在最后的 3 天里,他甚至对所开发的分析模型提出了相当有价值的改进意见和一些建议。

2. 某行业项目

当时在为某行业的一个项目做系统的分析和设计时,用户要求设计必须能导出代码。

这个项目在分析设计阶段的参与人员近 40 人。笔者在完成前一个项目后,马上参与到这个项目的原型开发过程。

在这个项目中,用户方技术组经过近两个月的交流后,完全可以对项目组所开发的设计模型进行相应的评审并提出他们的看法了。

3.1.3 初步结论

通过上述例子可以看出,用户并不是不能接受或者看不懂 UML 所描述的需求或者分析设计的含义,关键在于如何使用 UML 来表达你对需求的理解和设计的表述,绘制的图形是能否让用户看懂并接受。

除了上面的两个项目之外,笔者在后续的工作中也大量地采用 Rational Rose 绘制的 UML 图形来和用户进行交流,交流的方式在本章中都有详细的描述,甚至在给中国电信集团公司制定的一个全集团的业务需求规范和技术规范中的所有业务流程图也都是采用 Rational Rose 绘制的 UML 视图表达完成的,这些规范文档在 2003 年 3 月已经直接下发到全国各省级电信公司。在 2003 年第四季度参加的中国联通某项目的业务规范和技术规范的制定过程中,也采用 UML 视图来表述所有的业务流程和角色关系,并收到了较为理想的效果。

下面先从分析设计的定义开始讨论。

3.2 分析设计的定义

什么是分析?什么是设计?如何进行分析设计?分析设计中应该注意哪些问题?需要经过哪些具体的步骤呢?

这些将是在本章中需要讨论的重点内容。

3.2.1 什么是分析

分析,顾名思义即可思义。只是对于本书所涉及的系统而言,要做得就有了一个特定的限制范围:软件。

软件的分析,一般情况下是指对软件需求的分析,通过分析,将自然语言表述的软件需求进一步解析,获得半数字化的软件需求描述,以便于进行下一步的设计工作。

这个过程对应于 Rumbaugh^① 的 OMT 方法中开发过程的第一个阶段。

分析

基于问题和用户需求的描述，建立现实世界的模型。分析阶段的产物有：

- 问题描述；
- 对象模型 = 对象图 + 数据词典；
- 动态模型 = 状态图 + 全局事件流图；
- 功能模型 = 数据流图 + 约束。

在 Jacobson^② 方法中，对应的就是本书现在使用的分析模型^③了。

这里采用的分析过程是基于 UML 语言进行的实现。采用 UML 建模方式将需求进一步转化，成为有利于程序员理解的一种半数字化的表达方式。

3.2.2 什么是设计

设计，与需求和分析一样，也是使用非常广泛的词汇，不仅是在软件行业，在制造业、通信行业、电力、水力等行业，都会用到设计一词。

设计，是软件工程过程中至关重要的一步，以至于很多人认为：概要设计^④（对应于分析模型）可以省略，而详细设计（对应于设计模型）不可省略。或者，一些经验丰富的程序员^⑤可以直接从需求开发出详细设计来。

简单地讲：软件设计就是软件需求（分析）到软件实现必由之路。

软件设计的好坏将直接决定软件实现的质量。

所以，国内外软件行业的设计者对设计非常重视。国外最早提出的设计模式，也是对这方面的最好说明。

软件大师 Peter Code 认为一个好的系统设计应该有如下的性质：可扩展性、灵活性、可插入性。

- 可扩展性（Extensibility）：容易添加新的功能。
- 灵活性（Flexibility）：代码修改平稳地发生。
- 可插入性（Pluggability）：容易将一个类抽取出去，同时将另一个有同样接口的类加

① Dr. James Rumbaugh. UML 的创始人之一，他与 Grady Booch, Ivar Jacobson 被公认为建模领域的三巨头。

② Ivar Jacobson 博士在 Rational Software Corporation 担任软件开发策略副总（Vice President of Process Strategy）。他是 OOSE 方法的发明者，更与在 Rational 工作的两位面向对象科学家 Grady Booch 及 James Rumbaugh，创建了面向对象开发的标准化统一建模语言（UML）。Ivar Jacobson 多年来在软件开发领域中积极钻研，并作出了一系列的贡献。

③ 分析模型：use case 模型通过分析来构造。

④ 对系统需求的初步设计，通常被称之为概要设计。分析模型的工作对应着常规过程的概要设计。

⑤ 经验在软件开发中起着非常重要的作用，尤其是在中国国内目前的这种开发现状的情况下，因为开发这些 MIS 产品的技术相对而言都没有什么特别高难度的，所以，工程经验和领域知识就成了一个高手必备的素质了。

入进来。

设计过程对应于 Rumbaugh 的 OMT 方法中开发过程的第二阶段系统设计和第三阶段对象设计：

1) 系统设计：

结合问题域的知识 and 目标系统的体系结构（求解域），将目标系统分解为子系统。

说明：该阶段对应于本书所描述的软件工程过程中对设计模式的选择和定义过程，以及系统的分包过程。

2) 对象设计：

基于分析模型和求解域中的体系结构等添加的实现细节，完成系统设计。主要产物包括：

- 细化的对象模型；
- 细化的动态模型；
- 细化的功能模型。

说明：该阶段对应于本书所描述的软件工程过程中设计模型^①的开发过程。

在 Jacobson 方法中，对应的就是本书中现在使用的设计模型了。

在完成了对分析设计工作的简要描述后，本章的重点将转移到如何使用 Rose 把软件的系统需求变成基于模型的表达方式上，即进入分析模型开发阶段。

3.3 分析模型

本节将开始描述如何从需求调研得到的结果来构建系统的分析模型，这里将涉及的内容包括：模式的应用、分析类的获取、时序图的绘制等。

3.3.1 模式的选择与应用

一般情况下，在进行一个 Use Case 的分析过程中，需要考虑采用一些具体的分析模式。当然，也可以考虑按照个人习惯来进行 Use Case 的分析。

本小节中选择的是最通用的模式：MVC 模式^②。

① 设计模型：use case 模型通过设计来具体化。

② 关于 MVC 的说明：

当然，有些人会说：这是一种设计模式！其实分析和设计是很难清晰地划分开来的，很多分析的结果如果进一步地细化，那就是在做设计！

还有些人会说：这是一种体系架构的处理方式。这也正确，MVC 可以称得上是一个相当“伟大”的创造。它通过一种浅显易懂的划分方式，将一个复杂的业务系统分割成了具有逻辑关联关系的几个部分，将我们的思路很快从业务环境中转移到计算机实现的领域里面。

不过，MVC 还可以在不同的层面得到应用，有人认为：除了这种一次性的 MVC 的划分外，还可以分别对界面层（V）、模型层（M）、控制层（C）再进行一次 MVC 模式的细化，最后得到真正的实现结构。不过，这些内容就不属于本书讨论的范畴了。

笔者认为：设计的初始状态就是分析，一些设计模式同样可以通过调整 and 变化作为分析模式出现。

关于模式，本书在 3.4.1 节中有较为详细的阐述。

当然，分析和设计是不同的：分析是对需求的一种深入的实现性描述。

这种描述要比原来自然语言的描述更深入一些，它要求采用计算机的部分专业术语对整个需求流程进行描述。但不需要进行更深入的细化，下一步的细化就是设计。

这里，我们应用量变到质变的原理来分析一下：

分析是最初对需求的规划和设计，也就是对需求进行数字化描述的第一步，是对需求的一种抽象化的分解过程。随着工作的开展，当设计的成分越来越大超过某一个度量值^②的时候，分析就发生了质变，就变成了设计。

3.3.2 构建分析类

采用 MVC 的模式进行分析模型开发时，一般要初步规划为 3 个类：边界类、控制类和模型类（或者为数据类）。

因此，分析模型和需求间的关系如下：

需求 + 分析模式 = 分析模型

同样的，也可以得到：

分析模型 = 分析类 + 时序图

这里的时序图也可以是协作图，不过，笔者个人更习惯于称之为时序图。

1. 构建一个类的步骤

下面详细地描述构建一个类的操作步骤，并对一些需要注意的问题进行说明：

(1) 检查 Rose 的设置

选择“Tools”下的“Options...”选项，如图 3-1 所示。

说明：

图 3-1 中的这个菜单是可定制的，可以通过主菜单上的“Add-Ins”来进行个性化的设置。在图 3-1 中就去掉了笔者在项目中不常用的 VB、

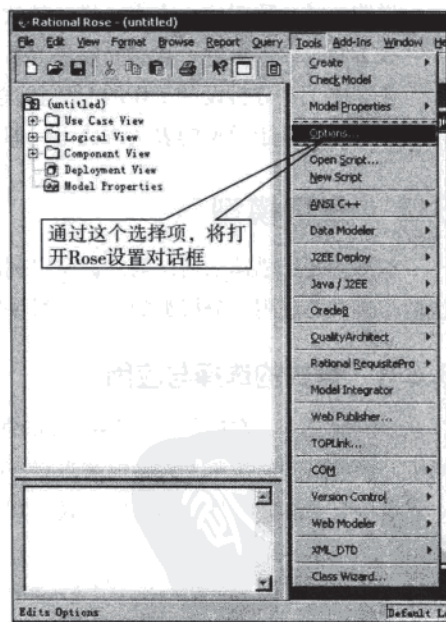


图 3-1 Rose 的设置

② 关于度量值的说明：

这个度量值到底是多少呢？

那就是完全非自然语言的描述加上模型化的结构来表达需求的时候，所得到的结果就是设计；而采用自然语言的描述方式，通过模型化的结构进行的需求描述的结果就是分析。

C++、Ada83、Ada95、CORBA、Rose C++ 等选项。

增加了这些选项后的 Tools 菜单，如图 3-2 所示。

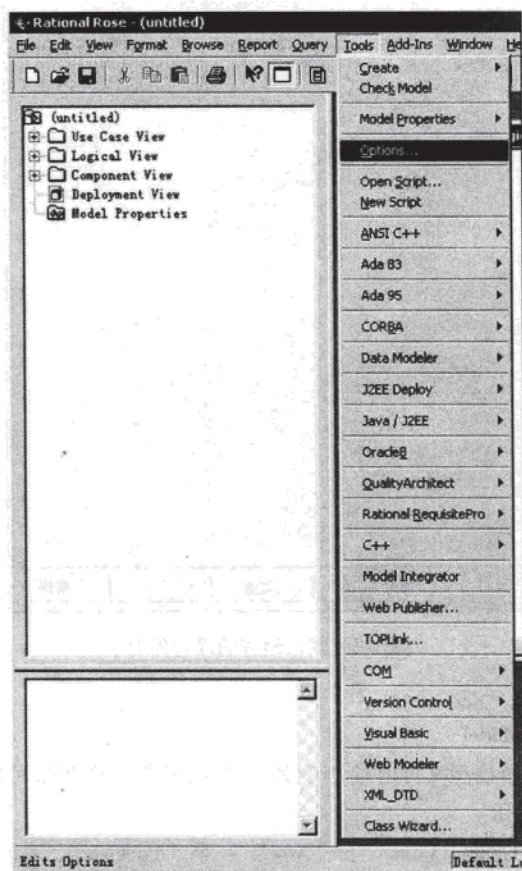


图 3-2 比较完整的 Tools 菜单

选择“Options...”选项会打开如图 3-3 所示的窗口，选择其中的“Notation”标签，设置的“Default notation”和“Default (language)”。

图 3-3 的说明：

No 1 选择建模规则，定义默认符号类型为 Unified，除非指定其他符号类型（如 Booch 或 OMT）。

No 2 注意，这里的 Default 实际上应该是 Default language。要确保你在进行分析模型开发时，选择的值是“Analysis”，否则项目将会受到具体语言的限制。原因是，通常项目在分析模型开发过程中，由于系统体系架构还没有确定，系统需要使用的开发语言也无法确定，因此，这里选择的语言应该是分析语言，只有在进入设计模型开发时，才会根据具体工程所用

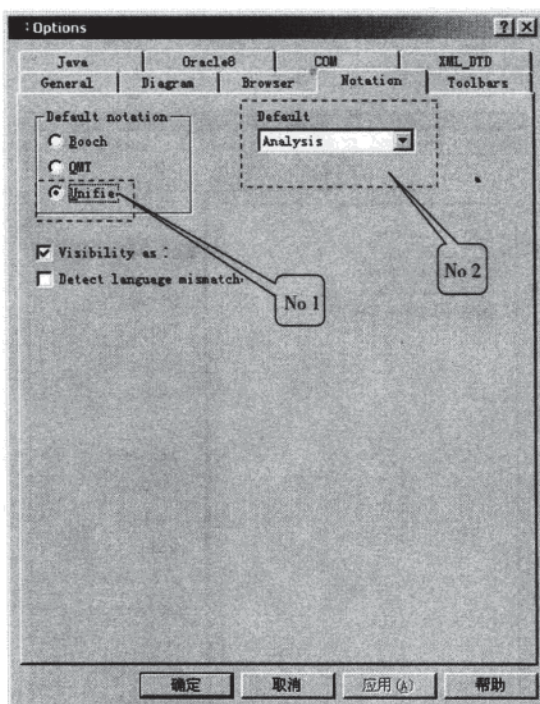


图 3-3 默认语言的设置界面

语言进行选择。

(2) 设置 Analysis Model

如果在前面创建整个模型文件时使用的是 RUP 的模板，现在就可以直接找到“Analysis Model”项，如图 3-4 所示。

否则，就需要在“Logical View”下“New”一个“Package”，然后把它命名为“Analysis Model”，如图 3-5 所示。

再往下的步骤对于两种模型的构建方式和过程就都是相同的，所以下面就不再分开描述了。

(3) 创建对应 Use Case 的分析模型

在“Analysis Model”下“New”一个“Package”，然后双击它或者在右键菜单中选择“Open Specification...”项，这时候会出现如图 3-6 所示的对话框供开发人员设置 Use Case 子系统的属性。

这时模型文件中就可以看到如图 3-7 所示中的内容。

声明：

千万不要忽略这些说明性文字，这有助于项目进行的中后期减轻你的工作量，提高工作

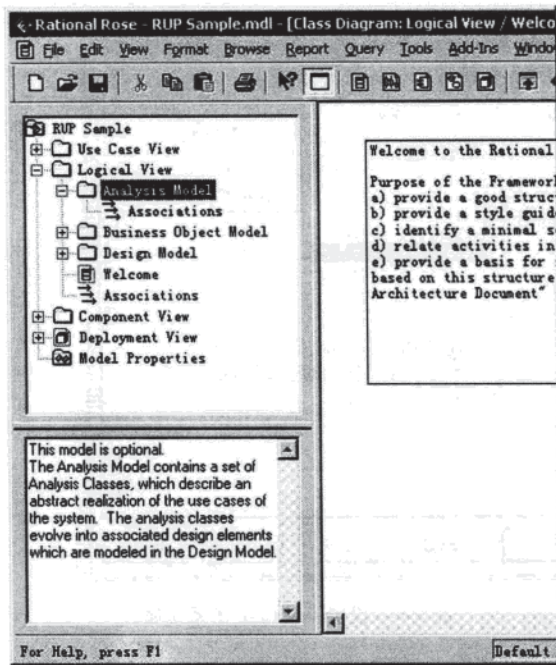
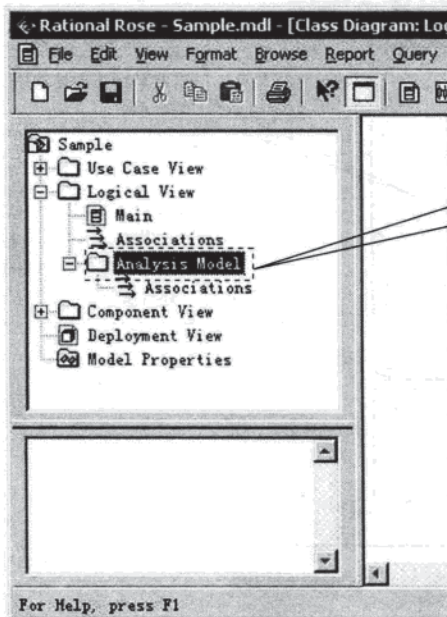


图 3-4 RUP 的分析模型



这就是我们刚创建的
分析模型标识

图 3-5 设置分析模型

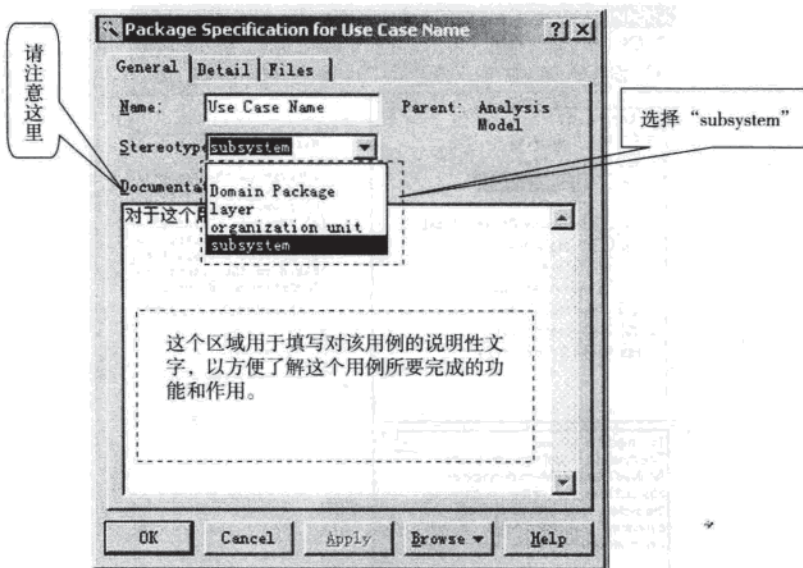


图 3-6 设置 Use Case 子系统属性

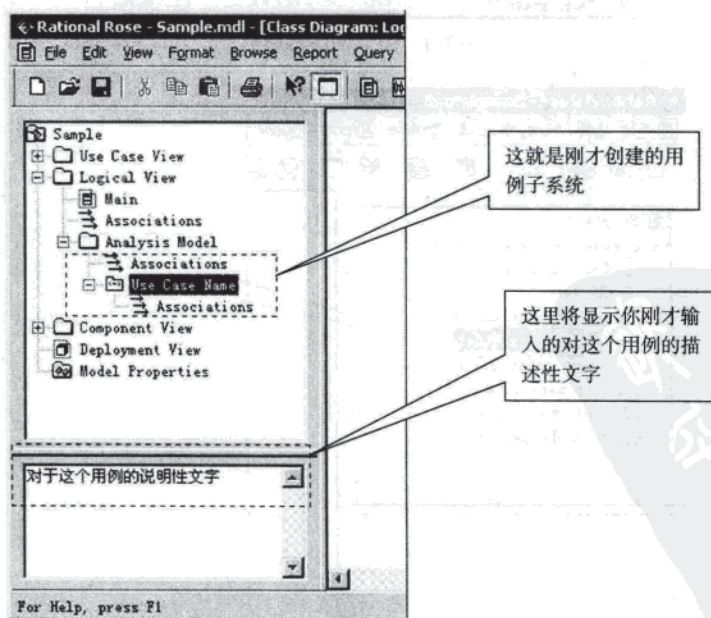


图 3-7 分析模型视图

效率，减少一些不必要的交流和无谓的讨论。而且，在从模型生成文档的过程中，这些文字

都将起到非常重要的辅助理解作用。

(4) 创建一个 Class

在“Use Case Name”下“New”一个“Class”，并选择“Open Specification...”来设置它的属性，如图 3-8 所示。

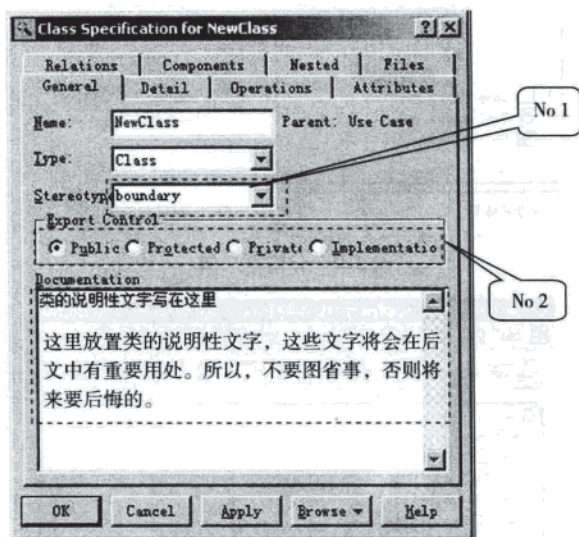


图 3-8 添加分析类

注意：

其他的标签选项一般情况下可以不做任何选择。

图 3-8 的说明：

No 1 这里要设置类的类别。由于采用了 MVC 的模式，所以，这里只选择使用其中 3 种：boundary（边界类）、control（控制类）、entity（实体类）。

No 2 在分析模型中，直接使用默认的“Public”设置。

图 3-8 确定后，就完成了边界类的创建。接下来继续创建剩下的两种类：控制类和实体类。

(5) 创建控制类和实体类

按照上面的步骤，开发人员可以依次完成控制类和实体类的创建，完成后，整个模型文件的视图就变成了如图 3-9 所示的样子，请注意它们图标差别。

(6) 得到分析模型

如图 3-10 所示是一个实际工程项目的分析模型部分。开发人员最终得到的分析模型中类图的示例基本就如此图所示。

图 3-11 是该分析模型时序图的局部示例。

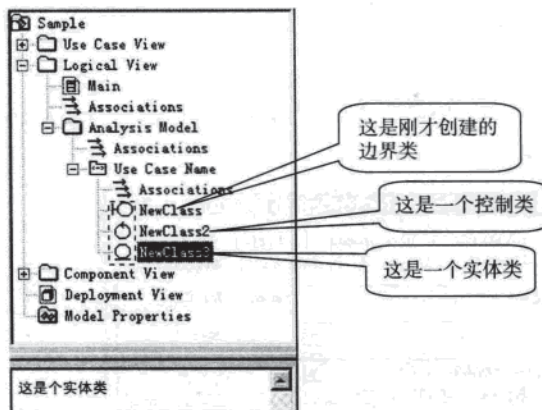


图 3-9 分析类示例

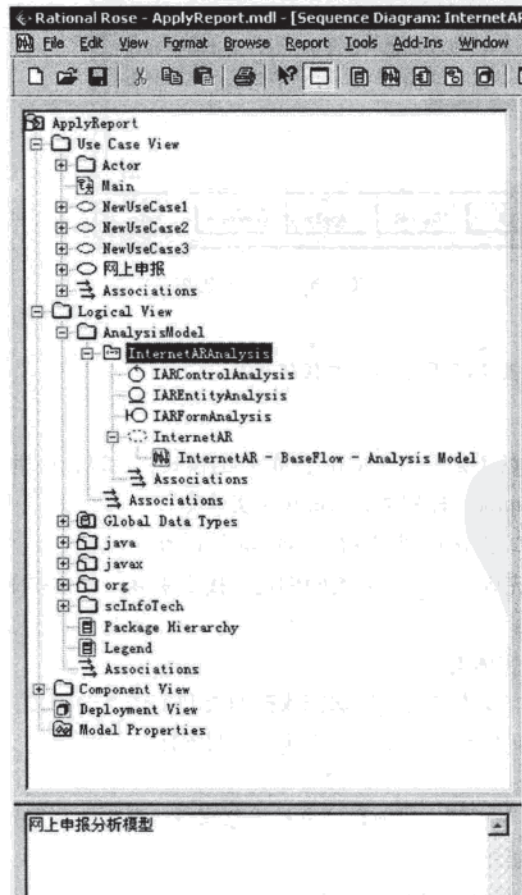


图 3-10 分析模型示例

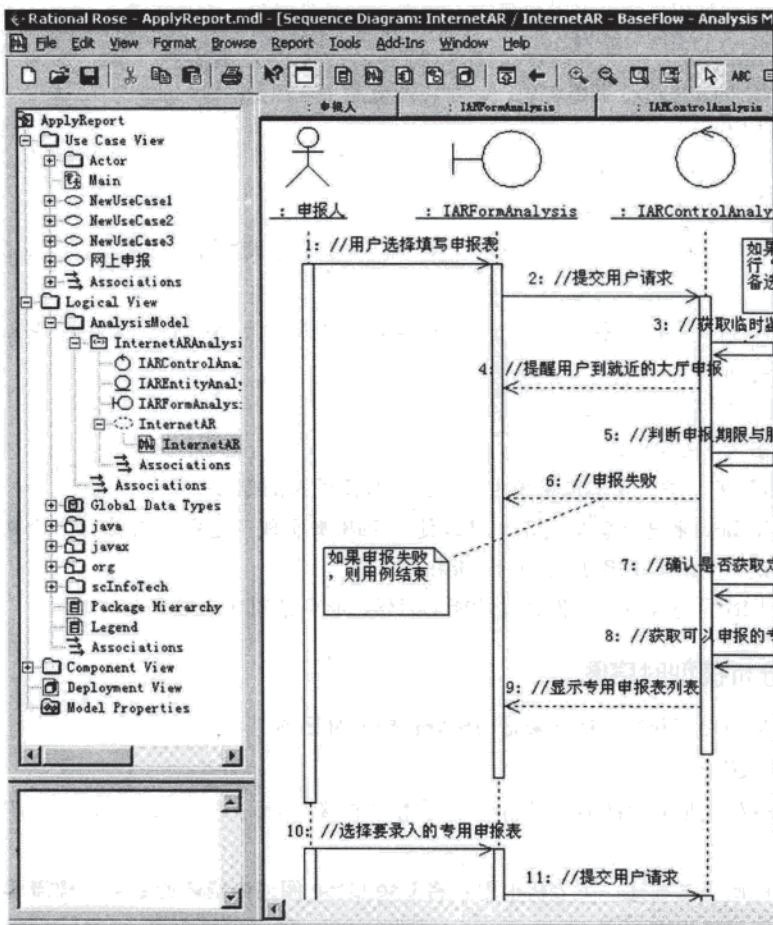


图 3-11 分析模型局部示例

2. 构建结果

下面将对各个分析类进行详细阐述。

注意：

因为笔者在 2006 年以前一直从事 Java 方面相关软件的开发，所以这里的分析和设计过程都将主要基于 Java 语言进行描述。

在 2005 年~2006 年期间，参加培训的学员都曾使用 VB、VC 等语言进行了建模开发的全过程实践。

(1) 边界类

——命名为 FormAnalysis

该类是表现与用户交互的直接界面。在进行设计的时候,对应于 B/S 结构中的浏览器展示的页面部分。

如果采用 Java 来进行系统实现,它对应的就是 JSP 部分。

(2) 控制类

——命名为 ControlAnalysis

该类是完成整个流程控制与业务处理的部分。

这个类将成为整个系统实现的核心,所有的业务实现和逻辑处理都放在这里,需要特别注意。

如果采用 Java 来进行系统实现,它可能会分化成后台的 Servlet、普通的 Bean 以及部分 EJB 中的 Session Bean 等。

(3) 模型类

——命名为 EntityAnalysis

这是数据/模型类,是实现系统中直接进行数据处理的部分。

如果采用 Java 来进行系统实现,可以使用 EJB 来实现,也可以直接写一个数据管理的类。一般来说,对应于 EJB 中的 Entity Bean。

当整个 EJB 只完成一个数据处理功能的时候,它可能对应于整个 EJB 层。

3.3.3 分析模型时序图

完成了类的初步开发,接下来是构建分析模型时序图。

1. 目的和作用

在构建分析模型时序图时,其目的是要用户能够看懂,或者说使初步了解 UML 的用户能够看明白。

用通俗的语言来描述一下分析模型:将 Use Case 阐述中描述的业务处理流程通过 UML 的时序图形式表示出来。

只要做到了这一点,那么,分析模型的最终目标就达到了。

换句话说,与编码更接近的细化事情,就交给设计人员去完成。

2. 如何绘制时序图

绘制时序图的具体步骤如下。

(1) 创建时序图

在 Use Case 子系统下选择 “New” → “Sequence Diagram” 选项,创建一个时序图,如图 3-12 所示。

(2) 为时序图命名

对时序图的命名并没有什么特殊的要求,需要注意的一点是:要让其他人能够方便地读懂你在时序图中所要表达的意思。也就是说,让其他人从名字上就可以直接了解该时序图所表示的内容。这一点尤其是在团队协作开发中显得格外重要。

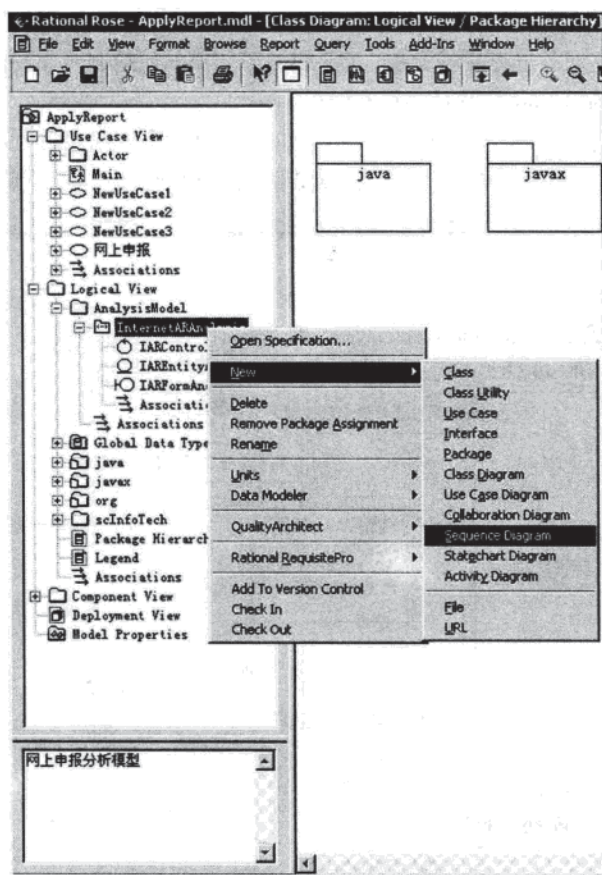


图 3-12 创建时序图

建议的命名规则：Use Case 名称 - 流名称 - 模型分类。

例如：

可将“登录系统（Use Case）的基本流分析模型的时序图”命名为

LoginSystem-Base Flow-Analysis Model

而“登录系统（Use Case）的第三子流分析模型的时序图”命名为：

LoginSystem-3rd SubFlow-Analysis Model

(3) 绘制时序图细节

完成准备工作后，就可以开始绘制时序图的细节了。

与 2.7 节相对应，交互建模时的用户操作成为 Actor 到边界类的操作关系，而系统响应则成为分析类到边界类的返回信息，然后根据系统的简单逻辑关系完成控制类与实体类之间

的行为联系。

所以，绘制时序图的过程并不复杂。如果仍然不知道如何绘制时序图的话，可以参考一些介绍 UML 和 Rational Rose 操作的书籍和资料。

本书在这里只是针对一些开发中需要注意的问题和通常需要用到的约定进行简要说明。

如图 3-13 所示，是一个实际工程项目中绘制的分析模型时序图的一部分。

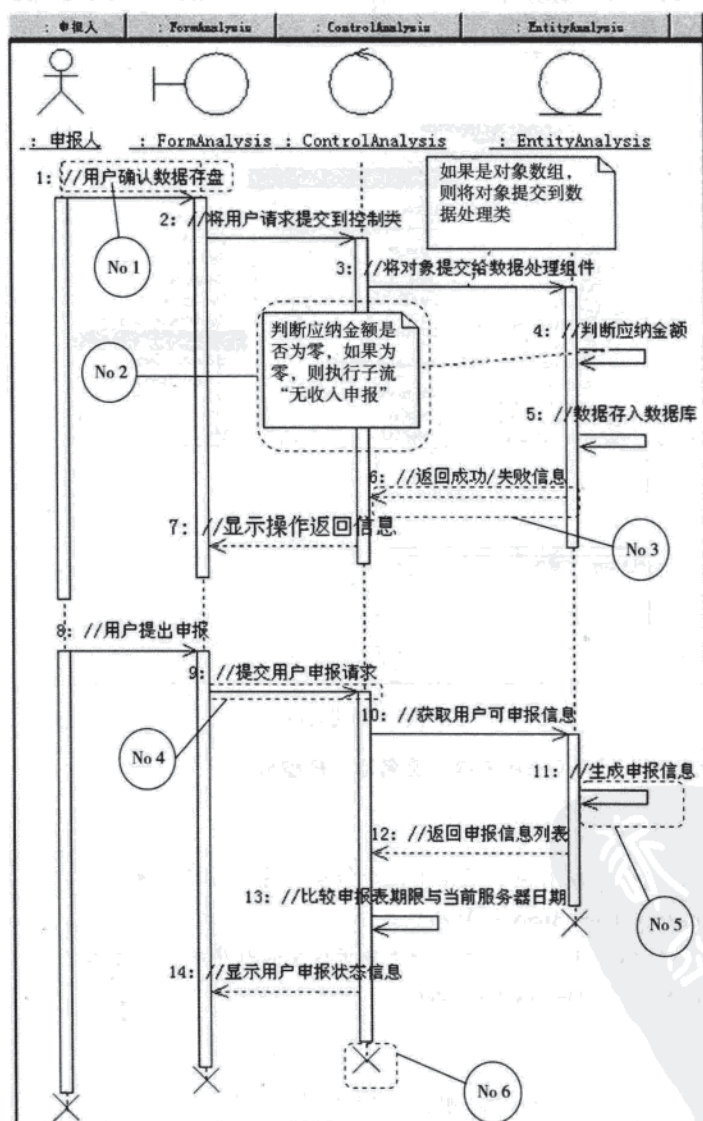


图 3-13 时序图示例

图 3-13 的说明:

No 1 分析模型的方法都是注释性质的,所以前面一般都要加上“//”标记。这里的方法和返回信息都可以使用中文,便于一般用户能看懂整个流程。如果不加“//”标记,那么 Rose 就认为这些是实际的方法,会将它们用符号直接添加到分析类的上面,这样也就失去注释的意义了。

No 2 在一些需要附加说明,或者系统发生跳转、调用、分支的地方,一般使用标签来做出说明。

No 3 返回信息要使用“Return message”箭头。

No 4 信息请求、信息发送、方法调用等一般使用“Object Message”箭头。

No 5 当发生循环调用、递归或者一些暂时无法判断从何处获得信息的方法时,可以使用“Message to Self”箭头。

No 6 流程终止时,最好使用“Destruction Marker”来结束它的向下延伸,以免影响对下面流程的查看。这一点在设计模型(参见 3.4.3 小节中的图)中就可以得到很好的验证了。

到此为止,分析模型的时序图也就绘制完成了。

3. 初步结论

至此,需求分析工作基本上结束了。此后的工作将已不再需要用户的直接参与及评审了。

也许有人会问:系统分析员到底是做什么的?需求、分析、设计之间是什么关系?

基于这些考虑,本书将在第 8 章的 8.4 节中讨论什么是系统分析。

3.3.4 参与人员建议

1. 主要开发人员

分析模型开发人员,包括部分需求调研人员和设计实现人员。

2. 主要审核人员

用户方代表、项目经理、行业专家、系统分析员(需求调研人员)、Use Case 阐述人员、设计实现人员、编码实施中的主要负责人员。

注意:

关于用户方代表参与分析模型评审的基本要求是,用户方代表中有较高水平的,对计算机软件知识较为熟悉的,同时了解 UML 基本知识或者能看懂 UML 描述的人员,否则,就不需要用户的参与了。

3.3.5 输入输出

1. 输入工件

前景文档、需求规格说明书、补充规约、词汇表、交互建模工件集、Use Case 模型。

2. 输出工件

分析模型。

3.3.6 UML 元素

1. 相关 UML 元素

Use Case、Use Case 图、Use Case 模型、状态/活动图、时序图、类图。

2. 用处

阅读前面开发出来的 Use Case 模型，根据 Use Case 阐述内容完成分析模型时序图的开发，同时创建基础的分析类，形成分析类的类图。

3.3.7 常见问题

1. 问题一：非注释性方法直接出现在分析模型中

很多人在开发分析模型的过程中对类间的操作未添加或者忘记添加“//”符号来表示这个方法是注释性的。

如果不添加“//”标记，则系统会自动认为你所添加的这个方法就是实际类中的方法，这将会对你所开发的工件在交付给其他开发者的过程中产生理解上的偏差。也就是说，你所开发的分析模型会被误认为是设计模型而拿去实施，这样的错误虽然可以在评审会上得到解决，但是，大多数公司在要求进度的情况下都会忽略评审工作。这样就会使这类问题有了滋生繁衍的场所，请务必谨记，下面是一个实际发生这类错误的例子，如图 3-14 所示。

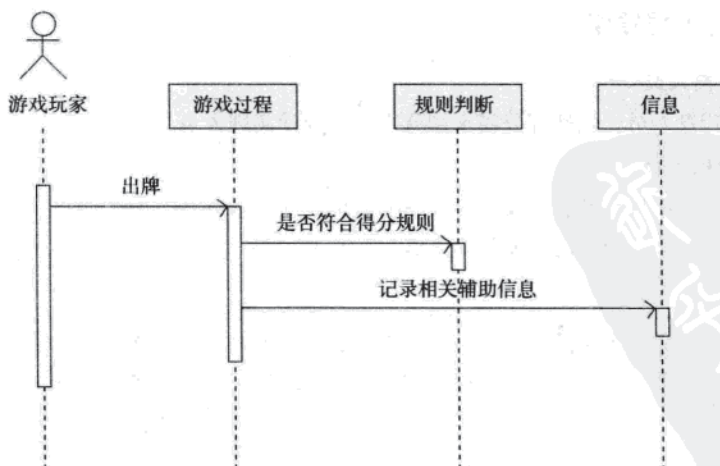


图 3-14 注释性方法错误地标记成类方法的示例

图 3-14 的时序图就会让 UML 工具给分析类“游戏过程”增加名为“出牌”的方法，其他类的方法增加也同理，这是错误的。分析类是不需要进行方法开发的。

2. 问题二：不标注方法名

有些人在开发时序图的过程中虽然思考了很多，但是，却没有写明每一个操作步骤是做什么的，这就产生了一种沟通上的障碍。

其他人将无法理解你需要表达的信息，也就是你对需求理解后得到的信息。设计人员无法根据你的分析模型来开发设计模型，这样会延误整个工程的工期，这样的例子如图 3-15 所示。

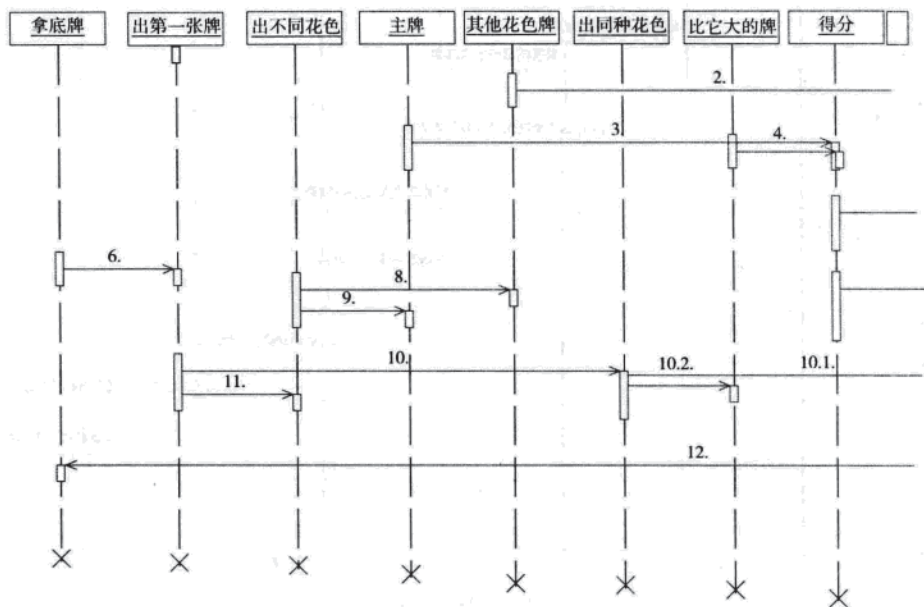


图 3-15 没有添加方法说明的示例

3. 问题三：数据流中断

如图 3-15 所示，每一个沿着虚线的细长矩形表示的实际上是数据流过程。

如果数据流发生中断，则表示后面是一个新操作的启动，而不是延续上面这个数据流业务过程的实现。

一个数据流中断的实际例子如图 3-16 所示。

特别说明：

2008 年 4 月笔者在南京某著名 500 强外企做 EA 工具使用培训的时候，看到其德国总部发过来的 UML 模型，他们的设计模型时序图就以该形式表示的，具体的例子此处不方便举。这也说明，国外在使用 UML 时同样，也存在有很多错误，甚至有些错误还很严重，不过，他们至少一直在用，而且敢用。

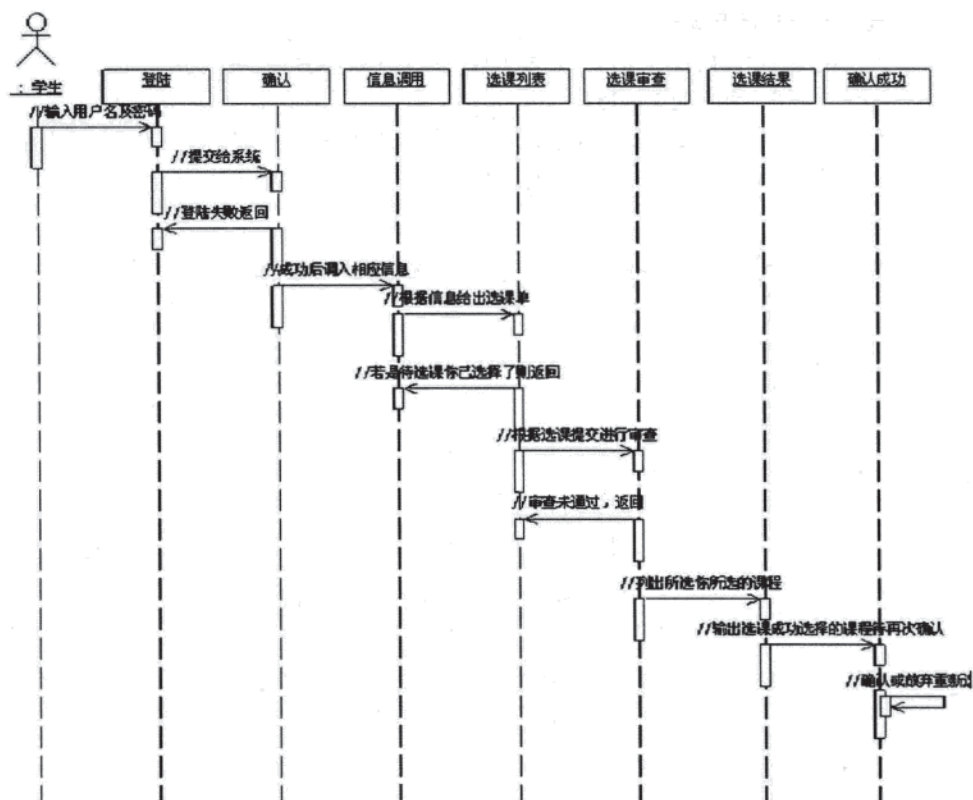


图 3-16 数据流中断示例

下面是数据流中断引发的另一个逻辑问题。

4. 问题四：Object Message 如何发生行为

Object Message 是一个行为，一个动作，是前一个类调用后一个类的某个方法引起的行为。但是，这个行为最初的形成，应该是一个有行为能力的 Actor 来驱动的，也就是说，应该有一个行为的起始者。

众所周知，任何类，都只能是一个被动者，而不是可以直接发出任何动作的行为者。

有人会说：这似乎不对？一些通过时间触发的类就应该是一个主动者，因为没有 Actor 来驱动这个 Object Message 的行为。

不，即使是时间触发也必须有一个 Actor 的主动性行为，那就是时间。同样的，一些开机后自动启动的程序所触发的动作，也是由于 Actor（人）启动计算机开关这个动作所引起的连锁行为，所以，这些都是被动的。

因此，在绘制时序图的过程中，一定要有相应的行为驱动者，也就是 Actor。即使是源于另一个 Use Case 的 Actor 的驱动所形成的连锁行为引发这个后续 Use Case 的行为。

一个行为错误的例子如图 3-17 所示。

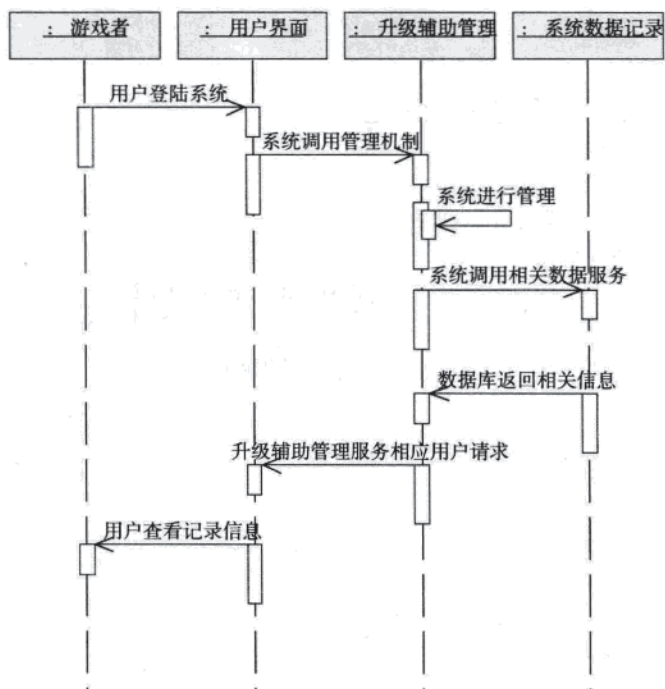


图 3-17 Object Message 的不正常行为

注意，这里发生的这类不正常行为往往都是因为数据流中断造成的，下面的问题五中描述的是另外一种类似的错误。

5. 问题五：Actor 本身是否可以有方法

曾有人在某个分析模型时序图中绘制出了如图 3-18 所示的图形。

注意：Actor 本身是一个系统外部的物，它可能是一个人，也可能只是一个外部系统。

在需要开发的系统中是不会对 Actor 进行任何修改或调整的，对于开发人员来说，这个 Actor 是一个完整的不可修改的系统，开发人员既不能也不需要考虑 Actor 本身有哪些方法，需要考虑的仅仅是 Actor 可以给现在这个系统提供哪些信息，以什么样的形式提供这些信息而已。

3.3.8 楚凡 UML 工具模型示例

下面的例子是 2009 年 1 月与西安楚凡合作培训中的培训实例：幼儿教育系统，如图 3-19 ~ 图 3-21 所示。

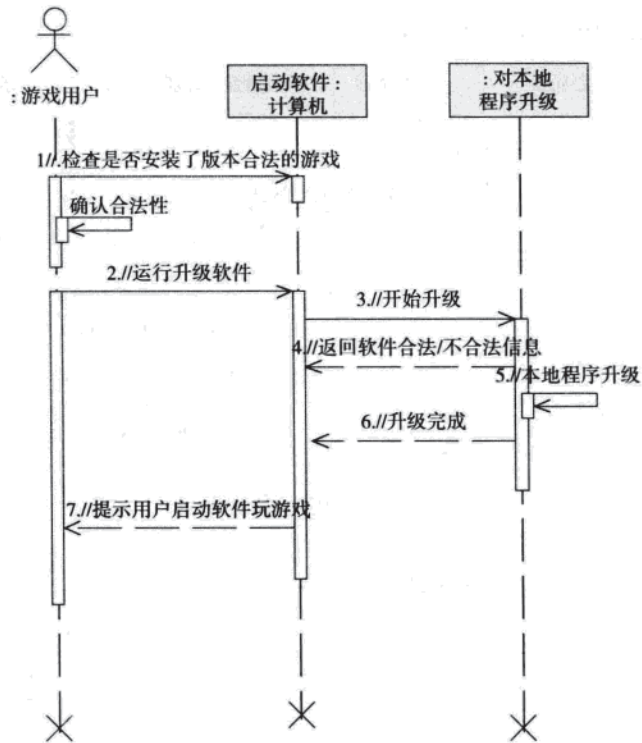


图 3-18 Actor 自身方法

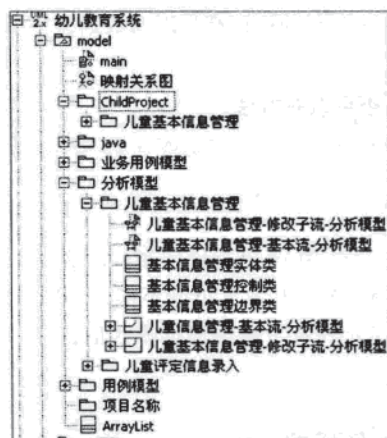


图 3-19 分析模型树形图

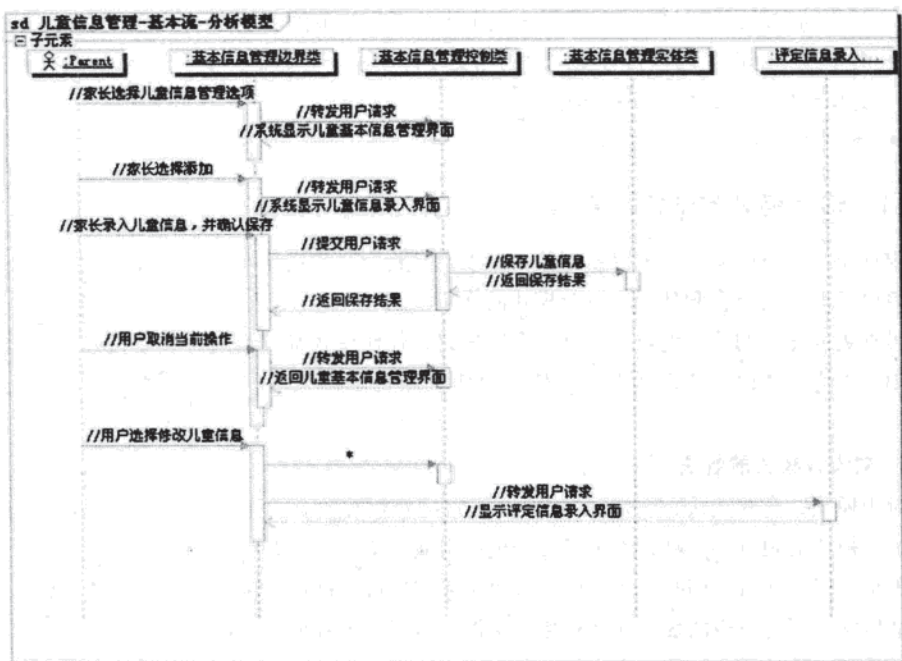


图 3-20 分析模型时序图基本流

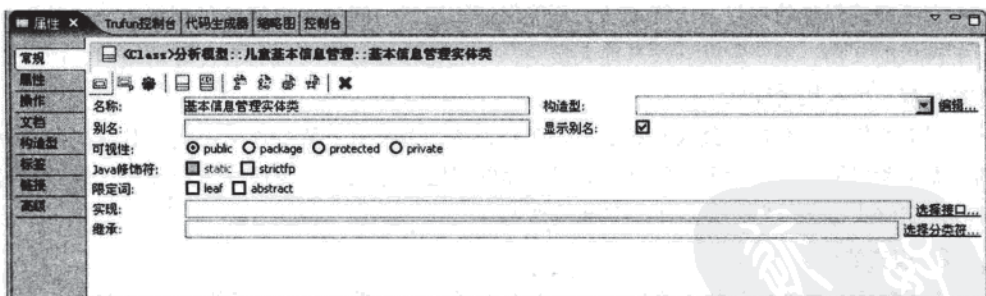


图 3-21 分析模型中的分析类

3.4 设计模型

设计模型将直接对应于编码实现，因此对开发人员的编码技术要求较高，同时还要求参与设计模型的开发人员具有较高的设计水平。

为了尽可能地做到设计与编码的分离，或者为了开发出更好的、更有效的设计模型，对设计人员一般有如下要求：

- 设计人员应用某种开发语言在相同方向上的软件开发至少进行过两个以上的同类实际

项目的辅助设计,参加过3~5个以上类似项目的编码;

- 对于这个方向上的项目可能应用的相关设计模式十分熟悉,至少要熟悉两种以上相关的设计模式,并能灵活地运用;
- 有较强的处理问题的能力。

3.4.1 设计模式的选择与应用

设计模型开发的第一步就是根据项目特点选择将要采用的设计模式。设计模式的选择要考虑该模式对应的语言和环境,以及模式的适用范围。

另外需要注意的是,绝对不能直接套用模式,任何模式都要经过一些变化和修改后才可以使用,因为这些模式都是从大量具体项目中抽象提炼出来的,而不是某一个项目的直接成果。

1. 对设计模式的看法

对于模式,有些人认为它是“银弹”,但紧接着就有关于不是“银弹”的观点,同时还出现了反模式的观点,于是就发生了关于模式到底是不是“银弹”的争论。

举个例子,如写文档时,我们已经习惯了采用模板,这样写出的文档风格比较统一,也容易被接受。因为风格的统一和规格化,文档整体看起来效果会比较好。

模式和软件项目之间的关系也是如此。

因此有了下面这样的结论:模式之于项目就像模板之于文档。

它不能确保你的项目一定能成功,不能保证所遇到的问题一定能够得到解决。这就像采用了模板的文档,不一定就会被其他人接受,只是比较容易撰写,具有良好的结构,因此比较容易被其他人接受而已。

同类的例子还有数学公式和数学问题间的关系,当采用了某个数学公式只对解决问题有利,而不能保证你一定能解决问题。对于实际的问题,你也许需要使用多个公式,然后通过计算才能解决这个问题。在这里也是相类似的。

2. 模式的分类

在设计模式这个名词提出之后,又逐渐有人提出了分析模式之类的模式名词。

模式在应用层面上可以分为架构级模式、设计级模式和编码级模式。

架构级模式往往与业务模式相互关联形成庞大体系的一个上层框架,然后内部通过设计级模式进行细化,最后通过编码级模式进行实现。著名的四人帮所编写的《设计模式》^①一书中介绍的23种设计模式其实是编码级模式的代表。

由于模式是人类从各种活动中抽取出来的一般性规律,所以根据其抽取的基础和结果分析,其中少量模式可以应用于多个层面,而大部分模式都有其具体的应用层级。

① 此书全名为《设计模式:可复用面向对象软件的基础》(双语版),其书号为ISBN 978-7-111-21126-6。——编辑注

著名的 MVC 模式就是一种多层面模式的代表,一些极端的 MVC 崇拜者认为 MVC 下还可以进行 MVC 模式的划分,每一个 MVC 层也都可以根据 MVC 的原则进行 MVC 的划分。

3. 模式的演变

这个话题目前提出来还有些过早,但是,我们至少看到在 UML 工具中已经集成了对编码级模式的应用。

2005 年 3 月,笔者曾写下了这样一句话:随着技术的进步和实现手段的集成化程度的提高,与底层接近的设计模式将会首先被集成为固定的或被自动执行完成的内容,而不再需要人工来编写修改了。

随着开发模式的演进,代码库(请参见第 6 章中 6.4.5 小节中企业代码库的内容)完全建立以后,系统就可以融合更高层次的设计模式,也就是设计级模式。

实践中,如果得到具体工具的支持,在固定的架构模式的支持下,按照全程建模方法论中的需求确定后就可以直接映射出该模块应该有哪些主要的类和对象,以及这些类和对象中的主要方法。也就是说,对于相同的业务或者相似的业务模块的开发,可以从需求直接映射出代码的实现,然后对设计模型和代码进行修补即可。

这也算是一种超越 MDA 实现方式的更高一层的开发方法。

4. 模式的选择

在前面进行的项目中采用了 J2EE 核心模式中的表示层模式(参见《J2EE 核心模式》一书(机械工业出版社出版, Sun 公司核心技术丛书)中对这个模式的介绍)。

下面将以这个模式为例对模式的应用和开发过程进行描述,所以在这里要对这个模式做一下简单的描述。

注意:

本书涉及的几个项目中,笔者都是采用这个模式对 MVC 模式进行扩展的,如图 3-22 所示。

这里,“JSP 页面”对应着“V”即显示层,控制类和助手类对应“C”即控制层,实体类和数据类对应“M”即模型层。

5. 该模式的特点和说明

这种模式有以下特点:

1) 可以减少业务逻辑的变化对前台的影响,也可以减少前台显示界面的变化对后台的影响,提高前后台之间的相对独立性,减少了耦合度。

该模式可以屏蔽前端与后台间的直接关系,后台不需要知道前台有哪些界面,控制类通过助手类的返回值进行组合后分发到具体的前端展现页面(JSP 页面),而前台的开发也不需要知道后台有哪些业务实现类(助手类和模型类),只要前台将请求发送到控制类,控制类对它进行解析后提交到相应的业务实现类。

2) 四个层面分工明确,便于分工协作开发。

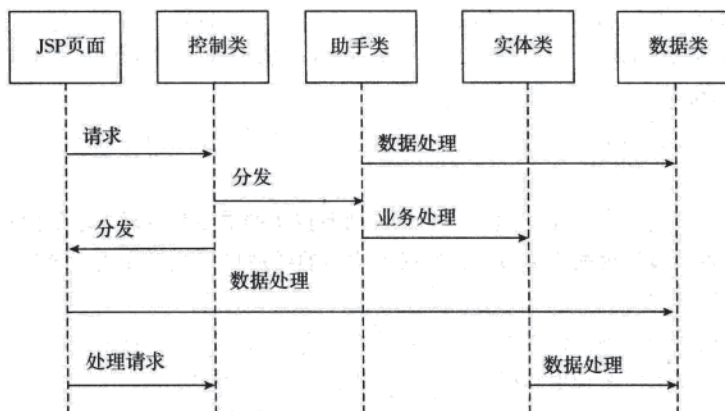


图 3-22 所采用的设计模式示意图

控制类管理流程跳转、JSP 页面负责界面展示、助手类处理业务逻辑、实体类实现数据处理；四个部分可以分给 4 个不同的开发人员进行并行开发，互不影响，相对独立，有利于提高了整个系统的开发效率，减少了开发周期。

这种模式，对于同一类的 JSP 页面的增加和修改，只需要对 JSP 页面进行开发，而不需要对后台业务实现进行修改；而后台的修改，除了返回类型增加时，需要增加相应的 JSP 页面外，也不需要控制类和其他 JSP 页面进行任何修改。

这样也就屏蔽了前后台间的直接交互，即使有人获得了前台的 JSP 页面的物理地址，也无法直接对后台进行任何操作，因为所有的分发都是隐性的，整个系统的安全性也可以在系统架构层面上得到保障。

3.4.2 设计类的构建

所有的设计类都是从分析类中衍生出来的。

在分析模型上应用设计模式后，分析类就会进行分化。一般来说，每一个分析类都会由于设计模式的介入而分化成几个设计类。

类的种类增加了，相应的技术要求也更高了，能够对模型进行评价的人也就更少了。设计模型对软件专业技能的要求比能看懂分析模型要高得多。

1. 如何从分析类生成设计类

如何根据模式将分析类转化成设计类，这样做的依据是什么？下面将进行详细的阐述。

在 MVC 模式基础上应用了所选择的设计模式后，分析类将被细化成如下 5 种设计类：

1) 分析边界类 FormAnalysis 仍然不变，因为在使用 Rose 进行 B/S 结构的系统开发中，一般不需要把浏览器展示的页面也绘制在 Rose 的模型文件中。

对 JSP 页面不同的考虑会有很多种不同的结果，开发人员可以在一个 JSP 页面中写完所有的界面，也可以一个结果写一个 JSP 页面形成多个 JSP 页面，甚至也可以将所有的结果都

放在一个 JSP 页面中进行处理。

注意:

如果重新创建了一个设计中的边界类,因为在 B/S 结构中,这个边界类一般不需要以代码的形式导出,所以,会在后面代码导出时生成一个不需要的类。笔者的习惯是不导出代码的类也就不重新创建,而直接引用对应的分析类来替代其在设计模型中的位置。

但是,如果是 C/S 系统的设计或者桌面软件的设计则需要对边界类进行设计开发。在 3.4.8 小节中有对应的设计模型实例可供参考,同时也提供了同一功能的 B/S 设计与 C/S 设计的对比设计模型。

2) 分析控制类 ControlAnalysis 类分化成控制类和助手类。

3) 分析实体类 EntityAnalysis 类分化成实体类和数据类。

例如,在某一个项目中,笔者把相应的类做了如下转化:

1) 边界类 FormAnalysis 转换成了所有的 JSP 页面。

2) ControlAnalysis 类转化成了一个控制类: IARControl.java; 3 个助手类: 鉴定信息助手类 IARApplyHelper.java、申报表助手类 IARApplyTableHelper.java 和申报扣款助手类 IARDeclareHelper.java。

3) EntityAnalysis 类转化成了一个实体类(数据管理类): IAREntity.java; 4 个数据类: 申报鉴定信息类 IIdentifyInfo.java、专用申报表类 IARApplyTableObject.java、申报信息对象类 IARDeclareInfo.java、申报人基本信息类 IPayerInfo.java。

也就是说,3 个分析模型中的控制类和模型类被转换成了 9 个设计类。

通过这样的划分,就完成了分析类向设计类的第一步转换。

2. 分析类与设计类的关系

从上文中可以得到这样的关系式:

分析类 + 设计模式 $\Rightarrow$ 设计类。

如果要更准确地表达出这种关系,应该是下面这个关系式:

分析类(某个) + 设计模式(修改后) $\Rightarrow$ 设计类(多个)。

注意:

这里很少会出现多个分析类在设计阶段合并成某个或者某几个设计类的情况,一般都是一个分析类划分为多个设计类实现。

3. 设计类的开发步骤

操作步骤如下所示:

1) 与分析类的第一个步骤相类似,但需要进行两项设置。选择“Tools”下的“Options...”选项,然后选择“Notation”标签和“Diagram”标签,如图 3-23 和图 3-24 所示。

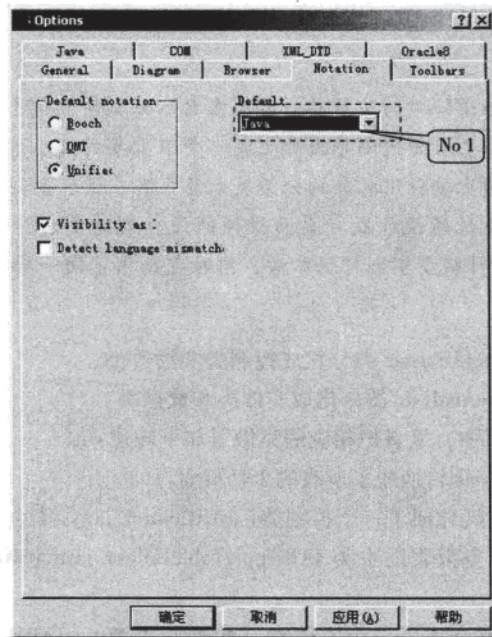


图 3-23 设定设计模型开发的默认语言

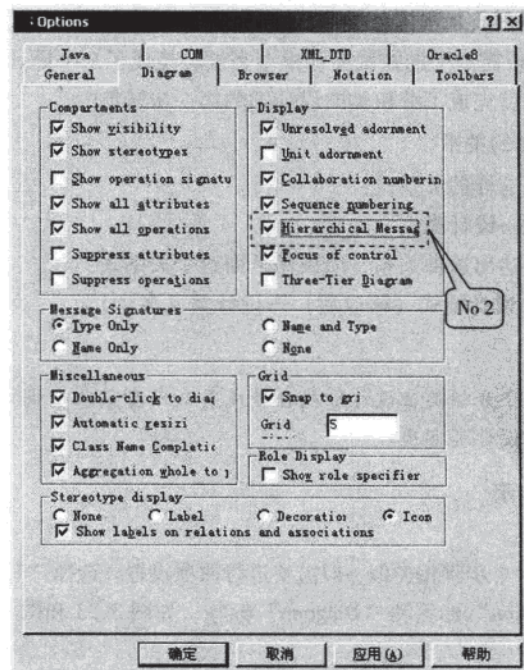


图 3-24 设置层级消息显示

图 3-23 ~ 图 3-24 的说明:

No 1 要确保在进行设计模型开发时, 选择要开发这个项目所采用的语言。笔者的项目是用 Java 开发的, 所以这里选择“Java”语言, 否则在进行设计时就会出现语言选择错误问题, 使得无法正确进行设计。

No 2 选择该项, 因为它可以使设计模型时序图更有层次感, 逻辑关系更明确。

如果选择此项, 时序图的层级关系将如图 3-41 所示。

如果不选择此项, 时序图将变成如图 3-25 的样子。

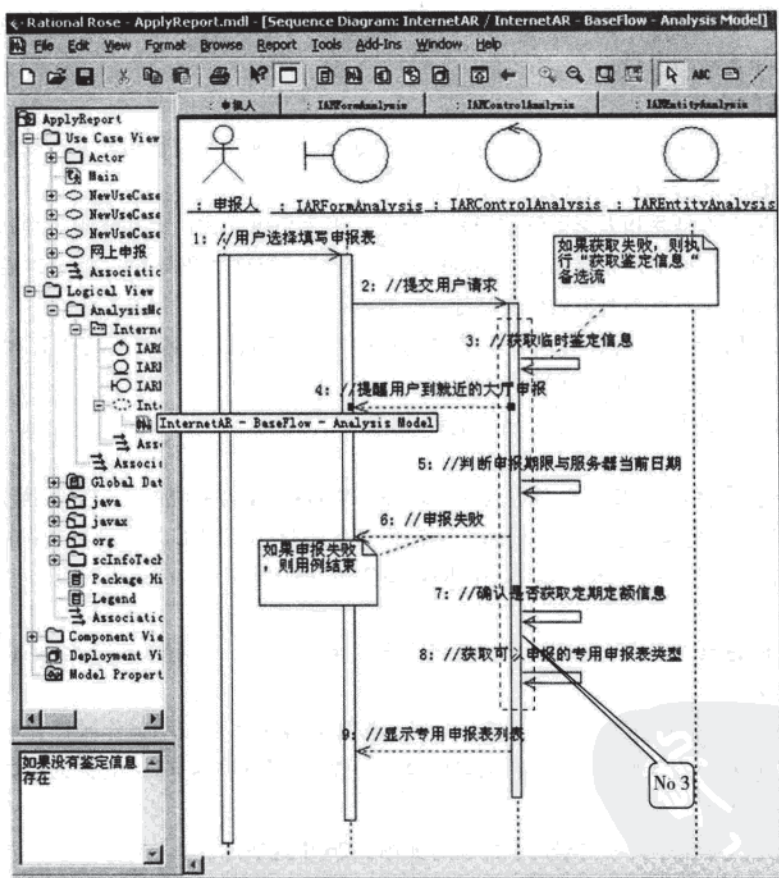


图 3-25 示意时序图

图 3-25 的说明:

这里的 No 3 就是与图 3-41 的区别, 可以明确地表示出是哪一个方法的返回或者方法的作用范围。

完成设置后, 就可以开始系统的分包和设计类的模型开发了。

2) 对于一个软件项目,如果不是很特殊的系统一般来说都需要做分包处理,请参看下面有关系统的分包部分的阐述;对于一些非常小的桌面软件,如果不需要进行系统的分包操作,那么就可以跨过这一步直接进行设计类的开发,请参见下面有关设计类的开发部分的内容。

3) 对于设计类的初步构建,下面将结合一个示例来进行描述,请参看下面有关设计类的开发部分的阐述。

4) 完成设计类的初步开发后,接下来是设计模型时序图的开发工作,这部分内容请参看 3.4.3 小节的介绍。

5) 完成时序图后,设计模型的开发工作基本结束,下一步的任务就是导出代码,详细描述请参见第 4 章的内容。

4. 系统的分包

在描述分包的时候,一般来说,可按如下的划分方式对分包进行描述:按功能划分、按位置划分、按业务关系划分、按操作顺序划分等。

在此,笔者不是要和大家讲解如何做分包,因为分包问题已经在需求工程过程中有过很详细地介绍了,这里是介绍如何在 Rose 中进行分包,以便代码导出后就可以直接在集成开发环境中使用,而不需要再做任何其他不必要的修改。

通常在使用 Rose 进行设计时,会按照与分析模型相似的命名规则将设计模型命名为“Design Model”,然后在其下面做包的划分。

为了确保后期代码能有效地导出,笔者建议:

不要使用类似分析模型命名方式的“Design Model”作为系统的主包名,因为一般的系统在进行部署时都不会使用这样的名字来作为代码目录,UML 工具导出代码的同时会将这个名称导出并支持后续代码模型的同步操作。

笔者习惯的操作方式是:按照项目的名称来定义主包名,对其下面的子包再做详细的划分和命名定义。

其下的子包可以按照 Use Case、按照功能、按照业务关系、按照操作顺序命名定义,也可以按照任何你认为合理的方式进行命名定义。一般来说使用 Use Case 名称的全称或者缩写加上分类进行命名。

例如:在某项目中,项目组对包的命名如图 3-26 中这所示。

这样,导出的代码所在的路径为:

(指定的目录) \ scInfoTech \ iar \

详细内容可参见第 4 章的图 4-17 所示。

此时,将你所参与的工程路径也设置成同样的目录结构,就可以实现 Rose 中代码导出到集成开发环境编辑器之间的代码位置的统一,不需要再进行任何路径设置就可以实现代码与模型的一体化操作了。

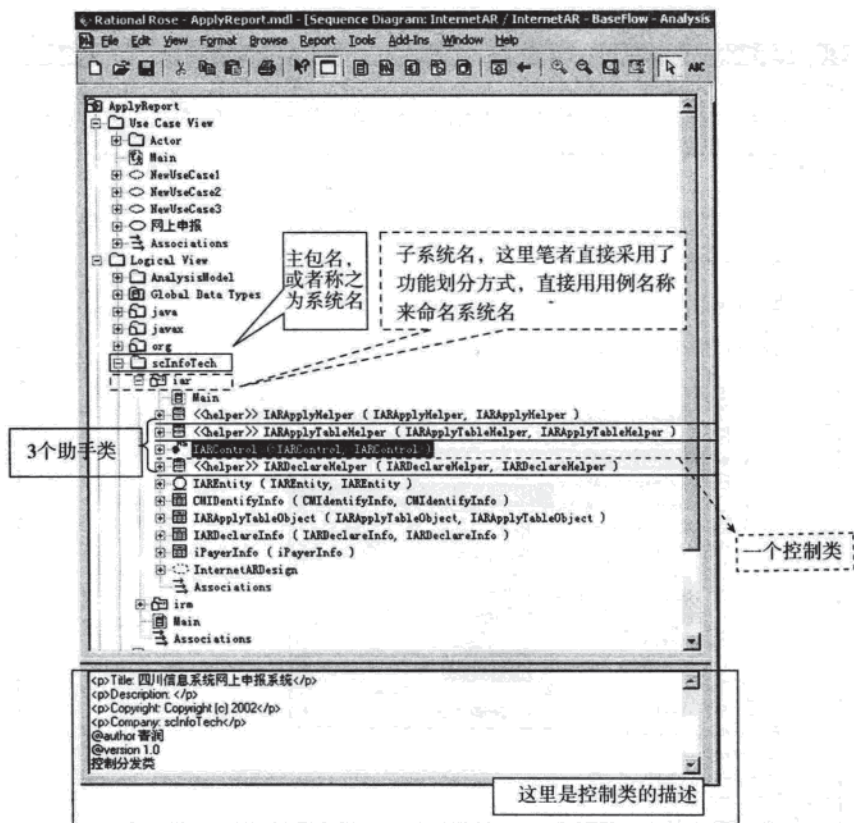


图 3-26 包的命名和设计类

注：其他的 UML 工具也都支持这样的导出定义方式。

完成了包的命名后，下一步就要进行设计类的开发了。

5. 设计类的开发（示例说明）

这里以上文中提到的控制类 `IARControl.java` 为例来说明一个实际可以导出有用代码的类的设计过程。

该控制类主要功能是进行系统的控制分发，一方面是把页面的请求分发到具体的助手类，另一方面把助手类的返回信息分发到具体的 JSP 页面（也就是边界类）进行前端展现。

第一步：创建类的过程

- 1) 在 `iar` 下 “New” 一个 Class，命名为 “`IARControl`”。
- 2) 选择 “`IARControl`” → “Open Standard Specification” 选项，如图 3-27 所示。
- 3) 在弹出的对话框中将 Type 设置为 “Class”，Stereotype 设置为 “Http_Servlet”，如图 3-28 所示。

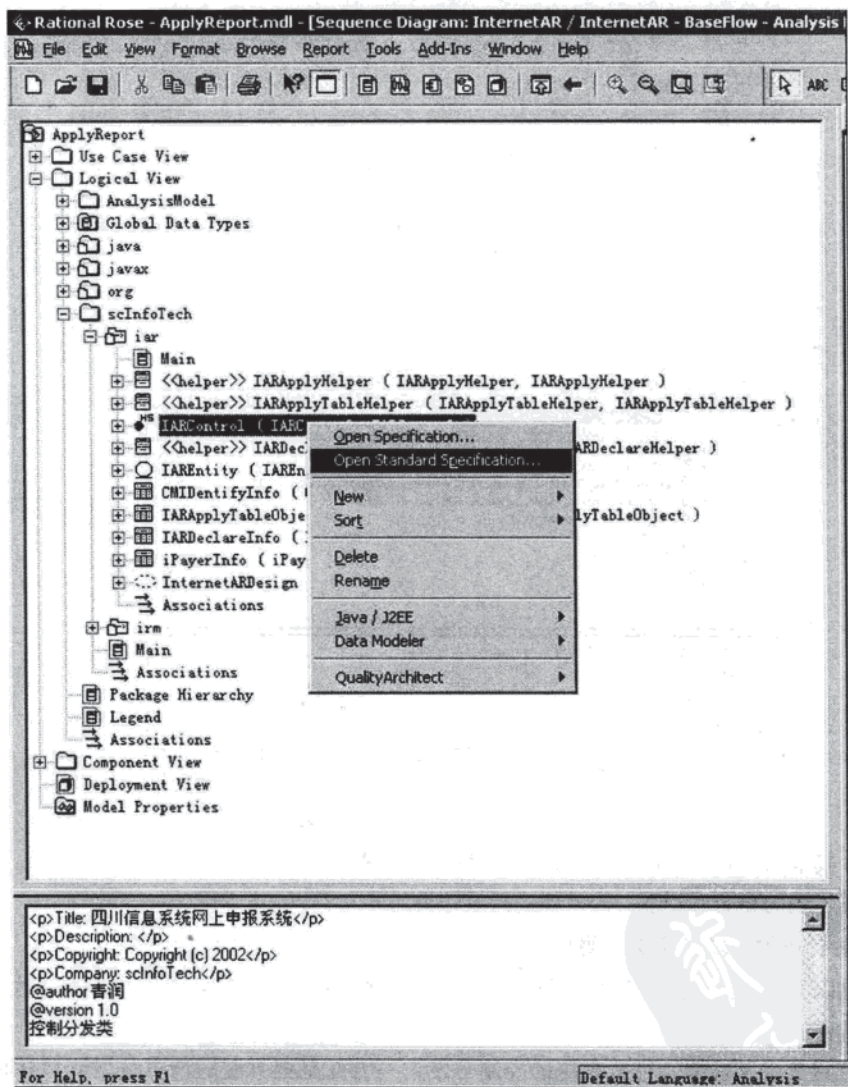


图 3-27 选择“Open Standard Specification”选项

注意，图 3-28 中的 9 个标签表示的是在类开发过程中要逐步解决的 9 个类的重要问题。

Genera 标签表示一般属性，General 中需要标识出类的名称（Name）、类型（Type）、版型（Stereotype）、输出控制（Export Control）、类的描述（Documentation）。

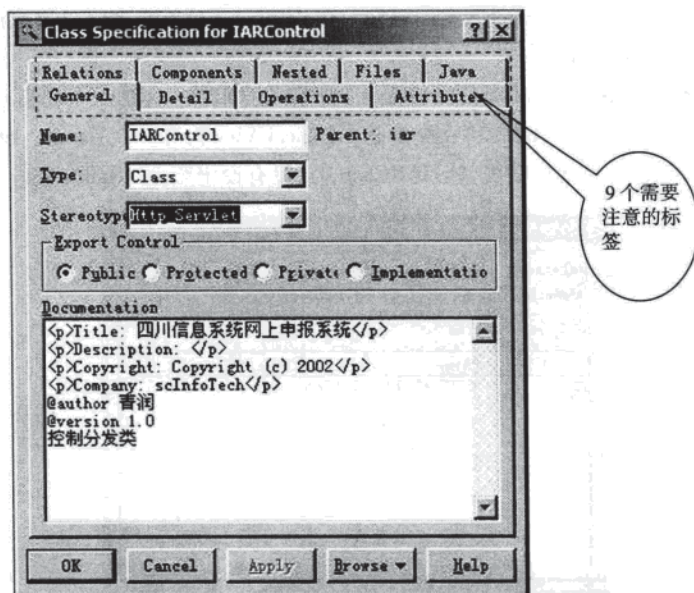


图 3-28 设置方法初始属性

图 3-28 中表示该控制类信息如下：

名称是：IARControl；
 类型是：Class；
 版型是：Http_Servlet；
 输出控制是：Public；
 类的描述是：
 “<p>Title: 四川信息系统网上申报系统</p>
 <p>Description: </p>
 <p>Copyright: Copyright (c) 2002</p>
 <p>Company: scInfoTech</p>
 @ author 青润
 @ version 1.0
 控制分发类”

4) 接下来，依次设置 IARControl 类的其他几个标签的内容。

通常情况下，不需要对各个标签页内容做特殊修改，直接使用默认设置即可以，只有在下面这两种情况下（如图 3-29 和图 3-30）需要按照要求进行设置。

Attributes 标签是类中所有属性的描述标签，对应于 Operations 标签中的方法，其中包含

所有自动生成的默认属性和开发者创建的属性。在 Attributes 标签页面也可以直接对该类的属性和值进行修改和添加/删除操作。

图 3-29 中显示了 IARControl 类的属性是 CONTENT_TYPE。这是一个私有属性，其初始化为："text/html; charset = GB2132"；类型是：Logical View:: java:: lang:: String；其描述是：Attributes declaration；他的父类是：IARControl，没有设定版型（stereotype）。

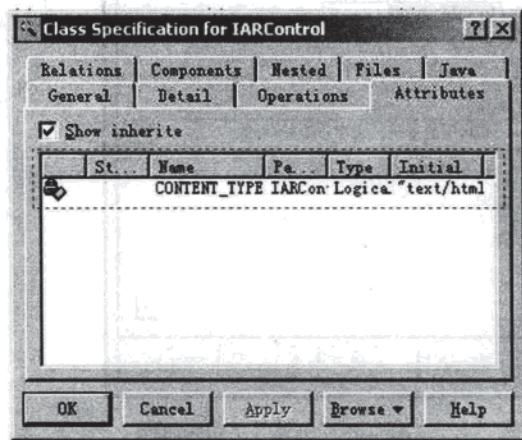


图 3-29 设置类的属性

选中“Show inherit”选项，表示要在这个界面显示出属性的继承关系。

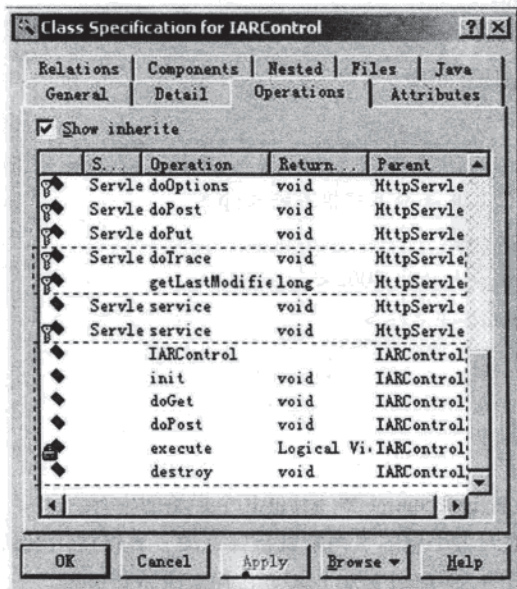


图 3-30 设置类的方法

Operations 标签是类中所有方法的描述标签,其中包含自动生成的默认方法和开发者创建的方法。在该标签页面可以直接对该类的方法进行修改和添加/删除操作。

图 3-30 中虚线框内的方法都是需要添加的,不过,也可以通过其他方式形成在这里添加方法并不是最常用的方式,所以也可跳过这一步,但在审核的时候,可以来这里进行检查。

另外,除了需要添加的方法外,对于 Servlet 还有一些已定义好的默认方法,如 doOptions、doPost、doPut、doTrace、service 等方法。

其他标签,诸如 Detail 标签、Relations 标签、Components 标签、Nested 标签、Files 标签都不需要额外说明,只有开发语言标签需要额外说明一下。

开发语言标签在这里显示的是 Java,是因为这个模型在前面设定开发语言时选择的是 Java,否则,这里显示的可以是你选择的其他开发语言标签,诸如:VC++, Delphi, 等等。

5) 打开 IARControl 的 Specification 选项,设置其相关属性,如图 3-31 和图 3-32 所示。

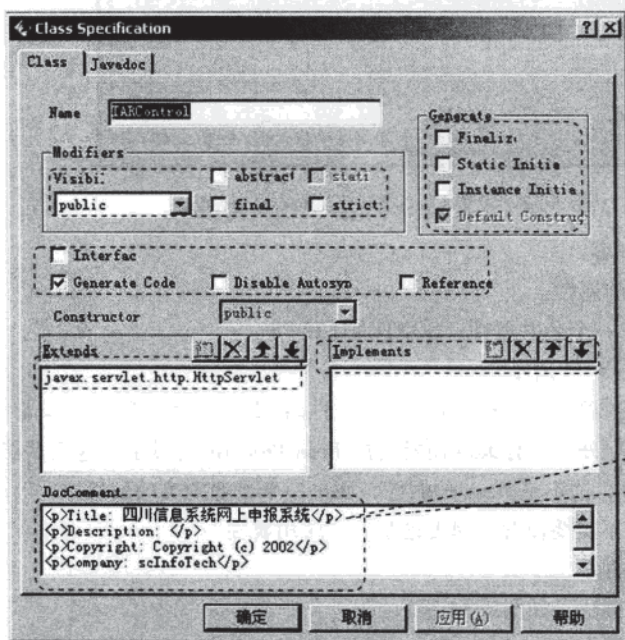


图 3-31 设置 Class Specification 的 Class

图 3-31 中是 IARControl 这个类的 Java 描述,可以看到大量与 Java 语言相关的信息和状态。如果在进入设计模型前没有设定语言,那么,系统将一直按照原始的设置继续执行下去,使得在代码编写阶段无法导出代码。

图 3-32 中描述了 IARControl 类的 JavaDoc,这是 Java 所特有的描述方式。

在这里定义好的所有内容都可以在代码编写阶段形成有用的 JavaDoc 文档,有助于其他人掌握你所开发的这个类的详细信息。

图中的内容都对应于 JavaDoc 的相关信息,这里不多做描述,读者可以参考 JavaDoc 相关

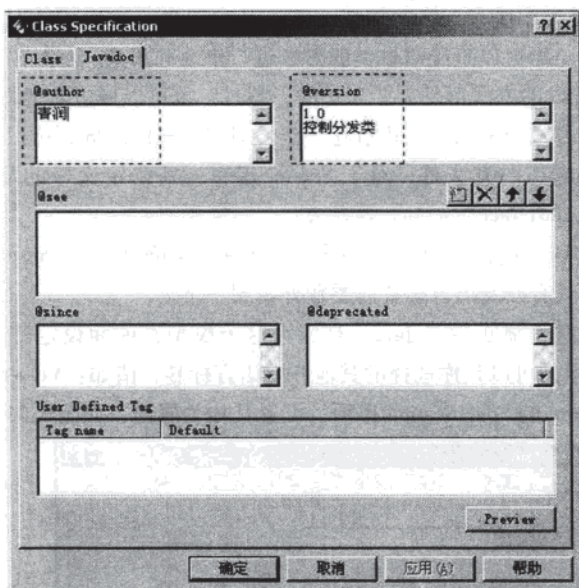


图 3-32 设置 Class Specification 的 Javadoc

的技术文档来了解这些内容。

6) 添加 IARControl 的属性。

这里 IARControl 只有一个公共属性: CONTENT_TYPE。

对它的详细描述如图 3-34 所示。

图 3-33 (续) 是图 3-29 中属性的 Java 版型。可以看出, 在图 3-33 中除了常规的属性描述 (如图 3-29 中的内容) 外, 还有 Java 所特有的 Bean Property 的设定, 这里默认的设置是: Not A Property。也就是说, 这不是 Bean 的属性, 而是一个普通的类的属性。

“Javadoc” 标签一般不需要设置, 因为这是一个通用属性。

7) 添加 IARControl 的方法。

IARControl 共有 6 个方法, 如图 3-34 所示。

其中, 方法 IARControl 不需要额外添加, Rose 会自动生成。其他的方法都需要进行添加, 这里以 doPost 方法为例进行描述。

doPost 方法的作用是, 将前端页面提交的请求转发给控制器进行处理, 并从控制器得到返回页面的信息, 将返回信息合成为返回页面的 URL 地址, 并分发到该页面。

第二步: 添加方法

添加 IARControl 下方法的具体操作步骤如下:

- 1) 首先在 IARControl 下 “New” 一个 “Operation”, 命名为 “doPost”。
- 2) 在图 3-35 中设定 doPost 方法的各种属性。

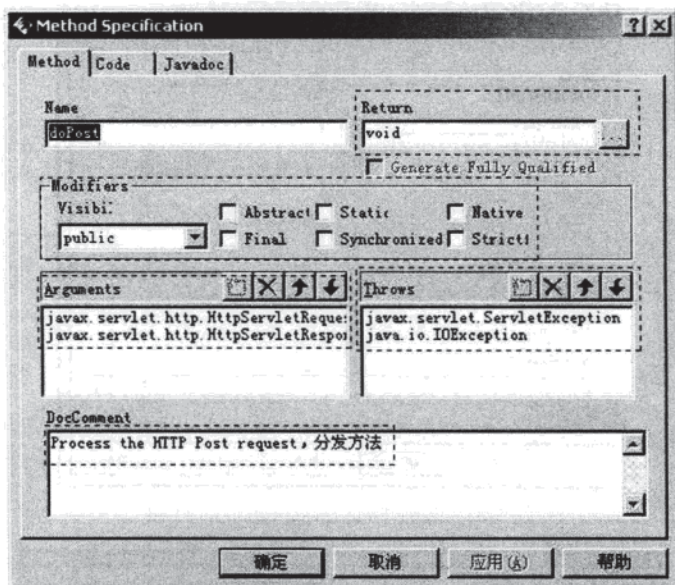


图 3-35 方法定义的 Method

这里要设置的包括方法的返回值 (Return)、修饰 (Modifiers)、参数 (Arguments)、需要抛出的异常 (Throws), 以及说明性文字 (DocComment)。

由于这些设置都是相对简单的 Rose 操作, 只要你熟悉 Java 语言基础知识即可完成, 所以, 这里就不再赘述。

3) 设定 doPost 方法的 Javadoc, 如图 3-36 所示。

4) 图 3-36 中 Code 标签的内容是不可编辑 (上面标示有 “Read Only”) 的。

Code 标签的内容只有当代码被反工到 Rose 模型中时, 该标签页面中才会出现实际内容。图 3-37 所示就是 doPost 方法在反工代码后的带有代码的图, 代码在没有进行反工时, 图中虚线框内是空白。

经过以上操作后, 就完成了该方法的设置。其他方法的设置过程是相似的, 这里就不再赘述。

5) 最后一个提醒点, 当一个类的实现逻辑非常复杂的时候, 就需要使用状态/活动图来对它进行更进一步的描述, 用来表达类中各个方法间的具体关系及与其他类之间的引用关系。

第三步: 状态/活动图

状态/活动图用于表示一些需要进行详细描述的流程。比如, 当一个类中某个方法的实现过程/逻辑比较复杂, 仅通过本小节中介绍的方法来进行设计开发并不能将它直接交付给编码人员实施, 此时就需要通过状态/活动图来进行进一步的说明。

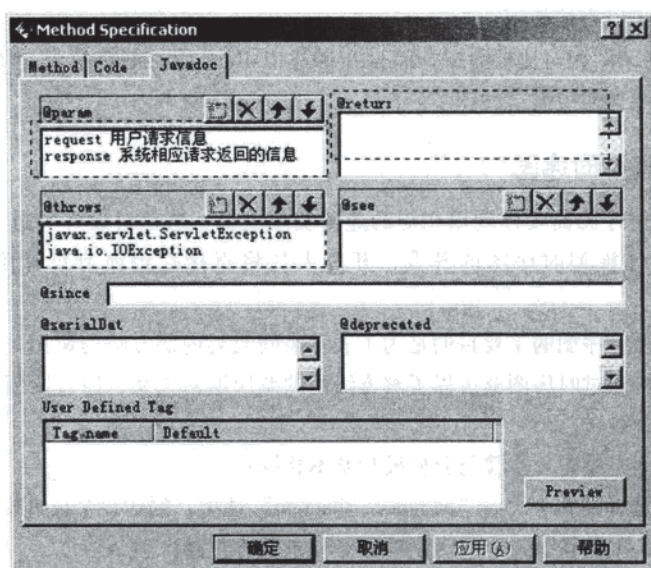


图 3-36 方法定义的 Javadoc

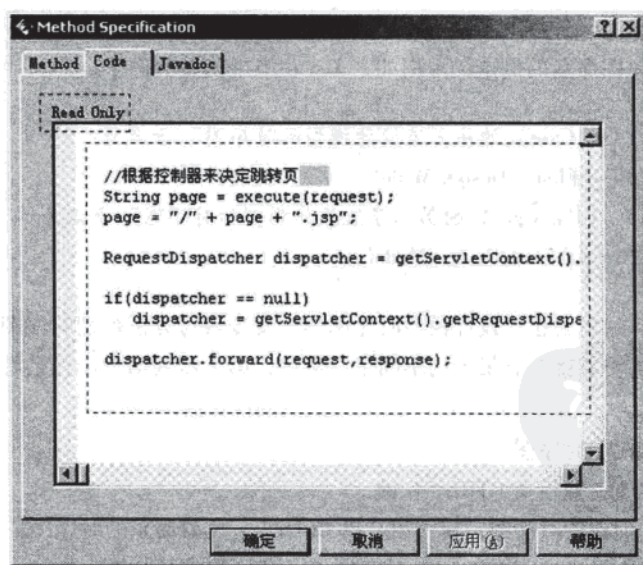


图 3-37 方法定义的 Code

笔者在使用状态/活动图时主要是为了描述类中方法的实现逻辑/过程，其实也可以用它来描述一些业务流程或者其他很多的过程性事务。

状态/活动图应该源于流程图，虽然，它增加了一些扩展，但是差别并不大，所以，几

乎不需要学习就可以直接使用。

因为状态/活动图相对比较简单,而且在讲述 UML 方面的书籍中也介绍的十分详细了,这里笔者就不介绍如何绘制这种图了。

3.4.3 设计模型时序图

设计模型的时序图需要体现出相应的业务实现逻辑,这是对设计类进一步深入的设计步骤。通过对设计模型时序图的开发,开发人员将直接获得所有设计类的方法和属性关系。

开发设计模型时序图的主要目的是为了表示类间或类内部方法的调用关系,使得程序员在进行编码时直接通过时序图就可以了解方法间的调用返回关系,通过调用过程来实现业务逻辑的实现层物理跳转。

建立设计模型时序图的步骤与分析模型基本相同:

1) 在 iar 下选择 “New” → “Sequence Diagram” 选项,创建一个设计模型时序图。

可以参见 3.3.3 小节中关于创建分析模型时序图的内容。

2) 然后为时序图命名。

笔者习惯的命名规则^①是:模块/包名称 - 流名称 - 模型分类(这里就是 Design Model)。

设计模型时序图命名的例子仍采用如 3.3.3 小节中模型的例子。

“登录系统 (Use Case) 的基本流设计模型的时序图”命名为:

LoginSystem-Base Flow - Design Model

而“登录系统 (Use Case) 的第三子流设计模型的时序图”命名为:

LoginSystem-3rd SubFlow-Design Model

可以看出:到这里为止,设计模型时序图的开发步骤与分析模型时序图的开发步骤很相似。不过,后面的步骤就会有很大的不同了。不过,第 3 步到第 5 步是笔者平时用到的偷懒步骤,可以跳过。

3) 打开分析模型时序图:

按住 Ctrl + A 键,然后按 CTRL + C 键。

4) 打开刚才创建的时序图,使用组合键 Ctrl + V,覆盖原来的时序图。

5) 下面的操作只用文字可能就有麻烦了。设计类已经基本上成型,这里缺的主要是类中的方法,因为类中的方法要通过时序图中的调用关系来初步产生,在实际的开发中还可

① 关于命名规则,只要一个团队内都使用统一的命名规则即可,不要每人用一种或者互相不一样。团队在一定程度上要保持一致性。这样的一致性对于客户来说是一种标准化的严格和体制上的统一,可以让他们感受到放心和安心。

能要提炼出一些公共的方法，甚至可能提炼出一些公共类，这也就是编码对设计的补充和对设计的合理化的修正。

在这一步操作中，把对应与分析模型时序图（其实，现在应该是粘贴到设计模型时序图中的分析模型时序图）中的分析类分别用设计类来做替换，同时把设计类也补充到时序图中。

替换过程很简单，用鼠标左键选中需要的设计类，然后将它拖动到已经打开的设计模型时序图中对应的分析类的图示上面，放开鼠标左键。此时，系统会给出一个提示，如图 3-38 和图 3-39 所示，同时这个图中类的名字就从分析类变成了设计类。



图 3-38 覆盖前的时序图

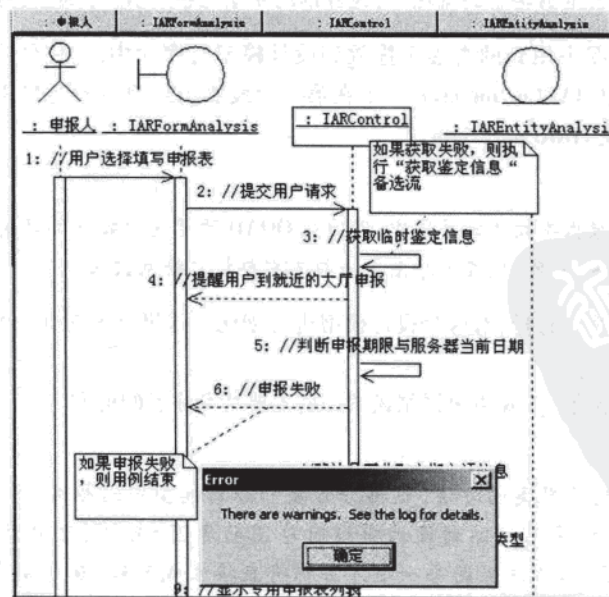


图 3-39 覆盖结束时的时序图和提示

由于已被标注的所有方法都无法从这个设计类中获取,系统将不得不把它们设置为自由文本,同时发出告警信息。

完成上述操作后的系统告警信息如图 3-40 所示。

```
09:52:21|
09:52:21| [ Change Object Class Assignment]
09:52:21| Warning: Message //将用户的输入提交给后台 received from object Untitled by
object Untitled does not correspond to an operation of class IARControl.
09:52:21| The message has been set to free-form text.
.....
(为了节省空间,这里省略了不需要的信息)
.....
09:52:21| Warning: Message //返回扣款状况 received from object Untitled by object Untitled
does not correspond to an operation of class IARControl.
09:52:21| The message has been set to free-form text.
```

图 3-40 系统警告信息

注意:

开发人员不要认为自己的设计是完美无缺的,或者说是绝对可以直接使用的。要知道再好的设计如果没有经过编码,它都是空谈。毕竟在开发过程中可能会遇到很多意想不到的问题。编码对设计的合理化修正是可能存在的,而且是非常必要的。尤其在当前国内现有设计人员进行设计的经验不够丰富的情况下,这种情况发生几乎是 100% 的。详情可参看第 4 章。

6) 将要在时序图中用到的类逐个拖放到设计模型时序图中,它们依次为: Actor、边界类 FormAnalysis、控制类 IARControl. java、一个在第一阶段要用到的助手类 IARApplyHelper. java、实体类(数据管理类) IAREntity. java。

注意:

一般不需要把数据类放在时序图中,因为在 OOAD 的开发中数据类只是一个对象,并没有直接给出被调用的方法,所以,在时序图中一般不需要把它绘制出来。

7) 按照分析模型的过程,对这个设计模型进行细化。这里就不对如何画线和标注的问题进行介绍了。

3.4.4 小节中会给一个示例来详细说明一些需要特别注意的地方。

注意:

除了从 Actor 到边界类的方法(这部分方法和分析模型中的相同,所以,这也是为什么笔者要把分析模型时序图粘贴到设计模型时序图的原因)外,这里创建的每一个方法都不再需要加“//”了,因为这里的每一个方法都将直接生成代码,在代码中表现出来。

关于类的方法设计,请参看 3.4.2 小节设计类的构建。

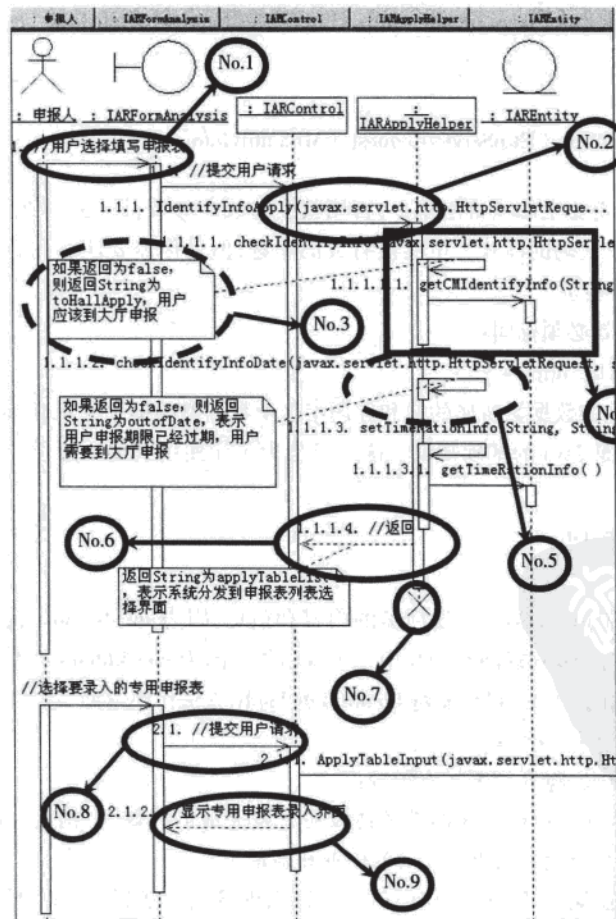
8)完成上述步骤后,就完成了设计模型时序图的创建,这样循环多次,经过评审后,完成整个设计模型的开发。

关于方法和属性的添加和修改设置的内容因为属于 UML 工具使用的问题,就不在本书中进行介绍了。

3.4.4 一个示例

图表 3-41 是某工程项目网上申报部分的设计模型时序图片段,该示例对应本书附带光盘中 ApplyReport.mdl 文件。通过这个片段,本书主要为了说明在绘制设计模型时序图中所需要注意的一些问题和一些需要共同遵守的约定,只有这样才能使你开发的时序图让其他人也能够看明白。

下面是对图 3-41 的说明,同时也给出一些可供参考的方式和操作手法。



1. No. 1 的说明

这里是用户在界面上的操作,也就是在浏览器中的操作。因为前面已经说过“开发人员不对浏览器中的 JSP 页面进行更详细的设计”,这部分放在界面设计中来完成。所以,这里的操作/方法名称仍然使用加“//”的注释性标注方式。

2. No. 2 的说明

这里就是一个完整的方法调用了,在上图中因为方法全称太长,所以除了这种将参数类型全称(带包名)写在括号内的做法。

还有人认为:可以把包名去掉,只放置参数的类型名就可以了。

例如:

这个方法的全名称是:

```
checkIdentifyInfoDate( javax. servlet. http. HttpServletRequest, scInfoTech. iar
. CMIdentifyInfo[ ] )
```

可以简写成:

```
checkIdentifyInfoDate( HttpServletRequest, CMIdentifyInfo[ ] )
```

不过,这一点一定要全部项目组统一,否则就会造成一些混乱。而且在导出代码的时候,这一点会影响到导出代码的样式。虽然没有大的问题,但是你需要注意在简写以后,需要 import 的包就会更多一些了。

如果使用后者,就必须标明:

```
import javax. servlet. http. * ;
```

下面这个是当你的数据类所属的包和你调用这个数据类的助手类不在同一个包中的时候才需要标注的,这都是 Java 的基础知识。这里只是为了让使用其他语言的读者能够注意这个区别:

```
import scInfoTech. iar. * ;
```

3. No. 3 的说明

这里的标签所描述的是第一个返回值的内容和信息,因为 IdentifyInfoApply 的返回值有多个可能,这里就给出了 No. 6 返回时的其中一个返回值。因为 checkIdentifyInfo 方法是被 IdentifyInfoApply 方法调用的,所以,它只能对 IdentifyInfoApply 方法作出返回,这里不能直接使用“return message”来返回到控制类。

通过返回值的定义,将会产生如下影响:

开发控制类的程序员就可以在助手类还没有开发完成的时候,根据已经统一定下来的返回值的信来定义/组合返回的 JSP 页面的位置和地址了。

与此同时,开发前端 JSP 页面的程序员也可以根据既定的逻辑获取到所有的需要开发的 JSP 页面的数量和内容,通过这个时序图就可以开始他的开发工作了。

4. No. 4 的说明

这里显示了助手类与实体类打交道的过程。

通过这个视图可以很清楚地了解到助手类 IARApplyHelper 的方法 checkIdentifyInfo 调用了实体类 IAREntity 的方法 getCMIdentifyInfo。

再看一下方法 getCMIdentifyInfo(如图表 3-42),就可以知道:这个方法需要调用方提供一个参数——iPayer_id, 申报人申报登记证号; 返回一个对象——CMIdentifyInfo, 申报鉴定信息对象。这个对象在 scInfoTech. iar 这个包中有定义。



图 3-42 IAREntity 的 getCMIdentifyInfo 方法

5. No. 5 的说明

这里表示的是一个独立的校验方法, 只是为了表示: 这里属于 IARApplyHelper 的方法可以是独立的方法。

另外, 对于一些提取出来的公用方法也可以采用这种方式表示, 不需要什么特殊标注。

6. No. 6 的说明

这里是两个类之间有返回时的标识方式。由于这种返回只是方法间值的传递，所以，这里需要加上“//”标识。

7. No. 7 的说明

这个符号和分析类中该符号的用法是相同的，表示这个类所参与的流程到此结束了。

8. No. 8 的说明

这个和 No. 1 是完全相同的，因为这个提交采用的是超文本标识语言中的表单提交，系统会自动调用 Servlet 中的 doPost 方法，所以开发人员就可以直接这样使用注释的方式来书写了，这是一个大家都知道的公共的开发人员不能作修改的方法。

9. No. 9 的说明

这个返回和 No. 6 中的返回是相同的，不过，这里的返回是对公用方法的返回，设计人员可以直接通过这种标识告诉编码人员下面应该出现的操作界面是什么。

3.4.5 参与人员建议

1. 主要开发人员

设计实现人员。

2. 主要审核人员

项目经理、行业专家、系统分析员（需求调研人员、Use Case 阐述人员）、其他模块的设计实现人员、编码实施人员、分析模型开发人员。

3.4.6 输入输出

1. 输入工件

分析模型、交互建模工件集。

2. 输出工件。

设计模型。

3.4.7 UML 元素

1. 相关 UML 元素

状态/活动图、时序图、类图。

2. 用处

通过分析模型时序图、类图及交互建模工件内容进行设计模型开发，完成设计模型时序图和设计类的开发，对于较为复杂的类通过状态/活动图描述类内部细节的实现。

3.4.8 楚凡 UML 工具模型示例

设计模型沿用分析模型中的幼儿教育系统示例进行展示，如图 3-43 所示。

时序图实例如图 3-44 和图 3-45 所示。

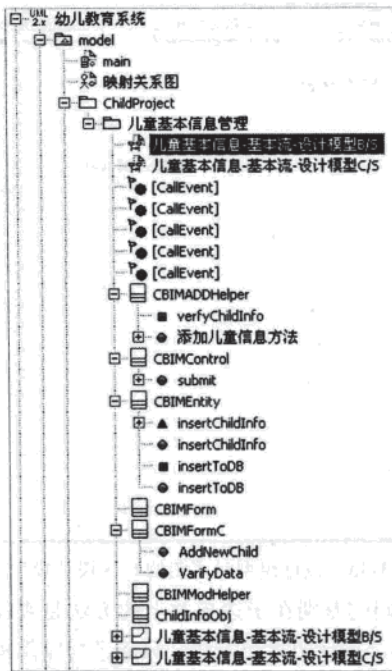


图 3-43 设计模型树形图

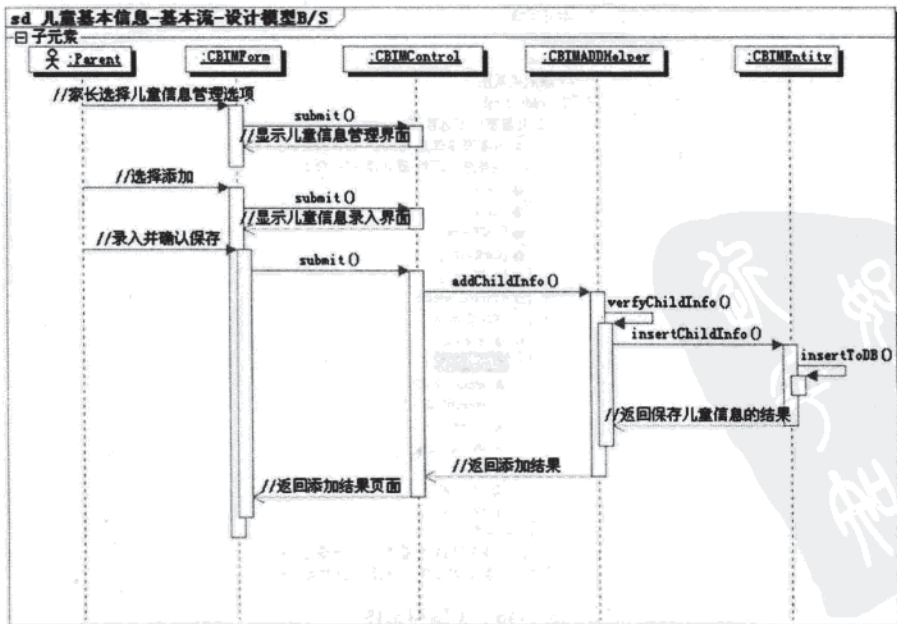


图 3-44 设计模型时序图的 B/S 设计模型

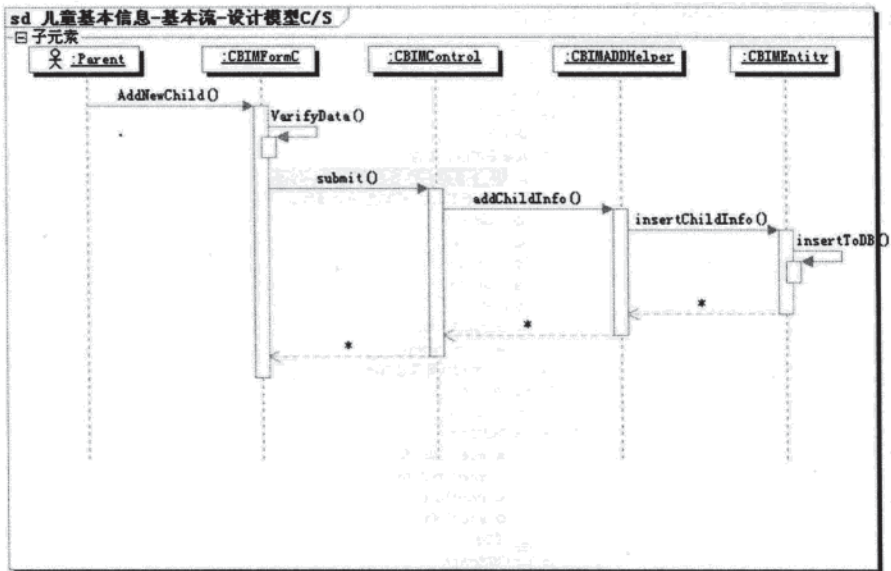


图 3-45 设计模型时序图的 C/S 设计模型

注意图 3-44 和图 3-45 之间的差别在于边界类上的方法是否需要设计, C/S 系统需要设计边界类方法, 而 B/S 系统不能设计边界类方法, 只能进行操作建议。

类图实例如图 3-46 ~ 图 3-53 所示。

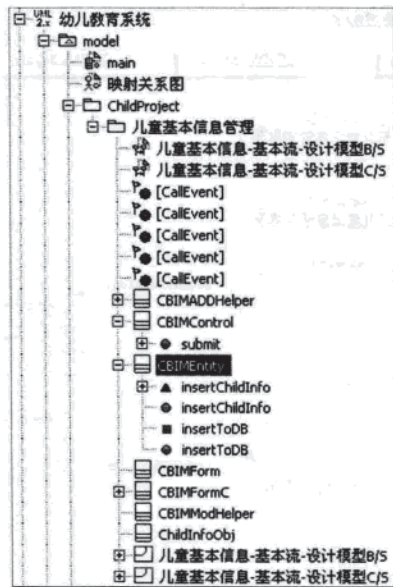


图 3-46 类图树形图

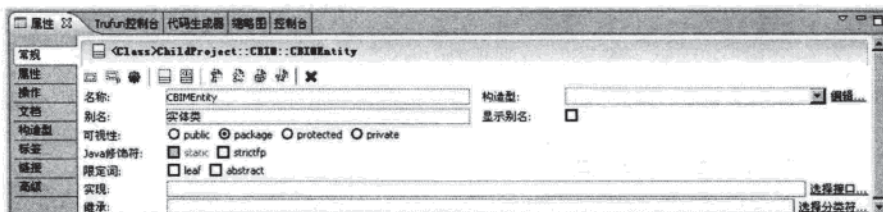


图 3-47 类的基本信息

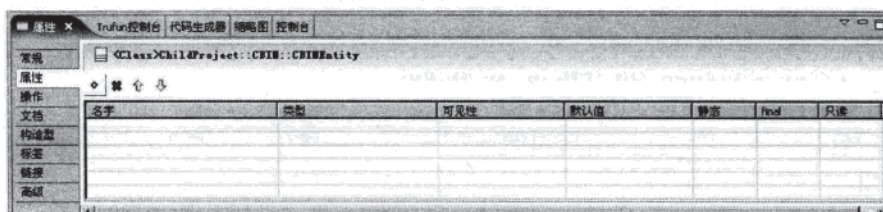


图 3-48 类的属性信息

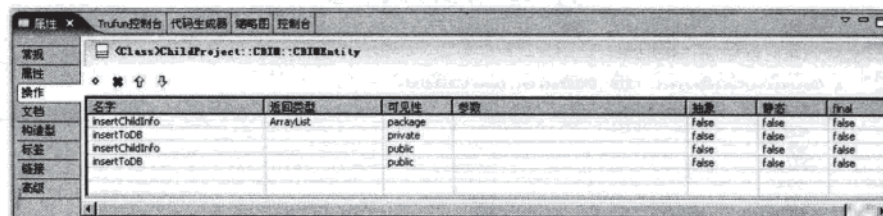


图 3-49 类的方法信息

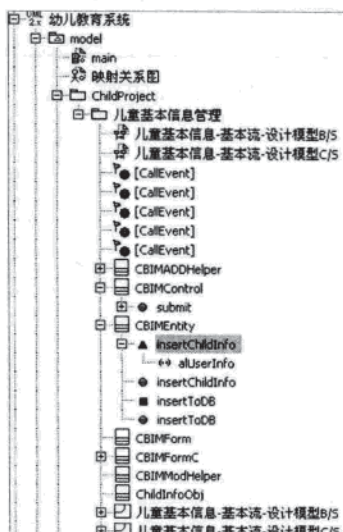


图 3-50 方法的树形图

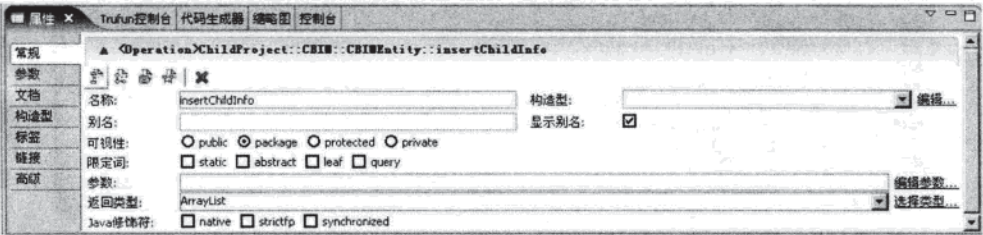


图 3-51 方法的基本信息

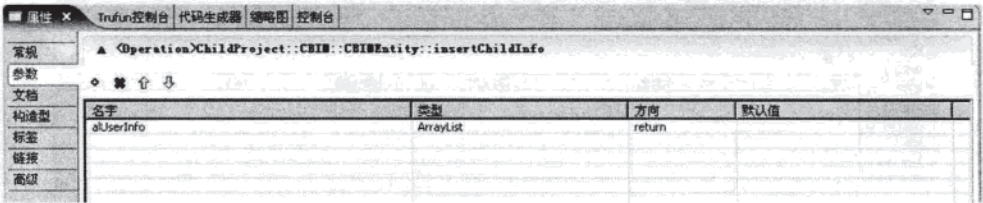


图 3-52 方法的参数信息

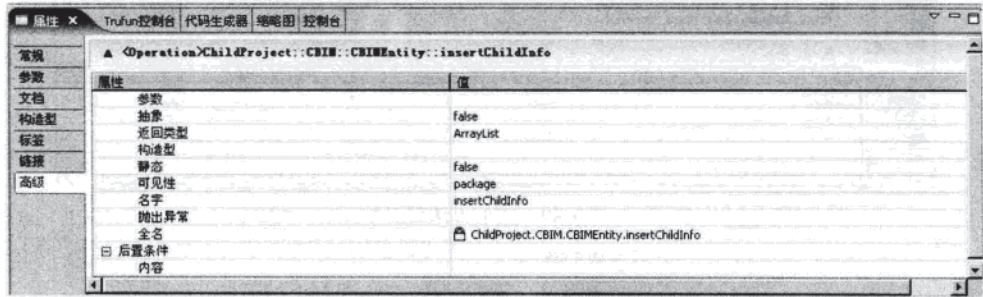


图 3-53 方法的高级信息



第 4 章

代码模型一致性

通过上一章的介绍，可以基本上完成设计模型的开发。但是，在实际工作过程中往往因为多种原因，使得所完成的设计模型存在一些问题和缺陷。

这些问题的解决，一方面项目组可以通过同行评审、测试等手段来尽可能地修正错误；另一方面通过规范和指导性培训降低缺陷发生的概率；最后，项目组还可以通过代码的导出过程对这些设计结果进行类形态结构上的检验。

在工程实现过程中，除了对类形态结构上可能存在的问题通过代码导出过程的检验来确保解决问题外，还需要考虑设计模型持续有效性的保障问题。为了达到这个目的，开发人员需要对设计模型进行反工。以上这些内容都是本章所要描述的重点内容。

本章内容包括：

- 生成框架代码；
- 维护设计模型；
- 反工。



4.1 概述

本章主要描述如何从设计模型导出代码，以及代码导出后如何在实际开发中保持设计模型的实用性。也就是说，如何保持设计模型与代码的一致性。本章的内容在软件工程过程中对应编码实施过程和变更过程。^①

编码实施过程代表传统的人们能直接了解到的程序员的工作范畴：编写代码形成软件。

但是，对于软件还有变更这个关键过程却是被很多人忽略或者所不了解的。

在很多实际工程中，建模工作在设计模型开发完成时就结束了，很少有人或企业能做到后面这一步。这一方面是因为设计模型可能开发得不够完善，无法导出代码；另一方面也是因为很多人并不知道如何保持代码与模型的一致性，也就是说，很多人不知道如何操作才能保持模型的有效性。

在实际工程中，往往是为了赶工期而省略了详细设计（设计模型）。项目还没有结束，就会发现，从需求到分析到设计的所有文档，全部作废了。编码阶段的开始意味着以前所有文档都变成了一堆废纸，这使得后期的维护工作不能正常地得到实施，直接导致工程软件无法提供持续不间断的升级服务的关键所在。

下面先从框架代码的导出过程开始本章的叙述。

4.2 生成框架代码

如何从模型生成框架代码，这需要经过3个关键性步骤：类的语法检查、ClassPath的设置、导出代码。

4.2.1 类的语法检查

在导出代码之前，开发人员要做的第一件事就是进行语法检查。操作步骤如下：

- 1) 可以如图4-1所示那样一次选择多个类，也可以一次只选择一个类来进行语法检查。
- 2) 检查过程中开发人员不需要操作，系统自动显示检查进度，如图4-2所示。
- 3) 如果检查成功，系统弹出检查成功对话框，单击“确定”按钮完成语法检查，如图4-3和图4-4所示。
- 4) 如果检查过程中发现了错误，系统会在类似图4-4的Log框中给出详细的提示，如图4-8所示。

① 系统开发过程中即将发生的任何与计划相不符的事件都称为变更。变更操作的流程就是变更过程。变更包括需求变更、设计变更，以及所有原因引起的需求变动造成的变更。

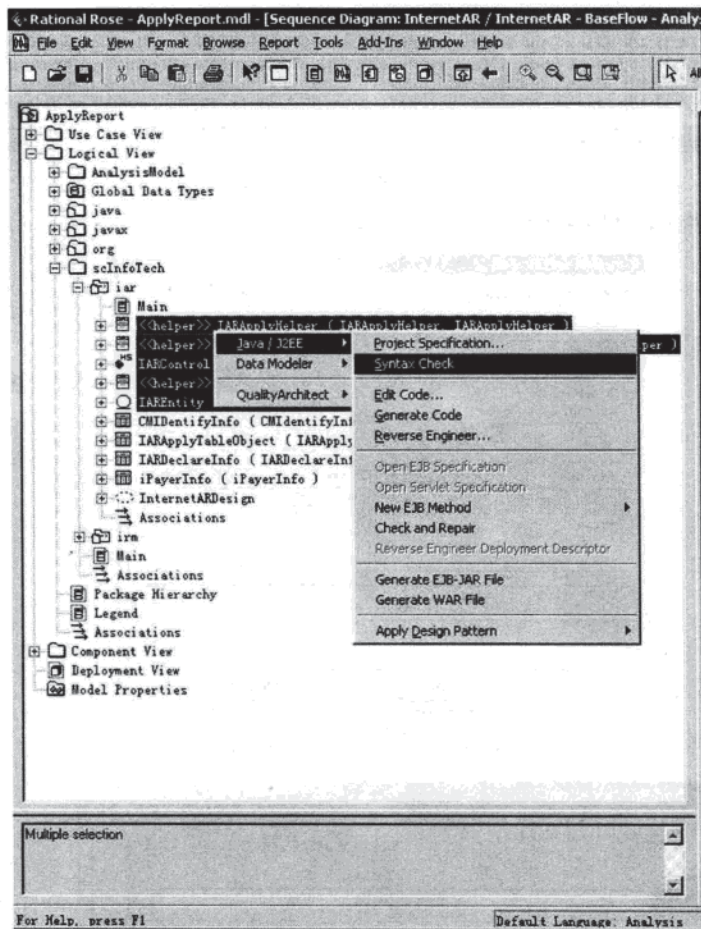


图 4-1 选择语法检查

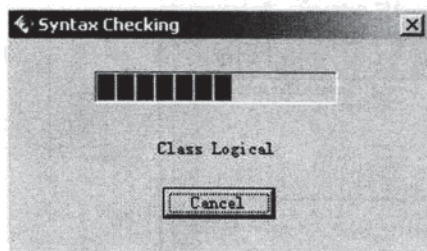


图 4-2 语法检查进度指示

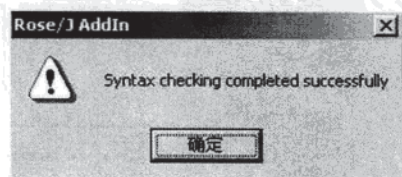


图 4-3 语法检查成功的系统提示

```

14:37:19] Class Logical View::scInfoTech::iar::IARControl - Starting syntax checking
14:37:19] Class Logical View::scInfoTech::iar::IARDeclareHelper - Starting syntax ch
14:37:19] Class Logical View::scInfoTech::iar::IARDeclareInfo - Starting syntax chec
14:37:19] Class Logical View::scInfoTech::iar::IAREntity - Starting syntax checking
14:37:19] Class Logical View::scInfoTech::iar::iPayerInfo - Starting syntax checking

```

图 4-4 语法检查成功的系统 Log 提示

例如：笔者进行从图 4-5 到图 4-6 的操作后。

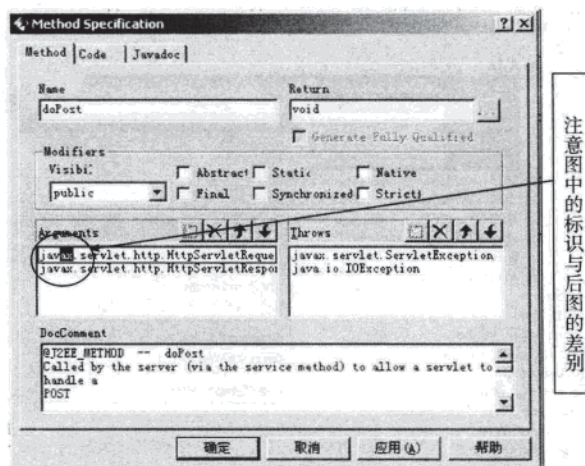


图 4-5 修改前的 doPost 方法

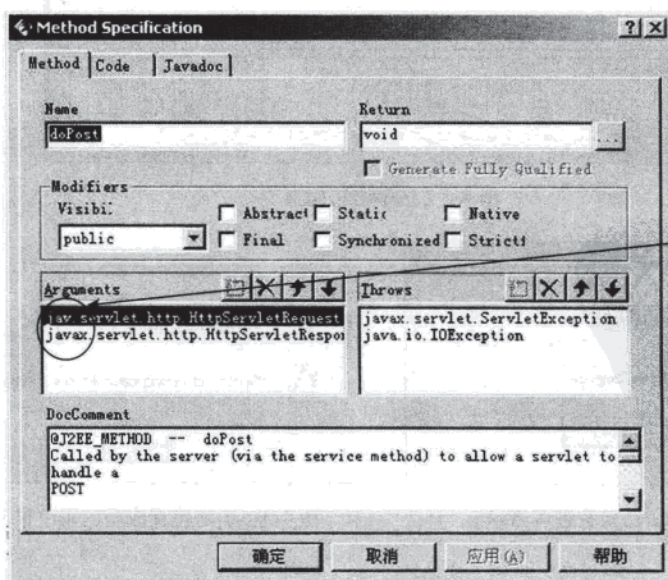


图 4-6 修改后的 doPost 方法

这时候再进行语法检查，系统提示如图 4-7 和图 4-8 所示。

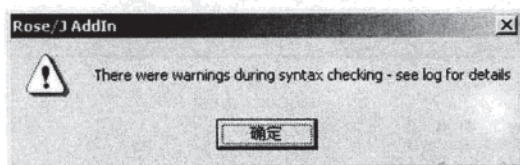


图 4-7 语法检查失败时的系统提示

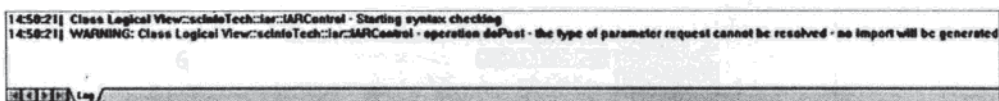


图 4-8 语法检查失败时的系统 Log 提示

因为图 4-8 中的系统提示太长，这里把它复制在图 4-9 中，以便于阅读。

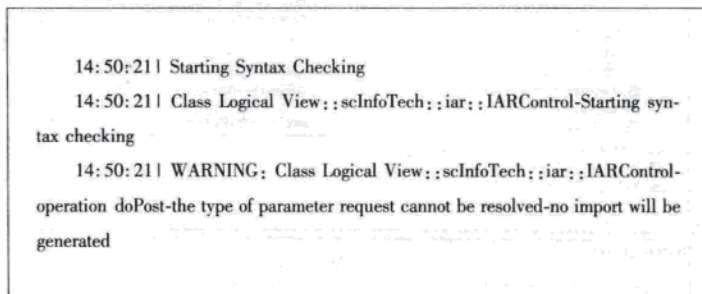


图 4-9 语法检查失败时的系统 Log 提示完整内容

从中可以看出语法检查失败的原因。

4.2.2 ClassPath 的设置

语法检查通过后，开发人员就可以进行下一步的工作：导出代码。

在第一次导出代码的对话框中，系统会要求对 ClassPath 进行设置。这时就应该考虑你的编辑器所建立的工程所在的路径，然后把这个路径添加到里面。选择这个路径进行导出就可以实现模型开发工具与编译器两者的直接结合。

下面是 ClassPath 的详细设置过程。

- 1) 在需要进行语法检查的类上单击鼠标右键，在弹出的菜单中选择“Java/J2EE”→“Generate Code”选项，如图 4-10 所示。
- 2) 如果是第一次生成代码，系统会弹出一个对话框，如图 4-11 所示。

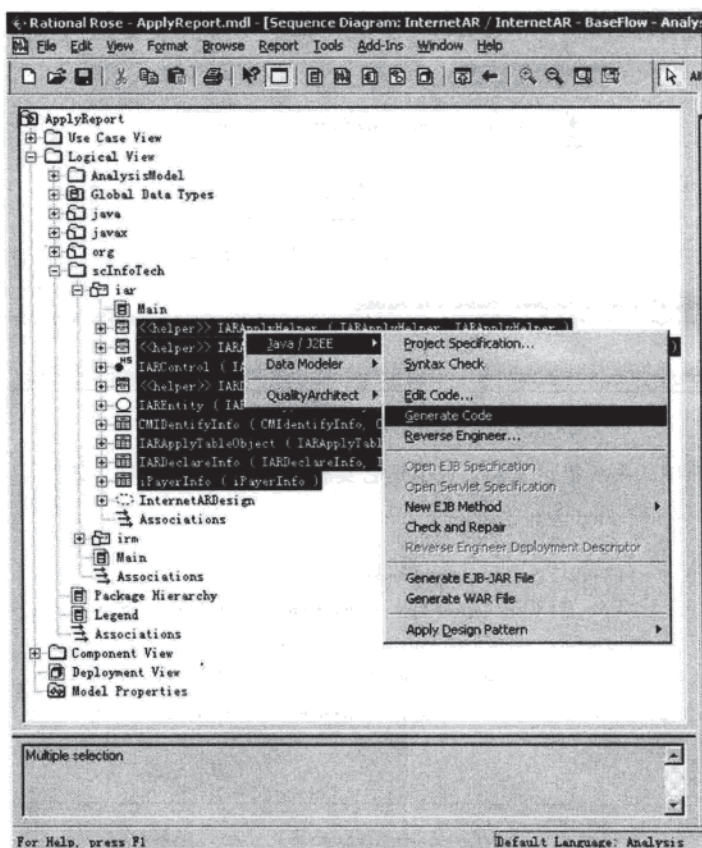


图 4-10 生成代码的选项

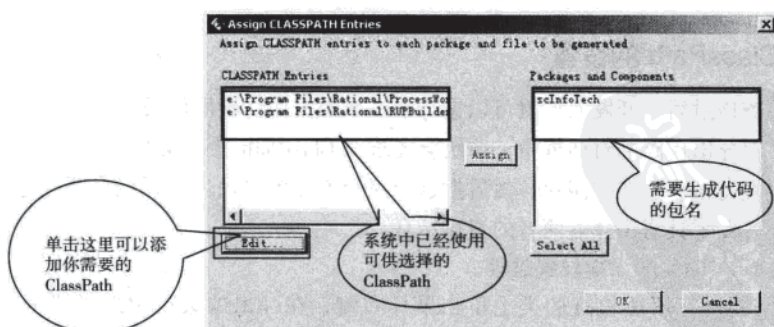


图 4-11 设置 ClassPath 路径

注意：

这里的 ClassPath 与 Java 中 ClassPath 的概念有些相似，但是 Rose 在这里所显示的并不是

操作系统中所设置的 ClassPath 值，这一点不要搞混淆了。

3) 不需要选择任何 ClassPath 条目，直接单击“Edit...”按钮后，Rose 系统弹出如图 4-12 所示对话框。对话框中显示的是操作系统所设定的 ClassPath 路径。

No 1 的说明：

单击这个按钮就可以添加希望生成代码的路径，然后单击“确定”按钮完成添加操作。

4) 这里笔者添加了如图 4-13 所示的路径。

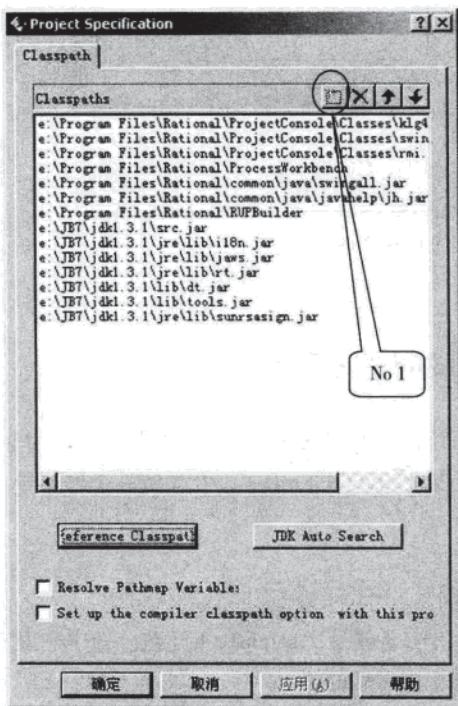


图 4-12 Project Specification 对话框

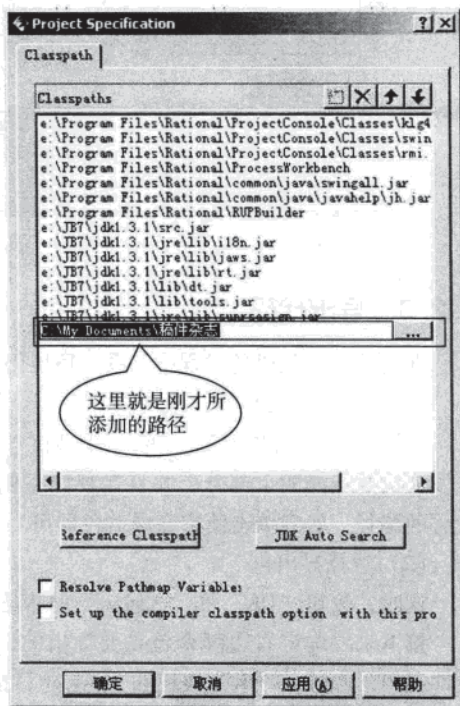


图 4-13 添加 ClassPath

单击“应用 (A)”按钮后可以继续添加其他路径，或者直接单击“确定”按钮，会保存添加或修改操作。

5) 此时，在如图 4-11 所示的对话框中就会出现“C:\My Documents\稿件杂志”这样的路径，如图 4-14 所示。

(6) 到这里开发人员就完成了 ClassPath 的设置，接下来就可以准备导出代码框架了。

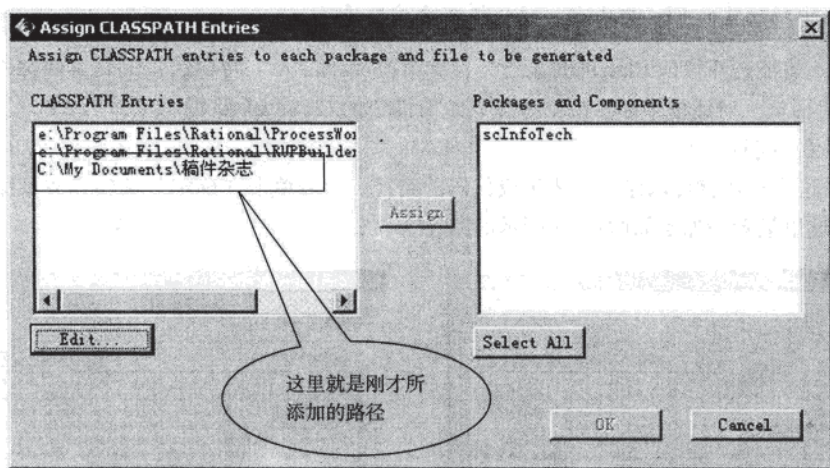


图 4-14 添加完成的 ClassPath

4.2.3 导出代码

在描述导出代码的过程中，系统环境仍然处于如图 4-14 所示的状态。

(1) 设置导出路径

从图 4-14 中可以看到“OK”按钮和“Assign”按钮都呈灰色（不可用状态）。这时候，开发人员要选择两个部分，并且在视图左侧的“CLASSPATH Entries”中选择一个你要导出代码的路径，然后单击使它变成蓝色即可。

(2) 选择导出包

这时，按钮“OK”和“Assign”仍然是灰色的。

接下来要确定右边哪个包是要导出的，这里很明显就是“scInfoTech”包。选中它及其组件视图，此时，“OK”按钮和“Assign”按钮都变成可用的了。

(3) 导出代码

单击图 4-14 中的“OK”按钮，出现如图 4-15 所示“Code Generation”对话框，代码生成过程开始。

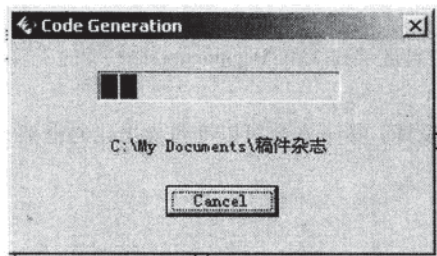


图 4-15 代码生成过程

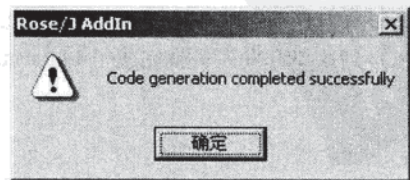


图 4-16 代码生成成功

单击图 4-16 中的“确定”按钮，成功生成框架代码。

代码生成了并不是最终的结果，开发人员还需要检查所生成的代码中是否存在错误。

(4) 检查生成结果

首先检查一下刚才设置的路径，如图 4-17 所示。

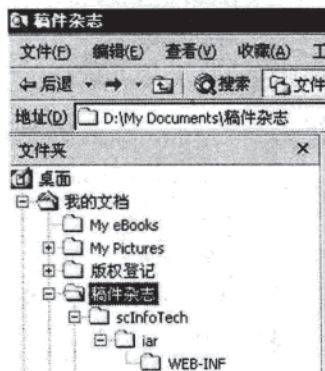


图 4-17 检查生成代码的路径

发现多了一个三级文件夹，刚才生成的代码就在目录 scInfoTech 的下一级目录 iar 下，如图 4-18 所示。



图 4-18 生成的类文件及目录

打开任一类文件可查看相关内容。这里我们选择打开控制类 IARControl.java 文件来查看生成的代码，如图 4-19 所示。

下面显示的是直接生成的 IARControl.java 的完整代码（用于与后面反工后的代码进行对比）。

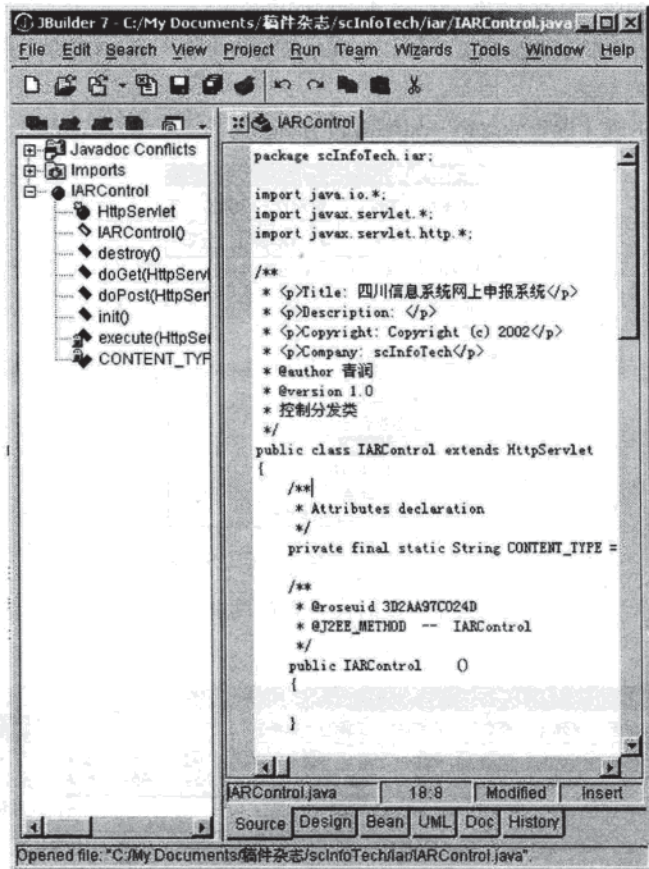


图 4-19 生成的代码

```

package scInfoTech.iar;

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

/**
 * <p>Title: 四川信息系统网上申报系统</p>
 * <p>Description: </p>
 * <p>Copyright: Copyright (c) 2002</p>
 * <p>Company: scInfoTech</p>
 * @author 青润

```

```
* @ version 1.0
* 控制分发类
* /
public class IARControl extends HttpServlet
{
    /**
     * Attributes declaration
     * /
    private final static String CONTENT_TYPE = "text/html; charset=GB2132";

    /**
     * @ roseuid 3D2AA97C024D
     * @ J2EE_METHOD--IARControl
     * /
    public IARControl ()
    {

    }

    /**
     * @ roseuid 3D2E9971022E
     * @ J2EE_METHOD--init
     * Called by the servlet container to indicate to a servlet that the servlet is being
     * placed into service.
     * 初始化所有的全局变量
     * @ throws javax.servlet.ServletException
     * /
    public void init () throws ServletException
    {

    }

    /**
     * @ roseuid 3D2E997301F5
     * @ J2EE_METHOD--doGet
     * Called by the server (via the service method) to allow a servlet to handle a GET request.
     * The servlet container must write the headers before committing the response, because
     * in HTTP the headers must be sent before the response body. The GET method should
     * be safe and idempotent. If the request is incorrectly formatted, doGet returns an
```

```

    * HTTP 'Bad Request' message.
    * Process the HTTP Get request
    * /
    public void doGet (HttpServletRequest request, HttpServletResponse response) throws ServletExceptionException, IOException
    {

    }

    /**
    * @ roseuid 3D2E9976036C
    * @ J2EE_METHOD--doPost
    * Called by the server (via the service method) to allow a servlet to handle a
    * POST
    * request. The HTTP POST method allows the client to send data of unlimited length
    * to the Web server a single time and is useful when posting information such as
    * credit
    * card numbers. If the HTTP POST request is incorrectly formatted, doPost returns
    * an HTTP 'Bad Request' message.
    * Process the HTTP Post request,分发方法
    * @ param request 用户请求信息
    * @ param response 系统相应请求返回的信息
    * @ throws javax.servlet.ServletException
    * @ throws java.io.IOException
    * /
    public void doPost (HttpServletRequest request, HttpServletResponse response) throws ServletExceptionException, IOException
    {

    }

    /**
    * @ roseuid 3D2E997A00F0
    * @ J2EE_METHOD--execute
    * 控制器,控制流程实现
    * /
    private String execute (HttpServletRequest request)
    {
    return null;

```



```
}

/* *
 * @ roseuid 3D2E997A0122
 * @ J2EE_METHOD--destroy
 * Called by the servlet container to indicate to a servlet that the servlet is being
 * taken out of service.
 * Clean up resources
 * /
public void destroy ()
{

}
}
```

这时候，就可以分配任务让其他的开发人员参与一同开发完成这些代码了。

到这里，代码的导出工作就告一段落了。

不过，此时还有一件非常重要的工作，如果不进行这项工作，项目组前面所付出的所有努力都将在开始代码编写的同时付之东流。那就是，如何保持设计模型的有效性？即设计模型的有效性维护。

4.3 维护设计模型

4.3.1 目的

设计模型的维护是非常重要的，如果你认为完成设计之后，在以后的开发中就不再理会设计模型，那么你就会很快发现你会淡忘这个设计，并且会重新回到最原始的过程中。

当然，即便这样也比刚开始就采用最原始的过程要好一些，因为你通过前面的开发过程已经对业务的实现过程有了相当深入的了解，这对于后续的工作还是有用的。不过，你是否会觉得可惜呢？辛辛苦苦的工作成果放弃了，将来维护系统的基础也丢失了，留下的仅仅是个人头脑中存在的一些对该软件系统的记忆，通过人的记忆来维护软件系统，可靠性够不够高？当然，如果你是为了个人职位的保护而故意这样做的，那也就用不着看本书后续的“废话”了。

下面的考虑并不是简简单单为了避免发生这样的问题，同时也是为了你的项目今后的考虑。当然，如果说你的项目是个非常小的项目，而且今后也不打算继续维护升级，那么你也不需要继续看后面的内容，到这里就足够了。

4.3.2 维护方式

其实每完成一个项目,开发人员都会积累一定的经验,会发现开发过程中的每一个步骤都会有一些与以往项目不同的地方,这些不同之处也是最值得开发人员关注的地方。

本小节要讲述的是维护方式所需要的一种基本认识或者说基本意识。

在笔者曾经参加过的几个大型项目中,或者说目前国内大多数较大型项目中,一个程序员本身对所开发系统的重要性远远超过其项目经理的重要性。

可以说,如果是项目经理辞职,那么只需要换一名项目经理即可。而如果是一名程序员要辞职,将直接面临的是整个项目的瘫痪,因为该程序员所负责的内容将没有任何其他人可以在几个月内就能接手。对于这类问题,不是纯技术手段就能解决的,这种问题的解决方式可参考本书第6章6.1节和6.2节的内容。

1. 维护重在持久

在导出代码后,仍然要不断地对模型进行相应修改。

在编写代码的过程中很多时候需要提取一些设计人员没有考虑到的可以提取的公共方法和属性,或者可以调用一些以前开发出来的可重用组件来实现设计的某个功能部分。这时,都会发生实现过程对设计模型的深入认识要求设计模型必须进行适当的修改和调整。

同时,在代码编写过程中,随着对业务需求的深入理解也会有更多的属性、参数和方法需要不断地修改,开发人员也可能会因为某些需求变动而对分析/设计模型做出相应修改,此时会影响代码的框架。此时,如果代码框架已经被大量修改,而无法与设计模型保持同步的话,需求的变动就只能直接对代码产生影响,而无法保持设计的有效性。这样的问题会造成在项目后期开发过程的混乱,出现问题后很难找到责任者,因为不知道是设计者还是实施者应该负责。

由此可见,持久保持设计的有效性是多么的重要。

2. 维护原则

维护原则包括以下两个部分:

- 1) 在需求发生变动时,尽可能从 Use Case 图开始修改,否则(时间不允许),略过分析模型,只修改 Use Case 图和设计模型。
- 2) 随时保持设计模型与代码的一致性,通常通过反工和导出操作来实现。

4.3.3 维护步骤

这里的维护步骤是指在实际遇到各种变更时的操作过程,本书强烈建议采用最完整的方式进行维护。

维护的过程分为以下两种。

1. 需求变动维护步骤

- 1) 需求人员跟踪需求,并记录需求变更情况。

2) 需求人员修改 Use Case 图和分析模型, 并将需求变更的情况和相关信息及时反馈到项目组。

3) 设计人员将已编成的代码反工到设计模型中。

4) 设计人员根据需求变更的情况修改设计模型, 将设计模型重新导出生成变更后的代码。

5) 代码检查。

6) 代码交付给编码实施人员继续进行实施。

2. 编码变动维护步骤

1) 编码人员将需要修改的类的意见提交给设计人员。

2) 设计人员将已编成的代码反工到设计模型中。

3) 设计人员根据变更的情况修改设计模型, 将设计模型重新导出生成变更后的代码。

4) 代码检查。

5) 代码交付给编码实施人员继续进行实施。

上述维护步骤是在各方对不同意见或者变更已达到共识的情况下进行的, 而不是任何一个角色认为需要改变就改变的。

因为每一次变动都是对过去开发模型的否定, 都是对过去开发成果的推倒重来, 都会存在大量的资源浪费和无形损失, 因此每一次变更都要经过缜密地推敲以确定是否需要变更。

关于对不同意见如何达成共识, 以及达成共识的过程采用的手段和方式, 读者可以参考关于变更控制、质量保证、同行评审方面的内容, 这些内容可以在网上找到, 也可以在本书第6章、第7章、第8章中找到相关内容和一些开发中的管理技巧。

4.4 反工

反工 (反向工程, Reverse Engineer), 简而言之, 就是从代码反工回模型的过程。

在一般情况下, 代码反工只能得到类图和分包图, 而不能得到相应的时序图、状态/活动图等这些表述流程过程的图。

4.4.1 作用

前面已描述了如何从设计模型导出代码, 也就是说当设计模型再次被修改时, 代码也需要再次被导出。

从4.3节的描述中可以看出: 除了第一次从模型导出代码外, 在代码编制过程中如果对设计产生的代码框架需要进行修改时, 都应编写新的代码重新导回到设计模型中, 以便保持代码与设计的同步, 这样, 在下次导出的代码中将包含现阶段已经完成的代码编写内容。这个过程就称为代码的反工。

特别声明:

在代码编写过程中,一般情况下不建议编码实施人员修改代码框架,如果确实需要修改,一定要和设计人员达成共识按 4.3.3 小节中的编码变动维护步骤实施,否则,会造成设计模型与代码间的混乱,也就很难保证设计的有效性了。

4.4.2 操作步骤

(1) 编写代码

为了便于说明,这里通过控制类 IARControl.java 的两个主要方法 doPost () 和 doGet () 来介绍详细的操作步骤。这两个方法的 IARControl.java 的源代码如下:

```
package scInfoTech.iar;

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

/**
 * <p>Title: 四川信息系统网上申报系统 </p>
 * <p>Description: </p>
 * <p>Copyright: Copyright (c) 2002 </p>
 * <p>Company: scInfoTech </p>
 * @ author: 青润
 * @ version: 1.0
 * 控制分发类
 */
public class IARControl extends HttpServlet
{
    /**
     * Attributes declaration
     */
    private final static String CONTENT_TYPE = "text/html; charset = GB2132";

    /**
     * @ roseuid 3D2AA97C024D
     * @ J2EE_METHOD -- IARControl
     */
    public IARControl ()
    {

```



```
}

/* *
 * @ roseuid 3D2E9971022E
 * @ J2EE_METHOD--init
 * Called by the servlet container to indicate to a servlet that the servlet is being
 * placed into service.
 * 初始化所有的全局变量
 * @ throws javax.servlet.ServletException
 * /
public void init () throws ServletException
{

}

/* *
 * @ roseuid 3D2E997301F5
 * @ J2EE_METHOD--doGet
 * Called by the server (via the service method) to allow a servlet to handle a GET request.
 * The servlet container must write the headers before committing the response, because
 * in HTTP the headers must be sent before the response body. The GET method should
 * be safe and idempotent. If the request is incorrectly formatted, doGet returns an
 * HTTP 'Bad Request' message.
 * Process the HTTP Get request
 * /
public void doGet (HttpServletRequest request, HttpServletResponse response) throws Service-
tException, IOException
{
    doPost(request,response);
}

/* *
 * @ roseuid 3D2E9976036C
 * @ J2EE_METHOD--doPost
 * Called by the server (via the service method) to allow a servlet to handle a
 * POST
 * request. The HTTP POST method allows the client to send data of unlimited length
 * to the Web server a single time and is useful when posting information such as
 * credit
 * card numbers. If the HTTP POST request is incorrectly formatted, doPost returns
```



```

* an HTTP 'Bad Request' message.
* Process the HTTP Post request,分发方法
* @ param request 用户请求信息
* @ param response 系统响应请求返回的信息
* @ throws javax.servlet.ServletException
* @ throws java.io.IOException
* /
public void doPost (HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException
{
    //根据控制器来决定跳转页
    String page = execute(request);
    page = "/" + page + ".jsp";

    RequestDispatcher dispatcher = getServletContext().getRequestDispatcher(page);
    if(dispatcher == null)
        dispatcher = getServletContext().getRequestDispatcher("/NoPage.jsp");

    dispatcher.forward(request,response);
}

/**
 * @ roseuid 3D2E997A00F0
 * @ J2EE_METHOD--execute
 * 控制器,控制流程实现
 * /
private String execute (HttpServletRequest request)
{
    return null;
}

/**
 * @ roseuid 3D2E997A0122
 * @ J2EE_METHOD--destroy
 * Called by the servlet container to indicate to a servlet that the servlet is being
 * taken out of service.
 * Clean up resources
 * /
public void destroy ()
{
}
}

```

代码中加粗下划线的内容就是编写的代码部分。完成了代码的编写，开发人员就可以开始代码反工操作了。

(2) 选择反工操作

打开 Rose 模型文件，在弹出菜单中选择“Reverse Engineer...”选项，如图 4-20 所示。

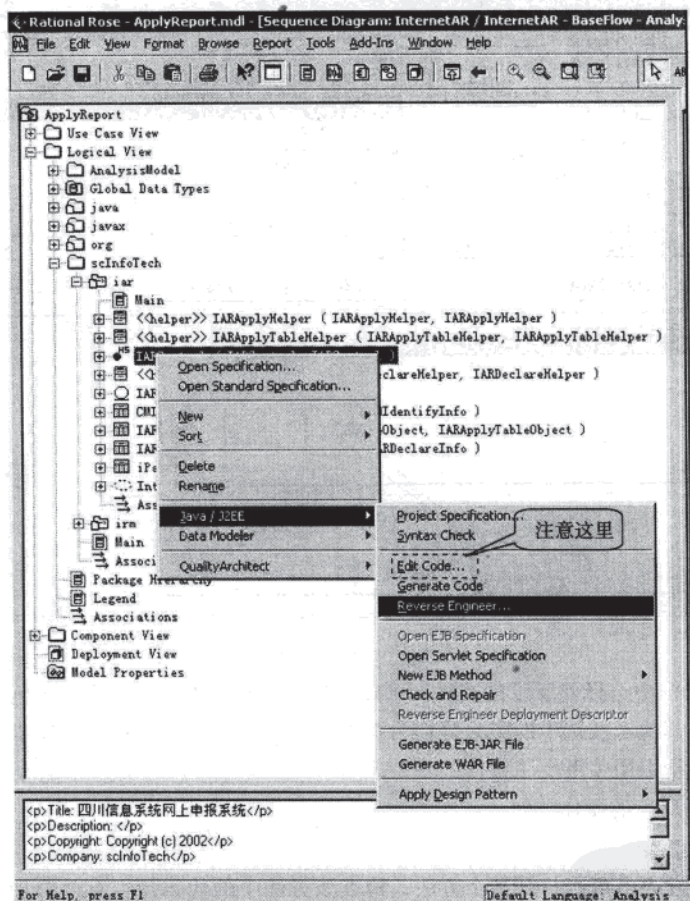


图 4-20 反工的菜单选择

注意：

图 4-20 中标示的是一个编辑代码的功能，该编辑功能，只是一个纯文本的编辑器，与 Windows 的 Notepad 相似，Rose 这个 Model 工具与 IBM 的集成开发环境的接口也许就在这里。

(3) 进行反工操作

在如图 4-21 所示窗口中选择要反工的文件进行反工操作。

图 4-21 的说明：

No 1 在这个框中选择要反工的文件放置的路径。

No 2 单击“Add”按钮，把上面的文件添加到下面被选定的反工文件框内。

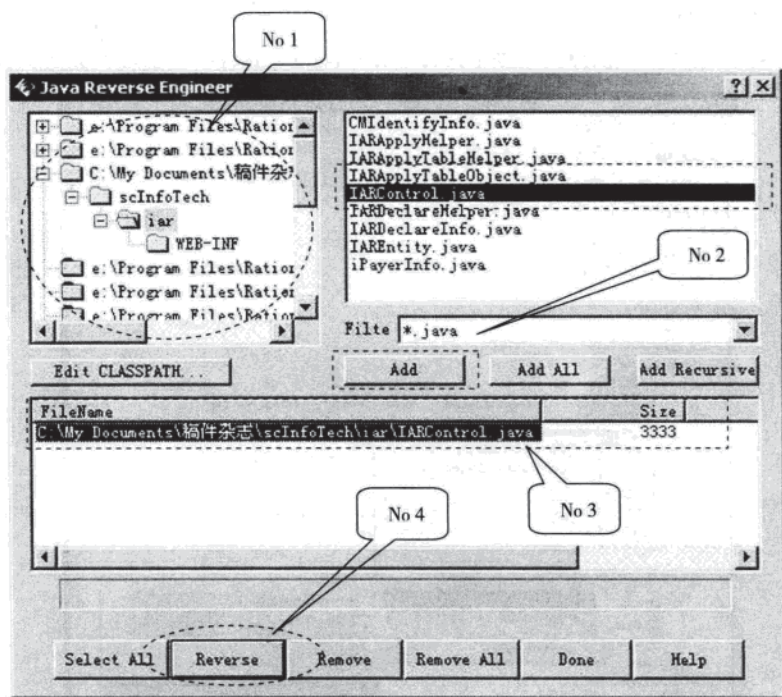


图 4-21 反工对话框

No 3 在这个框中选择要反工的文件，可以多选。只有选中的文件成为蓝色时才会被 Rose 执行反工操作，其他未被选中的类只是作为引用调用关系而存在，在进行反工操作时，对被反工类的接口进行引用关系反工操作的支持。

No 4 这就是反工的操作按钮。

(4) 检查反工结果

反工后，开发人员可以打开某个方法，检查该方法中的代码是否有效合理，如第 3.4 节中的图 3-37 所示。

当然，也可以使用 Edit Code 来查看和编辑代码，因为 Edit Code 只是一个简单的文本编辑器，所以，关于 Edit Code 的用法在这里就不做任何多余的描述了。

4.5 参与人员建议

1. 主要开发人员

本模块的编码实施人员、本模块的设计人员。

2. 主要审核人员

项目经理、系统分析员、本模块和其他模块的分析模型开发人员、其他模块的设计实现人员、其他模块的编码实施人员。

4.5.1 输入输出

1. 输入工件

设计模型。

2. 输出工件

代码。

4.5.2 UML 元素

1. 相关 UML 元素

状态/活动图、时序图、类图。

2. 用处

进行类开发的过程中，如果发现类内设计不合理的地方就应该与设计人员讨论确定解决方法，设计人员修订设计模型时会用到这个 UML 元素。

4.6 附加说明

4.6.1 需求变化时的策略

在实际开发过程中，当遇到需求变化，或者进行设计前对需求的了解不是很清楚或忽略了一些细节时，有以下两种解决方式：

第一种方式：如果项目时间允许，建议从分析模型开始进行修正，逐步延伸到代码，也就是完全按照 4.3.3 小节中建议的需求变动维护步骤进行。

第二种方式：当时间不允许的时候，至少要从设计模型开始进行修正，并重新生成代码框架。此时，Rational Rose 不会覆盖原来的代码，相反会在上面做出一些标示，将已经不存在的方法注释掉。

在这个注释的过程中可能会发生一些下面需要注意的问题。

4.6.2 需要注意的问题

这里需要注意的问题是：这种注释是傻瓜式的，它会与原来代码中的注释发生冲突，并造成注释不成功的语法错误。

例如，如果原来的代码是这样的：

```
/**
 * 处理合同文档编号的算法实现
 */
int iCount = 36;
.....
ContractNo = .....;
```


如果删除了这个方法，Rose 会把它变成下面这样：

```
/*  
/**  
 * 处理合同文档编号的算法实现  
 */  
int iCount = 36;  
.....  
ContractNo = .....;  
*/
```

结果是，集成开发环境就会认为出错，相信读者可以很轻松地看出错在哪里。

虽然这些错误修改起来比较容易，但需要注意的是，不要完全相信导出的代码可以做得很完善，开发人员在每次重新导出代码后，都要进行代码的检查，以避免出现这种问题。

经验告诉我们：**越低级的问题造成的错误越难以发现。**

如果一个 bug 经过 3~4 个小时的查找还是没有发现其产生的原因，那么，这个问题往往就是一个简单的错误造成的，例如：字母大小写错误、单词拼写错误等。



第 5 章

协作开发

完成了编码工作（代码模型一致性）之后，本章将介绍一些有关团队协作开发过程的内容，并为前面的几章做出一些必要的补充。

众所周知，现在的项目往往是需要多人协同进行开发的，本书从第 2 章到第 4 章分别从需求、设计和代码 3 个方面对软件工程过程中的建模方式和实施步骤进行了详细地描述，但是在这些描述中并没有涉及多人协作开发的问题，那么，项目组中该如何协同开发完成整个项目呢？

当然，如果你所从事的项目开发人员不超过 5 人话，那么，你可以不阅读本章的内容。因为 5 人以下的团队，通过人与人之间的直接交流和模型交互，如果分工合理，也许开发效率会更高。

本章内容：

- 协作开发过程；
- 多人协作开发；
- 文档模型结合。

新
知
识
PDG

5.1 协作开发过程

Rose 的协作开发是通过分割 .cat 包, 然后进行合并的过程。

这里介绍的是一种常规方法, 在本节的后面笔者会描述一种不太规范的分包操作方式 (参见 5.1.4 小节的内容)。

5.1.1 准备工作

(1) 检查项目状况

如图 5-1 所示, 系统中有的两个包, 现在要把这两个包分给两名开发人员来开发设计模型。

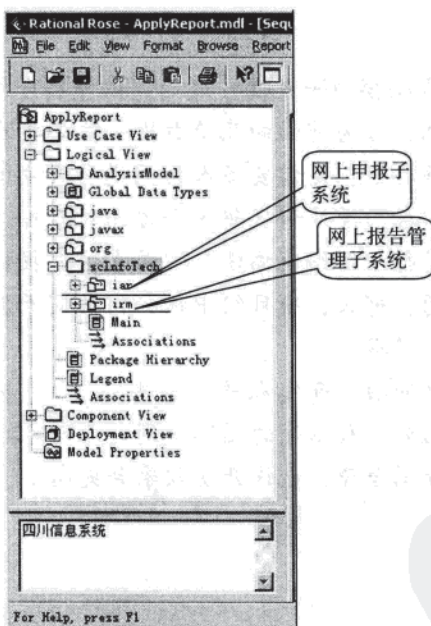


图 5-1 模型原图

(2) 设定模型文件的 Path Map

在设置 Path Map 之前, 需要加载系统所有的包, 并且模型文件和所有的子包都必须是可写的, 因为 Rose 要把 Path Map 的属性值写入到这些文件中, 如图 5-2 所示。

图 5-2 和图 5-3 的说明:

No 1 选择该选项, 系统会弹出如图 5-3 所示的对话框。

No 2 这里定义的是模型名称, 可以根据需要进行设定, 用于加载不同的模型包。也可以只设定一个, 用于所有的包。

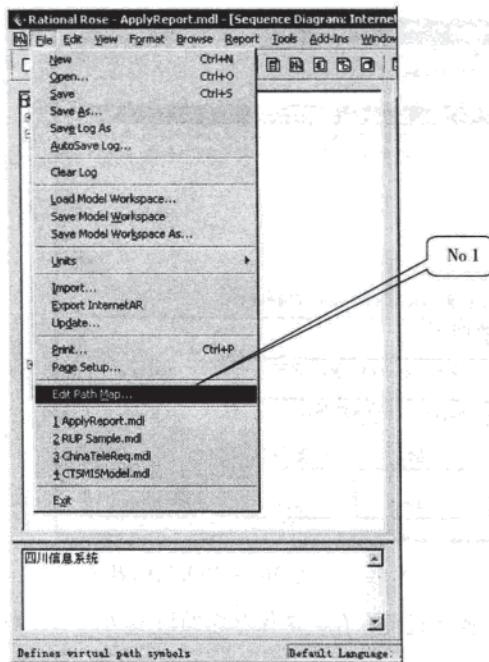


图 5-2 选择“Edit Path Map...”选项

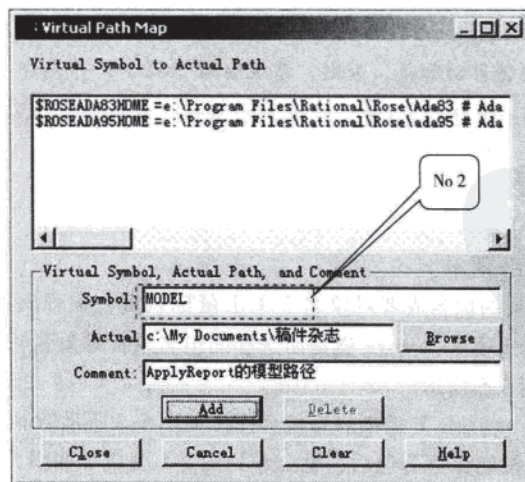


图 5-3 编辑 Path Map 的对话框

单击“Add”按钮保存你的操作，保存“Model Path”后的界面如图 5-4 所示。

(3) 单击“Close”按钮来确定你的操作

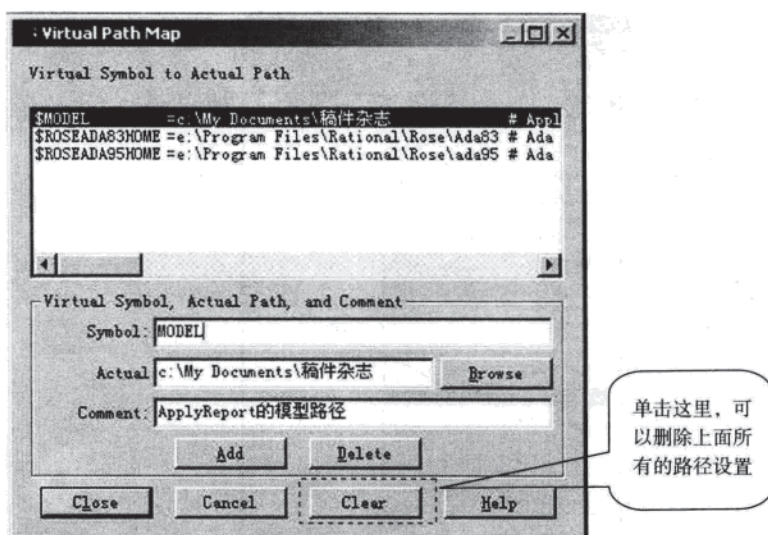


图 5-4 保存 Model Path 后的界面

如果还没有对所做的操作进行保存, 请保存你的模型文件, 否则下次打开文件时, 仍然会出现问题。

说明:

设置了“MODEL”参数后, 当再次加载这些 cat 包时, 系统会自动查找对应的“Actual”目录, 如果找到了, 就会自动加载, 否则, 系统会给予提示, 并允许手工查找。如果没有设置“MODEL”, 则无法加载这些 cat 包。这些在后面都将有图示说明。

5.1.2 分包

分包的具体步骤如下所示:

- 1) 在需要分离出来的包名或者对象名上单击鼠标右键, 在弹出的菜单中选择“Units”→“Control *****”选项 (***** 就是所选择的包名或者对象名), 然后将它保存到某一个特定的目录下, 如图 5-5 所示。
- 2) 选择“Control *****”选项后系统会弹出如图 5-6 所示的保存对话框。
- 3) 单击“保存”按钮保存分包后, 就会看到图 5-1 变成了如图 5-7 所示的界面。如果未加载该包, 图标就会变成如图 5-8 所示的样子。

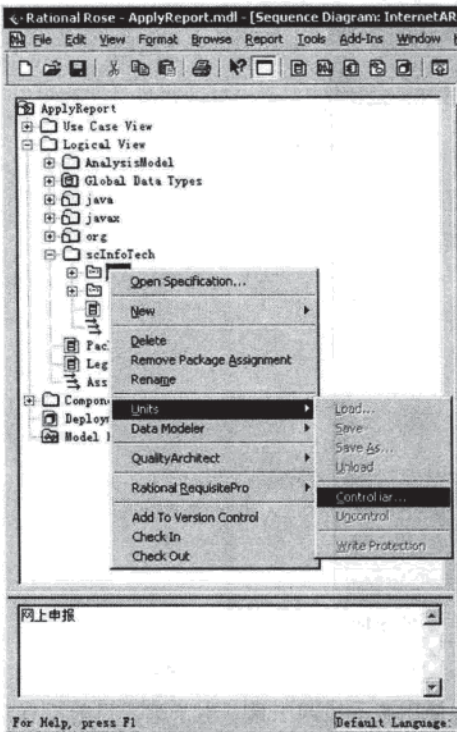


图 5-5 选择分包选项

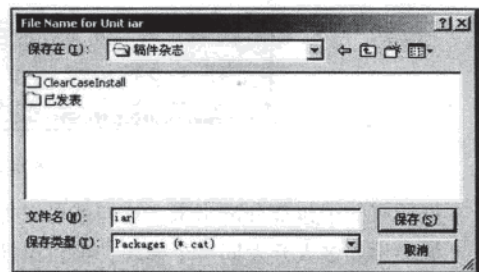


图 5-6 分包保存对话框

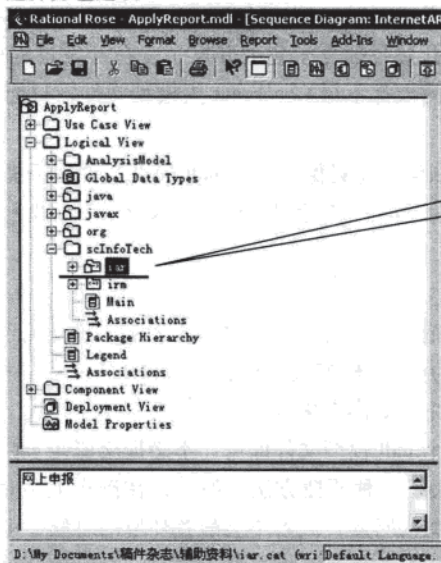


图 5-7 保存分包后的图标

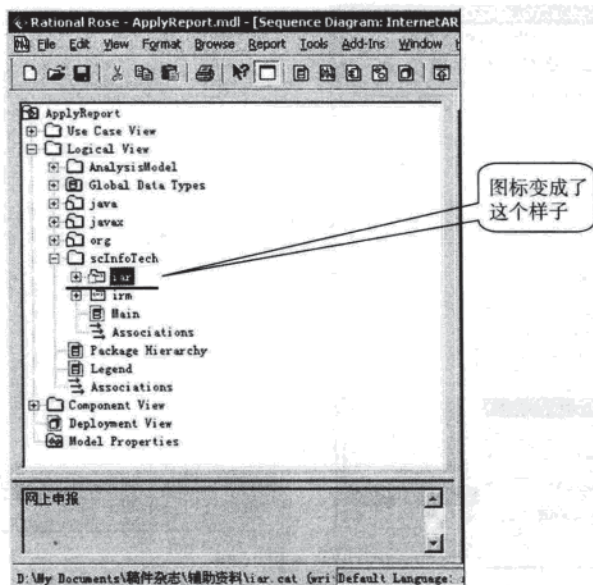


图 5-8 未加载子包后的图标

把所有的子包都分离出来，这样就可以每一个人开发一个模块做到互不影响。

如果想要多人开发同一个模块，项目组就需要使用版本控制工具。由于笔者目前没有常用的 3 种配置管理工具的环境（现在手边只有 MS VSS 的环境，也就无法进行对比分析），所以，这部分内容暂不做介绍。

结束分包后，开发人员可以开始各独立子包的开发了。在开发过程中有一点需要记住：除了主模型维护人员外，其他任何人都不得 Check out 主模型文件，只能 Check out 自己开发的 cat 文件包。

在完成模型的开发任务后，如何才能只保留一个 mdl 模型文件用于提交给公司或者用户，而不必提交一个加上如此多 cat 包文件的 mdl 模型文件呢？这就是下面我们将讨论的问题。

5.1.3 合并分包

完成系统开发后，在项目组把系统交付给用户之前或者某个模块已经不再需要后续的修改时，可以把这部分子包重新合并到主模型文件中去。选择“Units”→“Uncontrol jar”选项把这个包重新放入到模型文件中，如图 5-9 所示。

然后，项目组只需要提交一个 mdl 模型文件即可，不必提交那些 cat 包文件。

5.1.4 小节中的将要介绍方法是一种便捷方法，但是，在操作上不是很规范，建议用户不要轻易使用。不过，在比较紧急的时候，或者不会配置环境的时候，可以用来救急。

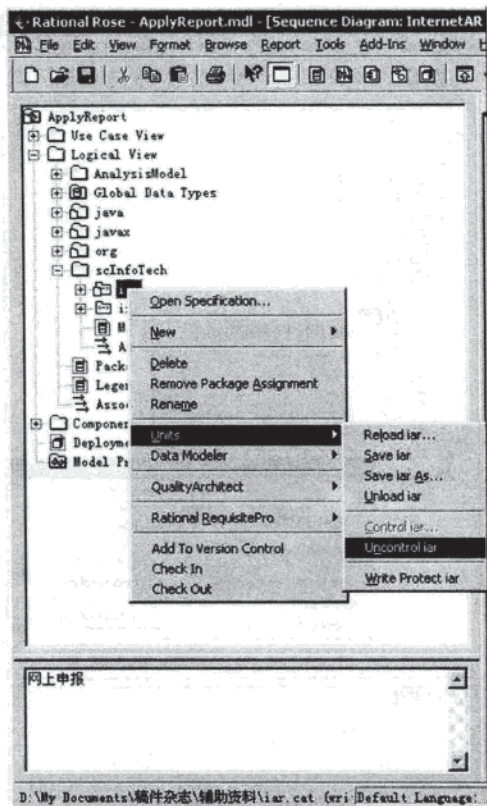


图 5-9 将包重新放回到模型中

5.1.4 分包特殊操作

通过这种方法,就不需要再经过“设定模型文件的 Path Map”这一步的设置,而直接进行分包即可。

在第一次加载已开发的 cat 包之前,需要将模型主文件设置为可写,然后 Rose 就会允许你加载 cat 包。这是因为,一般在配置管理工具管理下的文档都会被设置为只读属性来保护文件,此时, Rose 是无法对主文件进行修改的,那么也就无法使用户加载任何 cat 包了。

具体加载步骤如下:

- 1) 选择“Units”→“Load irm”选项来加载包,如图 5-10 所示。

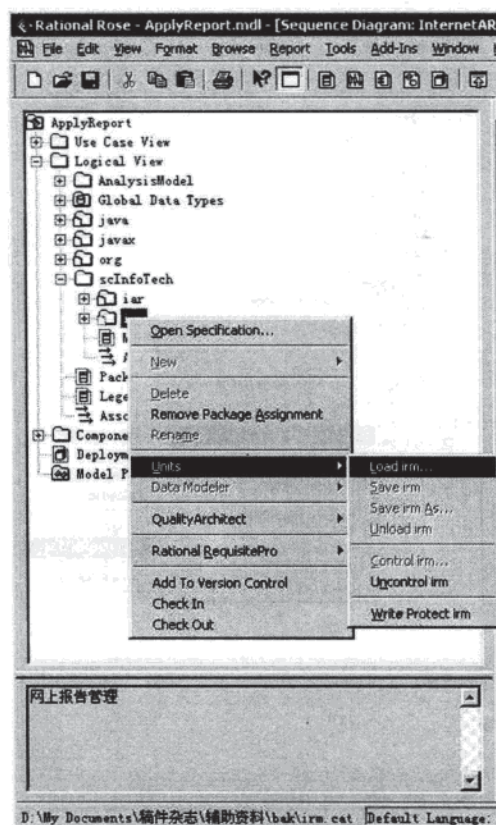


图 5-10 加载 irm 子包

2) 当需要选择其他目录来加载 cat 包文件时, 系统提示如图 5-11 所示。

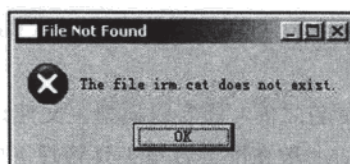


图 5-11 系统出错提示 (系统无法直接找到 irm.cat 文件)

3) 如果要进行选择目录操作, 就必须修改文件的属性, 将文件的“只读”属性去掉即可, 如图 5-12 所示。

图 5-12 中 No 1 的说明:

这里表示该文件具有“只读”属性, 修改将不被保存, 单击这里, 就可以修改模型文件的属性。

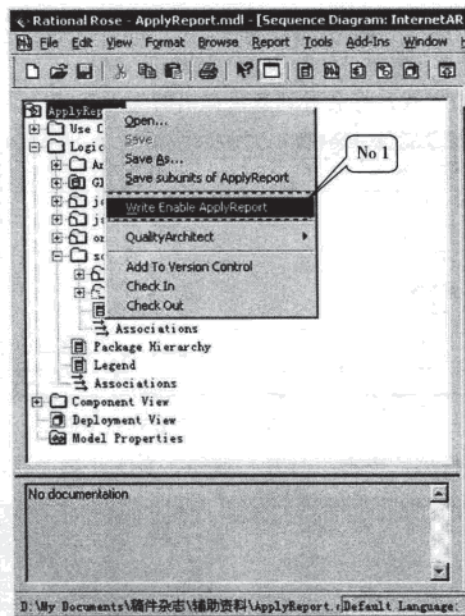


图 5-12 修改模型文件属性界面

4) 将模型文件修改为具有“写”属性,如图 5-13 所示。

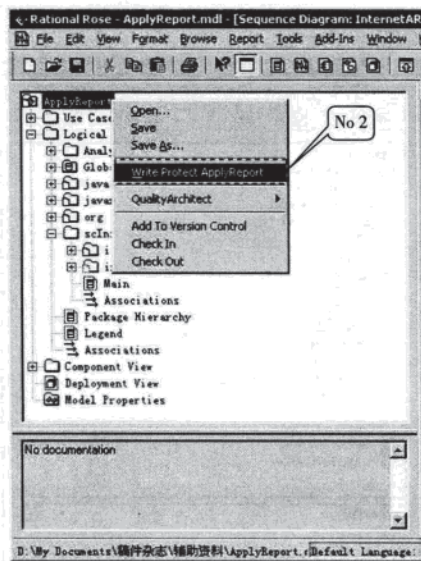


图 5-13 修改后的模型文件属性为“写”属性

图 5-13 中 No 2 的说明：

No 1 变成了现在的样子，表示 Apply Report 被允许写入。

此时再进行 cat 子包的加载，将会看到系统弹出的文件查找对话框，如图 5-14 所示。

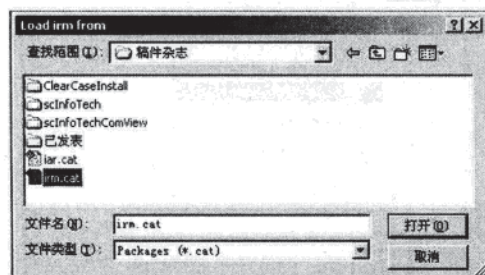


图 5-14 系统弹出的文件查找对话框

5) 选择对应的 cat 文件后，单击“打开”按钮，系统开始加载过程，如图 5-15 所示。

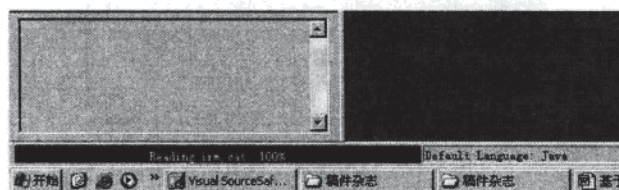


图 5-15 系统加载 cat 包的过程图

加载完成后，系统显示如图 5-16 所示。



图 5-16 系统加载 irm 子包后的图

请注意图 5-16 与图 5-10 的差别。

5.2 多人协作开发

5.2.1 第一种方法

这是一种比较简单的方法：在 1 个包下重新分出几个小包，然后交给几个人分工开发。在分包时应尽量保证这每个人所开发的部分不重合，同时定义统一的命名规则，使得文件具有统一性易于合并。

当每个人都使用共同的类和方法时，可先各自单独开发，然后在集成时进行相同类和方法的合并，或者在开发过程中随时进行沟通与协调创建公共的类和方法。

5.2.2 第二种方法

正常分工后，多人进行协同开发，应定义遵守共同的命名约定：在命名变量、时序图、状态/活动图等时，每个人要在名称前添加相应的子模块名称以示区分。这样当集成时，系统就可以很轻松地进行合并了。

5.2.3 一个特殊情况

有一些时候会遇到这样的问题：当一个模块确实需要几个人来进行协作开发，但又无法将他们分成几个相对独立的部分。

这个时候就只能通过配置管理工具来实现。通过配置管理工具对该模块进行分支管理后，就可以直接由多人在不同的分支下对“同”一个 cat 包进行开发，然后进行集成合并。

注意：

2003 年前对 Rose 模型文件的这种集成合并只能在 Rational 的 Clear Case 中进行，近几年配置管理工具有了很大的提升，理论上目前其他工具也均可以完成对 Rose 模型的这项操作，但由于作者没有实践，所以不能建议其他配置管理工具。

由于这样集成的过程本身也是相当繁复的，所以，笔者建议最好不要采用这种方式。2001 年笔者曾经为一次模型的复杂集成花费了 4 个工作日的时间。

后来经过深入思考研究后，笔者认为这种情况可以通过管理方式的配合实现 merge 操作的简化，具体方式可参考如下：

第一，每个人自行开发的方法和属性全部在后缀或者前缀增加个人标识编号作为区分。

第二，对原有公用方法、属性的修改一律采用新创建方法、属性的方式进行，明确规则参考上一条要求。

方法可以采用创建新方法的方式，内部调用原有公用方法来实现对公用方法的补充

修订。

属性采用传值方式将公用属性值传递给自定义的该公用属性的替代属性。

第三,如果传入参数需要指定具体方法,那么可以在公用对象的公用方法的参数中定义一个类别区分参数属性实现不同的人进入公用对象后执行各自指定的不同流程。

通过这三条方式的应用,基本上可以解决 merge 时大量同名方法、同名属性的冲突问题,可以极大地提高 merge 速度。如果开发组成员都遵循了这样的方式,并行开发的 merge 是不会有问题的,唯一的弊端就是这个公用对象或类的规模会成倍增加,这个问题可以在项目中后期通过重构的方式来解决。

5.3 文档模型结合

5.3.1 概述

在 Rose 的模型文件中,开发人员可以将对应的文件添加到模型中进行文档的统一管理。

通过这些添加可以实现在 Rose 中对项目所有文档和工具的统一管理与使用。除了与 Rational 公司的其他工具联合使用外,还可以通过操作系统所设置的文档管理方式来打开 Microsoft Office 中的任何工具以及其他的开发工具,通过这些方式可以实现 Rose 对整个项目的资源管理。

在图 4-20 中,已经提到 Rose 中集成了一个基本的代码编辑器,可以通过该编辑器来对项目的代码进行编辑。笔者也曾提到过: Rational 公司也许是在考虑将来在 Rose 中将集成全部可用的代码编辑,向这个方向发展也许是 Rational 公司的一种策略或者战略性的考虑。在 Borland 宣布收购 Together Soft 之后不久, IBM 公司就宣布收购 Rational,从而重组了 IBM 的整个开发工具产品线,这暗示着 Rational Rose 的下一个版本中将会直接获得 IBM 的集成开发工具 WSAD 的全面支持,从而完整地实现从建模到代码管理的全过程。

从一个 Borland 和 Rational 工具使用者的角度来看,笔者当时多么期待 Borland 和 Rational 的结合,哪怕 IBM 收购的是 Together。可是,毕竟商业不是技术,不能单纯地从理想的角度出发来获得所谓的理想的结果。

下面将描述文档与模型结合的实现步骤和操作过程。

5.3.2 文档模型的结合

文档和模型的结合实际上就是将文档添加到模型中进行统一的管理,这样才能实现整个开发过程的工具体化,减少不必要的工具使用和资源耗费。

文件具体的添加过程如下:

1) 在所选项目上单击鼠标右键,在弹出的下拉菜单中选择“New”→“File”选项,如

图 5-17 所示。

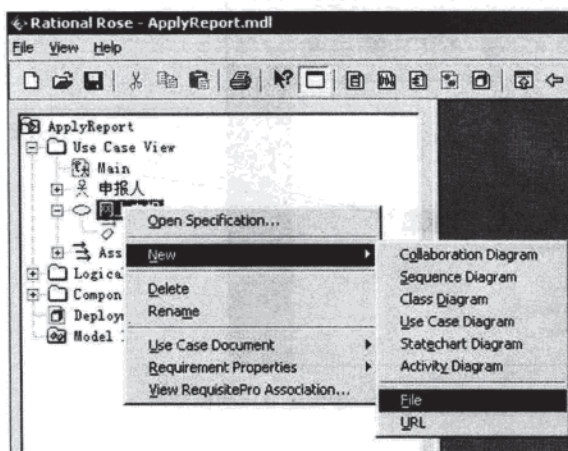


图 5-17 选择添加文件的选项

2) 然后系统会弹出一个文件选择对话框, 如图 5-18 所示。

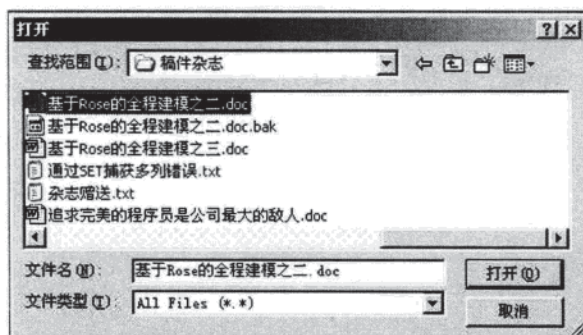


图 5-18 系统弹出的文件选择对话框

3) 选择要添加的文件, 单击“打开”按钮, 完成文件的添加操作, 如图 5-19 所示。

需要说明的是, 上面这个过程不是在配置管理工具中实现的, 所以, 在使用中将不存在版本管理和控制的功能。Rational 的 ClearCase 提供了这种针对 Rose 的文档模型结合管理功能。

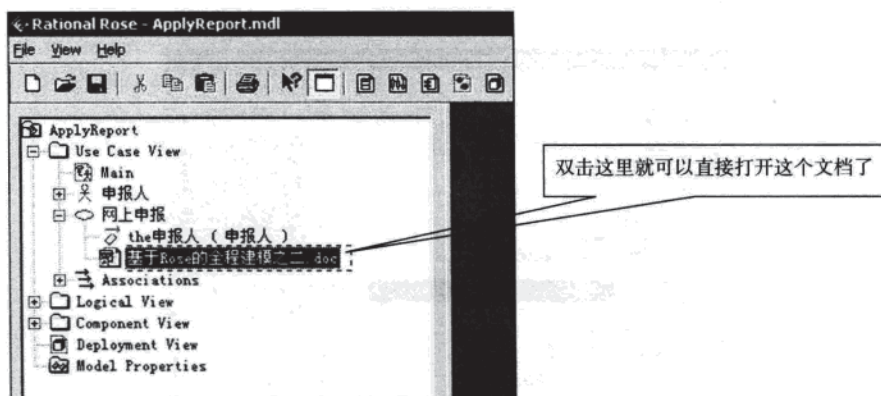


图 5-19 添加完成后的模型图



第 6 章

软件组织方法论

软件组织方法论指的是团队内成员间的合作方法以及相互关系，因此其内容包括人与人之间的关系，人与团队之间的关系，团队之间的关系，团队内的组别划分，人与企业之间的关系和团队与企业之间的关系等。

本章内容包括：

- 结对编程；
- 交换编程；
- 可度量绩效管理模型；
- 实用性知识库体系构建。

第 7 章软件开发中的心态对抗问题研究也属于软件组织方法论的内容，但是因为其涉及面较为独立，亦可自成体系，于是将其独立为一章进行介绍。



6.1 结对编程

6.1.1 产生

在传统的开发过程中，一般是一个人从一个模块的需求调研开始，做分析、设计、编码、单元测试，然后才会交给第二个人（专职测试人员）进行其他测试项目。这样的开发过程会因为开发人员的变动而对项目的进展产生较大的影响，所以有人提出项目中编码人员的重要性远比项目经理大。传统开发模式及其问题参见图 6-1 和图 6-2 所示。



图 6-1 传统开发模式

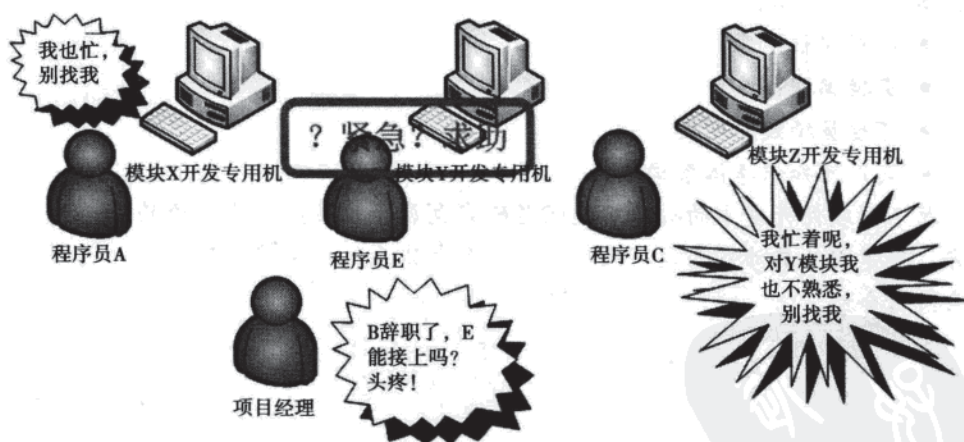


图 6-2 传统开发模式的问题

于是，在这样的实践过程中有人提出了结对编程的做法。

6.1.2 结对编程形成

结对编程的定义如下：两个程序员在一个工作台上共同参与、合作。就好像一人开车一人做导航员，而且最好能经常互换角色。这种合作对开发程序有很多好处。比如增加纪律

性，写出更好的代码等。

结对编程是 XP 编程中建议的 12 条实践之一，基本结对编程形式如图 6-3 所示。

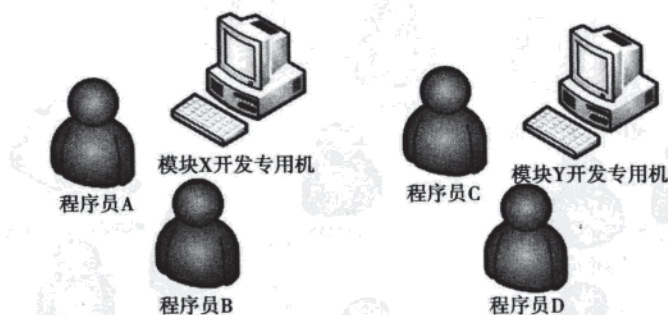


图 6-3 结对编程基本形式

6.1.3 结对编程的变形模式

1. 轮换结对编程

轮换结对编程模式如图 6-4 和图 6-5 所示。



图 6-4 初始轮换状态

这种形式就是基本结对编程方式的团队实现模式，通过轮换和交换的形式可以实现不同技能程序员之间的配合，快速地解决问题。

但是，看上去这种庞大的结对模式造成的印象上的人员浪费是不可避免的，因此，推动

起来是有较大难度的。

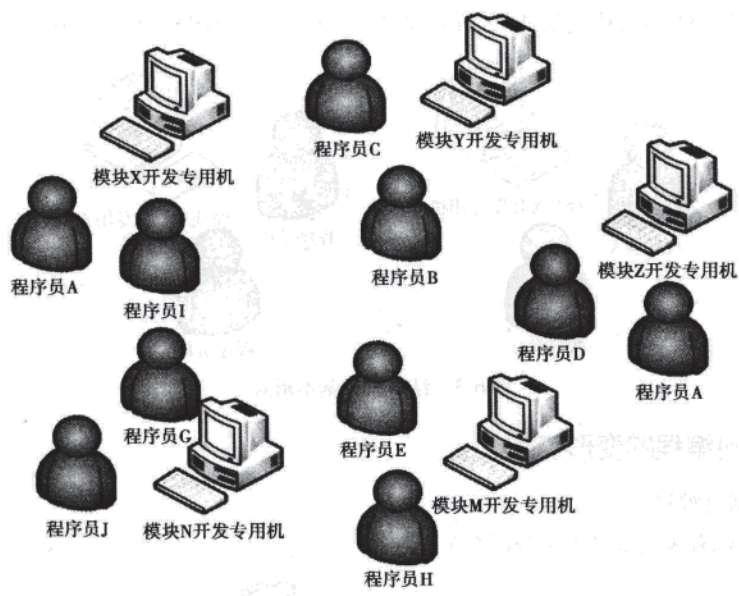


图 6-5 内层换位后轮换结对示意图

2. 指导结对形式

传统开发模式下，项目经理指导开发方式所形成的近似结对开发模式如图 6-6 ~ 图 6-8 所示。



图 6-6 轮换结对初始状态

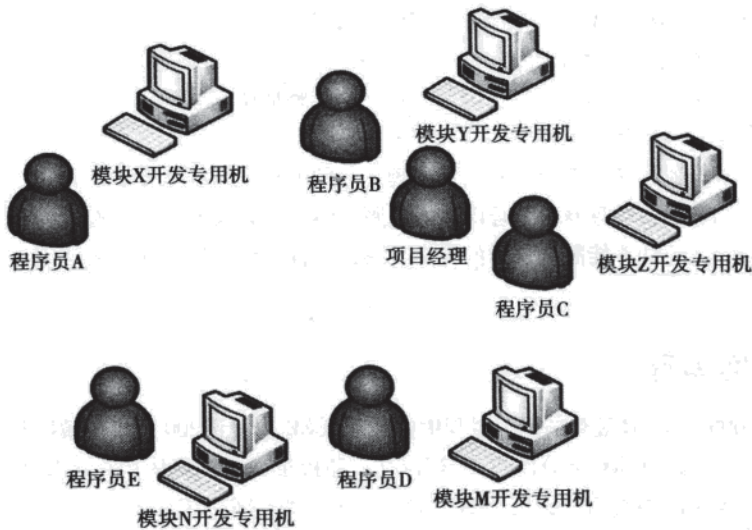


图 6-7 轮换结对实现模式之项目经理与程序员 B 结对

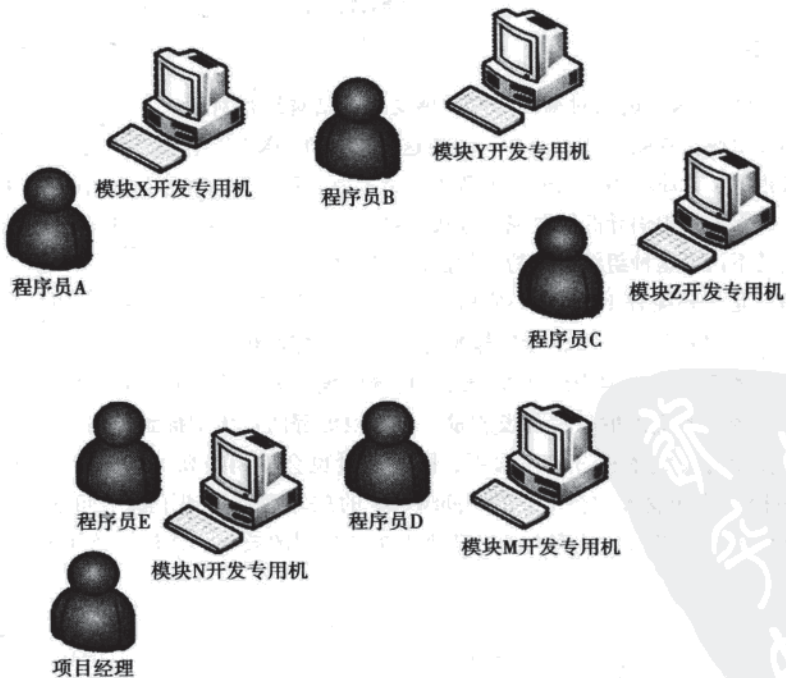


图 6-8 轮换结对实现形态之项目经理与程序员 E 结对

这种结对编程的团队组织形式类似于传统开发中项目经理指导个别程序员进行开发的模

式, 也应该是目前国内多见的开发团队形式中最为接近结对编程的团队组织模式。

但是由于双方身份的较大差异, 项目经理的指导下程序员自身的压力感会比较大, 虽然这种方式有一定的促进作用, 也会带来一些负面的抵触情绪。其问题的解决方式取决于项目经理的管理策略和与下属之间的沟通方式。

另外, 这种开发团队组织形式对项目经理自身的编程开发技能要求非常高, 要远远高于团队成员, 以强势项目经理身份管理团队。否则, 团队中必须配置一名专职的技术经理来实现这样的开发模式, 如不能配置这样的人员, 那么项目经理带领的开发团队则不能使用这种团队组织形式。

6.2 交换编程

本节在 2006 年 10 月最初的撰写过程中曾经从 IEEE 检索到 200 多篇与敏捷开发相关的学术论文, 其中涉及结对编程部分的有 73 篇论文, 除掉里面有两篇国内作者发表的文字有抄袭其中一个作者在国内的两篇论文之嫌 (其内容和数据完全一样, 仅仅是项目类别不同), 其余 71 篇论文均为有效论文, 这里提出参考的 31 篇参考文献中有 3 本书籍, 另外 29 篇为这 71 篇中较为典型并有较高说服力的论文分别排列放置于本书附录中“参考文献”。

6.2.1 引言

笔者在 2002 年参与的结对编程开发中感受到, 结对编程对两名相互熟悉的开发人员来说在一定程度上确实提高了代码质量, 但是这种方式在 2003 年笔者参与的另一个的项目中被领导直接拒绝了, 原因不言而喻, 浪费了一个人力, 对于公司而言是不可接受的。

因此, 极限编程中倡导的结对编程方式, 对于开发人员较少或者严重不足的项目中, 很多软件企业是不认可这种组织方式的, 他们认为这会浪费很多人力, 一加一未必大于二。

结对编程是极限编程 12 个核心实践之一 [14, 15, 16]。在结对编程中, 要求两名程序员使用一台显示器, 一套键盘鼠标来完成所有分配给他们的任务, 如图 6-9 所示。结对编程的支持者声明结对编程的开发方式与传统的编程方式相比有很强的优势, 这包括更高的团队开发效率和更高的软件开发质量。在极限编程的宣传中提到, 两个人结对在一段时间后可以使开发效率超过单人编程, 同时质量也会得到提高 [20, 21]。同样的, 在 2006 年的敏捷中国开发者大会上, ThoughtWorks 的总经理也提到了这个问题, 他解释说: 当两个人合作 3 个月以后, 效率会超过两个人单独编程的效率。请注意这里有一个时间前提: 3 个月以后。

我们认为, 3 个月未必是真实确凿的时间分界线, 它只是一个模糊的时间范畴, 其前提定义是两个人以前并没有配合开发过或者相互之间并不熟习, 然后在一个团队内进行结对开发。如果两个人配合得很好, 也许只需两个多月, 如果配合不好, 也许需要 4~5 个月, 或者更长的时间……我们相信这样的说法可能像 Martin Fowler 这样著名的软件工程大师也不会表示反对的。

结对编程也可能会出现一些可以想到的问题,诸如当结对中的一个人进行大段代码编写的时候另一个人走神或者接听电话,又或者突然有人过来与其聊天等,如图6-10和图6-11所示。

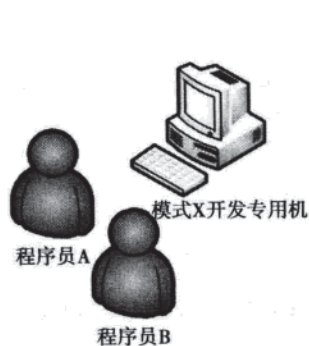


图 6-9 结对开发

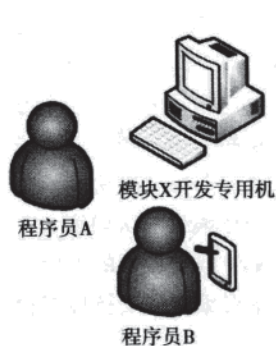


图 6-10 结对编程问题之分神

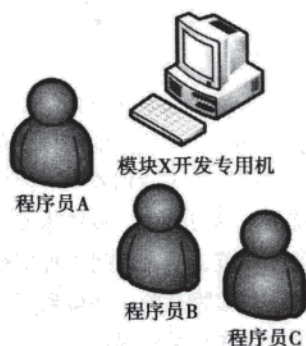


图 6-11 结对编程问题之聊天

但是,在笔者的实践过程中发现,几乎没有软件企业因为极限编程中这样的宣传而采用结对编程的方式进行软件开发。除了 ThoughtWorks 等少数公司外,几乎所有的公司仍然采用传统的单人编程的方式进行软件开发。

在所能查阅到的关于结对编程优劣势分析和数据采集的文献资料中,其研究的基础大部分都是在校学生 [22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41]。只有不到十分之一的论文是关于专业程序员进行结对编程的实验 [6, 30, 31],而这部分关于专业程序员实践结对编程的论文也并没有讨论结对编程开发效率是否真的能够达到两个同等水平的专业程序员单人编程的开发效率。几乎所有的论文对比数据基础都是基于结对编程的两人组和单人编程的一人组之间进行的,这样的数据无疑并不能让软件企业信服这种开发方法所具有的优势。

同时发现,一些论文在实践的基础上提出了相反的观点,如 Hanna Hulkko & Pekka Abrahamsson 的研究就认为结对编程在与单人编程对比的时候并不能始终保持高效率 and 高质量。他们在文中写道:“They indicate that pair Programming may not necessarily provide as extensive quality benefits as suggested in literature, and on the other hand, does not result in consistently superior productivity when compared to Solo Programming” [31]。

基于这样的实践和研究,我们更认为,在与结对编程和单人编程的对比上,交换编程具有其特殊的优势。

6.2.2 相关工作

2006 年笔者提出交换编程之前,我们所能查阅到的所有文献都是与结对编程有关的,并没有检索到任何涉及交换编程的概念和内容。因此交换编程属于全新的概念,提出过程受到了结对编程和单人编程的影响,可以认为是结对编程和单人编程结合的产物,如图 6-12 所

示。由于单人编程模式具有其固有的弱点，所以下面我们主要用交换编程与结对编程进行对比分析。

北卡罗琳娜州立大学的几位研究人员从 2002 ~ 2005 年间 [26] 进行了一系列的试验并发表了多篇结对编程论文 [25, 27, 30, 32, 33, 34, 35, 38, 40]。在他们的试验中全部采用的都是在校大学生，共有 1350 名从大一到大四的学生参与了试验。他们分别从课堂教学和学校软件实践方面进行了关于性别差异、团队构建、布鲁克斯法则、个性特点、学习类型、技能水平、对个体编程的认同度、工作道德规范，或者时间管理偏爱等基于教学考虑的方面进行了结对编程试验，并得到了一些结对编程的支持性试验结果数据和总结的观点。

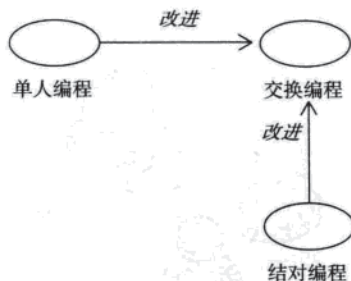


图 6-12 编程组队模式演进图

University of Sannio 的 Gerardo Canfora、Aniello Cimitile, Corrado Aaron Visaggio 等人对学习了一些编程课程的学生进行了结对试验，验证了参加结对开发的个体经验和技能对结对编程过程中知识构建的益处 [28]。

Matthias M. Müller 和 Frank Padberg 进行了一系列的结对编程试验，他们在一篇论文 [7] 中认为，通过使用一系列的度量方法来理解、模型化分析、评估结对编程在软件开发中的效果。在增加了人工成本的基础上，结对编程相对于传统开发过程更侧重于提高团队产量和代码质量。

Frank Padberg 和 Matthias M. Müller 的另一篇论文 [29] 中得到了这样的试验结论：结对编程的效率提高比例与结对编程的个人的程序开发经验无关，而是和结对编程过程中的个人感觉有关。他们认为结对编程过程中结对程序员感觉越愉快他们的开发效率提高得就越多。这篇文章也从同样的角度验证了北卡罗琳娜州立大学 2006 年发表的关于 Examining the Compatibility of Student Pair Programmers [30] 文中提到的“个性特点、学习类型、技能水平、对个体编程的认同度、工作道德规范，或者时间管理偏爱”的结对观点，认为如果不考虑这些因素，结对的效果就无法体现出来。

但是，Hanna Hulkko 和 Pekka Abrahamsson 的一篇论文的试验中引入了企业中有经验的软件开发人员。在这个试验中，它们认为：“回顾这些可以看到绝大多数这方面的实验研究都是在大学环境中进行的。几乎没有在真实软件开发项目中详细审查下的结对编程被已存在的系统化实验研究过。因此，假设目前仍然没有纯粹的经验验证这种优势的存在。”据此和他们的试验结果，文中提出了针对结对编程的反面观点：“我们的经验发现表明：提供了与结对编程所声明的优势对比的结果。它们显示出：结对编程也许不像相关文献中建议的扩展的质量好处一样的必要，并且另一方面，当与单独编程进行比较的时候，这些结果并不能证明结对编程可以一直保持较高的生产率。”

6.2.3 方法定义

与结对编程类似,交换编程的概念也是非常简单和直观的:

两位或者多位程序员轮流开发同一个软件系统的同一个模块的不同阶段的任务。

交换编程的方式更合适的说法应该是交换开发,这种方式不仅可以应用于软件项目,也适合其他研究开发型项目。相对而言,它是一种更容易被人们接受的方法。下面先分析一下它与结对编程的不同之处:

- 交换编程仍然采用传统的一人一机的开发方式,而结对编程是两人一机。
- 它在每个迭代间进行多人交换或者两两交换,结对编程没有交换的概念。
- 它与传统的编程方式之间的差别在于每个迭代间进行多人交换或者两两交换,而传统编程没有交换的概念。

1. 概念声明

- 轮流交换:3个以上的程序员之间相互交换所开发的工件,不只限于3人。例如,A1的开发内容交给A2做下一阶段开发,A2的交给A3做下一阶段开发,……,An的交给A1做下一阶段开发。这种方式称为轮流。
- 两两交换:两个程序员之间相互交换所开发的工件。仅限于两人之间。

2. 工程过程中的应用

交换编程中的操作与其他过程有较大的差异,建议在软件工程实施的各个阶段按照如下方式进行:

- 需求工程中,需求调研和需求分析进行轮流交换;
- 概要设计(分析模型)开发中,需求分析到概要设计也进行轮流交换;
- 详细设计(设计模型)开发中,概要设计到详细设计再进行一次轮流交换;
- 编码实施启动后,详细设计到编码的交换采用两两交换。

在全程建模开发方法下的交换编程应用方式如图6-13所示。

在编码以前全部采用轮流交换的目的就是为了让更多的人了解项目的进展,熟悉项目所开发的内容,并能提出自己的观点。有利于增加团队内成员之间的交流,让更多的人从不同的角度来研究某个系统模块所要实现的功能,以及用户需要解决的实际问题;不会因为某个人的定式思维而出现理解偏差,从而造成对需求的理解不足。

详细设计编码的交换采用两两交换方式,这是因为前期需求已经基本稳定,不需要对用户需求进行更多方式的理解,只需要实施并进行纯粹的编码工作即可。此时做轮流交换已经不存在任何意义,相反只能影响开发进度。

两两交换编程和轮流交换编程方式如图6-14~图6-18所示。

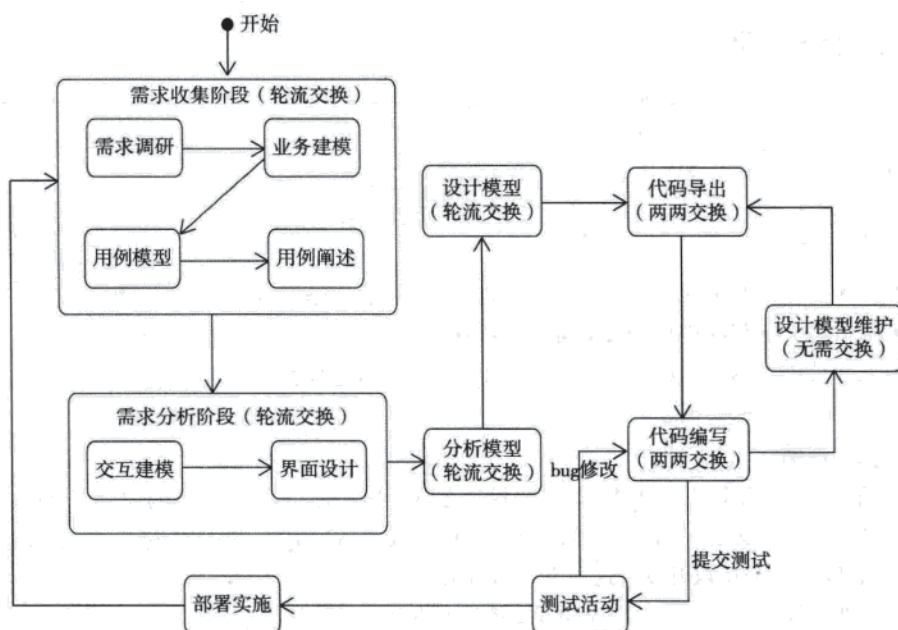


图 6-13 交换编程应用与开发阶段间的关系

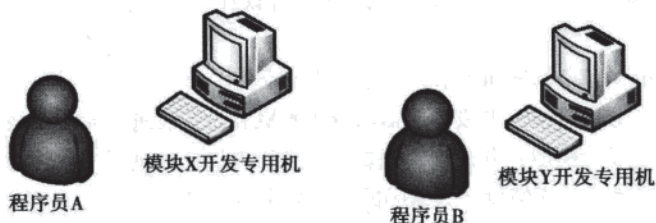


图 6-14 两两交换编程的初始状态

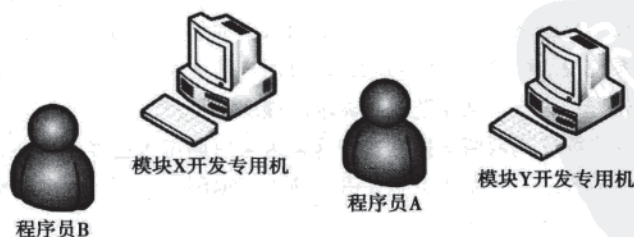


图 6-15 两两交换编程



图 6-16 轮流交换之初始状态



图 6-17 轮流交换之初次轮换



图 6-18 轮流交换之再次轮换

6.2.4 优劣分析

表 6-1 中表述了传统的单人编程、交换编程、结对编程以及交换编程 + 结对编程 4 种人员组织形式下在常见的软件开发相关问题上应用适用性以及表现出来的特点。

表 6-1 4 种开发结对模型的对比

对比项	单人编程	交换编程	结对编程	交换编程 + 结对编程	说明
开发时间看上去的浪费感觉	无	无	有	有	感觉结对编程会浪费一个人力
人员离职对项目的影响	可能造成某些模块无人能够承接, 必须从头开始	最多只需要从前一阶段结束的时候开始就可以了	可以继续进行, 基本上不受影响	可以继续进行, 基本上不受影响	

(续)

对 比 项	单 人 编 程	交 换 编 程	结 对 编 程	交换编程 + 结对编程	说 明
人员添加对项目的影	需要安排人来指导,对项目影响较大	在不同阶段加入新人略有影响	通过结对来带动新人,项目进度略有影响	通过结对来带动新人,项目进度略有影响	
团队内交流状况	很少	交流较多	结对内交流较多,结对间很少交流	结对内和结对外交流都很多	
团队的稳定性	差	稳定	稳定	非常稳定	
团队内技术水平较为均衡,没有技术水平特别突出的人员	适用	适用	不太适用	适用	比较符合中国大陆和大多数发展中国家的软件人才状况
团队内有技术水平较高实际经验超过 20 年的技术人员	对这个人会有较多地依赖	适用,但是会降低这个人对全部项目的直接影响	必须有,对这个人会有较多地依赖	适用	比较符合欧美发达国家的软件人才状况
结对适应期	没有	没有	3 个月的适应期	3 个月的适应期	结对效果是否能超过两个人独立开发的效率是未知的
项目周期短于三个月	适用	适用	不太适用	不适用	
项目周期在 3~6 个月之间	适用	适用	不太适用	不适用	因为时间较短,结对的效果很难发挥出来
项目周期大于 6 个月	适用,但要考虑人员变动问题	适用	适用	适用	6 个月一般是技术人员变动和适应的一个时限
团队人数 < 5 人	适用	适用	不适用	不适用	
5 人 ≤ 团队人数 < 50 人	适用	适用	适用,但有条件 ^①	不适用	
团队人数 ≥ 50 人	适用	适用	适用	适用,并建议采用	

表中的内容分别从团队稳定性、技术水平、开发周期、团队人员等 4 个方面对这 4 种团

① 对于团队人数少于 20 人的开发团队,企业一般不会允许采用结对编程的方式进行开发,因为这类项目大部分都存在时间紧、任务量大、承接项目的公司规模较小等特点。当然,对于类似微软、ThoughtWorks、AutoDesk 等对资源占用不敏感的产品开发公司中超过 4 人的团队就可以考虑进行结对编程。

队人员组成方式进行了分析。从中可以看到交换编程几乎对所有的项目都有其明显的优势,总体上看,交换编程无论在管理层面还是在团队组织层面或者是在项目变化上都优于单人编程和结对编程,参见图 6-19 所示。

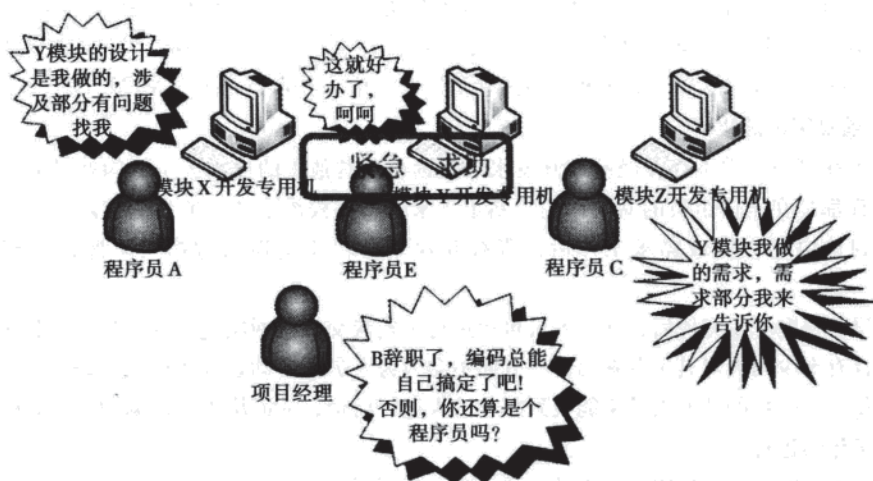


图 6-19 团队稳定性和可持续性提高

6.2.5 试验结果

2002 年 4 月在上海进行的一个类 ERP 项目中的技术人员采用了结对编程的方式来进行一些模块的开发。各自独立完成了模块开发的两名程序员携手进行第三个模块的开发时采用了结对编程的方式。两人只用了 4 天时间，就完成了这个新模块（其模块难度与前面完成的模块开发难度相同）的全部功能，并且该模块开发完成后几乎是一次性通过了全部测试。相对于此前做的功能模块来说，时间仅有其开发时间的十分之一。

由于结对编程会让人感到人力资源被浪费了一半，因此在 2002 年 10 月开始的一个中国电信项目中，项目组提出进行结对编程时被管理层拒绝。那么，如何能借鉴结对编程的优势，并能降低这种表面的浪费，让编程人员进行交流，同时还能提高团队的稳定性呢？下面我们来讨论这些具体情况。

1. 项目基本情况

该项目是电信 MSS 系统的核心业务系统部分，包括规划、设计、施工、验收、财务、合同等多个重要环节和多个部门的业务。这个项目在最初评估时，开发周期为第一个版本在 5 个月内完成，整个项目至少要做上 1 年以上，而最后的实际情况是，这个项目随着不断地升级和调整一直开发了 3 年多。

可以对比的数据是：2003 年初到 2004 年 10 月间 SAP 基于自有的 ERP 系统为中国电信提供了完成同样功能的一套产品来替代这个团队所开发的这个产品，SAP 也向中国电信承诺

要针对中国电信的行业企业特点进行定制开发,但是在2004年底产品推广应用前被宣告试验失败,延期推广。

2. 团队组成情况

当时团队开发人员为11人,人员技能较为均衡,不存在经验和开发水平远远超出其他人的技术人员。11人中3人有5年以上开发经验,两人有3年开发经验,另外的6人有1年以上的实际项目开发经验。后来开发团队开始扩展,从最初的11人扩展到23个人。

3. 项目实施过程:

由于开发团队中不存在经验和开发水平远远超出其他人的技术人员,因此技术方案的论证过程都是在全体组员的共同讨论中制定下来的。在权衡了整个项目的实际情况后,团队经过讨论决定在完成需求工作后,所有开发人员轮流交换来做第二阶段设计模型的开发。

在设计模型开发完成后,开发人员再次进行了轮流交换。两次交换完成后,保证了每一个模块都有至少两人对其十分熟悉,一方面不会因为人员的变动造成团队的不稳定,另一方面确保每一个模块的开发人员都能找到相应的人进行讨论,从而增加了团队内的沟通,方便了协调工作的进行。

在该团队的开发过程中,经常是大呼小叫,相互讨论,场面十分热闹。在Larry L. Constantine's *The Peopleware Papers* [18]一书中的第12章的“工作和游戏”一节中也有类似的描述。Larry L. Constantine认为这是创造性团队应该具有的表现形式,这也符合Brooks [19]的胶冻团队的一个特征。

由于当时没有进行实际数据的度量对比,我们在这里无法从量化的数据上说明问题,只能通过一些具体的事实进行说明和验证:当时这个团队在7个月内满足了南方11家省级电信公司和中国电信集团公司的基本业务需求。从2003年4月到2003年12月期间,基本完成了这些省级公司版本的二次定制开发任务。

6.2.6 小结

综上所述,交换编程的应用方式是有其适用环境的,并且具有较为明显的优势。交换编程的适应性较强,下面以团队状况和项目情况两个部分进行说明。

1. 团队状况

交换编程适用于人数超过两人的开发团队,因为交换一次至少需要两名开发人员。人数较多的团队也可以应用交换编程的方式来进行项目开发,要求团队内有1~2人具有2~3年以上开发经验,这是对软件项目(甚至技术难度较低项目)最基本的要求。

2. 项目情况

项目规模不限,开发周期的适应性较强,对于任何类型的项目都可以适用。

6.3 可度量绩效管理模型

6.3.1 引言

目前我国几乎所有的软件公司或者技术性公司都存在一种困扰人们的问题,即如何评定技术人员的工作量和贡献度。国际上通常情况下使用的是工作日报、周报、月报、年度总结和计划的绩效考核模式,让技术人员头疼欲裂却又不得不因为与工资和奖金挂钩而被迫填写这些东西。

由于技术人员工作的特殊性,加上市场项目的不确定性,使得技术人员的年度计划时有落空,而每天的日报填写更是虚空无物,最终归结为流水账。技术人员的工作量实在是难以衡量,比如说,1天中写了多少代码,能否保证这些代码有多少行是不需要修改的,这些代码带来的价值到底有多大呢?同样的,周报和月报也存在类似的问题。

绩效管理方法是笔者在2004年产生的想法,从2004年起至今天不断地进行补充和丰富,终于在2008年得到了一定程度的系统化,同时开始在一些公司进行推行。

由于绩效管理模式是为了实现有效的绩效管理,提高技术人员工作的积极性,同时为团队今后进行个人贡献度计算以及奖金分配提供绩效计算基础数据,因此采用这种以数据为基础的管理办法来衡量技术人员的工作量和贡献度。

本节涉及的内容将包括开发过程中的项目计划管理、风险管理、团队组成、配置管理和企业代码库构建、知识库体系等。

2008年绩效管理方法一文发表后不久,曾在CSDN软工/管理板块引起了一场关于技术人员是否要写日志的讨论,调查中超过80%的人都不同意或者不愿意写日报周报,大约有10%以上的人同意写周报月报,但是认为需要调整报告形式,大约有5%的人愿意写日报。国内甚至国外有很多企业采用这种日/周/月/年报的形式进行工作总结和作为审核依据之一,但是这种形式对于软件开发人员而言实在是过多地影响了正常开发思路的延伸。该模式从另一个层面上提供了一套方法,通过一套可度量的数据依据对技术人员的工作量和工作绩效进行评定。

注:本文是在对深圳一家游戏公司提供咨询后正式撰写完成的其他软件公司可参考后做调整实现。

6.3.2 团队组成与管理划分

1. 团队组建依据

团队的组成模式要根据各个公司的现状进行考虑而不能一概而论,尤其是对那些已有相对稳定的管理模式的公司就更需要根据具体情况进行考虑了。

对于管理较为混乱的公司,可以考虑自主选择和管理层认可的方式进行团队重组。例如,可以要求每个人填写第一选择两个,第二选择一个,然后根据人员情况和管理层的简

易, 将人员进行重新划分。

注意: 团队划分的内容可以参考下面“基本团队划分”部分的内容。

2. 人员平等, 技术至上

所有人员一律平等对待, 根据实际经验, 毕业年限, 为公司工作年限设定人员的基本工资, 然后根据项目组情况设定奖金和特殊奖励。

奖金部分根据所参加的项目组获得相应的奖金收入, 同时还根据所开发的基础组件当月被重用的次数获得额外的奖金收入(可参见 6.3.3 小节中的“企业代码库”部分的内容)。

3. 方向引导, 产品集中

在没有具体产品和用户需求的时候, 进行公司方向性开发和研究, 并根据所定方向进行团队组织和管理, 同意隶属部门经理或者某一级别的管理者管理。

在有具体用户需求、订单和产品的时候, 进行人员重新组合, 选择合适的人员设定为项目经理, 授权项目经理根据公司产品架构组的建议进行人员选择和搭配, 然后进行项目开发。

4. 基本团队划分

(1) 产品策划组

产品策划组考虑为无形态组的方式进行组织, 公司内任何人员都可以自行或者自由组队的方式进行游戏产品的策划和规划考虑, 并将相应的策划和规划案提交给公司产品架构组进行分析定位和评审。

通过评审的产品, 公司将考虑进行投资研发。

产品策划成形的相关参与人员都可以在公司投资研发的费用中获得 1% (一个设定的比例) 的奖励, 同时得到后期产品研发成功后的销售股份和提成。

(2) 内容开发组

内容开发组是负责游戏内容开发和实现的团队。

这个团队是根据具体项目需求进行组建的, 不属于常设团队。

(3) 工具开发组

工具开发组主要是进行机构设计和游戏辅助工具开发的团队。

这个团队是根据具体项目需求进行组建的, 不属于常设团队。但是, 每一个内容开发组都应该对应有一个同样的工具开发组进行配套支持, 工具开发组也可能独立成立来进行机构研究。

工具开发组人员的技能与内容开发组差异较大, 要求具有机械设计方面知识和技能的人为主要执行者, 有部分软件设计和策划人员参与辅助工作。

(4) 基础组件组

基础组件组是为了构建企业代码库所提供的一种组织形式。

基础组件组也是采用无形态组的模式进行组织, 公司内任何人员都可以提交自己开发的成品组件, 包括下列内容:

- 公司要求的设计说明书和设计模型；
- 组件详细功能说明书、接口详细说明书和详细设计文档（建议采用 UML 模型的表述方式来进行设计，代码直接导出，有利于重用和升级）；
- 可运行组件；
- 组件全部源代码和对应版本说明（注释不得少于 30%）；
- 采用本组件的示例性 Demo。

6.3.3 绩效管理办法基础

本绩效管理办法包括以下几方面的内容。

1. 项目管理

- 实施项目计划管理，制定项目计划和工程过程的基本规范；不需要太详细，可根据情况和需要进行扩充；
- 制定项目计划模板、项目周任务安排模板、项目风险评估模板和项目计划评审模板；
- 切实实行项目周工作例会，有效安排工作任务和工作总结；
- 每项任务完成后，必须经过文档的评审和代码测试来进行审核；
- 制定版本管理规则和开发基线管理规则。

2. 绩效管理

- 任务安排：周例会上进行工作量安排，结合工作完成后的审核机制以及再分配方式进行；
- 工作内容带绩点值；
- 绩点的合作分配规则（主管/组长/项目经理和主承接人员商议分配）；
- 配置库中的 check in 次数配合每次的 comment 内容进行绩点调整（通过次数体现时间加成和工作量加成，详情参见“配置库”的内容）；
- 任务分配不满造成每个月被分配任务所得到的绩点奖金低于前 3~6 个月平均值则按照前 3~6 个月平均奖金发放。每月按照 22 个工作日计算，如果每个月被分配任务所占用的工作时间少于 18 个工作日（这个数字可以按照企业现状来进行重新设定），则认定任务分配不满。
- 配合工作绩点根据情况通过加权累计方式计算额外奖金。

3. 配置库

配置库可以采用各种知名的配置管理工具来实现。

要求所有的使用者在每天 check out 其开发工件时，必须写明本次的任务。在每次 check in 其开发工件的时，应该写明本次 check in 的完成情况。这个信息可以写在 comment 中（各种配置管理工具都提供了这个功能）。

如果是撰写文档，则必须保持每次 check in 的内容和文档上的历史修订记录的内容时间相一致，时间由工具自动保持一致性，而修订内容则由技术人员自己填写，只需要填写一

次,进行复制粘贴即可。

另外,公司如果考虑到代码安全性的问题,那么建议采用下面的代码管理规则:

在软件进行系统测试前,代码完全组内公开,系统测试后,对于需要控制的代码实行专项工作小组模式进行管理和后续的开发延续,此时代码不再全部公开了。

4. 企业代码库

组件代码库的建立和支持办法可以在开发过程启动前进行相应的培训和指导。

代码库主要是构建企业代码基础库,降低企业的重复劳动,提高代码的通用性和共用性。

代码库补充形成的具体要求参见 6.4.5 小节的内容。

企业代码库的完成时间是闲暇时间或者任务间隙,也可以利用任务时间内来完成,但是不会因为要完成这个代码库的补充而延长任务时间或者增加人工计算。这是因为代码库的奖励是按照代码库的重复利用状况来计算的,能够做到一次完成,多次使用才是代码库的最终目的。

5. 绩效体系的执行时间^①

进行两个月的配置库和开发过程的数据积累,所有技术人员不但积累了数据同时还熟悉了整个过程。

第三个月开始制定绩效体系的数学模型,然后通过 3 个月的数据搜集或者根据情况加上第四个月的数据分别进行计算,确保模型的有效性和稳定性,确保平稳过渡。

算法完成后的执行期到稳定期大约需要 5 个月的时间,主要包括:

在基础模型算法的基础上,增加调整优化算法,其的作用是逐渐减弱的,即 6 个月后该调整优化算法的任务结束并退出模型。其目的是让所有人员的工资收入在潜移默化的微量变化中走向规范,而不是马上出现大的收入变化波动。

所以,整个绩效体系的执行时间为下面三个阶段:

前 3 个月进行数据积累,在第三和第四个月期间构建数学模型,同时完成调整优化算法,进行数学推演,反向计算每个人的收入变化情况和可能的今后 6 个月的变化情况。

最早在第五个月开始应用该模型,同时,调整优化算法起作用,直到第六个月底为止(这个时间可以根据具体情况进行分析,不是固定的 6 个月时间,而是会有变化的)。

从调整优化算法推出模型前两个月开始,对算法进一步地计算和修改,从第 12 个月开始,公司的薪酬体系就将完全按照该绩效管理办法来计算奖金和贡献度。

整个过程的实施计划表如图 6-20 和图 6-21 所示。

图 6-20 的这个计划表是按照每周 5 个工作日,每天 8 小时工作方式计算出来的,如果各公司的工作周期不同,则需要做适当地调整。

6.3.4 绩效管理实现模型

元工作任务的记录保存所有基础工作任务信息,这是系统模型形成的基础,后面实际工

^① 从这里开始,下面的时间预算都是按照 60 人以内的开发团队的模型设计进行的,如果人数过多,则可能需要延长相应的时间来配合。

标识	任务名称	工期	开始时间	完成时间	2008年4月				2008年5月		
					6	13	20	27	4	11	
1	现状分析	10工作日	2008年4月7日	2008年4月18日							
2	员工熟悉过程	12工作日	2008年4月21日	2008年5月6日							
3	数据积累第一阶段	44工作日	2008年5月7日	2008年7月7日							
4	进行数据检查	5工作日	2008年7月8日	2008年7月14日							
5	进行数据建模	12工作日	2008年7月15日	2008年7月30日							
6	数据积累第二阶段	22工作日	2008年7月8日	2008年8月6日							
7	进行数据验证	5工作日	2008年7月31日	2008年8月6日							
8	第一阶段实施	44工作日	2008年8月7日	2008年10月7日							
9	实施情况调查	10工作日	2008年9月4日	2008年9月17日							
10	算法模型调整	10工作日	2008年9月24日	2008年10月7日							
11	正式实施	22工作日	2008年10月8日	2008年11月6日							

图 6-20 项目实施计划表

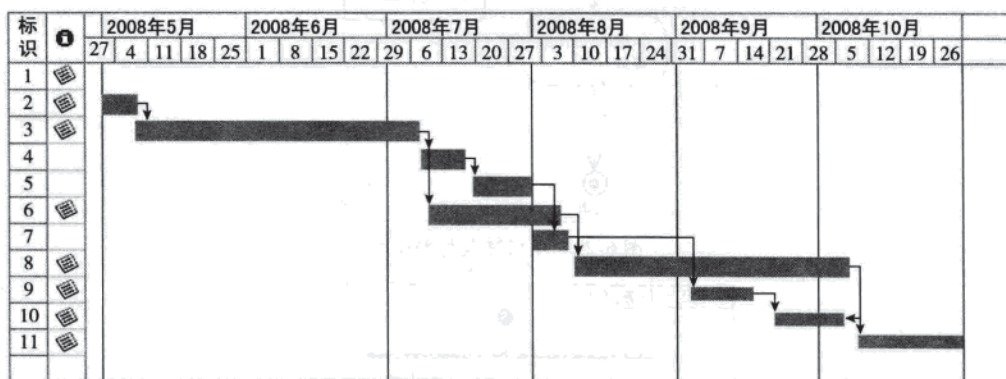


图 6-21 项目实施计划 (续)

作量完成情况的基础, 以及基于实际工作量完成情况计算的利益分配的基础数据。

阶段工作量积累是系统最终计算阶段贡献度和利益分配的有效数据, 年度/项目任务绩效计算是最终贡献度和利益分配的计算基础数据, 如图 6-22 所示。

配置库中绩效内容分类如图 6-23 所示。

对于项目组中的技术人员来说, 他的工作绩效由下面的内容组成:

- 周工作任务绩效: 周工作任务安排, 周工作任务审核结果。每次任务安排中都要划分出开发任务、测试任务和评审任务 3 个部分的绩效, 前两项任务分别由人来承接, 第三项任务根据测试完成的时间进行单独安排。
- 评审工作绩效: 所有的人员都可以获得的工作绩效。

每次评审根据工件的开发情况定义出评审任务绩效, 开发者可以在任务通过测试后提出进行评审, 同时建议参加评审的人员。根据最初定义的评审绩效数值确定最多和最少评审参与人员。

评审前每个人都必须留出相应的时间对工件进行检查, 并提交建议报告。同时, 这段时间将会占用该人员正常承接的任务时间, 因此相应的工作时间顺延。

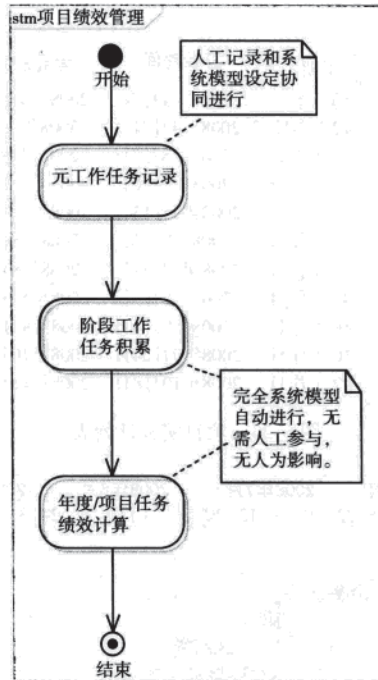


图 6-22 项目绩效管理模型

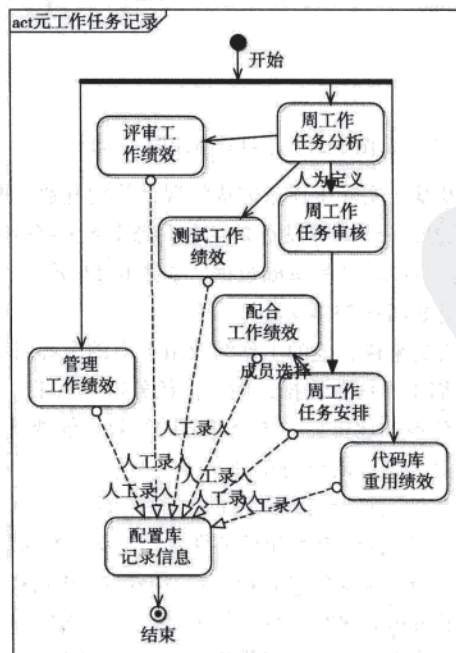


图 6-23 配置库中绩效内容分类模型

如果评审前的准备工作来做好或者敷衍了事, 当该工件发生问题时将会追溯评审人员的绩效进行扣除并处罚相应的绩效。

- 测试工作绩效: 这项任务对于测试人员来说就是主要任务, 承接到测试的任务和任务的完成效率和质量进行评定。对于开发人员应考虑单元测试任务的绩效。
- 配合工作绩效: 每个人都可能有配合工作绩效, 这要看主要开发者是否邀请。

关于配合工作绩效的分配采用的是 karma 原则, 绩效的分配由项目经理、核心技术人员(不多于两人), 加上任务主要承担者 3~4 人进行讨论确定。

如果该任务的主要承担者分配给他人的配合绩效偏低或者不分配, 那么时间长了, 其他人自然会远离他, 也不会给他分配绩效或者邀请他进行配合开发, 绩效偏低的人最后所获得的利益分配自然偏少。

- 管理工作绩效: 管理工作绩效只有项目经理、项目小组组长和一些配合管理的岗位和角色具有, 这些角色由技术人员兼任。
- 代码库重用绩效: 所有人都可以参与的, 即使不是编程人员也可以提供代码库的补充, 因为代码库中的内容是依靠重用数量进行绩效计算的, 所以即使有人试图用大量的网络代码来填充代码库获得绩效也不可能得到这种重用数量。

代码库中信息的编写制定都是有规则的, 网络上不可能找到如此完整的代码和介绍文档, 如果你不能很完善地填写代码的介绍资料, 那么其他人也无法对你的代码进行使用。另外, 随着后续的开发进展, 代码库中的代码是需要升级的, 也就是二次开发或者多次后续开发, 这种情况下, 如果原始提供者没有时间和精力来修订代码, 或者其他人提供了修订, 代码库的下次复用将给新的升级版本的开发者分配绩效指数。这一机制可以良好地保证垃圾代码的自动过滤和不被使用。

如图 6-24 和图 6-25 所示为模型正式实施后的执行过程。

根据配置库记录信息进行数据统计, 每一次任务的结束和任务中开发者 check in /check out 的数量进行统计, 同时根据每次 check in /check out 注释信息和时间进行分析, 每次任务分配第一次创建和通过评审后的注释都是规定格式带有绩效信息的内容。

项目贡献度/年度绩效计算时就从这些数据中获取。

因为配合工作绩效是多人完成的, 故需要单独进行计算。另外, 代码库重用绩效也是需要单独计算的, 虽然两者都记录在配置库中, 但它们都不是目前配置管理工具中的数据可以直接提供并抽取出来用于计算的。也就是说,

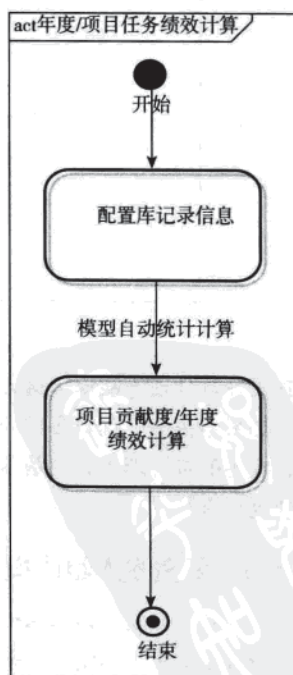


图 6-24 绩效模型的实施模型

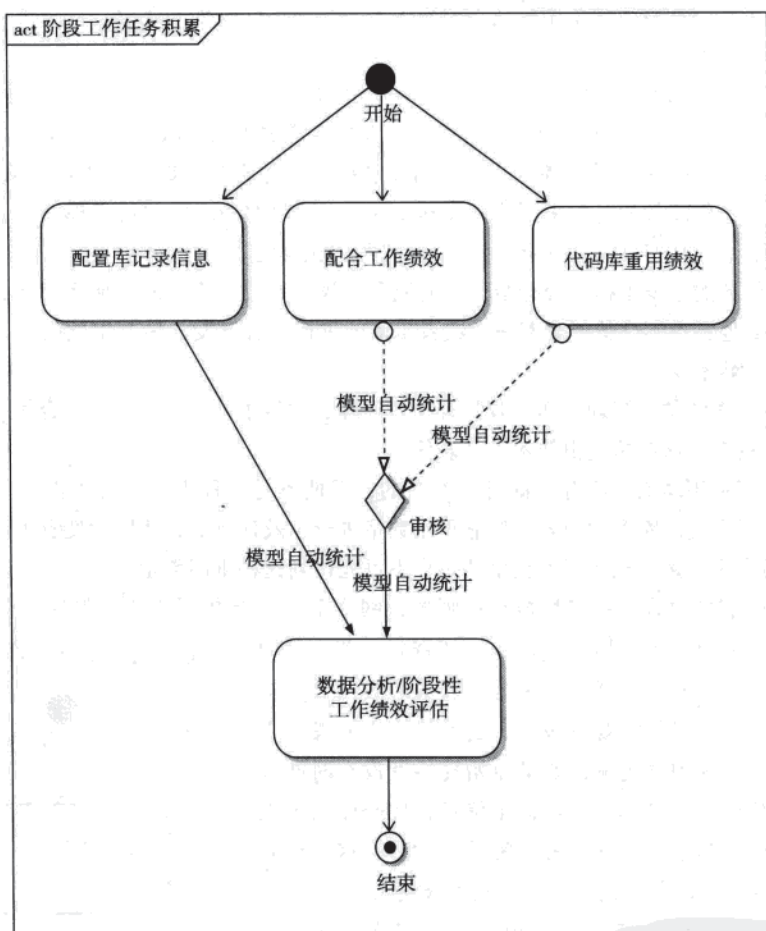


图 6-25 绩效模型的汇总绩效模型

配置管理软件还不能提供这方面的支持。

如果本模型得到较多的应用，那么有可能在将来的某一天会被一些配置管理工具所接受，成为配置管理工具的基本或者扩展功能之一，那时这 3 个绩效就不需要单独进行计算和单独进行审核了。

6.3.5 薪资收入组成模型

整个薪资的组成包括两个部分：一是基本工资，该部分是指每个月固定的收入，保证员工的基本生活水平不受到影响；另一个部分是绩效奖金，这个部分根据员工的工作情况和贡献度来进行计算。

1. 基本工资

基本工资采用下面这些计算加成：

- 参加工作年限指标：这是考虑到工作经验的影响指标。
- 进入公司年限指标：为公司工作的历史贡献影响指标。
- 学历指标：大专、本科、硕士、博士各有不同，这是考虑到每个人求学历程的时间，以及他们进入公司时的基础贡献的影响指标。

一些个人因素指标决定了基本工资的组成和数额。

2. 绩效奖金

绩效奖金分为了5部分：开发工作绩效、管理工作绩效、配合工作绩效、评审工作绩效、代码库贡献度。

(1) 开发工作绩效

根据任务安排和协助任务安排，及完成这些任务后所获得的工作绩效。这些任务包括正常开发相关的所有任务和测试相关任务。

这一项的范围最容易定义，但是，具体计算时却必须依据历史数据的积累进行模型分析。

(2) 管理工作绩效

参与管理工作所付出的劳动对应获得的工作绩效。

主要包括：团队工作任务计划安排、配置库管理和维护、资源协调和配置、人员安排和整合，以及人员情况和心理调整等其他事务性工作所占用的工作时间对应的工作绩效。

可以初步定义，一次评审任务为每位参与者一个基础绩点，并以此工作量来计算相关任务的绩点。

(3) 配合工作绩效

配合的次数越多，配合完成的任务难度/工作量越大，可能得到的配合工作绩效越高。

也有可能出现由于某人的 interpersonal 关系很好，其他人都愿意和他配合，但其本人技术水平并不高，可是配合的次数很多，从而获得较高的收入。这种人的存在类似团队润滑剂的效果，只要你能做到这一点，让大家高效率地完成任务，那么你的存在也是合理的。

(4) 评审工作绩效

参加评审的次数越多，评审工作绩效就越高。由于这项工作的绩效是参与者平分的，因此参与者的数量也根据绩效数值而受到限制。如果评审工作中没有尽职，那么就可能因为后续发现的问题而给参与评审者带来惩罚，扣除相应的绩效并惩罚性扣除一些绩效数值。

(5) 代码库贡献度

一个人对公司代码库的付出和这种付出在当月被引用的次数以及相应的加权。

代码库主要是对代码的分类管理，要求每一位代码维护人员撰写维护文档和代码功能说明书，企业内部给以相应的支持和奖励。

通过配置管理软件获得支持，或者自行开发一个简单的检索管理工具获得支持都是可

行的。

对于代码库的构建和使用,这里举一个简单的例子:每一个人创建一个新的类,完成它的所有说明文档和使用手册的编写后,公司给予20元的奖励,每被他人调用一次,则奖励5~10元或者一个绩点。如果发现有人应使用公司代码库中的现成内容而没有使用,则进行每次每模块20元罚款或者其他处罚方式。维护代码库,给代码升级一次奖励5~10元。

具体的规则应该根据公司本身的情况进行定义和分析,不可盲目选择一个数值进行执行,否则,可能会由于不公平而引起技术人员的不满。

代码库的进一步扩展是公司知识库的内容,同样的方法也可以用于知识库体系的构建和奖励使用上,详细内容参见6.4节的内容。

6.3.6 绩效管理算法模型

1. 模型概述

关于绩效管理算法模型的基本形式从上面可以看到一些,不过,更具体的算法和实现必须根据各企业的不同进行构建,简单来说,它包括基本算法模型和调整优化算法。

前者将是持续的,但要随着公司的发展每年或者固定时间进行一些调整;后者则是短时期的,主要在整个模型实际应用的前2~3个月内进行应用,过期后,其作用逐渐减弱并被消除。

2. 应用结果

在每个月进行工资和奖金发放前,只需要根据所有技术人员的绩点进行数字计算即可。

如果一个产品/项目获益,公司决定对项目组成员进行奖励,此时,就可以用该产品/项目总绩点数来除每个人的绩点,这样就可以相对合理地得到每个人应该得到的奖励,而不会因为管理者与员工之间的亲疏关系造成他人的不满,避免了公司因此带来的不稳定因素。

6.3.7 小结

通过上面的方式,就可以将公司的技术人员划入到一个整体范畴内,而不会因为个人因素来影响到公司其他技术人员的心情和变动。

将人为的数字影响放在周工作任务安排上,而此后的一切内容都是用模型和数字表示,因此可以在最大程度上避免因为个人因素而产生相对不公平事件发生,也可以在很大程度上避免一些弄虚作假的行为。更有利于促进软件企业内部利益的合理分配。

6.4 实用性知识库体系构建

知识库体系的构建目前最贴近的工具就是配置管理工具,本节采用配置管理工具加上论坛评论方式进行介绍,具体的创建和使用方法可以根据技术实现形式为企业内的员工培训。

6.4.1 基本原则

知识库体系的建设需要考虑到软件公司的各个方面,从新人进入公司,到技术人员的成长,团队的搭建和管理,公司开发所需要的技术工具和资料,代码库的积累和扩展使用,等等。

可以说,一个实用的知识库可以帮助人员快速成长并适应公司文化和氛围,可以减少员工在外部网络上搜寻资料所花费的时间,可以体现公司的技术积累和核心技术的表现,可以真正加快项目开发的进度,实现最大范围可控资源的共享和使用。

因此,知识库的构建需要遵循以下两个基本原则:

- 有效的数据积累和审核;
- 有效的公开和关键信息的有限保护。

遵循这两个基本原则,一方面是为了提供有效的知识库内容,另一方面就是最大限度地利用知识库的内容为团队和技术人员服务。

6.4.2 内容分类

一个基本的知识库体系应该包括如下5个内容分类:

- 企业基本制度和新员工学习材料库;
- 各项技术分类的文档资料和学习材料库;
- 企业代码库;
- 各行业客户资料和业务资料库;
- 企业知识库管理创建规则。

下面我们针对这5个内容分类进行逐项阐述。

6.4.3 企业基本制度和新员工学习材料库

1. 创建负责人

企业基本制度和新员工学习材料库的创建应该由人力资源部的人负责,组织财务、人力、档案、技术等部门的人员进行提供,并收集整理。

2. 内容

内容包括:

(1) 企业各项基本管理制度

公司的基本管理制度主要是企业目前状态和常规管理信息等内容。帮助员工了解企业的纵向深度内容。

(2) 单独的薪酬体系和劳动保障内容

这一项内容应该写明企业内的薪酬体系,各级别的薪资待遇,以及各个级别如何帮助员工合理避税,最大化员工收益的各种模式。

劳动保障内容应该写明员工应该享受哪些国家规定的基本劳动保障,以及,在什么阶段除了基本保障以外有何种其他保障或者收益。

(3) 企业各部门的划分和分类目标

主要介绍企业内各个部门的职能和划分方式以及划分原则。当企业部门发生调整时,应该在一定时间范围内进行调整,有利于新进入的员工更多地了解企业内横向扩展内容,以便于在适当的时候与需要的部门之间进行配合操作,降低无效的时间浪费。

(4) 企业年度目标和未来发展方向

企业应该明确自己的目标,一个可实现的年内计划目标和一个未来3~5年内的发展目标,要有具体的描述,真实可信,可以夸张,但是不能没有。对于员工来说,他们也希望自己能职业发展和提升,如果企业没有发展,他们自然不能求得发展。企业也可能会遇到短期的困境而无法给员工短时间内的提升,但是,长远的共同利益加上合理的短期利益的结合,不仅可以稳定人心,也可以让大家心有所属,共同为了目标而激发自己的积极性。

3. 使用规则

任何人都可以查看企业基本制度和新员工学习材料库,但是只有创建人员才可以修改,其他人可以根据文档内容提供附加建议文档和评论,以便于后续的修改和调整。

6.4.4 技术分类文档和学习材料库

1. 创建负责人

所有的技术人员均可创建此类文档资料。

2. 内容

企业的核心技术人员确定技术资料的分类目录,所有的技术人员都可以在对应的分类目录下补充自己认为有价值的文档资料,通过审核后的资料会显示在资料库内供大家查看和学习使用。

采用配置库和目前的论坛形式组成评论和学习方式,每一个文档被阅读后都可以直接查阅相关评论和补充内容。

具体的分类可以根据企业的目的和企业的技术要求进行定义,可以去掉一些公司在短期内不需要的技术资料,着重保留有用的资料,也可以采用兼容并蓄无所不有的方式进行资料库的分类管理。

3. 使用规则

使用规则中包括两个准则:

- 创建准则:创建该技术资料时必须至少带有基本的介绍性资料,更多的扩展资料可以在后续的过程中逐渐补充。
- 维护准则:任何技术人员查阅某个资料后都可以选择对它进行补充修订,也可以针对原有资料中的错误进行修订。每次修订补充都必须记录修订的内容和位置,以便审核人员进行审核并评估此次维护的价值。

技术相关人员可以查看、修改、评论,非技术人员则不需要相关任何权限。

6.4.5 企业代码库

1. 创建负责人

参与编码和项目开发的人员均可创建。

2. 内容

企业代码库的内容包括:

(1) 代码分类

代码具体的分类位置,方便使用者查找。如果用户无法查询或者查询很困难,那么被引用的次数减少,使得代码库获得的奖励部分就会减少。

(2) 代码

全部按照公司规定封装好的可执行代码和相关的库文件形式。

(3) 代码简要说明文字

代码的输入输出和一些功能介绍,一般公司可以参考 JavaDoc 的内容进行设计。

(4) Demo 实例程序

实际调用这段代码实现功能的例子程序,可以让用户知道为什么要用,是否能达到目的。

3. 使用规则

所有编码和测试人员均可使用,指定可以参加审核的特定人员也可使用。

代码库补充形成的具体要求如下:

- 1) 完成具体代码的全部开发和测试(建议和别人合作)。
- 2) 完成一个调用的 Demo 例子程序,以便于其他人使用。
- 3) 提交(提交内容按照基础组件组中的要求进行)到配置库中的审核区,等待审核。

这个代码库在一次完成后可以由本人升级,也可以由其他人升级完成,每次升级后进行评审,确定参与人员的贡献度比例,以便于今后相关绩点或者奖励的分配。

代码的贡献度包括两个部分:创建/维护升级贡献度,复用贡献度(根据被复用的次数进行计算)。这是因为被评价为有价值的代码也可能因为公司的项目发展方向和某些情况,在10个月内没有被采用,这并不能说明这段代码没有价值,在合适的项目中,它就可能体现出其自身的价值。

企业代码库的完成时间是指,每个人可以在闲暇时间,或者任务间隙来完成,也可以利用任务时间内来完成,但是不会因为他要完成这个代码库的补充而延长任务时间或者增加人工计算。这是因为代码库的奖励是按照代码库的重复使用状况来计算的,能够做到一次完成,多次使用才是代码库的最终目的。

注意,这里的代码必须是有一定功能价值的代码,如果有人企图使用一些简单代码来进行糊弄,在审核过程中一般会被查出来,而且,简单代码也不会被其他程序员认可,只要被

举报,就可以取消该代码的内容和对应贡献度。代码创建贡献度的奖励一般考虑在代码创建1年后进行,如果是年底创建的,那就要到下一年的年底才能获得奖励,或者定义超过10个月以上的代码库中的新创建代码才可以获得奖励,并可以根据在此期间被他人调用的次数确定该代码的贡献度。

6.4.6 行业客户资料和业务资料库

这个类别的管理内容接近CRM系统,但是,其内容与常规CRM系统的管理内容略有差异,除了基本的用户联系方式和关联关系信息的保存外,更重要的是客户方提供的各种技术资料、行业文档和需求调研资料以及调研中留下的各种音频视频,等等。

1. 创建负责人

公司的管理人员、市场人员,以及进行需求调研和售后服务的与客户接触的技术人员。

2. 内容

需要保存的主要内容有:

1) 用户方联系人信息,包括该联系人的基本个人信息,主要是职务、部门、所属上下级关系,与公司内派遣人员的接触以及有私人关系的公司内人员信息。

2) 用户方其他人员信息(这是个多项明细),其内容与上一条相似。

3) 用户方提供的行业资料,根据行业类别和项目类别进行多维关联存放。

4) 公司方进行某项目业务/需求调研过程中形成的行业业务文档,调研中记录的各种音频/视频资料,调研记录信息等。

以上内容可能根据行业 and 业务的不同,各公司的情况不同需要进行一些调整和补充,因此具体条目内容会略有不同。

3. 使用规则

除了可以参与创建的人员类别外,需要参与系统分析和设计开发的技术人员也有权力进行自己业务范围内相关客户资料和业务资料的查询使用,并可以根据情况提出自己的意见和建议供需求/业务人员进行再次确认和对具体内容的补充。

行业客户和业务资料库的升级规则也可以参考企业代码库的规则来使用,这里就不再做更多的赘述了。

6.4.7 企业知识库管理创建规则

企业知识库建立过程中会经历如下3个阶段:

- 大量创建,无人使用;
- 初创成型,少人使用;
- 不断补充,人人皆用。

这3个阶段体现了一种新生事物的出现、成长和发展过程。

创建之初,在开发和管理过程中还没有什么东西可供使用,因此就不会有人从该企业知识库中获取使用内容。

初创成型,当用户开始使用的时候,会因为规则不熟悉,或者不习惯,而不愿意使用企业知识库内提供的内容。同时,如果未能很好地执行规则中约定的审核机制,使得知识库中的内容存在大量无用的垃圾资料,在企业知识库创建一段时间后,仍没有见到效益,不仅员工会产生抱怨,管理者也会因此产生不满。

但是,只要有力地推动审核机制提高企业知识库中内容的有效性,迟早会跨越这个阶段而进入最后一个阶段,也就是企业知识库的良性成长阶段。

在良性生长阶段,企业知识库可以不断获得补充,员工开始大量使用知识库内容来补充自己提高开发效率,又不断从自己的开发中总结经验形成有效的内容来对企业知识库中的内容进行升级补充。

这3个阶段的执行持续多年后,这个时间有可能是5年甚至10年以上,到那时,国内甚至国外软件行业就会发生一次大的技术革命,随着知识库中代码库的形成,真正的MDA(模型驱动开发)才会形成,而不是以前噱头宣传的那种形式。

6.4.8 小结

企业知识库的创建有利于企业的长远发展,软件企业积累的不仅是为客户提供的业务系统,还可以不断获得内部的技术积累,这种技术积累就是可复用的软件代码形成的代码库内容。

一旦代码库形成,即使发生大的技术人员变动,公司自身保存的代码库也可以让后来的技术人员很快熟悉原来系统中所使用的大量代码,以便进行产品的升级和更新。很多时候,对代码库中代码的升级也同样会触发用户业务系统新版本的升级。

代码库的产生也将促使软件行业的开发方式发生大的转变,大量的软件工厂将被削减,更多的软件系统都可以通过拖曳方式和模型工具直接关联代码而完成,只需要高级的技术人员对模型进行修订,完成逻辑层面的开发即可。

代码的编写负荷会大大减轻,最后只剩下两个部分的工作:对代码库中没有的代码的开发,以及现有代码库中代码对当前业务系统的适应性修订方式的开发。

微代码的编写对程序员的技能水平要求较高,不可能交于代码工厂去完成,而对代码库中代码的业务适应性修订开发则是与现有外包开发内容不同的开发工作,这时外包主要存在的领域是这个方面的开发和大量的测试外包工作。

第 7 章

软件开发中的心态对抗问题研究

软件开发中的心态对抗问题如果提升到一定层面那就是心理学问题。众所周知，著名的咨询大师温伯格曾经写过一本名为《程序开发心理学》的书籍，书中着重描述了程序员在遇到各种情况时的心理和行为分析。在 china-pub 上也有人评价“慕名去书店拜读，也许是我希望寄托的太大，也许是受限于本人层次，反正我大概翻了一遍，并没有想象中的感动。我倒是觉得作者说的很对：我只是在不停地思考。这本书中有很多东西都是自己可以悟出来的，还有一些是别的书上提出过，或类似的东西，只要深入思考并不难得出结论。而这些东西是“学”不来的，与一个人的世界观，人生观都有关系，甚至这样的书，我认为倒更像是一种个人思想的纪实，只能了解，学不得。……在水木社区上也有朋友提到，温伯格的夫人是位心理学家，他的很多心理学的支持来源于此，他本人也认为他的《程序开发心理学》一书并不够完善。但是无论如何，温伯格都在一个相当高的层面上给我们提供了一种程序员心态的较为全面的描述，我们在工作过程中经常会感受到与之相应的心理状态，读完却心中空旷，总感觉很多地方言之不足，吾辈心有戚戚而空之。

于是笔者产生了想要撰写一些扩展性文字的想法，本章的 7.1 节和 7.2 节的内容曾经在 IT168 上首发，后来在水木社区 Thinker 板块被置底良久，由此可见人们对这类文字的总结和分析

的渴求。在本书写作过程中,笔者曾多次与出版社沟通,然后决定将相关文章置于其中,虽不完善,但渴求获得抛砖之效,能引起国内开发团队对心理问题的关注和对人性化开发方式的关注。

下面两节的内容是在已发表的文章基础上进行了重新的修订,以期使其融入到本书中,并达到更好的效果。

第三章 心态对抗

心态对抗是指软件开发人员在开发过程中,由于受到各种因素的影响,产生的一种消极、抵触、抗拒的心理状态。这种心态对抗的表现形式多种多样,如:对开发任务缺乏热情、对开发进度感到焦虑、对开发结果感到不满、对开发团队感到不满等。心态对抗的产生,往往是由于开发人员对开发工作的认识、态度、能力、资源等方面存在不足,或者受到外部环境的影响,如:工作压力大、时间紧张、资源匮乏等。心态对抗的存在,会对开发工作的顺利进行产生严重的负面影响,如:降低开发效率、影响开发质量、增加开发成本等。因此,了解和认识心态对抗,并采取有效的措施进行预防和解决,对于提高软件开发的质量和效率具有重要意义。

心态对抗的产生,往往是由于开发人员对开发工作的认识、态度、能力、资源等方面存在不足,或者受到外部环境的影响,如:工作压力大、时间紧张、资源匮乏等。心态对抗的存在,会对开发工作的顺利进行产生严重的负面影响,如:降低开发效率、影响开发质量、增加开发成本等。因此,了解和认识心态对抗,并采取有效的措施进行预防和解决,对于提高软件开发的质量和效率具有重要意义。

心态对抗的产生,往往是由于开发人员对开发工作的认识、态度、能力、资源等方面存在不足,或者受到外部环境的影响,如:工作压力大、时间紧张、资源匮乏等。心态对抗的存在,会对开发工作的顺利进行产生严重的负面影响,如:降低开发效率、影响开发质量、增加开发成本等。因此,了解和认识心态对抗,并采取有效的措施进行预防和解决,对于提高软件开发的质量和效率具有重要意义。

心态对抗的产生,往往是由于开发人员对开发工作的认识、态度、能力、资源等方面存在不足,或者受到外部环境的影响,如:工作压力大、时间紧张、资源匮乏等。心态对抗的存在,会对开发工作的顺利进行产生严重的负面影响,如:降低开发效率、影响开发质量、增加开发成本等。因此,了解和认识心态对抗,并采取有效的措施进行预防和解决,对于提高软件开发的质量和效率具有重要意义。

心态对抗的产生,往往是由于开发人员对开发工作的认识、态度、能力、资源等方面存在不足,或者受到外部环境的影响,如:工作压力大、时间紧张、资源匮乏等。心态对抗的存在,会对开发工作的顺利进行产生严重的负面影响,如:降低开发效率、影响开发质量、增加开发成本等。因此,了解和认识心态对抗,并采取有效的措施进行预防和解决,对于提高软件开发的质量和效率具有重要意义。

7.1 项目管理者心态问题研究

这个话题是针对项目经理而提出的，本文的定位在于给项目管理者一个心理分析的启示，该启示有助于管理者做好项目管理工作，协调好上层与技术人员的关系，同时让所有的开发人员具有更高的工作热忱。

关于项目管理者分类请参看 7.4 节中的内容。

7.1.1 引言

项目管理在当前已经是一个被所有人都了解并熟知的概念，不过，当人们提到项目管理时，大多是指计划、管理、人员、资源、任务等的管理，而从来没有人提到过心理这个词。相信很多人都看过温伯格的《程序开发心理学》，也许很多人都认为程序开发心理学到这里也就足够了。其实，温伯格的《程序开发心理学》一书只写了程序开发中的一部分，即程序员心理学，而实际上，程序开发心理学或者推而广之的软件项目开发心理学中应该至少包括投资者心理学和客户心理学这两个方面。

有人会问：客户的确和软件项目关系很密切，但是投资者和软件项目似乎关系不大，所以应该不考虑投资者的心理。这里所指的投资者与传统意义上的投资者有所不同，其定义取决于环境。例如在公司内，你的高层管理层就是这个项目的投资者，他们虽然是从客户那里拿到钱来做这个项目，但实际上，他们也在衡量拿到的钱和做这个项目的开销之间的比例关系问题，如果不合适，他们也会中止这个项目的。另一种情况是，找真正的投资者，或者称之为风险投资者或天使投资者来投资你的软件产品，这时投资者的心理状态也同样会对你的产品开发过程产生影响。所以，投资者心理学也是必须研究的一个课题。

好了，到这里，我们谈到了关于心理学的问题，但心态对抗的话题好像还没有引入。下面我们一步一步分析。

上面提到投资者心理学、客户心理学和程序员心理学 3 个方面，而对于项目管理者或者项目经理也有他本人的项目管理心态，或者称之为项目管理者心理学。这 4 个心理学组成了完整的程序开发心理学。

上述 4 种人的 4 种心理状态会因为软件项目的存在而相互碰撞，这样就必然会产生心态对抗的问题。这种心态对抗如果处理不好，可能会让本应进展很顺利的项目面临危境，而如果处理得好，也许能让濒临绝境的项目起死回生。因此，心态对抗或者说心态分析也就成了这个方面的重点研究内容了。

鉴于本文的篇幅所限，我们不可能将这 4 种心理状态做非常完整的分析，只能有针对性地做一些相关分析。由于本节主要是讨论项目管理的心态问题，所以项目团队的管理就是其首要任务，下面我们主要进行项目管理者心理和程序员心理之间的对抗分析。

7.1.2 角色定位

在基本的项目团队中，主要存在两种角色：项目管理者 and 程序员。

1. 项目管理者

项目管理者一般称为项目经理。对于程序员这类技术人员来说，他们一般认为项目经理应该是一种技术角色，而不是单纯的管理角色。而在常规定义中，项目经理都是管理角色，而不是技术角色。这种角色定义的差别使得程序员与项目经理之间的冲突变得非常激烈。

中国的传统观念是，学而优则仕。在软件行业中，所谓学而优即指你的技术水平比别人高。而学而优则仕的传统观念又使得技术人员先天的技术崇上意识被强化了数倍，于是非技术人员来做项目经理的难度就大大增加了。另外，很多在中小公司的项目经理其本人的项目经验较少，因此就难以服众。我们经常可以看到类似这样的问题：

1) 如果有3个人负责一个小项目，你是组长，你应该做什么？另外两个人总是意见不统一，你该怎么办？

2) 如果一个团队中程序员的素质都较差，该如何管理呢？

例如有如下4个人：

- 满脑子装的都是私人感情问题，虽然，白天工作还是很认真负责的，但是一点也不思进取，得过且过。
- 对待工作很积极，具有唐僧精神，不达目的绝不罢休。但是有健忘症，很容易忘事，即使反复强调的事情也会忘记，而且从不按需求来做事，只凭自己的心情，不管客户的要求。
- 心理年龄很小，做人做事都像小孩，说话做事都不计后果。对待工作还是很认真负责的，虽然有怨言，但能完全按照需求来做事。缺点是极端不思进取，有平平淡淡才是真的人生观。
- 什么事都不想做，认为每天只要人到公司，就可以直接拿工资了，别人叫他做事，他不停地抱怨。如果与他私人关系较好的同事将道理向他讲明白，他还是会全力以赴工作的（我曾给他通读《人月神话》）。

这样的团队，导致我们公司每个项目都会拖延至少3倍的时间，老板很生气。

上面这些都是项目经理对程序员可能产生的一些常见看法。关于为什么产生这样的看法，以及如何解决这些问题，我们在后面进行详细分析。

项目经理的一个主要职责就是进行计划管理并按照计划协调项目进展。但是在软件项目中程序员往往既是设计者，又是实施者，因此在对项目进度的把握中如果项目经理自身经验不足，那么他不可能比程序员对这个项目的计划进度把握得更准确。即使他有这样的经验，

但在自己不参与开发时做这样的估算在时间进度上都会存在偏差。如果估算的时间偏短或者用自己已往的开发时间来规划普通程序员的开发时间,是不切实际的,于是就使得项目的估算工作量偏小,最后带来的是程序员和项目经理之间的不和谐关系。

2. 程序员

软件项目中程序员是主体,软件项目各个阶段的工作都是由程序员来完成的。国内的程序员以硕士、本科或大专毕业生为主,绝大部分人都具有较高的素质和文化修养,有较强的个人自主性,而且很多人都是因为兴趣才走上这条职业道路的,所以,这些人的自尊心和信心都是需要得到尊重的。

由于写程序本身是项具有创造性的工作,这与科学研究有着很强的相似度,所以,才有如此多的人投入到了软件行业中,其中不乏具有很优秀的人才,这些人也逐渐成为了这个群体的顶尖力量。

程序员由于自身具有较高的素质,他们对自己的看法也有较强的执著性,因此要轻易改变这些人的想法和观点是相当困难的。

程序员对自己的职业诉求因为上面的基础,更期望能够在有一定保障的条件下进行新的技术探索和新产品的开发,尤其是做一些其他人没有做过的项目或者没有解决的问题。

当然,从事程序员这个职业的人员也是有一定的层次划分的,因此这里面也有一些人是贪图享受的,只希望借着程序员的名声获得一些看上去似乎相对较高的收入,而进入以后才发现,完全不是自己想象的样子,于是就开始自甘平庸或者很快转行。

3. 周边角色分析

周边角色,主要是指客户和投资者。

对于客户来说,主要关心的是最终产品的质量优劣,是否既能提高效率又能节省人力,另外就是调研需求的人是否合他们的心意。

对于公司来说,所关心的是收支问题及团队的稳定性。比如,有人提出这样的问题:请问怎么能够使公司的技术实力不随技术人员的变动而变动?

例如,一个公司有6名开发人员,如果有3人辞职,然后又新招进3人,那么公司的技术实力就会发生很大的变化。如果6个人全部换掉,公司的技术力量将就由新来的6人组成。如何能使公司的技术实力不随技术人员的变动而变动呢?或者说尽可能地减小影响。

山东一家公司的总经理也问过笔者同样的问题,他甚至对笔者说:好不容易把人培养出来了,结果个个都敢和他叫板,让他每天觉都睡不好,天天都在担心,人走了怎么办?他说,虽然有备份的技术人员,但他们大部分年龄偏大,而且从事着管理工作(后面的话他没有多说,话外之音,即这些人员备份也未必可用)。

7.1.3 心态对抗解析

为了说明这种心态对抗是如何产生的,在前文心态描述的基础上,下面我们看一下这些心态的分析和对抗解析。

1. 项目管理者与程序员

项目管理者经常遇到的问题如下:

- 1) 项目周期紧,必须尽快完成,如何促使程序员能够积极高效地完成开发任务。
- 2) 工资低,程序员积极性不高,不配合工作。
- 3) 虽然有一定的技术经验,但是对目前的项目不想插手,原因是:怕不能按要求完成工作,或者自己不想再写代码,或者不想将自己的技术水平表现出来。
- 4) 程序员懒惰,不想做事情。
- 5) 程序员经常丢三落四,不按要求进行开发。

还有很多,这里就不一一列举。

概括起来主要有下面几个问题:

- 1) 程序员待遇低(对应上面的2)。
- 2) 对任务的分析与程序员之间有分歧(对应上面的3)。
- 3) 程序员不配合工作或者积极性不高(对应上面的2、3、4、5)。

项目管理者由于职责要求,必须保证项目按期保质保量地完成,而在实际工作中,约90%的项目都会延期(参看《死亡之旅》),其原因不仅限于上面几种,但本节主要是分析项目管理者与程序员之间的关系,所以外部关系和原因暂不考虑。

对于第一个问题,这是涉及公司整体管理的一个问题。当然,众所周知,国内程序员工资收入水平是相对较低的,这个问题的解决往往是项目管理者没有权力做到的。对于项目管理者而言,你可以做到的是根据团队中程序员的表现向高层管理者反映相关情况,并获得高层管理者的理解,至少在项目周期内可以得到一定数量的月度奖金分配权利(这一点是很多公司在项目进行中都会采取的手段,即使老板很自私,他也会考虑到项目的进展和结项问题而在这方面做出适当的让步,并给予项目经理相应的权力)。而对于国内绝大多数的中小公司,只要项目管理者真的为团队内的程序员进行了争取,所有的程序员都会看到,并会感激你,至少在你可控的项目进程中,程序员会相对主动地配合你的工作,而不是完全不配合或者完全消极怠工。

对于程序员而言,无论公司多大,基本上真实发生的事情大家都会听到并有所耳闻,他们如果看到项目管理者是这样在为他们争取利益,保证他们的利益,任何人,不仅仅是程序员都会因为对这个项目管理者有一种报恩的态度而主动地完成一些自己份额内的事情。

对于第二个问题,这种分歧有可能会扩大化,也有可能消弭于无形,关键在于项目管理者如何协调与控制。一般情况下,大多数程序员会比较坚持自己的或者认同的观点,而不会

轻易改变自己的观点去附和他人的观点。项目管理者应该考虑到这一点,进行适当地让步,让氛围轻松下来,甚至可以考虑让程序员在一定的范围内进行自己所认同的实践,这样既体现了对程序员本人意见的尊重,同时也表现出了项目管理者的大度,而且,如果这个程序员的看法是正确的,也可以在一定程度上避免项目走入歧途。相反,如果项目管理者固执己见(因为不少项目管理者本人也是或者也曾是相对优秀的程序员),那么这种对抗就会因为无法认证而持续下去,既浪费了时间(讨论的时间大大延长),也耽误了正常的项目进度。

对于第三个问题,程序员的不配合或者积极性差的时候,项目管理者更需要详细地了解程序员本人或者包括他最近的行为以及家庭的情况,看看问题的关节点是在公司内部还是在公司外部。如果问题的关节点在公司外部,那不妨作为项目管理者或者程序员的领导,请这个程序员出去吃顿便饭,聊聊天,进行一下贴心地分析,让他感受到你对他的关心。而如果这个问题在公司内部,即使是看上去有些不合理的要求,也可以将问题进行记录并向高层管理者反馈并争取能够做到解决或者局部解决。相反,即使问题最后并没有解决,程序员看到你的确在真心实意地帮助他解决问题,帮他考虑事情,他对你的感激也不会少于问题全部解决,至少在你的项目周期内,他不会轻易地考虑辞职或者不负责任的工作。

对于项目管理者而言,无论你曾经或者现在的技术水平多么高,只要你现在的位置不是在做这个模块的开发,那你就需要考虑一下协调工作和分析心态的事情。程序员不会主动来分析你的心态替你考虑,但是他会想你这个领导是不是真的在帮他考虑,如果是,他一定会感激你并配合你,如果不是,他也会反抗你。

常言道:站着说话不腰疼!这句话是说,不做事情的人都是站着说话的,干活的人都是弯着腰的,当然你如果不弯腰,你是不会知道腰疼的。当然你的计划和考虑与已经干了很长时间并感到疲乏的人的做法所不同时,项目管理者做法就会体现出他本人的管理水平了。

2. 项目管理者与周边角色

分析完项目管理者与程序员的关系后,下面就说一下项目管理者是如何与周边角色进行协调的。

首先我们来看项目管理者与公司高层管理人员间的心态解析。

项目管理者了解到程序员的要求后,必须向高层管理人员反映,并期望得到高层管理人员的认同,以便于获得相应的技术和资源的支持,这样才能真正有效地解决程序员的问题及软件项目开发中遇到的问题。

对于高层管理人员来说,节约开支,留住有用人才,尽快完成项目,保证合同资金能够及时收回,这些都是很重要的事情。而节约开支与留住需要的技术人员之间存在冲突,但是如果留不住这些有用人员,那么项目就不可能按期完成,合同资金也就无法及时收回。

通常,高层管理人员不会直接面对技术人员,所以,他们并不清楚哪些技术人员必须留下,因此,项目管理者应该向高层管理人员说明自己为什么要这样做,为什么需要这些额外的奖金或者为什么需要给这些技术人员提高工资。

如果项目管理者不能很好地与高层管理人员进行沟通,高层管理人员一定会拒绝项目管

理者的要求,并且对项目管理者后续的要求也会产生怀疑,势必造成程序员心态的不稳定性,同时给正在进行的项目带来内部的风险。

由此可见,项目经理应该具有很强的逻辑思维和推理能力,同时还要具有很好的语言表达能力(关于技术能力不是这一节需要描述的内容,因此不在这里说明)。很强的逻辑思维和推理能力,这一点是几乎所有优秀的程序员都具有的基本素质之一,但是,绝大多数技术人员因为很少与外界接触,尤其是优秀的技术人员因为长时间面对机器而不是面对人,所以,语言表达能力都不是很强,甚至很多人的语言表达能力非常差。可以说,让一个优秀的技术人员写一段异常高难的代码,比让他去说服一个小学生都容易,甚至很多程序员根本就不愿意开口和别人说话。所以,中国传统的学而优则仕的观点在软件项目管理者的选拔中是需要特别小心的。IBM的非技术项目经理的定义是笔者比较推崇的一种项目管理模式,这属于笔者定义中的弱势项目经理的角色,主要是承担协调任务,而不是强调管理。因为在中国,所有的人都喜欢这个“管”字,因为管下面是“官”字,当官,才能发财,才能扬名,才能光宗耀祖,所以,如果一个公司的高层管理者能够选拔出能言善辩的优秀技术人员那是最好,这就是笔者项目管理定义中的强势项目经理。如果做不到这一点,那么,不妨把你们的项目管理角色分开,一个专门做项目协调工作的项目经理,另外是一个负责技术和实现把握的技术经理,这样既能避免程序员本身的弱点,同时也能发挥程序员本身的优点。这个项目协调人员担任的项目经理可以是一个市场人员,也可以是一个管理人员,甚至可以考虑用能言善辩的美女(程序员大部分都是男性角色)来担任。

下面我们来说一下项目管理者与客户之间的心态对抗解析。

前面提到,弱势项目经理的主要任务就是协调工作,除了项目内部和公司内部的协调工作外,还需要协调与客户间的关系,甚至要辅助参与商务谈判。

项目经理在安排到客户方进行调研工作的技术人员之前,一定要到客户现场进行前期调查或者称之为观察。仔细考查客户方的人员情况和工作环境。比如说客户方大多数人员的年龄偏长,那么就需要安排一名很稳重的程序员去进行需求调研,而如果客户方大都是年轻人,就需要考虑安排一名活跃些的程序员去进行需求调研。当然安排的这些程序员都必须懂得日常礼仪,懂得如何谦虚地与客户讨论问题,同时具有相当的技术水平,不拒绝琐碎的工作,不是去指手画脚地指责客户的行为或者错误的判断。

这里有一个2005年发生在中科院的例子:

2005年底一名研究员(海归)到深圳某国家机关单位调研并进行设备的初步安装调试,在此期间,他居然直接指责对方的总工程师。在查看设备状况时,还曾指着那位总工程师说,“你,把那个螺丝刀给我递过来”(说这句话的时候,他身边就站有两名他从北京带到深圳的技术人员),对方虽然碍于面子按照他的话做了,但因为他的这种行为让对方极为反感,在他返回北京后,对方就要求立刻撤掉所有设备,如果不是相关市场人员进行协调以及和高层管理者的介入,这个项目险些就此终止。



3. 程序员与周边角色

程序员因为其工作的特殊性很少与外界打交道，在公司内部也主要与项目管理者以及相关的技术人员在一起。关于高层管理者如何留下公司需要的技术人员的问题，也就不在这里讨论了，后续章节中会进行分析。

程序员与客户打交道的过程属于需求调研工作的内容，或者说带有部分市场工作性质的工作活动，限于篇幅也就不再继续讨论了。

7.1.4 小结

本节从4种角色进行了软件项目开发过程中的心态对抗分析，同时着重对项目管理者和程序员进行了深入的心理分析。一个团队是否可以顺利地完成任务，在不考虑任何公司外部条件变化的基础上，一方面要看团队是否有相应的技术实力，另一方面取决于项目管理者如何在项目进行过程中留下技术人员并激发技术人员的工作热忱，否则，这个项目必然延期甚至死亡。

7.2 高层管理者心态问题研究

这个话题是针对企业高层管理者而提出的，目的在于给企业高层管理者一个心理分析的启示，这个启示应该有助于你做好项目管理工作，让你的项目进展得更顺利，让你的项目管理者不至于过于为难而无法开展他的工作，同时让所有的开发人员具有更高的工作热忱。

7.2.1 引言

软件开发心理学包括投资者心理学、客户心理学、项目管理者心理学和程序员心理学4个部分。4种人的4种心理状态会因为软件项目的存在而相互碰撞，这样就必然会产生心态对抗的问题。这种心态对抗如果处理不好，可能会让本应进展很顺利的项目面临危境，而如果处理得好，也许能让濒临绝境的项目起死回生。因此，心态对抗或者说心态分析也就成了这个方面的重点研究内容了。

对于企业内部的项目而言，企业的高层管理者就是以一种投资者的身份在参与项目，如果项目失败，他的投资全部被浪费，如果成功，他的投资就可以从客户那里获得回报，因此高层管理者心理学是投资者心理学的一个重要组成部分。或者说投资者心理学分为两个部分：高层管理者心理学和投资人心理学。对于投资人来说，只关注结果，通过结果和过程中的阶段点对所任用的企业高层管理者评判审核，而企业高层管理者则要在过程中关注项目进展情况并选择项目经理，还要与最终客户进行沟通交流，以期完成公司的终极目标：企业持续盈利。

鉴于本文的篇幅所限，这里不可能将这4种心理状态做非常完整的分析，在7.1节中我们已经有针对性地进行了项目管理者/程序员的心态分析和项目管理者/程序员之间以及他们

与周边角色之间的心态对抗分析,本节将针对企业高层管理者进行一些心态分析和研究。由于是针对企业高层管理者的,那么企业盈利就是其首要任务。为了企业能够盈利,他必须保证项目能及时、高效地完成,在客户那里他可以完成本次合同/订单的任务,甚至得到更多的合同/订单;如果处理不好,发生问题就可能失去这次合同/订单,所以这里我们主要做企业高层管理者心理与项目管理者/程序员心理之间的对抗分析。

范围限定:主要针对软件研发公司,对于销售公司和其他类型的软件相关企业这里不考虑兼容性问题,可以借鉴,但是不要照搬,因照搬产生的相关责任请自行承担。

7.2.2 角色定位

在一个公司内部有如下角色之分:企业高层管理者、项目管理者、程序员、其他人员。公司的外围主要是公司的客户。

1. 企业高层管理者

正如前文所说,企业高层管理者的终极目标是:保证企业持续的盈利。

为了这个目标,企业高层管理者必须做到留住必要的员工,减少不必要的开销。其实,留住必要的员工要比减少不必要的开销相对容易,但是,很多企业的高层管理者却为此伤透脑筋,他们反而认为减少不必要的开销更容易。下面将以实例进行说明。

(1) 案例介绍

公司背景:

到2005年为止一家成立了5年的公司,前身是一家国内做通信的老牌公司,目前核心成员5人都是股东,他们中大部分都是技术人员出身。目前主要市场立足于通信、电力、安防行业。笔者的一位朋友是5位股东之一,目前主要负责市场。

事件说明:

有一次笔者的那位朋友在与笔者聊天时,提到一个问题:如何留人。

他说到(下面都是他原话的真实意思,但并非完全一样,毕竟笔者没有那么强的记忆力),他们公司一部分人做软件平台开发,一部分人做硬件开发,而做软件平台开发的人随时都可以更换,因为这些人很容易重新招聘。而做硬件开发的人,在山东很难找到,花费3年时间培养一个人,结果,这个人前几天(2006年的某一天)再次向他提出辞职,他立刻从深圳回到山东。利用周六休息时间,打电话约这名员工出来吃饭,闲聊,谈生活,聊工作。最后问他有什么想法时,那个人却说不走了,继续干。他才放下心来。但是,这个人已经是第二次提出辞职,这让他很担心,万一哪天这个人真得走了,怎么办?他做的产品就没人做了,他的工作没有人能够接替。

朋友还说,虽然他那里还有一些备用技术人员,但是,这些人员大部分年龄偏大,而且目前都从事管理工作(后面的话他没有多说,话外之意就是,这些人员能否还胜任这份工作)。

后来他的公司推出了措施,凡是在公司工作满多少年,级别达到某种程度的技术人员,一律可以向公司申请10万元无息贷款来买房,今后每个月在工资中扣除相应的贷款按揭数量。另外,还准备为员工提供同类的买车或者其他服务,但是,要求在按揭期限内,不得离开公司。通过这种手段,使一些技术人员稳定下来。

笔者的朋友还是一直忧心忡忡,总觉得这样还不够,但也找不到更好的解决办法。笔者向他提供了一些解决方式,不过有些观点他并不是很赞同,不管怎么说,至少对于技术人员来说,笔者的方法应该是足够贴心的,但对于管理层来说,是否能够得到认可还需要一些时来进行磨合。具体的建议将在后文中逐步介绍。

这里仍然用7.1.2小节中的例子:

一个公司有6名开发人员,如果有3人辞职,然后又新招进3人,那么公司的技术实力就会发生很大的变化。如果6个人全部换掉,公司的技术力量将由新来的6人组成。如何能使公司的技术实力不随技术人员的变动而变动呢?或者说尽可能地减小影响。

(2) 分析

上述两个例子都表达出了一个观点:高层管理者为了公司的利益一定要让整体技术力量能够稳定并发展,而不至于后退。同时,能够通过公司的技术力量及时地为客户供货,完成客户方的合同和订单,以此来保证公司的收入和整体利益。

这其实是一个关于企业高层管理者与员工之间的心理对抗问题。对于企业高层管理者来说,必须了解员工的想法,这样才能有针对性地提出自己的对策,才能留住自己需要的员工,当然,同时也会送走自己不需要的员工。

其实,真正的解决之道应该是将时间消弭于无形,在事态发生之间就将事情解决,这才是最高的管理哲学或者可以称之为管理上的王道。要做到这一点实在太难了,后面我们还会举出一些2007年前后发生的相关的例子,也将针对这些例子提出企业高层管理者的具体考虑和做法的建议。

2. 项目管理者 and 程序员

这两个角色的定位分析请参考7.1.3小节中对应的分析内容。

3. 周边角色分析

周边角色,主要指客户和公司的投资者。

对于客户来说,主要关心的是生产出来的产品是否好用,是否既能提高效率又能节省人力。

公司的投资者只会关心公司是否盈利,当公司遇到严重问题时,会全力帮助公司解决具体的问题。毕竟投资者的利益直接与公司的存在与发展紧密相连。

7.2.3 心态对抗解析

为了说明这种心态对抗是如何产生的,前面我们对每一种角色在项目开发中所产生的心态问题进行了描述,下面我们来看一下对这些心态的分析和对抗解析。

1. 高层管理者与项目管理者

要讨论高层管理者与项目管理者之间的关系,首先要了解程序员的心态,只有这样才能让项目管理者理清自己的思路,才能让高层管理者明白项目管理者为什么会如此思考问题。关于程序员的心态问题请参看7.1.2小节和7.1.3小节的相关内容。

在项目进行中,项目管理者经常遇到的问题如下:

- 1) 团队内部分或者全部程序员待遇偏低,领导不给涨工资。
- 2) 对任务的分析与程序员之间有分歧。
- 3) 程序员不配合自己的工作,或者他们的工作热忱不高。
- 4) 在其位而无实权,没有基本的奖惩权力,无法对程序员的积极工作进行表彰或者错误进行惩罚。
- 5) 担心主要技术人员在项目进行中发生变动。

在项目进行中,高层管理者除了正常的管理工作外,还会对如下问题进行考虑:

- 1) 项目进展是否顺利,能否按期完成。
- 2) 产品按期交付,能否保证运行质量,让用户满意,尽快付款。
- 3) 项目成本是否能按预计那样而不超出,或者更多地节约。
- 4) 项目经理不要在团队中拉拢帮派,架空公司,牟取私利。

总之,项目经理是从项目层面上考虑团队/项目的管理问题,而企业高层管理者则是从企业盈利的角度来考虑项目管理问题的。

由此可见,两者的想法之间有不少冲突,对比如下:

- 1) 项目管理者1和高层管理者的3。
- 2) 项目管理者3和高层管理者的1、2、3。
- 3) 项目管理者4和高层管理者的3、4。
- 4) 项目管理者5和高层管理者的1、2、3。

项目管理者由于职责要求,必须保证项目按期保质保量地完成,而在实际情况中,几乎90%以上的项目都会延期(参看《死亡之旅》),其原因不仅限于上面几种情况。除了在7.1.3小节中已经分析过的项目管理者与程序员之间的关系,本小节将针对上述几个问题重点分析高层管理者与项目经理之间的关系,其他外部关系和原因暂不考虑。

对于第1个问题,涉及公司整体管理问题,同时还会影响到项目的成本,所以,直接和高层管理者关注的第3个问题相冲突。程序员工资收入水平相对较低,不是项目管理者权力范围所能解决的。对于一名优秀的项目管理者而言,他能做到的是根据团队中程序员的表现向高层管理者申请,至少在项目周期内争取一定数量的月度奖金分配权利。但是,在提出这

种申请的同时又会触发高层管理者担心的第4个问题的产生,因此,项目经理往往很难得到这样的支持。这就是冲突之一。

其实从正常的心态分析,一名优秀的项目管理者不会背离公司来做一些事情。对于每一位程序员来说,他们很清楚,即使项目管理者积极为他们争取利益,但如果没有公司高层的理解和体谅,他们也不可能拿到这些奖金。

在团队内部如果只是部分程序员收入相对偏低,那还可以理解,对于项目管理者来说这方面的压力并不是很大。而如果普遍偏低,那带来的后果是,虽然因为人的适应性可以调控,但是,却不会因此让人保持沉默。可以说,在普遍的低收入状况下,只要有机会,人们都会去选择新的机会,哪怕新的机会并不一定有什么新的好处或更好的未来,至少对于任何一个人来说,都会憧憬新的机遇能够带来相对较好的未来。

对于那种不负责任的项目管理者来说,可能会带着团队在自己承接的项目过程中跳槽离开。而一名优秀的项目管理者永远不会劝说自己团队内的技术人员在项目进行中辞职。但是,他会劝说得不到公平待遇的技术人员在项目结束后辞职。这并不是说他对公司不忠,而是他,作为一名优秀的项目管理者应该做到的对上和对下都负责的处世方法。公司的高层管理者必须体会到这一点,并且体谅该项目管理者的做法。因为他的这种做法才能在最大限度内保证公司当前进展的项目得到最高效的完成,最大限度范围内的降低公司的开发成本。

如果是公司的运营和经费出现问题,那么高层管理者应该将这些问题向项目经理表述清楚;你既然敢把一个项目组交给他来管理,就应该将公司的危机问题也告诉对方。要知道,一个项目的失败可能会比公司财政危机的现状泄露更危险,如果你不能信任这个项目管理者,那么你不应该把团队交给他来管理了。

对于第2个问题,则和高层管理者无关,只和程序员有关,这个问题的讨论请参看7.1.3小节的内容,这里不再叙述。需要说明的是,如果这个问题项目管理者都不能合理地解决好,那么高层管理者似乎应该反省一下自己,是不是选错人了。

对于第3个问题,程序员不配合工作,或者他们的工作热忱不高都将会导致项目延期完成,整个项目进度的拖延,会带来交付日期的延后,那么根据合同要求,客户方也会延缓相关合同款项的支付。这种延期必然造成项目成本上升,客户的满意度下降和合同履行延期等问题。

对于这个问题上,高层管理者一般都会支持项目管理者的做法,来促进团队内程序员的工作积极性。出现这种问题原因有多个方面,如果是技术方面的问题,项目管理者必须有能力来解决或者协调解决,而对于非技术方面的问题我们会在相关部分进行讨论,这里就不再继续了。

这种情况下除了正常的解决途径外,如果是因为长时间开发任务使得技术人员过于疲劳而产生了工作任务延期的问题,可以考虑笔者下面这种方式:

在开发进度遇到延迟的时候,可以考虑给所有的开发人员更换一种开发环境。因为,当

一个人到达一种新的工作环境时，往往会因为外界的刺激，让人的神经在2~3个月内保持相当高的兴奋度，这种兴奋度可以在一定程度上与人对周围环境的疲劳度进行抵消，让技术人员持续高效地完成一些开发任务。

这种方式笔者曾提供给那位总经理朋友，他很高兴地说：这是个很好的建议，他们在威海有足够多的场地，可以将公司总部的技术人员调动到那里去做开发，他回去后立刻就去实施。

对于第4个问题，是我们最常见的项目管理者的一种困境，这也是很多公司很难轻易解决的问题。甚至可以说，很多项目最终失败的原因并不是选择的项目管理者存在问题，或者技术人员存在问题，而是因为这个问题没有解决好带来了负面的效应造成的。

最有效的管理方式莫过于奖罚分明，而奖罚，在公司中能做到些什么？我们可以考虑一下。

将可能的处罚措施罗列如下：扣工资/奖金、延长加班时间/减少休息日、训斥、体罚、打骂、辞退，等等。

其中延长加班时间/减少休息日对于程序员来说简直不算是处罚，就是日常工作状态；剩下的除了扣工资/奖金、训斥、辞退这3种方法外，其他方法是不可行的。而辞退，对于一名仍然对公司有价值的程序员来说，是不能轻易实施的方法。扣工资/奖金，如何扣？扣多少？扣多了，员工会辞职，那么就成了变相的辞退，对项目不利。扣少了，员工会怠工，本来收入就不高，还要扣除，结果仍然等同于辞退。于是，就剩下最后一种处罚手段——训斥，那么我们来看看训斥的结果如何。如果项目管理者技术很好，属于强势项目经理类型，说话让人信服，那你训也就训了，大多数技术人员都不会反抗，因为你说得有道理，他会反省自己。但是，如果项目经理的技术平平，不能让人信服，那后果如何，几乎所有做过项目管理工作的人应该都能想象得到。

可能的奖励措施有：增发奖金、增加休息日、公开表扬、提职加薪、请他吃饭，等等。

一般来说提职加薪的可能性不大，因为你只是一个项目经理，公司并没有给你那么大的权力。对于企业高层管理者来说，升值加薪代表着成本的提升，一般情况下不会考虑如此行事，而如果一个项目组整体加薪，还可能引起项目进行过程中项目组成员之间对加薪数额的攀比，其结果可能更为不利，所以，这一措施基本上是行不通的。

如果说请他吃饭，这是可行的方法，但是，如果让你花自己的钱请他吃饭，你是不是觉得有点亏呢？而在公司给你的待遇也不够高的情况下，公司又不给你报账的情况下，或者说，公司也在同时压低项目管理者本人的收入，那你还能有心情为了公司的事务自掏腰包请别人吃饭吗？

公开表扬，这也是可行的方法之一，但是，光是口头表扬，其实意义不大，所有的技术人员都已经很清楚这种空头支票的价值几许了。

增加休息日，这个方法看似可行，但是，想想你能增加几天？少于两天的话你可以回想一下：这些程序员每天加班时间是多少？他们被占用的休息日有多少？增加休息日的价值大

吗？很多时候对于程序员来说，休息日本来就不存在，尤其是增加的，所以，增加休息日的方法真得可行吗？增加休息日超过两天，高层管理者能同意吗？

那么最后一个办法就是增发奖金。一提到钱，至少对于我们这些收入微薄的技术人员来说，不可能不感兴趣，所有人都会认为这是最实在的表扬方式了。这就遇到下一个问题：高层管理者是否同意增发奖金？毕竟这种奖金是额外增加的，对于项目来说，将会造成成本的提高，对于公司来说将是一种额外的开支。但是，发不发，是一回事，发多少好像又是另一回事。

这里举一个笔者曾经经历的公司 2001 年的政策制度：

当年那家公司为了激发技术人员参与项目的热情，在月初发布了一项政策，允许项目中对部分人员进行高薪聘。也就是给这些技术人员一个短期内拿到较高工资的机会。当时很多人听到后都很激动，都觉得公司在为员工考虑了。要知道，当时很多人正在为半年一次的提升机会名额过少而感到不满。

可是，在下个月初上报项目高聘工资之前 3 天，公司发布了新的政策，指出一个项目组中的高聘人员不能超过项目总人数的 5%（具体数字记不大清楚了，但绝对不会超过 10%），高聘最多高聘一个级别，并对部门整体的高聘人数进行了限制，每个部门的高聘人数比例不能超过 3%（比前一个数字还低）。后来又增加了一条限制：高聘的理由必须经过专家委员会的审核，通过后才能得到高聘，同时，每个月都要重新申请一次。

于是众人哗然！然后在项目中，有位项目经理直接给自己办理了高聘，理由很充分，如果我的工作都没做好，整个项目自然更不会好。另一个项目组中的项目经理没有给自己做高聘，而是给下面的两位技术人员办理了高聘，而其中一位后来因为部门总人数比例超标，被取消了高聘资格。所有的人都会觉得第二位项目经理比第一位强，但是，这位项目经理自己心里也很委屈，自己做那么多事，却没有额外的收入，还要平复大家心里的不满。这个事件 1 年后，这批技术人员都开始逐渐离开这家公司。

对于第 5 个问题，如何确保项目组中的主要技术人员的稳定性，让他们安心地在项目组努力工作，主要在于，项目管理者要向高层管理者详细说明他们为什么重要的原因，获得高层管理者的理解。

前面的例子曾提到过笔者的那位总经理朋友的公司推出了一项新的政策：在公司工作满 3 年的员工，如果需要买房结婚，公司可以提供 10 万元无息贷款，在 5 年内按月从工资中扣。

通过这种方式，公司留下了几位有实力的技术人员，但是，还是经常遇到问题，比如提出要辞职的那位做硬件开发的技术人员，就再次让他头疼了很久。

于是,笔者给出了建议:无论是软件开发,还是硬件开发,都可以采用交换编程的方法,来进行人员备份工作。但是,他对此却产生了异议,并说在他们那里找到一位合适的技术人员很困难,那些有技术实力的人都跑到北京、上海、广州、深圳等地了,所以,让他找个这样的人员来做交换编程,难度太大了,而且目前的技术人员也都是经过2~3年多才培养起来的。

从这个例子可以看出一个问题:公司对员工的投入是否能够让员工感到安心并产生报恩的感觉。其实,除了正常合理的工资支出外,通过管理方式,尤其是人与人之间的沟通交流,是可以解决大多数问题的。

笔者的这位朋友一直都是把员工只当做员工看待,并没有把他们当做朋友来看待,这也是他对这些技术人员严重不放心的根本原因。他曾对笔者说,只有当一个人做到一定程度,被他认为可以作为兄弟的时候,他才可以放心。针对他的这句话,笔者没有当面反驳,因为觉得他的思路进入了一个误区,即他并没有真心地和下面的技术人员交心,只做表面文章,或者说,无奈之中做了违背用人不疑,疑人不用的古训。

因此,笔者没有对他提出更多的建议,因为他主要考虑的是如何为企业直接盈利,而不是如何长期地保持企业团队的技术稳定和未来发展。随后,笔者与他的合作也产生了一些分歧,就没有再和他续签后续的合同。虽然俩人每次见面都会一起吃顿饭,聊聊天,但是,笔者却感觉,他的公司模式不是笔者能够完全认同的,不是一种能推动笔者所认同的整体的管理思想和方式的模式——其根本原因,就是因为企业高层管理者对员工的本质想法所致。

针对上面这几个问题,其实更深层次的问题就在于,公司是否足够信任项目经理的管理方法。对于高层管理者来说,他也很担心项目管理者在公司里面形成自己的独立实体而与公司对立,但是,却又希望项目管理者能够搞好内部团结,留下这些技术人员。其实,对于高层管理者来说,他本人也喜欢使用自己熟悉的人,中国历来都有一朝天子一朝臣的说法和做法,不拉帮结派,怎么搞好关系?怎么才能团结人?

这里顾虑的深层次原因有两点:一是高层管理者对程序员的心态不够了解,二是过于顾忌项目管理者的人际关系好可能会给公司带来不利因素,却没有考虑到另一个方面。

程序员的心态相对来说持传统的中国文人/技术人员的心态居多,更多的是只有项目管理者的人际关系足够好,才能让技术人员和他交心,才能使项目团队获得稳定。其实,高层管理者更应该从另一个角度去思考,既然项目管理者有能力将团队内的人员留住,这难道不是一件好事吗?作为高层管理者本来应该面对项目团队的每个人,而现在,只需考虑一个人,即项目管理者,让他去留住整个团队,保证项目正常实施。相反,如果项目管理者留不住项目组中的技术人员,无法团结他们,这才是高层管理者应该担心的事情。对于管理问题,将矛盾集中于一点进行解决,比矛盾散播于多点的解决方案要简单得多,往往不需要全部解决,只要部分解决也一样能极大地缓解矛盾冲突。这和写代码不一样,写

代码更倾向于将接口简单化，而不是将所有的接口集中于一点。反过来想一下，如果高层管理人员不能留住自己的下属（项目管理者），那么又如何才能留住自己团队的技术人员呢？

对于一个软件公司来说，技术人员就是劳动者，如果没有技术人员的参与，再好的公司，也就空谈，善待技术人员，这样才能保证你的舟不会轻易颠覆。

赋予项目经理以一定的权力，让他有权对项目组的成员进行奖励和处罚。要知道，平时的加班已经是一种违反规定的处罚方式了，因此，为了团队的稳定，付出少量资金来奖励一些技术人员也是应该的。

2. 高层管理者与程序员

一般来说，企业的高层管理者很少与程序员进行交流。下面举例说明。

2010年1月底，笔者去一家原托普人员组成的公司谈合作，开车来接的人是当年跟着宋汝华（托普集团的创始人，目前在北美）创建公司的第一批人之一，交谈中提到笔者对成都尤其是对原托普公司的看法时此人多次陷入沉思。

应该有很多人看过2004年《程序员》杂志上发表的《论托普的倒掉》一文，后来CSDN的总裁蒋涛提到，这期杂志成了很多软件企业总裁的案头卷，在很多国内大型软件企业的总裁办公桌上，这份杂志都摆放了很长时间。

中国的基本管理体系中，官本位出现以后，官民之间就成了对立关系，能把官民关系理顺的官员被称之为青天大老爷。老爷的称呼是古代中国甚至近现代中国对有权势有地位的人的尊称。这样的称呼也就体现了官与民之间的距离，两者很难融合。

由于在企业中，高层管理者就是最高权力的所有者，而程序员往往是最底层的工作人员，这两者在一定程度上，高层管理者是主动行为者，他的任何行为都不会受到直接的惩罚，而程序员的一些主动行为则可能带来直接的惩罚性结果的出现。也就是说，高层管理者如果在一些可能危及公司或者损害基础员工利益的行为发生后能够意识到它的危害性并尽快进行弥补，那么一切都还来得及挽回。但是，由于官本位思想的作祟，高层领导者往往不愿意认错或者弥补，于是，最终的结果是，当此类现象屡次发生后程序员被迫选择离开。

其实认错并不可怕，害怕认错的人，其恐惧只是在于他自己本身对认错的错误认识，往往主动认错，主动改正，大家都是可以宽容并理解的，毕竟没有人是完美的，公司也一样，都会犯错误，就怕犯了错误不认错，还强硬不承认错误到底。

由于程序员心理的问题在温伯格大师的《程序开发心理学》一书中已经做了很全面地分析，所以，本书就不做更多的介绍了。

笔者提出的建议是：主动认错和弥补错误是度量的体现，不是形象的损失。

也就是说，主动认错并不会降低管理者的身份和影响力，相反，会给你带来意想不到的好处。

另一个例子：

2009年7月间，笔者在一家医疗相关的公司帮忙做技术总监，构建团队，当时谈好了，所有的技术和开发问题全由笔者负责，公司总经理只需做好配合工作，使得开发能够顺利进行即可。该总经理据说是英国留学回来的MBA。

但是，开发期间发生了一件事情，笔者认为可以成为软件企业管理上的一个典型事例，在这里列举出来供读者分享。

第一天，笔者从16名测试人员中面试招聘了两名测试人员。

第二天，两人来上班，这一天是星期五。

周日，公司总经理突然发现公司的一台机器上少了一个光驱，而这台机器使用者，就是其中一位测试人员。

该总经理的处理方式如下：

首先给那位测试人员打电话，问他明天是否来上班，然后提到公司丢了一个光驱，等等（具体的对话内容无法确实考证）。

然后又给笔者打电话，说要报警，言辞非常激烈。笔者当时无语，就说，等周一上班一起处理吧。

周一早上，那位新员工上班不久，光驱找到了，是以前辞职的一位员工卸下来，装到另一台机器上进行软件安装后没有放回去。

下面我们来分析这个事例。

如果光驱没有找到，作为管理者应该如何处理这一问题？

从两个方面考虑：

- 1) 相信员工，进行寻找。
- 2) 不相信员工，报警或者进行处罚。

继续上面的例子：

在周一上午，这名测试人员离开公司后（据总经理说，该人态度非常恶劣，然后走人），总经理当面询问所有开发人员，你们对这个事情有什么看法？

所有的人都回答，没有看法。

这段对话，总经理在不同的时间当着不同人（包括主要投资人）的面向笔者重复过两次，一直坚持自己的处理方式是正确的，而且对投资人说，她不懂技术没法审核技术人员，只想将主要投资人的计算机看管好，总不能连计算机设备都看不好呢。

其实，只要她改变一下态度，同时按照笔者最开始的提议，让笔者周一来进行处理，一切事情都不会发展到如此极端的程度。这位总经理的固执很快使她失去了公司总经理的位置，但是，因为她本人还是一个小投资人，利用来公司的机会把所有的资料全部上锁，称之

为防止机密流失（该公司在当时还没有任何有价值的代码和资料出现，以前的开发过程更是一片混乱），此外她还有一系列极端的对抗行为使得后面的开发工作无法开展，这家公司在这次事件一个月后就完全倒闭了。

3. 高层管理者与周边角色

高层管理者与周边角色的关系主要是指市场推动活动以及相关的资源信息管理，或者说带有很强市场性质的工作活动，不属于本文的重点讨论内容，限于篇幅不再赘述。

7.2.4 小结

本节从4种角色进行了软件项目开发过程中的心态对抗分析，同时着重对高层管理者和项目管理者进行了深入的心理分析。一个软件企业能否走得很顺利，其核心重点有两个，一个是市场上的项目和合同签订情况，另一个就是其技术团队的稳定性和发展性，这两者，缺一不可。国内的企业大都强于前者而弱于后者，更主要的原因是很多企业高层管理者并非技术出身，但是，他们的想法和思路往往会决定他们是否能留住企业内的主要开发人员。这也就决定了他们的团队是否可以顺利地完成任务，如何让项目管理者在项目进行过程中留下技术人员并激发技术人员的工作热忱。

有关管理者心态问题研究讨论到这里基本上告一段落，如果这些内容能够得到一些公司的认同和采用，那么讨论的目的也就达到了。只希望国内能够多一些百年老店，少一些一次性作坊；多一些关心技术人员的企业，少一些恶性盘剥技术人员的公司；多一些管理者与被管理者之间的融合的氛围，少一些矛盾冲突的激烈。

7.3 管理中的权力使用分析

在本节中将借用一些电视剧中的片段进行配合分析，讲述一下企业管理甚至团队管理中的权力下放以及权力限制的方式。由于涉及权力下放和限制中双方甚至三方的心态变化，因此单独列为一节进行讨论。

7.3.1 管理中的放权与分权

所有的人都知道，要做事情，必须有一定的权力，中国有句古话：名正言顺，名不正则言不顺。

在企业中更是如此，如何给项目管理者一定权力，这些权力包括哪些内容，这都是企业高层管理者需要考虑的。而项目经理如何给开发组的组长放权，让开发组长也有一些可以控制的权力，这是很多企业高层管理者都感到头疼的事情。另外，给了权力，如何控制，这更是一件让人头疼的事情。权力放大了，可能造成项目经理一人独大，甚至出现项目经理带团队与公司对抗的悲剧产生；而权力小了，项目经理又无法处理事情，团队成员不听指挥，那就是一盘散沙。

这里，我们可以看看《亮剑》中的386旅的旅长是如何给团长李云龙放权的，而李云龙

又是如何给下属的营长连长放权的。

1. 旅长给团长的放权

赵刚听李云龙说派了一个营过去牵马回来，立刻就急了，说这是擅自调动部队。李云龙也急了，他们的重要对话如下：

1) 旅长说，我有装备我要你干什么，你既然能当团长，就有能耐去搞枪，要不然就回家抱孩子去，别在这儿给我丢人现眼（要知道李云龙连老婆都没有，自然没有孩子可抱了，所以就只能去搞枪了，否则，当时在世面上他就没得混了）。

2) 李云龙说，让我搞枪，没问题呀，可是你不能给我戴紧箍咒呀，对不对，你总得给我点自主权吧，不能什么事儿都让你大旅长占了，又想让我搞枪，又想让我当乖孩子，这叫不讲理呀。

3) 旅长说，去去去去去，自己想办法，我什么都不管，我什么都不问。我警告你，李云龙，你少拿这些屁事来烦我。

好了，这就是386旅旅长给李云龙的放权过程。

在权限范围内的事情，放手让下属去做，过多的干预反而会影响下属的积极性，那就成了对话中出现的“又想让我搞枪，又想让我当乖孩子”，那就是“不讲理”。国内很多企业高层管理者总是担心项目经理自己做大，于是处处限制，担心他把团队拉走自立门户，殊不知，这正是后面项目失败的原因。

中国有句古话：用人不疑，疑人不用。尤其在必须让别人承担责任的事情上，那就应该信任，否则，就不要任用此人。

2. 团长给骑兵连长的放权

搞到了骑兵装备，下面就是训练，其实这也就是项目组中各个小组工作的管理问题。小组负责人应该有什么权力，这里可以看到李云龙与赵刚之间的第二次正面冲突：营连长打骂体罚战士的权力问题，也就可以看到李云龙是如何给下面的小组负责人放权的。

(1) 骑兵连长的体罚

骑兵连连长孙得胜没有什么文化，因此在训练中对战士的惩罚就只有体罚的方式（惩罚，这是一种管理措施，当时八路军没有军饷，所以，也没有其他办法，训练必然有人好有人差，如果不给小组负责人惩罚的权力，那就会有人根本不听他的指挥。在软件开发中也是一样，如果没有惩罚权力，只靠向上面打报告做惩罚，一个是时间长，效果不好，另外还会让其他人认为你爱打小报告，引起更多人的不满。可以说，打小报告，是“人神共愤”的事情，呵呵，说得严重点了，但是，对于这做法，相信很多人都深有体会，或者心有戚戚的）。

(2) 政委反对体罚

赵政委就说，不许他体罚战士，要做思想教育。

很明显，如果不允许惩罚，只能靠说教，往往很难有明显的效果，毕竟很多时候，处罚在管理中是必须采用的手段，有奖有罚才能激发团队的积极性。

八路军当时没有军饷，没有装备，人又不多，不能说，你不好，就开掉你，让你回去当步兵，都不现实。那么剩下的手段是什么呢？不让你吃饭？那饿坏了，就更不能好好训练了，就只有上鞭子了，给你一下子，让你知道疼，下次你就会小心，小心第二鞭子还会疼的。同时，这鞭子下去，不会伤筋动骨，你还能训练，也不影响其他的，可以认为是当时最好的惩罚措施了。

(3) 团长婉转地拒绝

说教是不行的，就像李云龙所说：“对于不好好训练的战士就说，我求你了，老哥，我给你跪下了，行不行？”——这肯定不行，一个管理者如果随便就给人跪下，那算是什么事情？这样也有损于管理者应有的威严，于是通过这种方式获得了政委的让步（这就属于另一个争议解决中的技巧问题了）。

(4) 结果

所以，最后的结论是：你只要让骑兵连两个月后能形成有效的战斗力，其他的，你看着办。

孙得胜最终获得了自己应有的处罚权力和对自己团队的控制力。所以，在后面的战斗中，才会以全员牺牲的方式，给主力部队（步兵）赢得了撤退的宝贵时机，而不至于被鬼子的骑兵追杀。

这里附加一句：在军事作战中，最怕的就是追杀，尤其是骑兵对步兵本身有着速度上的优势，一旦建制丧失而撤退，哪怕再多的军队，都会面临灭顶之灾。例如，三国史上的官渡之战，赤壁之战，夷陵之战，南北朝时期的淝水之战，英法半年战争中的阿尔库金战役，等等。

李云龙对骑兵连连长孙得胜最后说了一句话，体现了他放权方式的本质：

“孙连长，骑兵连就交给你了，怎么训练是你的事儿，一句话，我要的是能打硬仗的部队，两个月之内骑兵连没有任何战斗任务，你得把部队给我训练好，两个月之后，骑兵连应该像一把出鞘的利剑……”

(5) 分析

放权，要选择合适的人，对这个人进行全面的放权，同时整个集体还必须为此付出一些应得的利益。这里大家看到了李云龙的放权，付出应得的利益其实也在这里做出了表示：两个月，两个月的时间就是一个付出。当时，八路军因为不发军饷，那就是这些人的吃用和时间，他不能在这两个月内去要求骑兵连怎样做。这就是对部队的培养和付出，对新人的付出——国内绝大多数的公司老板，都看不到这一点的作用，都担心这会让自己一无所得，都担心下面的人跑了。当然，这种担心是可以理解的，但是，如果你看错了人，那就是你选人的责任，而你对一个错误的人进行了投入，那就是你的问题了。选对人，对所选的人进行足量的投入，这是必需的，企业团队要发展，必须有新人和新的技术的融入，不能只依靠原来的技术。毕竟技术的发展和变革是很快的。

3. 开发中实际现象的对比分析

我们在开发中经常遇到的一个问题就是：技术团队没有人来掌管。

这时该怎么办呢？

让公司人力资源部去招人吗？招聘行政管理人员，人力资源部或许有办法，但是如果技术人员完全让人力资源部去找，大概这个公司也就离死不远了。

让领导去找吗？领导也未必说得清楚具体需要什么样的人，而且，即使招来人，这些人能否帮助你的团队解决问题呢？

这就存在一个矛盾：自己培养人，还是招人？

这时候，就存在一种资金使用权和培养人的权力问题。

实际上，国内很少有公司给项目经理放权到可以让他使用一些资金来提高团队战斗力和效率，这也是国内很多项目遇到开发人员积极性不高的原因之一。很多项目严重滞后的时候，高高在上的领导们则需要考虑一下：是否给这些可怜的项目经理有效的激励机制和激励权力呢？既不给他们权力，还要求他们必须做到，那岂不是不讲理吗？这就是让人家当乖孩子，还要让人家去搞“枪”。

386 旅旅长的放权是正确的，允许李云龙擅自调动部队来搞副业，不追究他的任何责任，只要他不闹出大问题，不出格逾矩，就不管了。

这样一来，一年的时间，新一团什么都有了，歪把子，九二式，掷弹筒，迫击炮，就靠这些家底，他才敢和坂田联队硬碰硬地去干。

一个联队的编制和团是同等的，在当时抗日战场上，从来都是一个日本联队至少需要中国一个旅乃至一个师才能对抗，团对团的对抗从来都是必死无疑的，比如：该电视剧中提到过，晋绥军 358 团曾经就被坂田联队击溃过。

注意：到了这里，有些高层管理者会强调说，担心放了资金使用权力项目经理会以权谋私，或者收买人心，然后带团队集体跳槽。关于这个问题就涉及用人和信人问题了，给了资金使用权利，你也可以在资金的使用用途上做些限制，但是不要因噎废食，为了防止项目经理谋私而使得项目经理根本无法使用这笔资金来对团队成员进行奖惩。关于跳槽的问题，如果企业能够信任员工，并且对员工足够好，集体跳槽的事情一般是不会发生的，而这两个问题的核心都在管理者的用人与疑人上，这些问题已经在 7.2 节中做过介绍了。

同样的，我们也经常看到国外的公司，诸如 IBM、HP、爱立信等都有内推的做法，那就是让技术人员推荐自己熟悉的有技术实力的人员进入公司，然后给予一些推荐奖励。这些企业应该是看准了一点，那就是，如果你所供职的企业效益不好，你不会推荐自己的朋友来公司，否则，你迟早会被朋友指责的。你能推荐人，说明你在一定程度上对企业有相当的认同感，这样你推荐过来的人，企业也比较放心，因为你不会因此欺骗公司最后让自己受到牵连。

讨论完了放权，下面我们要说说如何限制权力的使用，不能说给你权力，你就随便了。

如果说给了资金使用权，就可以自己拿钱去开公司，那么对于原来的公司而言，岂不是亏大了吗？

7.3.2 管理中的权力限制

如何限制权力的使用，往往比放权还困难，因为一旦加以限制，就好像给下属加锁了。如何才能委婉地做到这些，同时又不会影响项目组的积极性，还能有效地保证收权后项目组的进度不受影响。我们来看看当时 386 旅旅长是如何做的。

1. 如何限制权力

李云龙搞到了一个骑兵营的装备，而他自己只有一个团的编制，下面设一个骑兵营是违反常规的。一般来说，骑兵属于特殊编制，除非是专门的骑兵部队，否则，都要相隔一个级别进行军队建制组建。对于整个 386 旅来说，旅长为自己的部下能搞来资源而高兴，但是，他必须考虑在下面的团级建制之间进行一下平衡，不能随意让他们超越编制限制发展，因此必须收一些装备上来，否则，他这个旅长就难当了。

于是，在李云龙搞到装备的第二天，旅长的电话就来了：

李云龙呀，你这一次又捞了多少好处啊（领导开始试探下属的口气，他知道这是不好要的吧）？

不多，不多，就一个骑兵营的装备，我准备组建一个骑兵营。

看把你能的，你一个团就想闹一个骑兵营，那老子这一个旅，是不是该弄一个骑兵团？

应该，应该，应该，冲你旅长的名声，配个骑兵师都不多（李云龙知道领导要从他那里抢东西，所以，他开始耍滑头，糊弄事情，说旅长下面应该配个师——这不是明摆着给领导戴高帽嘛）。

你废话少说，你留下一个连的马匹装备，剩下的统统给我送到旅部来（正式的打劫开始了）。

干嘛呀，旅长，打劫呀，你干脆枪毙我得了，不行不行不行，坚决不行（李云龙发现不好，只好考虑强行抗命，希望能够混过这一关，他心里也知道，旅长是真的要抢了，自己不好过关，但是，总归是自己抢来的东西，拱手送上去，实在是心有不甘）。

不行？那好吧，那咱们俩就得说道说道了，你未经请示擅自调动部队，这罪过可不小吧（旅长一看，说好话要不过来，只能施加压力了，而且，要往重的地方说，越重越好，否则，这小子不会轻易就范的。旅长心里也清楚，他想：老子如果不是从你口中抢食物，是有点不好意思，前面才不会说那么一大堆客套话，你小子居然敢顶撞起我来了，想吃独食，有了好处，至少要给我分点么——这也算是当时的腐败之一）？

（李云龙一听要追究他擅自调动部队的罪过，这个实在是太大了，比战场抗命可能还要严重，弄不好再被撤了，别说这一个营的装备了，一个连自己也留不下了，自然着急

了)我以前可是(有点说不下去),哎,旅长你同意过呀!

我同意过?谁能给你证明啊(继续施加压力,适当的时候,言而无信还是有好处的,另外还好当时没有留下字据,旅长心里暗自庆幸,施加的压力效果达到了,心说,你不给,我就让你小子多着急,我看你给不给)。

旅长啊,这红口白牙,你可不能不认账啊(继续耍赖,试图通过承诺获得支持)。

少跟我扯淡,啊,现在两条路由你选,要么你把马匹装备给我送来,要么我就上报总部,追究你擅自调动部队的事,李云龙,你打算怎么办(要么你就范,让我限制你的权利,分我点好处,要么,我就拿大帽子压死你,让你小子什么都得不到,别说这个营,我连个连都不给你留,全拿走)?

哎呀,官大一级压死人哪,行行行,旅长,我认了,你要想打劫你就明说,找这些借口干啥呀(只能认了,否则,自己又是什么都没有了)。

哈哈哈哈哈(电话里传来的笑声),我不找借口你能给我吗?我说你小子怎么学得跟山西土财主一样,就这么仁核桃俩枣的,你还给我抠抠搜搜的,像个男子汉吗(好了,目的达到了,再给你加个帽子,让你舒服舒服)?

剩下的对话内容就不再多描述了,总之,经过上面的博弈和对抗,旅长的目的达到了,李云龙最后也高高兴兴地上交了多出来的装备。

2. 分析

我们的企业高层管理者也应该考虑考虑到底如何放权和如何限制权力的问题,而不要什么事情都板着脸来做,那样,可能适得其反,效果不会很好的。

总结一下上述对话,旅长做到了如下几点:

- 通过电话进行商谈,以免见面的尴尬;
- 话语中尽可能地缓和语气,不发生直接冲突;
- 适当的时候施加压力,表明这是规定,表明他也是受限于制度或者压力,实在没有办法让下属全拿下;
- 目的达到后,给下属以另一种形式的鼓励/奖励。

在这里不选择开发实例进行分析是因为国内企业基本上没有给项目经理较多的权力,因此也不存在项目经理权力过大需要限制的问题。通常情况下,是项目经理权力过小,使得很多事情无法实施而发愁,甚至最后导致团队逐渐解散或者以项目失败而告终。

从这里可以看到,对下属的权限限制,或者权限回收也要讲究一些策略,不是僵硬地收回或者处罚就能达到目的,毕竟,你还需要他真心为你工作。从上面旅长的做法总结以下几点:

- 合适恰当的谈话方式;
- 语气舒缓,避免发生激烈冲突;
- 适当的压力,明确的态度;

- 打一巴掌，最后还是要给个甜枣吃。

7.3.3 项目管理者如何与组员合作

众所周知，一个项目组，必然有一个带队的，称为项目经理。项目经理如何和组员合作是一个争议较多的话题，有人问与手下人如何合作？手下人不干活怎么办？团队成员间关系如何协调，等等。

项目经理与组员间的关系如何定位，这是一个既简单又复杂的话题。

笔者以前也有一个当经理的同事说过，他是靠着弟兄们吃饭的，所以……

但笔者认为，在《亮剑》中，我们也可以得到不少的启示。

1. 亮剑的过程

李云龙是如何做团长的，大家在第20集中可以看到，为了被杀的魏和尚，李云龙不惜关押了同级别的新二团团长孔捷等人，同时动用兵力直接攻打黑云寨。结果，仇报了，他被降职为营长。

快意恩仇地一刀，杀掉了山猫子（二爷）。痛快地一刀！

笔者相信只要有血性的中国人，都会喜欢这一刀的。

这一刀也同样体现在他攻打平安县城的时候，也体现在第1集中，返回去救出张大彪的过程中。

这一刀的效果体现在了很多地方，比如骑兵连连长孙得胜带着全连冲锋到阵亡的过程，冲锋到了只剩下一条胳膊，还是继续冲了过去，等等。

2. 亮剑的启示

这一刀表现出什么呢？

表现出李云龙从来没有把团里的人当做手下人看待，而是拿他们当亲兄弟。亲兄弟死了，自己哪怕拼上一死，也要为他报仇。降职，当不当官都无所谓。

这就是全心地为团队成员考虑的一个典型。

这种管理是不需要任何技巧的，需要的就是一颗心，一颗负责任的心，一颗不考虑自己利益的全心为了团队成员的心。

为什么李云龙带的兵愿意为他玩儿命，其原因也就很明显了，就是他愿意为这些兵的生死发展玩儿命，种瓜得瓜，种豆得豆。

在现代的团队管理中，很少会涉及某件事情要一个人生死的时刻，但是，这时团队的管理者应该考虑的就是成员的利益。保证团队所有成员的利益，团队成员才会发挥出自己最大的能动性。

说得刻薄一些：管理者和投资人获得的资本升值其实就是资本论中提到的剩余劳动力和剩余价值的部分。但是，如果团队成员都没有心情干活，在目前这种工资固定的薪酬形式下，剩余劳动力的数量必然会极大地降低。

如果能够让团队的成员完全主动地工作，激发热情地工作，那么即使加些工资，剩余劳

动价值的数量也会相应地增加。

3. 结论

可惜的是,国内很少有老板会意识到这一点。所以,才会有那么多的劳动纷争。

员工工作的时候总是提心吊胆,下班的时候匆匆忙忙,早晨上班惶惶张张,上班伊始神情恍惚,午饭后身体困乏没有精力,……问题越来越多,会让员工产生骑驴找马的心态,人自然不稳定。

如果一个团队管理者遇到了管理上的问题,不妨先问问自己,是否做到了以上这些要点的内容。

7.3.4 一些常见问题

1. 保密问题

问:文档是否需要技术人员保密,如果需要,那么假如我们向组长或者 PM 索要“不相关”的文档,他甚至可以拒绝吗?所说的不相关,比如是这个人开发的模块不涉及,也不需要 peer review,那么就是“完全不相关”。

答:对于一些行业而言,文档需要根据情况进行一些保密,另外对于不同级别的技术人员也可能根据需要进行一些保密。

如果索要无关的文档,项目经理的确可以选择拒绝,不过,一般做出这种决定的组长和 PM 应该属于强势项目经理。也就是说,这种管理者本身应该是技术高手。否则,这样的决定会让其他人受不了。这是强势项目经理与弱势项目经理之间的差异。

对于弱势项目经理而言,则需要征询主要技术人员的建议后做出拒绝会比较合适一些。

另外,组长也有权力限制组员看到一些不必要的文档,这样可以减少时间的浪费。但是,如果因此发生了配合上的偏差,组长是要负责的,同样的责任项目经理也应该承担。

如果一名程序员因为被限制看某些文档而不努力工作或者做一些其他的事情,只能说明他个人的职业素质有问题,这样的人在任何公司都将是不被容纳的。

另外,在管理学上,是绝对禁止一个人看自己权力范围以外的不被许可的文档和资料的。如果你看了,你就违规了,或者走人,或者等候处罚,并且随时可能因为所看文档和资料的泄密而成为嫌疑人或者承担责任。

管理上有这么一句话:管理者是有情的,管理是无情的,制度是绝情的。

2. 学习和提高问题

问:有人说,他希望能看更多的文档,以提高自己,如果被限制了,就可能失去了提高的机会。另外,认为只有和高手在一起才能得到提高。

答:笔者并不认为公开文档就能提高一个人的能力。

相反,对于一些情况而言,这样的公开反而是有害的。

比如说,有些大学生,还没有毕业就到处找文档,查资料,让别人给他发送项目文档。他要这些文档做什么呢?不是为了学习,也不是为了应用,而是修改后拿去糊弄老师。当他

毕业时,还可以修改出一些文档来糊弄招聘者,这样的人是非常多的。笔者就见过很多这样的人,大学刚毕业,就声称自己如何了得,开口闭口自己是 sa,不能做编码工作,要做就做 pm 或者 sa。

另外,人的接受能力和领悟能力并不是跨越式发展的,适当的时候看适当的提高文档和资料是有用的,也是有好处的,而如果在不适当的时候看过高的文档,往往只能让一个人毁灭掉。就像现在很多人想做 pm 和 sa,他们想做这个职位的原因是,这个职位高,收入高,更重要的一条是,不用那么辛苦地去写代码了。可以这样说,如果一名程序员拒绝写代码,他永远不会成为一名好的软件从业人员。可以说一个软件技术从业人员一旦停止了写代码,也就意味着他的软件职业生涯的终止——这句话说得可能有些过火。但是,很多人想要的提高就是为了躲避写代码,这是非常严重的问题。

一位好的领导者是会根据成员的实际成长而不断地给他一些提高的机会,给他看一些有助提高的资料,但绝对不是把所有的资料都给他看就是对他的帮助。相反,这在目前的情况下,是一种非常严重的伤害。

3. 另一个提高的问题

问:假如我现在是一名程序员,从事这项工作已 5~6 年了,我希望能有机会锻炼自己的项目管理能力——这是我“主动的、自觉的”想。

但是如果按照前面的思路,除非我真的做 PM 了,否则一直得不到实践锻炼的机会。也就是说,无法在实际项目中提高自己(总不能光看书吧),这么一来,我将被注定这辈子永远做一件事情——仅仅因为我得不到锻炼自我的真实环境。当然还有别的可能:得到升迁,或者跳槽后,有机会去做 PM,才能在实践中锻炼自己。这好像是一种悖论。好像 PM 都是游泳健将,我真希望有朝一日像他们一样下水试试。

又比如,我一直希望能和水平比我高的朋友打篮球,从而提高自己的水平,但实际上经常和水平差不多的朋友一起玩,一直得不到那样的机会。于是,我打篮球的水平永远得不到提高,就算有所提高,提高的速度也会比较慢。

答:首先,这种说法是有问题的。不见得必须和水平高的人交流才能提高自己的水平。每个人都应该学会在任何环境中总结工作积累经验,这样才能最有效地提高自己的能力。

其次,提高得快与慢只是个人的感觉。水平高的人不见得总有时间。任何人都一样,都是普通人,没有谁是真正的超人。

就拿跑步来说吧,如果你认为别人都很慢,难道就没有考虑过你不能只是跑在他们前面而已,一定要超过他们两圈的想法吗?

认为必须和水平高的人交流的想法,只会让人产生一种严重的依赖心理。也就是说,你将丧失自学的能力,而主要是靠别人的提醒和帮助来学习。如果是这样,你将不可能成为一名绝顶高手。

最后,帮助水平比你低的人,并不见得就不能提高自己。也许他们会遇到很多你没遇到的问题,毕竟每个人的经历不同。

补充一点：帮助水平比你低的人，不代表帮助他查找一些基本手段就能找到的概念和文字就是帮助，这样低层次的帮助应该是避免的。因为对技术人员来说，他必须有基本的技术资料阅读和查询的能力，这个能力是最基本的，没有这个能力，他也就失去了作为技术人员的基本资格。

下面是 CSDN 一个网友 jane 在讨论中对这个问题的最后看法：

一般来讲，无论公司或项目经理都会根据项目情况适当地控制文档的管理权限，程序员如果想向上发展，必须做到以下几点：

- 要出色地完成本职工作，尽量比其他人做得都好；
- 通过其他途径，比如上网查阅等了解项目管理的相关知识；
- 以积极的工作态度得到项目经理或公司领导的认可。

这样一来，也许不用你提出要求，公司领导或项目经理就会考虑培养你。

如果你在一家公司工作3年以上，仍没有向上发展的趋势，首先审视自己的工作态度和能力等，如果认为问题不是出自本身，此时可考虑换个工作环境了。

7.4 项目经理分类

读者可从网上或其他书籍中查到很多关于项目经理的信息。几乎所有的行业都有项目经理这一称呼，随着时代的改变，项目经理这一称呼开始发生变化，在本书中多处采用的是项目管理者一词，表示负责项目管理职责的人员。

项目经理是项目管理者的习惯称呼，有些单位或团队称之为项目负责人。但在本书中根据笔者多年来参与项目的一些经验和对项目管理的一些看法，提出了一种相对于传统称呼的全新的概念，弱势项目经理和强势项目经理两种称呼。

本文分为两个部分：

1. 项目管理之系统

这一节介绍传统概念下的项目管理和项目管理者的定位，以及由此产生的一些问题和一些项目中因管理而出现的矛盾冲突。作者从管理学的角度结合中国人的传统心态来看待项目管理在国内的一些表现，从而提出了项目管理人性化方式的建议。

2. 管理者的定位分类解析

(1) 强势项目经理

(2) 弱势项目经理

基于人性化项目管理的理念和实践，管理者自身应该如何定位自己在团队中的角色。强势项目经理和弱势项目经理的概念，以及这两者定位的差异和在同一项目中应该采用的管理策略和手段。

7.4.1 项目管理之传统

下面是几个必须涉及的基本概念：管理、项目管理、项目经理。

1. 什么是管理

这是一个很复杂的概念，由于篇幅所限我们不可能拓展开来进行管理定义的讨论，但是我们必须知道管理的基本概念是什么。下面引用的内容来源于网络，是相对比较认可的关于管理定义的阐述：

管理是指在一定组织中的管理者，运用一定的职能和手段来协调他人的劳动，使别人同自己一起高效率地实现组织既定目标的活动过程。

哲学意义上的管理（是对各种管理的抽象表达，是各种管理的本质意义）是指，集中人的脑力和体力达到预期目的的活动。管理不仅表现在对人与人之间关系的调整上，也决定如何运用自己的体力和脑力上。比如，早晨起来锻炼身体，然后去上班；又比如工作先干什么，后干什么，采取何种手段。无论干什么，都需要集中自己的脑力和体力，否则就无法完成目的。

管理是为实现共同的目标，多众员工参与的共同劳动，如进行计划、组织、指挥、协调、控制等。

管理是人类与生俱来的行为。

2. 什么是项目管理

这里引用了有关项目管理的概念：

“项目管理就是指把各种系统、方法和人员结合在一起，在规定的时间、预算和质量目标范围内完成项目的各项工作；有效的项目管理是指在规定用来实现具体目标和指标的时间内，对组织机构资源进行计划、引导和控制工作。”——《项目管理知识指南》

项目管理，是一种管理学术语，是指在项目活动中运用专门的知识、技能、工具和方法，它使项目能够实现或超过项目利益所有人的需要和期望。

3. 什么是项目经理

这里引用了如下关于项目经理的概念：

项目经理是指为项目的成功策划和执行负总责的人。

注意，这里提到了项目经理对项目负总责，也就是说，项目的成败都要体现在项目经理身上。

在继续下面的内容前需要明确一点，即项目管理的终极目的是什么？我们对该问题认识是：项目能够完成，成员开心工作。

项目能够完成，项目经理责任最大，但是如果员工不能开心工作，那你的项目是否能完成也就出现了问题，即使完成了也许也要花掉更多的资源，诸如人力、财力、物

力、时间等。因此项目的完成不代表项目成功，因为一个成功的项目应该是在一定的资源投入范围内，项目完成。如果一个项目远远超出了预期的时间，那它也只能算是一个失败的项目，同样，人力、财力、物力等任何一样大幅度超出，都是项目失败的标志。

因此，在传统概念中，人们认为项目经理就是一种官。而且，很多公司把项目经理作为一个岗位级别来设置。这样的结果是，使得很多项目经理任职之初就获得了一种所谓的高人一等的地位，而这种地位往往伴随着很多不同的冲突，于是就衍生出了职位定义对人的影响、职位产生的差异，或者就是职位产生的最终问题。

人们习惯上称呼管理者为“官”。

“官”这个字在我国传统习惯中有着非常重要的地位，于是项目经理也往往会把自己看做一名领导者，技术人员也会把项目经理当做领导者，不仅是管理上的领导者，同时也应是技术上的领导者。

于是，这个项目经理就可能出现拿着项目经费为自己谋取私利，克扣成员工资，动辄以扣工资开除为口头禅对付自己的项目团队成员等。

而实际上，项目经理仅仅是这个项目的一位负责人，他的主要职责是确保项目按照合同的要求完成，而不是一个正确意义上的领导者。

项目失败了，项目经理是必须负责的，无论是因为何种原因失败，该角色的人都必须承担。

中国传统的习惯是“学而优则仕”，如果你不够优秀，就不能仕，否则，所有的人都会认为不公平。仕途在汉语中表达的意思就是当官的过程。所以，在软件团队中担当项目经理的人必须技能出众，才能领导别人。但是，在实际中，我们很难找技能和管理能力都优秀的人，于是在本书中将项目经理划分为两种类别：强势项目经理和弱势项目经理。

7.4.2 强势项目经理

在所有软件开发的从业人员中，都希望能有一个强势项目管理者，都希望自己的项目经理能在技术上给予团队成员帮助，又懂得管理的技巧让团队成员开心工作。

但是，由于技术人员多年来对技术的钻研和探求，往往忽视了人性上的一些关系，于是很多技术水平很高的人员被评价为高 IQ 低 EQ。

这些年软件项目早已不再是 20 世纪 90 年代中期以前的那种几个人就能完成的工作量，大批的项目都在规模上不断地提升。因此，一个合格的强势项目经理往往是大家所期盼的。

但实际上这样既懂技术又懂管理的项目经理却很难找到。

在笔者的 blog 中提到过一位中科院的研究员，这位研究员在技术水平上可以说是很高的，国际上排名也是前几名，但是，他却总是言而无信。

例如，第一天下午告诉你去做某件事情，第二天上午就问你昨天做了什么，你如实告诉他以后，他会说，他什么时候让你做这件事情了？让你拿出他让你做这件事情的证据，诸如此类之事数不胜数。

2005~2006 年期间与笔者一起在他那里工作过的人员陆续离开之后，2008 年年底发生了一次大规模学生叛逃，他的 18 个硕士博士中 9 个人集体提出换导师，其中两人是 2005 年就跟随他做博士学位研究的博士生。

正因为真正合格的强势项目经理很难找到，所以，我们就提出了另一种项目经理：弱势项目经理。

7.4.3 弱势项目经理

在 IBM 工作的每个人都有两个职称级别，一个是管理级别，一个是技术级别，有些人主要提升技术级别，另一些人主要提升管理级别。在他们的团队中任命的项目经理主要做的就是团队管理和支撑的事情。

弱势项目经理应该如何做呢？笔者曾经在中科院自动化所模式识别国家重点实验室工作了 1 年，刚去时，学生并不配合，认为你没有资格来领导我们，你又不是学模式识别出身的。但是，当笔者离开 1 年之后，有一位湖南的朋友（目前在北美读博士）在网上遇到笔者问，你是不是自动化所那位传说中的项目经理。到现在，那位老板原来的学生和笔者之间的关系都非常好，并且在考虑一起创业的问题。

弱势项目经理的做法和强势项目经理有很大的区别：

首先，他不能使用强硬的口气来指挥团队成员进行某项任务。

其实所有的人都知道领导分配的任务必须完成，但是，在一些情况下大家可能不太乐意做这样的工作，或者工作内容不是他们感兴趣的，于是推托和躲避的现象就会发生。

这时，弱势项目经理应尽可能地采用协商的口气，耐下心来和技术人员聊天，并尽量为技术人员争取应该获得的利益，比如时间、空间、后续的任务分配选择权利，等等。

其次，在探讨和谈话过程中应该体现出自己的判断能力，表现出自己的主见。

虽然是弱势项目经理，但是做事情一定要有主见，主动地推动一些事情的发展，有主见的选择下一步需要进行的任务内容，避开可能出现的风险和不可控因素。

如果你是一名完全没有主见的弱势项目经理，那么技术人员对你不会尊重，只会看不起你。技术人员不会因为你不懂某项技术而轻视一个人，只会因为你没有责任心和主见而看不起这个人。

表现出自己主见的同时，要表现出做出这种选择的依据，该依据要客观，并且大部分都

来源于主要技术人员的建议，这样就能争取到尽可能多的团队成员的认可。

而对于强势项目经理，就可以更多的根据自己的经验和阅历来进行选择，甚至不说明选择的理由也可能会有较多的团队成员认同，因为强势项目经理明白主要技术成员为什么会如此思考，而这一点正是弱势项目经理的不足。

再次，项目中出现偏差和问题时，应勇于主动承担责任。

既然选择了下一步的任务和操作方式，那么，一旦发生了问题，弱势项目经理也应该主动承担相应的责任。

不逃避，不推脱，是一个项目管理者的基本素质，无论你的角色是强势项目经理还是弱势项目经理都一样。这也是给团队中技术人员以信心的根源，只有这样，团队中的技术人员才会信任你，才会跟着你努力完成项目。对于技术人员来说，怕的不是犯错误，而是犯了错误不承认错误，这往往是国内甚至国际上一些曾经凝聚力很好的团队最终溃散的根本原因。

曾经的一个例子：

项目经理在带队成功完成几个项目/产品之后，产生了自大与自满情绪，担心如果自己一旦犯错会遭到他人的轻视，于是当自己真的犯错后就不由自主地考虑如何推卸自己的责任，或者转嫁责任，以维护自己所谓的形象。

结果，团队成员开始不满，却不敢言，这种不满日积月累，就会离心离德，随之开始有人辞职离开。

项目经理为了隐瞒人员离开的真实原因，就开始散播一些言论：谁走了对团队都无影响，团队都能继续存在，甚至过得更好。

然后，金玉其外，败絮往往其内。当原来的核心人员纷纷辞职之后，项目的开发与进度开始出现严重的滞后问题，任务无法完成，最终导致项目以失败告终，项目经理的辞职也就隐瞒了更深层次的问题。

承担责任与否，体现了一名项目管理者的责任心，但是由于企业中很难做到利益与责任的并存，而项目管理者只有责任没有利益，所以，这也是大量项目失败的一个核心原因。

最后，为团队成员争取利益。

既然你是项目经理，那么你就是这个团队的领导，一名负责任的领导，不仅要考虑自己的利益，更要考虑团队成员的利益。哪怕因此可能损害了自己的个人利益，也要为团队其他成员争取利益，因为团队的稳定和项目进展的顺利才是你这个项目经理的最大利益。

一些额外的奖励即使最终争取不到，只要你努力了，团队的其他人员都会看到，他们会默默地支持你，并主动配合你的工作。

如何为团队成员争取利益涉及多方面的因素，部分内容可以参考7.1节中关于利益分配方面的分析。

7.4.4 题外话

对于一般软件企业而言，项目经理的工资要比一般程序员高出很多，由于他们不需要参与开发工作，所以很多对开发兴趣不浓厚的技术人员就特别希望自己能够当项目经理。

我们认为，项目经理待遇高有两个原因：

第一，项目经理需要安抚团队内技术人员的情绪，因此有时需要请他们吃饭，那么这种费用就包含在其中。待遇高是为了什么，就是让你请客吃饭的。

第二，项目经理应该为团队内技术人员承担责任，来自上级领导的指责项目经理应一力承担。项目经理是做什么用的，就是用来承担责任的。

很多企业并没有给项目经理足够的收入让他来如此操作，所以，很多时候，项目经理自己也很难。

笔者在自动化所工作期间，和同事们出去吃饭，只要不是事先说好的哪个人请吃饭，一般都是笔者请吃饭，偶尔有些吃饭的钱可以考虑合理的方式进行报销，其他的都是自掏腰包了。

7.4.5 小结

对于强势项目经理和弱势项目经理之间的做法区别是，强势项目经理可以在一定情况下采用强硬的口气进行命令性任务分配，而弱势项目经理绝对不能如此做。

一名好的项目经理应做到如下4点：

- 客观的评判分析；
- 公正无偏的争议处理；
- 主动承担责任，不推卸责任；
- 为团队成员争取利益。

只要做到了上面这4点，无论你是哪一类的项目经理，都可以让团队成员心甘情愿地跟着你拼命了。

7.5 论本本主义

本节的内容已经在程序员杂志上发表过，但是目前仍然大量存在对产品、技术、标准这3个名词的崇拜和盲从者，所以，在本书写作过程中时候重新对本节的内容进行了编辑和补充，希望警示世人。

7.5.1 2009年的几件事情

2009年发生了如下几件事情：

- 1) 开心告开心。
- 2) 参加了软件营销论坛，听了一场关于国际上软件行业标准制定的“闹剧”。

3) 去帮一位医生的公司做电子病历之类的网络平台,结果,经历了一名英国某非牛津大学 MBA 出来的总经理的很“牛”的管理压制。

4) 模式识别技术的应用之一——森林火灾报警检测方案的最终设计完成。

这几件事情从表面上看,似乎没有什么关联,但是,从本质上它们都包含了如下内容:

- 都是软件产品或者都是针对软件产品的;
- 都存在技术竞争和对立;
- 都需要或者在一定程度上需要一套标准来定义支持。

7.5.2 引发的思考

1. 开心事件

开心告开心的事件让人感觉真的很开心,从2008年开始笔者发现很多老同学和周围的朋友在玩开心网,可是,笔者却一直拒绝去玩,主要是因为比较厌烦开心网的垃圾邮件推广方式。另外,笔者对这类游戏并不感兴趣——很多人刚开始并不承认开心网也是游戏网,他们认为这是一种消遣的娱乐方式。毕竟对于很多人游戏网来说,直接宣传自己是游戏网,会引起很多人的拒绝,而变换一种方式来操作,行游戏之本质,会让很多拒绝游戏的人不知不觉地进入其境地,当他们意识到自己已经是在玩游戏的时候,就只能给自己寻找借口,说自己的网游是不花钱的,云云。

但是随之而来我们看到的开心告开心,以及漫山遍野的开心农场、开心鱼塘之类的产品不断推出,大家不禁开始关注,谁才是正宗的,谁是山寨的?那么随之而来的问题就是:谁的才是标准,谁是跟进的?玩家会问:我们玩谁的游戏最后不会没得玩?开发者会问:我们跟进谁的 OpenAPI 做开发,最后不会做不成?——大家都知道,对于 SNS 来说,只有拥有较多的客户群体,开发者跟进的开发才可能因此赚到较多的收益。

因为所有的人都会有一个共同的默认认知,那就是:标准才能持久,其他所有非标准的东西总有一天要跟着标准进行修改,否则就会被淘汰。

那么,到底标准应该由谁来制定呢?标准制定后,如何才能服众呢?

2. 国际标准如何制定

在2009年的中国软件营销论坛上,有幸听到北京邮电大学的一位女教授讲述了她参加国际软件行业标准制定委员会的工作内容和工作方式。

从她的描述中,我们得知:

第一,绝大多数国家在参与软件标准制定的时候,都是以国家利益为基础进行竞争的,并不是根据是否符合软件开发的实际情况来制定的。

第二,各国之间的利益竞争,从各自的互利(互相支持对方的标准)模式和争议表决方式(大家举手投票表决)中存在更多的是利益,而不是对问题的解决方式。

第三,各个标准组的定义和人员的权限,是根据国家的实力和影响进行划分的。

上面这3点,从商业和国家的利益角度出发,笔者并不表示反对,但是,从一种单纯的

技术角度来看,笔者只能评价其为一场“闹剧”。

由此,我们看到了国际标准到底是什么?是否还值得软件从业人员在日常工作中去追随、跟进、执行呢?

3. 电子病历和模式识别的应用

这两个话题具有很强的相似性,所以放在一起进行分析。

在欧洲举行的一个关于医疗改革方面的全球大会上,美国的一位发言人表示,他们前期资助 1000 万美元的电子病历失败,后面准备投入几亿美元进行新型电子病历的研发。

“到了去年 12 月,圣巴巴拉郡的数据交换计划却悄然落下了帷幕。在耗尽了 1 000 万美元的资助经费后,该地区的医疗机构却看不到这个项目还有什么继续下去的价值。尽管在该区域还有许多医生仍在使用电子记录,但跨诊所共享数据、轻松地跟踪病人治疗梦想却已黯然消退。

.....

美国总统布什在 2004 年曾号召,到 2014 年要让大多数美国人享受电子健康记录。但经过几年的不懈努力,电子记录的使用依然处于落后状态。目前仅有 10% 的医生办公室使用电子记录。医院推广电子病历时,往往还没碰到医疗机构间交换数据的最大难题(尤其是竞争对手之间的数据交换)时,技术本身就已出现了问题,如去年由医疗卫生机构凯泽永久医疗集团(Kaiser Permanente,下称凯泽永久)所运行的医疗数据网络就发生故障中断。还有法律问题、隐私问题、围绕技术的竞争压力问题和投资回报问题等。而数据共享的具体实践也还未在现实世界中获得广泛测试。”

——以上内容引自《美国的电子病历之痛》,出处:信息周刊,作者:赵红权,编译日期:2009-06-08。

在国内,也有众多的公司在进行电子病历的开发,还有一些有前瞻性的医生也在投入自己的精力,希望能够制作出让医生和病患都满意而且使用方便的电子病历,当然,在这个基础上,他们自己也能因此获得不小的回报。

最终的结果是,只能一家作为标准或者修订后作为标准来进行全国甚至全世界的推广,不可能出现多家标准的竞争问题。谁是最后的标准,这将决定谁能获得国家最后的投资和支持,以及将来的所有收益。

同样的事情在模式识别应用的智能视频分析技术上也有体现,只是在这个领域更多的竞争是在国外技术对国内的渗透中与国内技术之间的冲突和对抗。

模式识别技术的实际应用应该是在积累了 20 多年后的 2006 年才开始真正投入的,国外也不过比我们提前了最多 1 年左右的时间(这个数字是在和多家国外厂商进行交流后得出的结论,因为很多宣称已经在海外投入使用的基础数据中的有效信息部分他们都无法提供,只能宣称应用了哪些行业,却无法准确地通过结果数据让我们相信其应用效果)。

2006 年笔者在中科院自动化所工作期间,与公安部一所(专门制定安全相关标准的部

门)进行过多次洽谈,但是迄今为止,仍然没有一个国家标准出台,更不用说什么行业标准。因此很多送去检验的产品执行的都是企业标准,特定环境内达到特定效果的检测,然后很多企业就拿着这个检测报告对外宣称自己能够解决多少实际问题(这些问题其实大都不在评测中限定的特定环境之内)。

那么,我们应该如何看待标准,如何使用标准,标准和技术之间有什么样的对立统一关系,如何在标准的指导下应用合适的技术来完成产品,这是一个十分复杂的话题。

7.5.3 产品、技术、标准

对于一个企业而言,应该如何看待标准,如何使用技术,如何完成产品,如何获得市场的认可,这是一个最终的核心问题。这里我们就分为3个层次对这3个方面进行尽可能详细地阐述。

1. 标准的选择和看待

一般而言,标准分为三个层面:国家标准、行业标准、企业标准。对于不同的标准有不同的限定要求,而对于软件行业而言,我们还可以看到很多世界级标准或者说影响达到了世界层面的标准,如CMM/CMMI系列就是从研究机构推动并逐渐被世界接受的标准。

下面以CMM/CMMI标准来进行分析。

笔者当年所在的公司于2001年就参与了CMM3级的部门评估,于2002年开始编写并发表。“CMM不是认证是评估”,当年我们那位来自印度的CMM的主任评估师一直强调这一点。但是直到现在,我们发现国内还是在说CMM/CMMI认证……诸如:

“截至2003年3月,全国共有近50家软件企业通过CMM认证,其中通过2级的32家,3级9家,4级2家,5级的4家。而全国仅有1400多家软件企业,实施CMM认证的企业比例已经高于世界平均水平。”

“CMM/CMMI认证前的准备工作:

8. 申请CMM/CMMI的认证费用有多大?

关于CMM/CMMI的认证费用的问题很难回答,从上面题目的讲解中你可以看到它主要取决于评估者及所属的机构。”

(因为以上两个是反面题材就不注明出处了)。

此外,CMM/CMMI的评估在SEI的建议是:每半年到1年需要进行一次新的评估,每次的评估都只是告诉这个企业你到评估时为止的软件开发程度可以达到什么级别,并不是告诉人们,评估完成后,这个企业的项目开发程度就一定是什么级别的。每年发布的SEI报告中也会提供,目前为止达到的企业数量和国家分布情况。

那位主任评估师还强调的一点是:“我是来帮你们寻找问题的,……”。找到问题后,他会给我们提供相应的指导建议和策略,所以,一次完整的CMM/CMMI的评估一般都会持续6

个月以上，而不是很短的时间。

当然，从另一个角度来看，这是 SEI 成功的基础，SEI 因为它的这种不断评估不断推动的模式，为它自身和它的评估师也带来了大量的现金收入。

下面我们针对 CMM/CMMI 中的一个关键问题进行分析，看看我们应该如何看待标准。

很多人对 CMM/CMMI 的抱怨在于：文档过多。甚至有人认为 CMM/CMMI 的结果是：“项目经理几乎成了文员，而一名程序员却可能成为文档编写家。所以最终惹得大家都抱怨过程文档太多了，70~80 个模板，哪里记得清楚。”

实际上，CMM/CMMI 从来没有制定一个必需的软件过程框架进行执行，它只是一个抽象层面上的分析结果，团队是否规范，不仅仅是文档，而在于记录和稳定的有效性上。

一般而言，出现这种抱怨结果的企业往往都是采用了 RUP 或者类似的开发过程作为企业软件核心过程来推动自己的软件过程改进时出现的问题。采用 RUP，甚至国外曾经有过的一种更重量级的过程模型 EUP，它们都是在团队层面要求较大的模式下形成的，而不是针对一般性的中小软件企业，而且其产生的环境是欧美社会制度下的软件企业生存环境，这和我国软件企业的生存环境有着很大的差异。同样的，在 RUP 中也给出了裁减建议，但是，即使按照它的裁剪建议去执行，其结果通常地是一般企业不能接受的。如果采用了敏捷开发过程中的一些轻量级方法或者笔者的全程建模方法论，再结合迭代过程模型，可以在很大程度上减少文档的编写，这些方法同样可以让你达到所谓的 CMM/CMMI3 级以上的要求。因为 4~5 级别的内容主要是在组织层面和数据积累层面的提高，不再单纯是团队层面的提高，所以，可以说能够“真正”达到 3 级评估的团队，在团队层面的开发管理上应该是已经成熟了的。

有些企业通过评估的主要目的是拿政府的奖金，这也是可以理解的。标明自己的企业如何紧跟标准，往往只能说明这样的企业在市场或者技术层面的怯懦。就好像一款低档次的汽车上非要贴上一个奔驰的标签，来显示自己的出身高贵。要知道，什么样的标签并不代表的企业有好的效益，欺骗总有现形的那一天。

对于所谓的国际标准，都是需要进行斟酌度量的，如果不是国家要求必须去做，则不要被标准所误导。前面抱怨文档过多的企业，一般都是犯了“本本主义”的错误。总认为别人说的就是圣经，圣经就能解决一切问题。但实际上，并不是这样的，圣经也要拿给懂得它的人来读，来分析，来讲述，才能让更多的人接受，几百年来基督教的传播在这方面显示出了其强大的生命力。同样的，我们也可以看到，仍然有很多教士对福音书和教义的传播是失败的，最后很惨烈，这在基督教的发展史上并不鲜见此类事件的发生。同样的事情，我们也可以从佛教在中国的传播发展过程看到，佛教在汉末唐初时期与道教、儒教进行了几百年的融合和相互学习，最后形成的是很有中国特色的佛教，而不是原始的印度起源的佛教。

如果没有听过 2009 年软件营销论坛的那个关于标准议题的朋友，可以去下载视频资料听一下，因为那位教授很实际很准确同样也很隐晦地告诉了我们，所谓的国际标准都是利益的划分和争议，根本不是以工程实际为基础来看待如何解决问题的标准。也就是说，国际标准制定的终极目标不是为了对软件工程活动进行指导，而仅仅是各国之间的利益

划分。

2. 技术的应用与实践

做好一个产品除了恰当地应用好标准外，还要采用适当的技术和适当的方式进行推动。

从开心告开心和众多的开心农场、开心鱼塘的出现，可以看到同类技术的模仿是非常快的，人们也认为最初的那个开心也是模仿照抄 facebook 的东西，并不是自己所独创的。于是，他的东西也很快被别人所模仿照抄，tencent 现在也在做开心农场之类的小游戏，依托他巨大的用户群推动着他的新业务，同样也给开心网带来了不小的冲击。

技术的应用往往不在于其先进性，而在于其合理性。

但是合理的技术应用通常很难形成较高的技术门槛，容易被仿冒。在目前阶段，信息交流无边界限制的情况下，很难确保你的技术具有足够高的门槛，更多的门槛应该在于新功能、新应用的推出和对用户市场的把握程度。

如果能够在自己的产品被别人效仿之前就推出新的版本和新的应用，那么，别人的效仿也不可能带走你的主要客户群，也就不可能对你产生有效的冲击。这在一定程度上形成了自己的技术壁垒，这样做的根本在于产品规划的合理性和递进性。

完全依靠所谓的高新技术是很难长期占有市场的，更不可能依靠市场人员的单纯推动和关系来完全占有市场，因为客户也在不同产品之间进行比较。

有朋友问过笔者这样的问题：用户对产品的忠诚度到底有多少？

他说，他希望依靠客户对产品的忠诚度来形成自己的市场份额，希望不要被他人的模仿带走客户。这种思考的方式和定位基础是存在问题的；尤其是在互联网应用上会存在更多的弱点，对于互联网应用而言，用户没有多少忠诚度。他们会有一定的产品黏着度，也就是你的产品如果能够获得更多的用户使用，使用的年限越长，那么他的黏着度就会越高。因此，如何长期保持用户的使用，而不是用户的变化和流失，就成了一个十分关键的问题。这才是技术上当你无法形成有效技术壁垒时需要更多考虑的问题。

比如，微软的 Windows 操作系统就是这样吸附用户的，即使这些用户没有购买能力，也要让他们无法从自己的系统轻易离开，因为在微软的操作系统上用户体验做得非常精彩。好像从没有人评价过微软 Windows 系统的技术有多么的强大，反而有人在几年前曾评价说微软 Windows 2000 实现的绝大部分技术其实就是 5~6 年前和 Windows 95 做竞争的 IBM 的 OS/2 已经实现了的，但是，IBM 的 OS/2 完全牺牲了，而当时技术相对落后的 Windows 95/98 却活了下来。这个问题不得不让我们深思。

因此，在没有足够技术壁垒的产品上，产品规划（新版本的不断推出）将是至关重要的问题。在可以形成技术壁垒的产品上，用户体验将是决定你的产品被用户接受的程度和忠诚度。

3. 产品规划和开发管理

说到产品的规划，笔者总是不禁想到 2001 年底在托普成都中央研究院工作期间曾经完成的一套移动增值业务的两年期 6 个产品线的规划。其中有一个很小的功能当时就得到了四

川移动 SP 业务负责人的赞赏，可惜的是托普的目标不在产品上。2002 年初只立了第一个 GPRS Modem 的项目，导致这个规划最后成了泡影。笔者也很快离开了这个没有希望的地方。4 年后，这个精巧的功能送给了北京一个做 SP 业务的朋友，他在山西网通从事 SP 业务。其他功能因为部分是具有时效性，还有一些开发难度较大的，所以就被放弃了。

曾经一个朋友问笔者，中国的软件企业和国外有多大差异？笔者回答说，技术上已经没有多大差异了，主要问题在于管理上的差异。

为什么说技术上已经没有多大差异呢？

这是因为随着互联网的发展，加速了国内外技术的交流。在 20 世纪 90 年代以前我们的技术基本上要滞后于国外 5 年左右，而现在无论从哪个方面来看新技术几乎都是同步发送到世界各地的。

我们主要的缺陷在于管理和人员意识，这是不可能在短时间积累并形成产生效果，因为管理不仅仅是某些人理解了管理的方式和方法，而是在于所有参与人员对管理的认识和执行状态。

从团队的角度看，如果有人无法理解管理上的一些策略和执行方式，那么开发过程的执行就会出现严重的分歧，在项目计划的制订上就会出现很大的争议。

国外用了 50 多年时间发展到现在的管理水平，我们不可能只用 10 年时间就全部理解国外的管理水平。另外，国外的企业和用户状态与我国目前情况有着较大的差异，直接照搬国外目前的管理方法和手段往往是不能立刻获得收效的，只有针对国内的实际情况进行管理模式和管理方法的变更，才能有效地执行。

有一位曾经做过电信老九七工程的人对目前国内行业软件开发评价说：国内目前除了税务、电力行业外，其他行业的信息化水平基本上处于 20 世纪 90 年代的电信信息化状态，用户意识也处于那个状态，在和用户的交流中可以深切地感受到，用户提出的问题都是我们 10 多年前做电信老九七工程时用户问过的同类问题的另一个行业的翻版。

在 2009 年 7 月笔者看到的一期医学信息化的国内刊物上，还在阐述 VPN 技术和局域网等给医疗信息化带来的好处，国内的电子病历和医院的信息化仍然处于一种很基础的状态。

这样的差距，也同样存在于我国和国外的软件开发之中，不是技术的差距，而管理意识上的差距。

另外一个例子：

2004 年底，中国电信集团内部宣布终止 SAP 在中国电信各省市试点公司的 ERP 试点，从此宣布了历时两年多投资过亿的中国电信 ERP 应用结束了。本来作为过渡阶段预计在 2004 年底被 SAP 逐渐取代的电信计划建设项目管理系统（笔者在电信工作时完成的最大的软件项目，其实质就是电信 MSS 系统，也就是 ERP 系统的核心业务层系统），一直继续被使用而存活了下来，到 2010 年，仍然不断地改进和更新。

这就是一个典型的套用国外数学模型直接来约束中国本地企业的案例。同样的 ERP 实施中，只有少数制造业的 ERP 系统得到成功，其他在中国大陆实施的 ERP 系统几乎全面失败。

SAP 当年给中国电信承诺的投标书中，愿意为适应电信需求进行相当数量的二次开发，而这不能解决数学模型层面的差异。

目前国外的 ERP 系统基本上都是国外企业在几十年的数据积累上形成的一套数学模型框架下实现的，而基础数据的不同使得不同地域不同行业的企业产生了对同一类别数据的不同应用需求和响应结果，而采用同一个数学模型只能导致同一类别数据产生同一类别的响应结果。于是冲突就在最底层发生了，而进行宣传的市场人员和一般的管理人员并不明白其根源的差异性，于是盲目地上马和投资就这样带来了不可磨灭的损失和凄惨后果。

这样的失败源于需求方的无知和厂商利益驱动的结果，厂商不负责任的推广方式谋求自己最大利益的实现，而需求方在只看到厂商在国外应用的良好结果，却不知道其本质差异，于是就被蒙蔽了，形成了客观存在的本本主义的意识形态。

7.5.4 软件开发本本主义

在软件开发的实践过程中，我们可以看到很多的本本主义的实例，这些本本主义者不断的打击着开发者的积极性，不断为自己的利益而奋斗拼搏，不断地将一个一个的项目和团队送入火葬场，不断地把公司从兴盛推向毁灭之路。

1. 名词崇拜者

在 2009 年 4 月 2 日，一个 QQ 上的人提出了下面的问题：

某人 17:39:29

hi

某人 17:39:50

有没有试过用敏捷的方式来项目啊？

青润 17:40:26

??

某人 17:41:01

比如极限编程。

青润 17:41:12

我不懂什么是敏捷，我只知道什么能给我解决问题。

青润 17:41:20

什么能解决问题，我就用什么。

某人 17:41:25

惊讶的表情。

从语气和对话中，笔者感受到一种强烈的名词压抑感，笔者真的希望学计算机的和做软

件开发的朋友们能够灵活一些，不要那么多的本本主义，名词崇拜派（笔者给本本主义者的命名）。这类的话题，在以前曾经提到过，比如那篇被引用上百或者几千处的文字：《精通它，才能毁灭它》（没看过的朋友直接搜索这个关键词，就能找到）。

网络上热捧如潮的概念和名词把一个个学习计算机的孩子推向了这个概念和名词的创造者，让他们成为这些创造者攫取利益的牺牲品。使得他们只知道概念，只知道提出概念，却不知道如何解决问题。软件科学的实质是软件工程学科的实现，我们是做工程的，写软件的，不是做学问的，我们不需要看到名词就想它如何深奥、如何高贵，来源于欧罗巴、美利坚，多少个世界 500 强在用！我们需要的是解决问题。

让我们的项目做完，让我们把产品交付给客户后，客户不骂我们或者不多骂我们，至少不会说，这是什么垃圾！这么简单的功能却实现成这种样子。

这么说，就是不想让大家浪费太多的时间在国内的商业企业为了其商业目的推动的所谓名词面前匍匐倒地，顶礼膜拜。

2. SOA 版本

大家应该记得，一些商家（都是国际大公司）曾经争论过关于 SOA 版本号的问题，如 SOA1.0，SOA2.0。由此推及，一个企业内是不是还要定义 CTO0.9，CEO1.1 才对？

要明白，商业推动利益下产生的大多数名词，都只是那些排名前列的公司做给他们的股东看得，不是给技术人员用的。从上面有关国际标准如何制定的内容中可以看到，国际标准的制定也是基于利益的，这样的利益在一定层面上来看，不是技术人员应该追求的范畴。

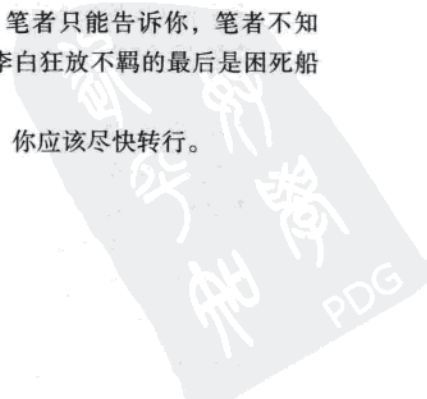
对于技术人员而言，需要的是如何解决问题，以及解决问题的具体方法和过程。

7.5.5 最后的分析

几十年前，毛主席就说过这个问题，如今，本本主义似有愈演愈烈之势，好好读读老人家的著作《反对本本主义》，那是多么经典的一篇文章。

最后，拜托各位，不要问笔者，你知道不知道某个概念。笔者只能告诉你，笔者不知道，笔者是文盲。谢谢了——诙谐的背后是心里最深的痛楚，李白狂放不羁的最后是困死船舱的凄惨结局。

技术人员，就要多一些务实，少一些轻浮，如果没有兴趣，你应该尽快转行。



第 8 章

技巧和相关文档

本章主要介绍的是笔者在开发过程中积累的一些经验和方法，其中包括使用 Rational 工具过程中的一些小的经验和技巧，还有一些关于软件工程远景的展望。虽然这些技巧对于软件工程过程的全程实现无关紧要，但是采用这些技巧却能大幅度地提高开发效率，减少一些不必要的时间浪费。



8.1 cat 包的修改

8.1.1 示例环境

这里介绍一种笔者摸索出来的简捷构建时序图的方法：在 Rose 分离出来的 cat 包中都有一个标识符，修改这个标识符，Rose 就可以把本来不是自己直接分离出去的 cat 包当做是自己分离出去的。

例如，在前面提到的两个包：iar. cat 和 irm. cat。

Rose 中截图获得的两个包的内容如图 8-1 所示。

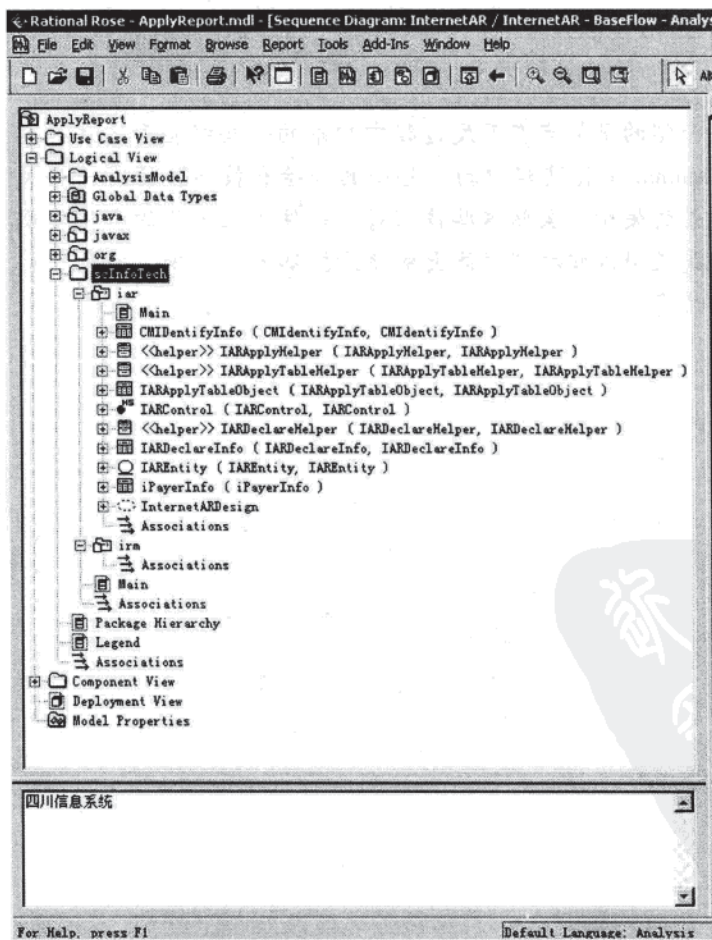


图 8-1 Rose 模型截图信息之原图

iar.cat 的内容如下:

```
(object Petal
  version      45
  _written     "Rose 7. 6. 0109. 2314"
  charSet      134)

(object Class_Category "iar "
  is_unit      TRUE
  is_loaded    TRUE
  quid         "3D293F0802E0"
  documentation "网上申报"
  stereotype   "subsystem"
  exportControl "Public"
  logical_models (list unit_reference_list
    (object Class "IARApplyHelper"
      attributes (list Attribute_Set
        (object Attribute
          tool "Java"
          name "BodyDanglingComment"
          value "")
        (object Attribute
          tool "Java"
          name "GenerateFinalizer"
          value FALSE)
        (object Attribute
          tool "Java"
          name "GenerateStaticInitializer"
          value FALSE)
        (object Attribute
          tool "Java"
          name "GenerateInstanceInitializer"
          value FALSE)
        (object Attribute
          tool "Java"
          name "GenerateDefaultConstructor"
          value TRUE)
        (object Attribute
          tool "Java"
          name "ReadOnly"
          value FALSE)
        (object Attribute
          tool "Java"
          name "Final"
          value FALSE)
        (object Attribute
          tool "Java"
          name "Static"
          value FALSE)
        (object Attribute
          tool "Java"
```

```

        name    "Strictfp"
        value    FALSE)
(object Attribute
  tool    "Java"
  name    "GenerateFullyQualifiedExtends"
  value    "")
(object Attribute
  tool    "Java"
  name    "GenerateFullyQualifiedImplements"
  value    "")
(object Attribute
  tool    "Java"
  name    "ConstructorIs"
  value    ("Ctor_Set" 62))
(object Attribute
  tool    "Java"
  name    "GenerateCode"
  value    TRUE))
quid    "3D29668500BD"
documentation
|<p>Title: 四川信息系统网上申报系统 </p>
|<p>Description: </p>
|<p>Copyright: Copyright (c) 2002 </p>
|<p>Company: scInfoTech </p>
|@ author 青润
|@ version 1.0
|申报助手类
.....
.....
.....
.....

(object ClassView "Class" "Logical
View::scInfoTech::iar::IARApplyHelper" @ 183
  ShowCompartmentStereotypes  TRUE
  location    (880, 432)
  font    (object Font
    size    10
    face    "宋体"
    charSet    134
    bold    FALSE
    italics    FALSE
    underline    FALSE
    strike    FALSE
    color    0
    default_color    TRUE)
  label    (object ItemLabel
    Parent_View    @ 183
    location    (706, 407)
    fill_color    13434879
    nlines    1
    max_width    348

```



```

        justify    0
        label      "IARApplyHelper")
    stereotype    (object ItemLabel
        Parent_View @ 183
        location   (706, 366)
        fill_color 13434879
        anchor     10
        nlines     1
        max_width  348
        justify    0
        label      "< < helper > >")
    icon_style     "Icon"
    line_color     3342489
    fill_color     13434879
    quidu          "3D296685008D"
    width          366
    height         152
    annotation     8
    autoResize     TRUE))))))

```

irm. cat 的内容如下：

```

(object Petal
  version 45
  _written "Rose 7. 6. 0109. 2314"
  charSet 134)

(object Class_Category "irm"
  is_unit      TRUE
  is_loaded    TRUE
  quid         "3D76D0D3000F"
  stereotype   "subsystem"
  exportControl "Public"
  logical_models (list unit_reference_list)
  logical_presentations (list unit_reference_list))

```

注意上面加粗并带下划线的内容。

8.1.2 操作步骤

- 1) 先从 Rose 中将这两个子包卸载，如图 8-2 所示。
- 2) 将下面文件内容中加粗带下划线部分的内容互换，并将文件名互换。
如果不互换，Rose 将会报错。
- 3) 修改两个文件中的部分内容。

iar. cat 修改后的内容如下：

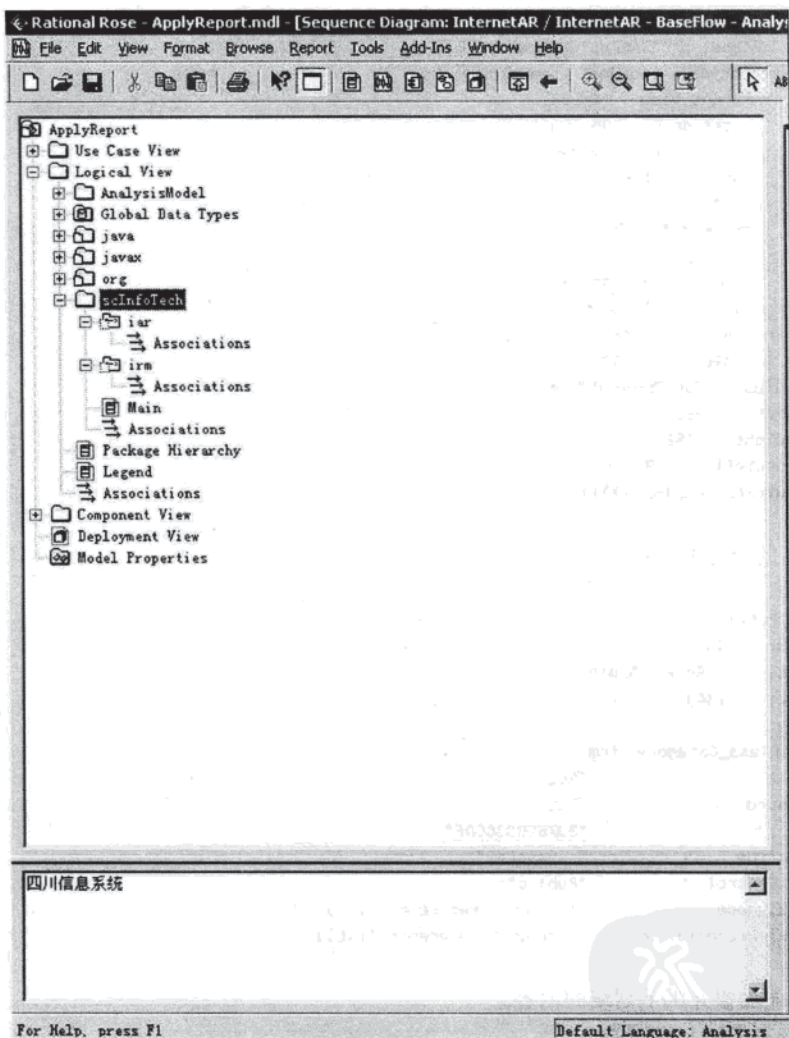


图 8-2 Rose 截图信息之卸载图

```
(object Petal
  version      45
  _written     "Rose 7.6.0109.2314"
  charSet      134)
```

```
(object Class_Category "iar"
  is_unit      TRUE
  is_loaded    TRUE
  quid         "3D293F0802E0")
```

```

stereotype      "subsystem"
exportControl   "Public"
logical_models  (list unit_reference_list)
logical_presentations (list unit_reference_list))

```

irm. cat 修改后的内容如下:

```

(object Petal
  version 45
  _written "Rose 7.6.0109.2314"
  charSet 134)

(object Class_Category "irm"
  is_unit TRUE
  is_loaded TRUE
  quid "3D76D0D3000F"
  documentation "网上申报"
  stereotype "subsystem"
  exportControl "Public"
  logical_models (list unit_reference_list
    (object Class "IARApplyHelper"
      attributes (list Attribute_Set
        (object Attribute
          tool "Java"
          name "BodyDanglingComment"
          value ""))
        (object Attribute
          tool "Java"
          name "GenerateFinalizer"
          value FALSE)
        (object Attribute
          tool "Java"
          name "GenerateStaticInitializer"
          value FALSE)
        (object Attribute
          tool "Java"
          name "GenerateInstanceInitializer"
          value FALSE)
        (object Attribute
          tool "Java"
          name "GenerateDefaultConstructor"
          value TRUE)
        (object Attribute
          tool "Java"
          name "ReadOnly"
          value FALSE)
        (object Attribute
          tool "Java"
          name "Final"
          value FALSE)
        (object Attribute

```



```

tool      "Java"
name      "Static"
value     FALSE)
(object Attribute
tool      "Java"
name      "Strictfp"
value     FALSE)
(object Attribute
tool      "Java"
name      "GenerateFullyQualifiedExtends"
value     "")
(object Attribute
tool      "Java"
name      "GenerateFullyQualifiedImplements"
value     "")
(object Attribute
tool      "Java"
name      "ConstructorIs"
value     ("Ctor_Set" 62))
(object Attribute
tool      "Java"
name      "GenerateCode"
value     TRUE))
quid      "3D29668500BD"
documentation

```

|<p>Title: 四川信息系统网上申报系统 </p>

|<p>Description: </p>

|<p>Copyright: Copyright (c) 2002 </p>

|<p>Company: scInfoTech </p>

|@ author 青润

|@ version 1.0

|申报助手类

.....

.....

.....

.....

```

(object ClassView "Class" "Logical
View::scInfoTech::iar::IARApplyHelper" @ 183
  ShowCompartmentStereotypes  TRUE
  location    (880, 432)
  font        (object Font
    size      10
    face      "宋体")
  charSet     134
  bold        FALSE
  italics     FALSE
  underline   FALSE
  strike      FALSE
  color       0
  default_color  TRUE)

```

```

label (object ItemLabel
  Parent_View @ 183
  location (706, 407)
  fill_color 13434879
  nlines 1
  max_width 348
  justify 0
  label "IARApplyHelper")
stereotype (object ItemLabel
  Parent_View @ 183
  location (706, 366)
  fill_color 13434879
  anchor 10
  nlines 1
  max_width 348
  justify 0
  label " << helper >> ")
icon_style "Icon"
line_color 3342489
fill_color 13434879
quidu "3D29668500BD"
width 366
height 152
annotation 8
autoResize TRUE))))))

```

8.1.3 结果

加载到 Rose 中, 就可以看到最终的结果了, 如图 8-3 所示。

注意:

上图中可不是简单地改名, 而是对 cat 包文件内部信息的直接修改。

下面再用一张图来说明这种方式的好处。

将一个 iar. cat 包修改后添加到 irm. cat 包的位置上, 然后, 将两个包同时加载, 如图 8-4 所示。

8.2 ClearCase 安装指南

网上有很多人都在问 ClearCase 如何安装, 而且 ClearCase 不仅使用起来比较困难, 安装的过程难度也大。

本节的内容是笔者 2001 年 4 月在深圳安装正版 ClearCase 的过程, 期间曾和 Rational 中国公司的技术支持进行过多次交流后才完成整个安装过程的。

8.2.1 概述

安装 ClearCase 的时候有一些特殊的要求, 因此很多人不会安装, 也就造成了很大的麻

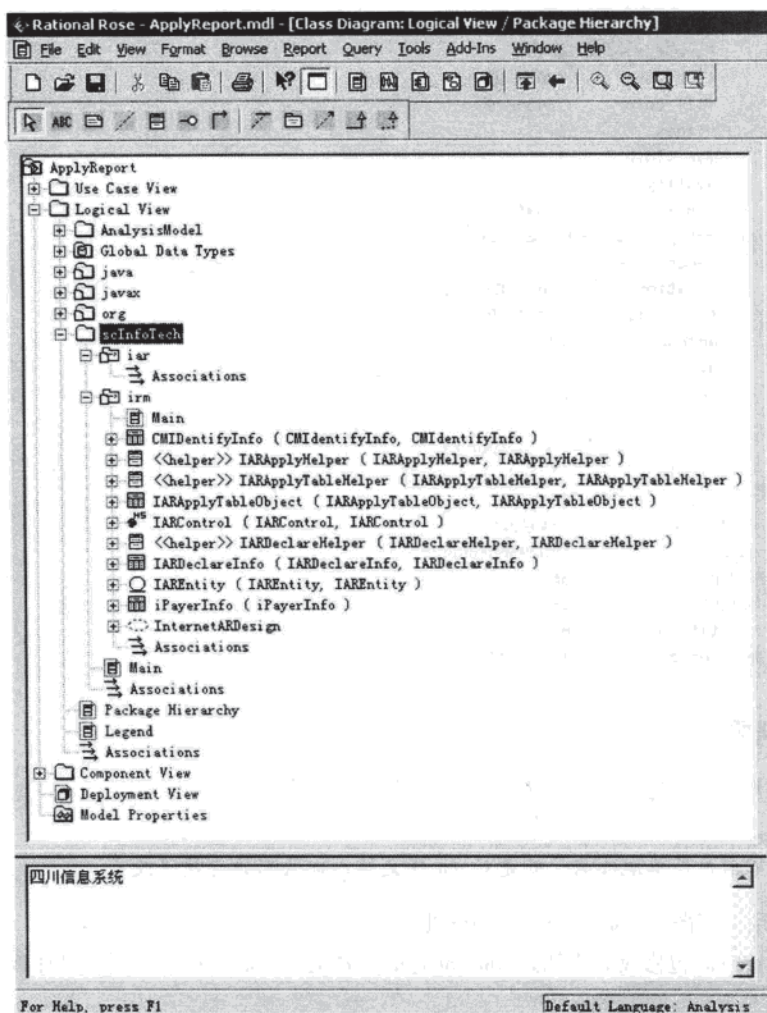


图 8-3 交换后的模型

烦。下面就根据笔者安装 ClearCase 的过程来描述一下如何安装 ClearCase，以及如何安装 License。

8.2.2 安装前的准备

安装 ClearCase 有一些环境上的要求。

1. 硬件上的要求

硬件方面主要是服务器的安装，这里不需要做更多的描述，只需要注意两点：

- 1) Rational 公司建议要有 512MB 以上的内存，实际上 256MB 就可以了，当然是越大



图 8-4 修改后的 Rose 图

越好。

2) 硬盘剩余空间越大越好, 因为任何的版本控制和管理都要有大量的备份, 这也就造成了对存储空间的非常要求。

一般来说, 对于一个 10 人参与, 时间为 3 个月左右的项目, 至少需要 2GB 左右的硬盘空间, 其他的依次类推。

2. 软件上的要求

软件方面则有一些特殊要求:

1) 应该选用 Windows 的服务器版软件, 最好是 Windows 2000 Server, 如果安装的是 NT4

Server, 那么需要打上 Service Pack 6 以上的补丁才可以继续。

2) 这台服务器必须是主域控制器。

3) 最好使用固定 IP, 或者在这台机器提供 ClearCase 服务的时候应该拥有同一个 IP (不能变化, 否则, License 就会出现问题的, 服务也就无法正常提供了)。

4) 如果是 Windows 2000 Server, 则需要安装 Active Directory, 否则, 会有一些麻烦。

5) 每一台客户端机器都要登录到该 ClearCase 域, 成为域中的一个成员。

8.2.3 配置安装

安装 ClearCase 之后, 保持默认设置即可, 不需要做更多的修改。但要注意, 在安装结束后, 系统会要求重新启动计算机。重新启动后, 系统会出现服务未启动的错误——这是很正常的, 否则, 才是不正常。因为此时, ClearCase 组中唯一用户没有足够的权限来启动该服务, 所以, 才出现了这个错误。

登录系统后, ClearCase 会弹出一个界面要求对 ClearCase 进行配置, 在进行配置之前, 首先为客户端授权。

在打开的“Active Directory 用户和计算机”窗口中, 可以看到一个特殊的组和一个名为 clearcase_aldb 的用户, 如图 8-5 所示。

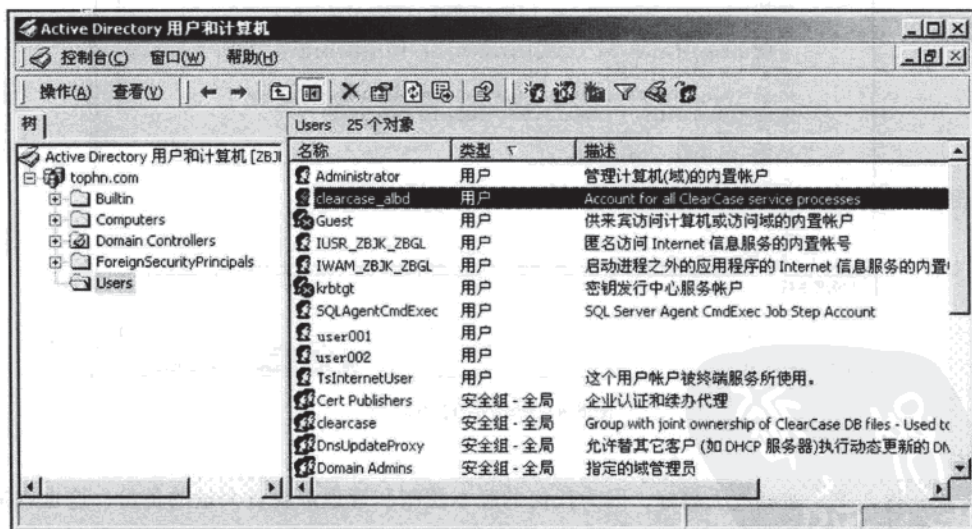


图 8-5 ClearCase 用户

仔细查看这个组和这个用户的特点, 后面将会用到。

如图 8-6 所示, 显示了 ClearCase 用户的属性。

图 8-7 中显示的是名为 clearcase 的特殊安全组。

ClearCase 组的常规属性如图 8-8 所示。

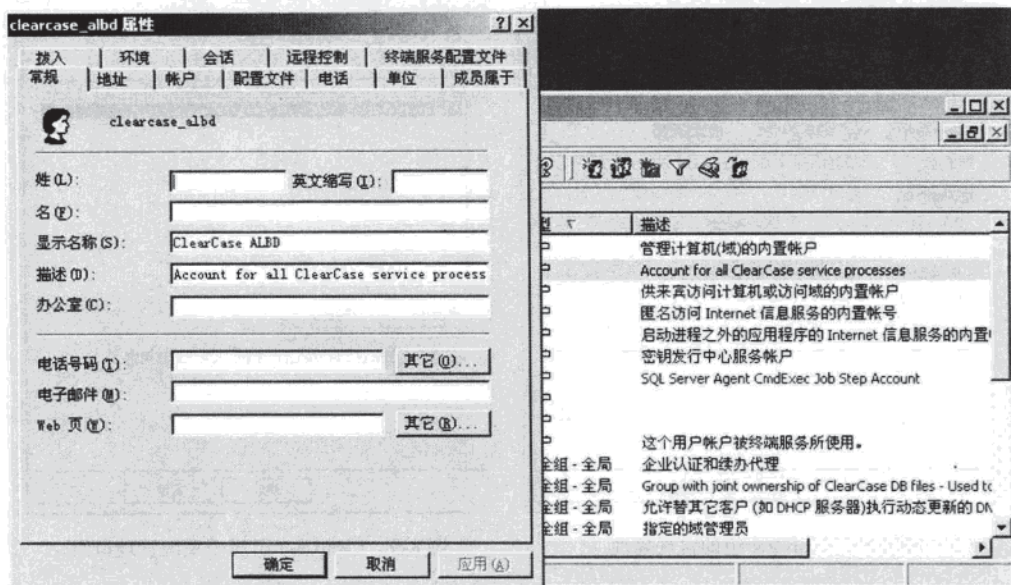


图 8-6 ClearCase 用户属性



图 8-7 ClearCase 组

ClearCase 组属于系统管理员组，如图 8-9 所示。

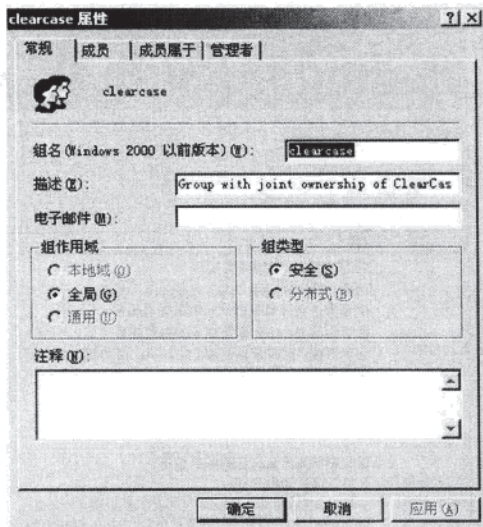


图 8-8 ClearCase 组的常规属性

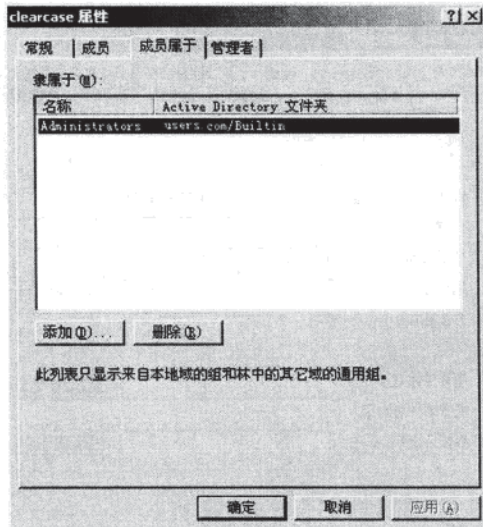


图 8-9 ClearCase 组属于系统管理员组

ClearCase 成员属性如图 8-10 所示。

从图 8-10 中可以看出：用户 clearcase_ aldb 就是 clearcase 这个组的。同样的，也可以把系统管理员身份添加到这个组中使用系统管理员来启动 ClearCase 的服务。

另外，其他使用 clearcase 客户端的人必须使用这个组中的用户名来登录 ClearCase 域，这样才可以使用 ClearCase 的服务。

进行完这样的授权操作后，就可以开始配置 ClearCase 服务了。

注意：

在进行授权以前，可以提供多个这种客户端的用户安装，这个数量可以多于 ClearCase 的 License 的数量，不过，这需求申请动态 License，然后才可以采用这种分时段的方式进行使用。



图 8-10 ClearCase 成员属性

8.2.4 关于 License 的申请安装

接下来的工作是，获得所安装的服务器的 HostID，然后填写 Rational 公司的一个 License 申请表，并将申请表附加到邮件中，发送给 Rational 公司申请 License 序号。

在回复的邮件中，可以找到所需的 License 字符串，将这部分字符串粘贴到 License Server 中，就完成了 License 的安装，然后可以正常地启动 Clear Case 服务了。

注意：

由于服务器本身要占用一个 License，所以，如果想使用 Clear Case 进行配置管理，最少要购买两个浮动 Licenses 才够用。记得在 2001 年一个浮动 Clear Case 的 License 的价格是 4 万多元人民币。

8.3 关于 CMM 的一些观点

本节将站在较为客观的角度来解释一些 CMM 中的常见概念和问题。其实，这些概念/问题应该是源于软件工程的，之所以将本节标题定为“关于 CMM 中的一些观点”，主要是因为这些观点在我国首次出现或者成为热点而被关注时，大都是在与 CMM 相关的讨论中形成的。

说明：

本节中的大部分内容都曾发表在 2002 年的《基于 Rose 的全程建模实例》中。

8.3.1 SQA 和 SQC

SQA，英文全称为 Software Quality Assurance，中文翻译为“软件质量保证”。SQA 是 CMM2 级评估中的一个 KPA，通过图 8-11 中 SEI 发布的评估结果统计数据可以看到这个 KPA 的相对难易程度。

说明：

KPA，英文全称为 Key Process Area，中文翻译为“关键过程域”。CMM 中共有 18 个 KPA，分别是：RM、SPP、PTO、SSM、SQA、SCM、OPF、OPD、TP、ISM、SPE、IC、PR、QPM、SQM、DP、TCM、PCM。

图的统计数据是基于 521 次 CMM 正式评估的结果得到的（这也是到本书 2004 年第 1 版完成之日，SEI 所发布的最新统计数据）。由此可以看出，SQA 是 CMM2 级中实施难度最大的一个 KPA。

SQC，英文全称为 Software Quality Control，中文翻译为“软件质量控制”。

这里将不对这些概念做更多的解释，只介绍软件质量保证和软件质量控制之间的关键性差别。

软件质量保证在于审核软件实现的全过程，通过对软件工程过程是否符合公司的规定和要求进行审核和监督，并将审核结果提交公司高层领导，让公司高层领导可以详细了解项目

Key Process Area Profiles - 1 Organizations Appraised at Level 1

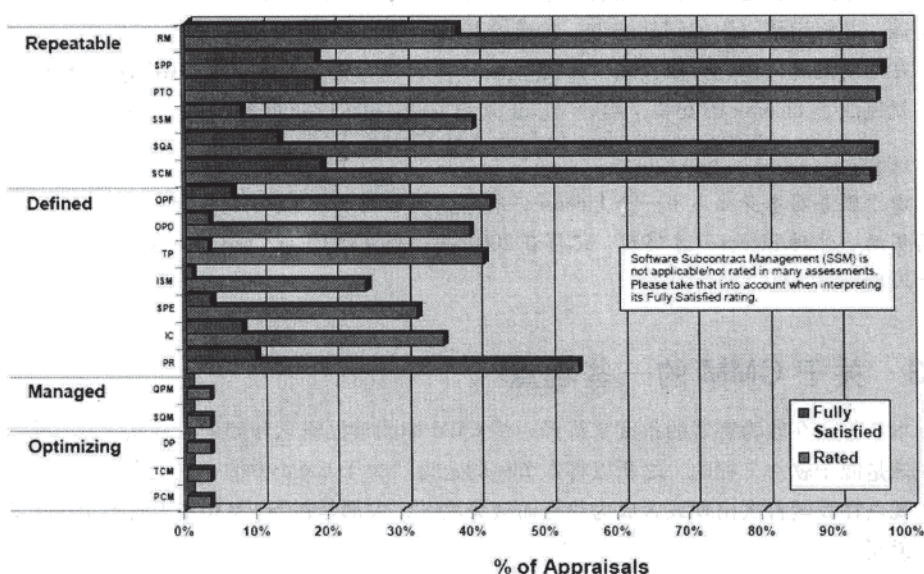


图 8-11 2003 年 4 月 SEI 发布的评估结果统计数据汇总表

开发的状态和过程；通过与项目组的沟通来解决工程过程执行中发生的不符合公司规定的过程和操作方式。

软件质量控制在于审核软件实现的结果，通过测试和检查的方式对软件项目某一阶段的结果进行管理，通过与项目组的沟通和交流来解决这些过程执行结果中发现的问题。

软件质量保证与软件质量控制之间具有以两个相似之处：

- 1) 两者都是为了帮助项目组更好地在公司的有关规定下完成软件项目开发的工作。
- 2) 在一些特殊情况下，两者都需要高层领导的介入才能解决问题。

它们的不同之处在于，软件质量保证不对项目组直接产生任何影响，仅对于项目的执行过程是否按规定进行审核、检查，或者检查中发现的问题将依附于软件实现的大前提，或者有可能一些不规范的操作会更有利于项目的开展时，需要高层领导的直接协调来解决。通常，SQA 的工作人员会被看做是项目组的对头，或者认为是高层领导派来的“间谍”。

而软件质量控制将对项目组产生直接的影响。软件质量控制过程中所发现的任何问题，项目组必须给予相应的解决，否则，将无法进行项目的验收，也就无法完成并提交给用户使用。一般都不需要高层的介入，项目经理对此也会相当关注，因为他知道如果不能及早发现并解决的问题，一旦提交用户使用后发现问题，结果会更严重。

8.3.2 CMM, 评估、认证

通常人们见到的认证太多了, ISO 认证, MCS D 认证, SCJP 认证, 等等。只要有人过来审核都认为是认证。其实不然, CMM 并不是这样, CMM 只有评估, 没有认证。

CMM 的评估是一个持续不断的过程, CMM 的目的是为了帮助目标企业改进其软件开发过程, 而不是简单的认证, 因此, 每过一段时间就需要进行重新的评估, 如果再次评估时没有达到上次评估的要求, 则在 CMM 的级别发布中会修改其级别。

8.3.3 只对组织中的某一个部门进行评估

CMM 只对组织^①中的某一个部门评估, 不会对整个组织进行评估。

CMM 的评估只是针对部门, 同时在 3 级以上才需要组织提供一些环境和配套措施, 但它并不是对整个组织的评估。

有些人经常会说: 全公司通过了 CMM 的某种“认证”, 这本身就是一种错误说法, 给人一种误导, 也是对 CMM 的一种形象损害。

说明:

本书在这里的评论只是一种基于学术观点和软件工程领域的知识性评论, 如果有公司因此指责笔者, 笔者将一概不予理睬。毕竟商业行为和学术观点是有着很大差异的。

8.3.4 评估是反复不断的

每次评估, CMM 的主任评估师都会说:

“我是来帮助你们改进过程的。每过一段时间, 我们都需要对一下你们目前的开发过程所处的状态进行评估, 并针对缺陷提出我们的意见和建议以供参考, 这有助于改进你们的开发过程, 提高软件开发管理水平”, 等等。

因此, 按照规定每过半年, 该部门的开发过程状态就需要进行一次重新的评估, 以确保其项目仍然能达到此前评估中认定的级别, 检查该部门是否出现新的问题或者其级别是否已经有所提高。

如果过了期限而没有参加评估, 则不会被认定仍然具有该资格。在 SEI 那里记录的也只是曾经通过评估的部分, 并不能说明现在还拥有资格的有哪些部分, 所以, 在查看 CMM 证书时一定要注意。

8.3.5 关于 CMM 过级时间的个人观点

在 CMM 的 5 个级别中, 有效的其实只有 4 个: CMM2、CMM3、CMM4、CMM5。这 4 个级别中 2 级和 3 级的差别并不大, 通过它们的时间有可能是一样的, 因为它们不要求更多的

① 一般来说概称一个公司、一个集团等为一个组织。每个组织下面分多个部门。

积累。

一般来说,如果说得难听一些:一个公司只要投入到位,很多东西都可以赶制或者伪造出来,而且这些“东西”完全可以骗过评估师的眼睛。

例如:在3级中主要的要求是建立组织级的很多东西,其中最关键的是组织级的软件开发过程,而这个过程是可选择的。比如直接借鉴 RUP 这个可以达到 CMM3 级的软件工程过程中的很多东西,就可以使你能较快地达到目的。

而在4级的评估中,它需要大量项目真实数据来形成统计表,如果要这些统计数据有效,至少需要6个以上的项目按照公司组织级的开发过程进行研发,即使每次两个项目并行进行,那么也需要1年多的时间,否则,提供的数据很可能是假的。如果没有这么多的项目,或者无法并行进行,那么至少需要两年的时间才可能完全通过。

对于5级,由于笔者没有实践,所以考虑得也很少,此处不做过多评论。不过,有一种观点认为:如果不是造航天飞机之类的项目,就根本没有必要通过 CMM5 级。

在2003年4月 SEI 发布的 *Process Maturity Profile—Software CMM® CBA IPI and SPA Appraisal Results 2002 Year End Update* 一文中第29页,笔者看到了可以证明笔者前面观点的内容,现引用原文如下:

For organizations that began their CMM - based SPI effort in 1992 or later, the median time to move from:

- maturity level 1 to 2 is 22 months
- maturity level 2 to 3 is 21 months
- maturity level 3 to 4 is 25 months
- maturity level 4 to 5 is 15 months

由上面可以看出 SEI 对于通过 CMM 评估的时间统计结果。由于上述结果是根据各种不同的级别完全重新实施项目并达到评估要求的结果所需要的时间,所以,这里和笔者的观点有很多相似的地方,诸如:从3级到4级的过级时间至少需要两年左右 (maturity level 3 to 4 is 25 months)。由此可见,几个月就能通过跨级评估的企业实际上是不可能达到实际水平。

说明:

SEI 关于 CMM 的评估报告一般是每年的3~5月份到五月份发布截止到前一年年底的统计信息,每年的8~10月份发布截止到当年上半年的统计信息。历史上发布信息的时间如下: April 1994、October 1994、April 1995、November 1995、April 1996、November 1996、May 1997、October 1997、May 1998、December 1998、March 1999、August 1999、March 2000、August 2000、March 2001、August 2001、March 2002、August 2002、April 2003。

关于这类信息，可以从 SEI 的网站上搜索：

<http://www.sei.cmu.edu>

这份评估状态报告文档的下载地址是：

<http://www.sei.cmu.edu/sema/pdf/SW-CMM/2003apr.pdf>

8.4 什么是系统分析

人们通常认为，在软件工程领域系统分析员应该是个非常高深的角色，甚至很多系统分析员在一些大公司的待遇比部门经理还要高。所以，这也成为很多程序员人生中的一个目标。

8.4.1 定义

那么，什么是系统分析呢？系统分析的工作在最初的定义中是指，系统分析员能将用户的原始需求转换成程序员可以理解的形式，由程序员来进行编码实现。

系统分析员一般要完成3项任务：

- 需求调研：带领需求调研人员与用户打交道，以完整获取用户的需求。
- 需求分析：分析用户的需求，同时不断与用户进行交流，形成需求的稳定版本——即通过对需求的分析获得系统整体架构，然后指导分析人员构建完成分析模型。当然这也是需求管理的一个重要步骤；
- 设计：指导设计人员熟悉分析模型，并据此完成设计模型的开发工作，然后交给程序员进行编码实现。

8.4.2 现实状况

在实际工作中，一般开发人员都是一人兼多职，每一个人几乎都是从需求调研开始一直做到编码结束。而项目经理则更要求是多面手，从项目管理到需求、分析、设计、编码，甚至要求其无所不会、无所不能。

因为项目经理太忙没有时间制订计划，往往会造成工作失去计划，或者只制定了一个形式上的计划，使得项目计划本身的可操作性非常差。

另外，因为项目经理还要参与开发，使得他没有精力来进行项目过程的监控管理，没有时间跟踪其他开发人员的开发进度，或检查其他开发人员开发结果的有效性，最终导致整个项目的完全失控，或者远超时限的延期。

8.4.3 一些建议

对于一些比较大的工程项目，或者一些可以进行规范化管理的软件公司，这里给出如下建议：

如果要进行真正的工程化开发，在软件成为一个产业的时候，2.3节中提到的一些原始

方式就已经不可用了。因为它使得很多高素质的人员不得不从事一些比较低级的工作，无法更好地发挥一个高层次人才的作用，造成了很大的资源浪费。

比如说：写一个简单的递归调用，完成一个基本数组的增加。

这样的方法，目前每一个开发人员都可以做到信手拈来，但是做这样的事情往往是在浪费时间。因为系统分析员也不得不从事普通的编码工作，来编写这样的简单过程，项目组应该让他们把自己的精力集中在一些难点上，一些一般的开发人员无法解决的问题上。

现在，开发人员既要了解业务，又要熟悉开发，这的确是很麻烦的。如果项目组进行了人员的职能划分，这样就可以大量减少开发人员对业务的了解，而提高开发效率，另外一部分人（下面的需求人员）专门从事需求工作来对客户的业务进行了解。

在进行分析模型开发工作时，建议由系统分析员带领需求人员进行绘制，设计人员参与讨论——主要是为了让设计人员明白需求中一些需要特殊注意的问题，了解需求中与自己相关业务的基本过程。

如果一个公司的职能划分足够明确清晰，并且开发人员素质达到了一定的层次，分析模型由需求人员来绘制即可，设计人员只需要负责开发设计模型就可以了。

8.5 项目计划如何做

8.5.1 什么是计划

“计划”是一个多涵义的术语。计划的本意是人们对事业的未来发展所作的部署和规划。莫里斯·博恩斯坦曾给计划下了一个不褒不贬的定义：

“计划”是未来行动的方案，它包括3个主要特征。

- 1) 它必须与未来有关。
- 2) 它必须与行动有关。
- 3) 必须有某个机构负责促进这种未来行动。

作为一种日常行为，计划可以看做是一般的“有意识、有目的”的活动，但作为在学术上有价值的计划概念，显然就不能停留在这个层次上了。

百度百科中计划的定义工作或行动以前预先拟定的具体内容和步骤。

管理学中计划的定义：确定组织未来发展目标以及实现目标的方式。

计划是一个使用得非常广泛的用语，在不同场合的含义可能不完全相同。有人给计划下的一个“简明的、不带褒意或贬义的定义”是：“计划”是未来行动的方案。

计划也是软件开发过程中的一项重要活动，软件开发过程中到处都充斥着计划活动的存在，比如：测试计划、软件开发计划、配置管理计划、风险管理计划、需求管理计划等。

8.5.2 评价你们的计划状态

很多书中和资料中都可以查阅到计划的制订和执行过程以及相关的内容,但是,在软件开发中,公司的计划活动做得如何呢?我们不妨做一个评估,看看公司的计划是否真得有效果:

- 1) 公司的项目有计划吗?
- 2) 公司的项目开发计划有哪些? 只有一个整体计划,还是有其他计划?
- 3) 公司的项目计划中包含哪些内容?
- 4) 在按照模板填写计划时,是否感觉有很多内容总是无法下手?
- 5) 在计划的执行过程中是否会遇到与计划不一致的地方? 处理这些不一致的地方时,选择的操作手段是否合理? 有没有一种用来评价手段是否合理的办法或者标准?
- 6) 如果计划发生变更,如何操作? 修改、评审之间的过程关系是怎样的?
- 7) 公司的计划变更频率有多大? 或者说一个项目执行过程中会发生多少次变更,每次变更都修改计划吗? 当有变更时,是否会不修改计划而继续执行,甚至最终抛弃计划?
- 8) 项目结束时是否会审核实际执行过程与计划的一致性问题? 它们真的能够保证一致吗?
- 9) 对项目计划执行过程中发生的各种情况是否有统计数据可供后续计划制定和项目执行时参考? 这些数据存放在哪里? 是否是有效的? 如何保证它们有效性?

获得上面9个问题的答案。下面我们来看看一个计划应该如何制定和执行,什么样的计划和执行才能起到作用。

8.5.3 项目中的计划

这里只考虑整个项目计划,不考虑测试计划、质量保证计划等非整体项目时间任务安排的计划。

项目中的计划大体上可以分为整体计划和阶段计划。项目的整体计划一般来说只有一个;阶段计划包括里程碑计划、月计划和周工作计划;里程碑计划对于小项目(2~3个月内的项目)一般不需要考虑,月计划在一些控制较好的项目中只是作为给领导汇报的材料而已,周工作计划则是具体执行的细节计划。

这里只讨论最常用的整体项目计划和周工作计划。

1. 整体项目计划

项目整体计划一般是对项目的全过程进行计划的起始。中国有句古话:一年之计在于春,一日之计在于晨。这里说到的就是计划的问题,一年之计可以类似于把一个项目的整个过程对应到一年的各个阶段中,可以趣称之为春、夏、秋、冬四大法则。

春天,万物复苏,计划从这时候开始制定,似乎一切都很美好,一切也好像都按照刚刚

完成的计划在进行。

夏天，天气炎热，各个物种在不同的阶段进入繁盛时期，项目的开发情况一片火热，大家热情高涨，但是，问题也随之发生，虫子和各种病原体也都逐渐出现并开始生长。

秋天，似乎到了收获的季节，可是我们的项目还没有结束，因为夏天的一些疾病到了秋冬季节才开始爆发，比如流感的大面积暴发，大都发生在气温骤降的深秋，这时候人们会进入各种集市进行商品交换，为冬天做储备。而随着大量人与人接触，也给了这些病菌快速传播的途径和机遇。

冬天，万物萧条，气温降低，冰雪覆盖大地（南方除外）。已经过了项目验收的时间，可是客户还是不签字，因为有些问题仍然没有被解决，时间越拖越长，开发组成员的心也越来越凉。老板因为项目无法结束，项目款收不回来而开始从项目组抽调一些人员到其他新的项目中，为明年（公司的新项目）做准备，人手严重不足，加上历史问题的存在，计划再一次濒临绝境。

2. 周工作计划

一日之计在于晨说的就是周计划的内容，因为我们不可能把一个人的行为直接附加到一个项目组的团队上面，随着人数的增加，计划的周期必然扩大，否则，每天 24 小时就只剩下开会和制订计划了。

另外项目中的每一项具体任务都可能会持续几天甚至几个星期或者几个月，所以，每天都制定新的计划对于项目来说是不太现实的。

周计划，一般来说是项目组的最小计划范围，这个计划可以细化到天，计划的审核点应该细化到小时。比如，星期二下午 2 点对模块 A 的设计方案进行评审，那么模块 A 最迟应该在星期一下午 5 点以前完成设计方案并提交到配置库，由配置管理员或者项目组内部的分发机制软件将该模块设计方案的审核权利分配给参加审核的具体人员，同时发送邮件或以其他方式将星期二下午 2 点的评审会时间确定，星期二上午每一个要参加下午评审的人应该留出相应的时间来对模块 A 的设计方案进行审阅并在上午 11 点以前提交评审问题表（关于评审如何做，请参考 8.6 节的内容）。

8.5.4 计划的生命周期

所有的计划都有其生命周期，一般来说，计划都存在制定期（编写完成计划），执行期（执行计划，产生问题），修订期（分析问题，修订计划，进行审核）三个阶段，如此往复，不断轮回。

根据项目管理者的经验和项目的执行状况，这个生命周期的长短也是各不相同的。一个生命周期有可能是 1~2 个星期，也可能是 1~2 个月，也可能是 1~2 天。

所以，所有的计划，尤其是大的计划，都是要在即将结束的时才有可能出现一个完全可以执行到结尾的状态，否则，变化始终是计划的唯一特性。

那么,一个项目计划到底应该包括哪些内容呢?

1. 计划的内容

项目的计划一般来说,应该包括如下内容:

(1) 人员安排

项目组中有多少人,每一个人有什么特点、专长以及分配的职责。

(2) 任务安排

各个阶段的任务安排由哪些人来参与完成,需要什么样的资源配置。

(3) 资源分配

每个阶段可以从公司或者投资方获得哪些可用资源,这些资源的数量和持续时间等特性都要描述清楚。比如,公司会在什么时候发奖金作为激励,奖金的数量和获得的前提条件是什么;又比如,什么时候可能需要安排加班,是否要安排食宿并解决一些生活上的问题。另外,公司可以提供什么样的计算机配置,以及其他设备资源配置以便于大家使用,这些资源的持续时间是多久,是否会因为其他项目组的使用而发生冲突,等等。

关于上述3方面的内容就属于计划制定的范畴。

2. 计划的制订

一个完整的项目计划在制定之初只能绘制出大的阶段划分情况,因为不同的阶段,人员、资源都可能发生变化,任务也可能因为用户方的变更或者前期的分析不足而产生变化,所以不要奢望刚开始制定的项目计划就能细化到天,只需粗略地根据进度和要求做一个基本合理的划分就可以了。

每次制定的计划如图8-12所示的样子就足够了,然后根据项目的进展情况不断地丰富和细化,将它完善。

除了甘特图以外,还需要一份文档说明,该文档的内容可以很简单,只需包括上面计划内容的3个方面即可。

3. 计划的审核与修改

计划的审核过程可以像图8-12所示那样执行(这是笔者新公司以前的一个产品计划)。

从图中可以看到第二代产品开发的计划是比较细致的(因为第一代产品已经开发完成了),执行到了具体的阶段才会细化出来,到了第三代和第四代产品就没有细化,只展示了一个大的阶段。

标识	任务名称	工期	开始时间	完成时间
1	资金开始进入	2 工作日	2009年6月1日	2009年6月2日
2	场地租赁管理	25 工作日	2009年6月3日	2009年7月7日
3	租赁办公场地	10 工作日	2009年6月3日	2009年6月16日
4	装修办公场地	15 工作日	2009年6月17日	2009年7月7日
5	财务/资金计划	390 工作日	2009年6月3日	2010年11月30日
6	财务细节计划	20 工作日	2009年6月3日	2009年6月30日
7	财务计划执行	370 工作日	2009年7月1日	2010年11月30日
8	融资计划	200 工作日	2009年6月3日	2010年3月9日
9	人员招聘	30 工作日	2009年7月8日	2009年8月18日
10	技术人员招聘	30 工作日	2009年7月8日	2009年8月18日
11	市场人员招聘	30 工作日	2009年7月8日	2009年8月18日
12	办公开发设备购置	15 工作日	2009年6月17日	2009年7月7日
13	市场开拓计划	121 工作日	2009年6月17日	2009年12月2日
14	宣传印刷品制作	21 工作日	2009年6月17日	2009年7月15日
15	设计宣传印刷品	10 工作日	2009年6月17日	2009年6月30日
16	审核宣传品	1 工作日	2009年7月1日	2009年7月1日
17	宣传品审核修订	3 工作日	2009年7月2日	2009年7月6日
18	再次评审和修订	2 工作日	2009年7月7日	2009年7月8日
19	制作宣传印刷品	5 工作日	2009年7月9日	2009年7月15日
20	进行电视/广播/报纸	60 工作日	2009年7月1日	2009年9月22日
21	进行南京电信的市场	100 工作日	2009年7月16日	2009年12月2日
22	进行其它市场拓展活	100 工作日	2009年7月16日	2009年12月2日
23	新产品开发计划	277 工作日	2009年7月8日	2010年7月29日
24	第二代产品开发	96 工作日	2009年7月8日	2009年11月18日
25	制定第二代产品	5 工作日	2009年7月8日	2009年7月14日
26	产品软件开发	50 工作日	2009年7月15日	2009年9月22日
27	产品测试	20 工作日	2009年9月23日	2009年10月20日
28	测试修改	20 工作日	2009年9月23日	2009年10月20日
29	产品硬件设计	30 工作日	2009年7月15日	2009年8月25日
30	代码硬件移植	20 工作日	2009年8月26日	2009年9月22日
31	整体产品测试	30 工作日	2009年9月23日	2009年11月3日
32	产品问题修改	30 工作日	2009年9月23日	2009年11月3日
33	产品试运行	60 工作日	2009年8月26日	2009年11月17日
34	产品发布	11 工作日	2009年11月4日	2009年11月18日
35	产品发布准备	10 工作日	2009年11月4日	2009年11月17日
36	产品发布	1 工作日	2009年11月18日	2009年11月18日
37	第三代产品开发	96 工作日	2009年11月4日	2010年3月17日
38	第四代产品开发	96 工作日	2010年3月18日	2010年7月29日

图 8-12 公司产品计划图

产品规划甘特图如图 8-13 所示：

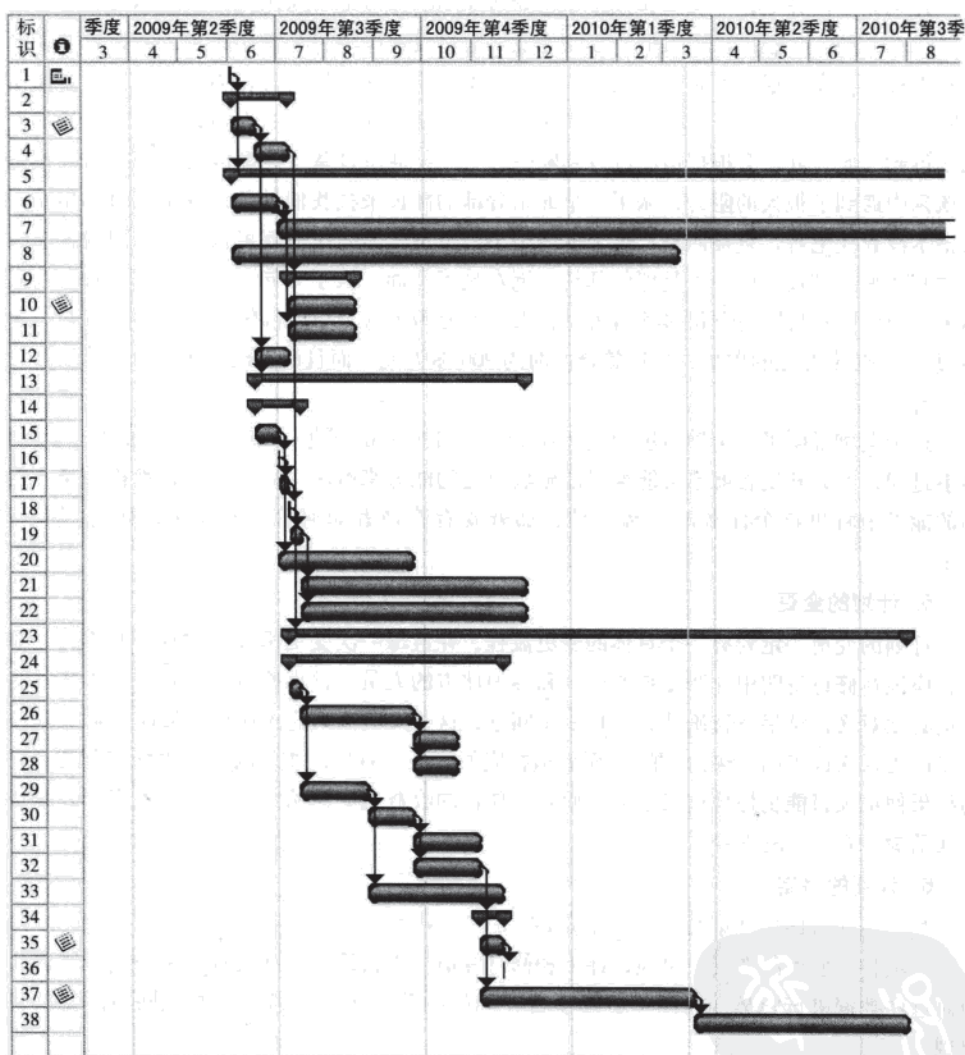


图 8-13 公司产品计划甘特图

4. 计划的执行

计划的执行是指按照计划的内容，把任务分配给具体的执行人。

执行过程中对于小的任务只需要考虑结束前的审核即可，较大的任务一定要考虑分阶段进行审核，与任务承担者进行沟通，确定任务执行过程中的有效性，及时发现项目风险，并尽早进行处理。

一个笔者曾参与的例子：

在某集团商务处，一个工作人员——业务主办或者更高级别的人员，经常是4~5项目同时进行。领导会根据每个人的特点和专业方向将具体的项目分配给每个人，每周五都要向处长进行的工作汇报，处长会把询问你这一周手中项目的进展情况；周三副处长有可能随时询问一些进展情况，其他时间都是自己做自己的事情，除非发生了有争议的事情。比如有厂商投诉到，此时处长会根据情况进行判断——当年某通讯设备生产商因为一个项目在与笔者的谈判中遇到了很大的阻力，派了一个北京分部的部长来找我们处长谈这个项目的情况，处长根本没有找笔者，直接回复：这个事情是小白负责，你们找他就可以了，不用找我。根本就没有和那位部长见面。于是那位部长与笔者见了一面，谈了几句，就走了，接着很快发出话来，说他们和集团还有很多项目可做，大家不要为了这个小项目伤了和气（这可是一个初始报价1700多万元的项目，最后签合同时为900多万元，而且所有硬件小型机都从半配变成满配）。

这个案例说明了一个项目进展过程中的审核和放权的问题，如果管理者管理过于死板，事事过问，那么开发者就不可能很好地施展自己的能力来解决问题，但是，承担者是否有足够的能力来解决这个任务的全部问题，那就要看管理者如何管理和衡量，如何判断和审核了。

5. 计划的变更

计划的变更一定要有一个具体的变更流程，并且每一次变更都要经过评审和修订，修订时，应该在修订说明中注明变更原因，和参加评审的人员。评审修订后，每一个参加评审的人员必须对修订结果进行确认，一旦发现问题，这些人应该为这次评审的失误（假设这个问题是因为这次评审的结果造成的）而承担相应的责任。但是，如果没有发生任何问题，评审的结果使得项目能更好地进展下去，而且，项目的收益也应该给参与评审的人考虑。具体的模式请参考6.3节的内容。

6. 计划的终结

当一个项目即将结束时，应该完成如下的工作内容：

对项目发生的变更、风险、评审和修订等信息进行统计，同时根据解决情况和发生原因对这些数据进行归类，以便于后续项目在同样的阶段进行参考，以避免同类问题的再次出现。

对项目计划进行最终的修订，补充项目的完成时间和最终评审情况。

对项目全过程中所有参与人员的绩效进行审核和计算，获取每一个人的绩效贡献值，便于计算项目收益时采取有效的分配方式。

8.5.5 计划中的常见错误

1. 大而全的计划

问：一开始就制定大而全的项目计划吗？

答：非常致命的错误，这样的计划不可能得到执行，因为随时都可能产生新的问题和新的变化。另外，这个过程非常费力，经过几次修改调整，项目管理者会发现自己大部分精力都用在修改项目计划上了，没有时间进行项目内的其他管理工作。随后就会出现，因为疲劳，而不再对项目计划跟踪修订，最后使得项目计划逐渐失效，一切都重新回到最原始的混乱无计划开发状态下了。

2. 迭代计划的各迭代内容

问：对于有多次迭代的项目计划，最开始就要写清楚各次迭代的内容吗？

答：如果是这样，就会出现计划制订完成后，大家都做得很好，使得项目经理没事可做了。

这个回答必然是否定的，因为在实际项目中对于时间跨度越大的行动其计划就会越概括，对于迭代也是如此。因此在很多时候，项目的迭代计划刚开始只能粗略地写，也就是初步对项目的认识和评定后的结果；随着项目的进展逐渐细化，将你下一迭代要做的内容填写细致，那么这个迭代的计划也就制定完成了。

在每一个迭代完成后，下一个迭代开始前，要修改计划，调整迭代内容，补充马上要开发的迭代细节。同时，对前一个迭代进行总结分析，进行数据管理和统计，这个行为将对下一个迭代计划的执行和制定产生直接的影响。同一个项目内的两个迭代之间的参考性是最强的，通过它们之间的关系可以最大力度地避免或解决一些可能发生的问题。

3. 项目计划的格式

问：项目计划有一般的格式吗？rup 的模板怎么样？

答：格式无所谓，只要大家都能接受，那就是格式。不一定别人的格式就对你的项目有效。每一个项目都有自己的特点，必须根据项目特点来做修改，不能完全使用同样的内容来填写。

rup 的模板太繁冗了，一般不建议直接照搬。不仅如此，也不建议所有的项目都照搬其他项目的文档模板，因为不同的项目有各自不同的侧重和内容。

4. 计划模板的内容

问：计划模板内容很多，如何填写？能删除吗？

答：如果已经采用或者必须采用一些传统的计划模板来填写内容，那么，就应该做好下面的事情：

对于无法确定的内容，将可选的几个方面都写上，不用太多，简明扼要地描述，每段内容不超过 200 字就足够了（200 字是个概数，不用刻意限制，有可能不到 100 字，甚至十几个字就足够了，但也有可能需要 400~500 字的描述）。

对于无法填写的内容,比如因为时间未到,或者资源缺乏而无法填写,那么就注明无法填写的原因即可。

对于和当前项目不适用的内容,比如一些不需要硬件的项目,而项目计划中却有硬件部分的内容描述,那么,完全可以将它空在那里,或者只需要简单地写上:不适用即可。

针对模板,不能说没写的内容都删除掉,其实很多内容是要根据项目的进展,才能逐渐补充上去的。项目结束时,所有的内容也就全部都写完了。

5. 什么人看项目开发计划

问:项目计划应该给哪些人看?谁又是最需要看的?

答:所有相关人员都应该看。没有谁更需要,根据分配的任务,不同角色的关注点不同而已。另外,参加项目计划审核的人不仅仅是看,而且还应该参与到与自己工作内容相关的计划的制订过程中,这样才能使计划更客观,更具有可行性。

8.5.6 小结

计划是软件开发过程中的一项重要活动,本文并没有对计划的所有方面都进行描述,因为那需要专著类书籍才能较为完整地说清楚,本文只是立足于针对具体项目的执行过程问题而描述,表述一些可以解决问题的办法和处理手段。

由于近几年发现很多软件公司都存在一些概念上的误区甚至操作上的严重问题,本节的目的将软件开发过程中的各个活动都单独写出来,作为提示要点以供读者参考。

8.6 评审如何做

8.6.1 什么是评审

在各行业中,评审活动也都是大量存在的,比如说,项目招标评审、质量评审、设计评审、合同评审、订单评审,等等。

在软件开发过程中,评审也是一项重要活动,如测试计划、软件开发计划、配置管理计划、风险管理计划、需求管理计划等的评审;另外,开发过程文档的评审,包括需求文档、分析文档、概要设计文档、详细设计文档等;开发过程管理规范也许存在的评审,如开发过程规范文档和流程的评审;同样的,开发产品也需要评审,如代码走查、系统验收其实也是评审活动的表现形式。

8.6.2 对评审状态的评价

很多书籍和资料中都可以查阅到评审的过程以及相关的内容,在软件开发中,企业评审活动做得如何,可针对以下几项进行评估:

1) 企业的评审通常有几个人参加,其角色是什么?各个阶段的参与人数及其角色是否有明确的定义?

- 2) 企业在评审前是否给参加评审的人员足够的准备时间? 他们是如何利用这些时间的?
- 3) 评审前, 企业的评审资料是否准备充足? 是纸质资料, 还是电子文档?
- 4) 企业的评审会持续多长时间?
- 5) 一次评审会一般评审多少份待评审工件?
- 6) 评审时提出的修改意见如何记录?
- 7) 评审时是否直接在原文档上进行修改?
- 8) 是否有专人做评审记录? 评审记录是如何做的?
- 9) 评审后, 一般多长时间给工件开发者进行修改?
- 10) 修改后的工件如何进行修改确认? 如果不能确认不能是否还会安排评审, 通常如何做?

请记录上面这 10 个问题的答案。

8.6.3 评审过程

1. 评审的准备

评审的准备包括如下内容:

(1) 待评审工件

一般来说, 一次评审的待评审工件数量不宜过多, 因为需要让参加评审人员能够集中精力关注工件的内容和开发质量, 过多的工件会分散评审人员的注意力, 同时也会影响评审的效果。

待评审工件一般一次为一件最好, 宁可多开评审会, 也不要弄一次工作量过大时间过长的评审活动。

(2) 参加评审的人

参加评审的人员一般包括:

- 工件的开发者, 负责工件的讲述和简要介绍。
- 项目组负责人, 负责会议的协调和评审过程的控制。
- 工件评审者, 需要至少 3 人, 最多 7 人, 奇数人数, 以便于评审意见的投票。一般是根据待评审工件的属性进行选择评审者, 如开发类工件选择其他项目组中相关的开发人员来参与, 其他亦如此。
- 会议记录者, 只需 1 人, 负责会议记录。

(3) 评审准备时间

一般要提前 3~5 天告知参加评审的人员, 给每个参加评审的人以确定的准备时间, 而不是工作之余的时间。

如果不给固定的工时来做准备, 并记录绩效, 那么很多人就没有时间来专门看评审工件的资料, 不能提前做好准备, 最后就会造成评审时间过长, 无法保证其效果。

评审前评审人员要把已找到的问题通过邮件方式发送给参加本次评审的负责人。

具体的工时根据评审会的工件数量和评审时间进行确定,通常情况下,如果评审一个工件需要1个小时,那么至少给每位参加评审的人员两个小时以上的文档审阅时间,这个时间基本上是评审时间的两倍(这只是一个经验数字,笔者目前还无法提供可供参考的实际数字)。

(4) 评审工作准备

待评审工件最好进行纸质打印,不建议使用电子文档的模式,因为通常会在纸质文档上做标记,或者写评审记录(这些在CMM3级的要求中是必须留下的)。

当然,如果企业有足够多的手写设备,那也可以考虑全部采用电子版本的评审模式。

(5) 其他工具准备

记录工具,这里建议采用录音笔或者速写笔和足够的纸张,建议考虑前者。如果公司资源允许也可以考虑进行视频记录。

2. 评审进程

有关评审过程的细节在很多书籍中都有介绍,这里建议读者去看看刘寅斌著的《系统分析之路》一书中的介绍。

评审过程基本如下:

1) 评审会开始后,主要开发者进行任务的简单介绍,每个工件的介绍时间不要超过5分钟,因为所有参加评审的人已经过了认真的准备。

2) 参加评审人员对工件进行提问,开发人员进行解答,使用录音笔和记录人员同时进行记录。这个过程每个工件不要超过30分钟,如果工件内容过多可适当延长时间。

3) 评审负责人进行评审意见总结,时间为10~15分钟,然后根据修改的工作量确定评审后的修改时间和最后的审核时间。

4) 最后,所有参加评审的人员在评审总结上签字确认。

3. 评审后的修改与审核

评审后的活动经常被人们忽视,这里笔者就详细写一下相关步骤:

评审会后,通过录音和现场记录将评审时的评审记录通过邮件发送给所有参加者。

开发者根据评审记录修改开发工件。

修改完成后,将修改后的开发工件提交给所有评审者进行最终审核。

审核通过后,标记工件版本,确定此工件的阶段性版本记录。

8.6.4 评审常见错误

1. 评审时间过长

整个评审会议过程要快速有效,不能拖延,否则时间超过一个小时以后,评审人员会感到疲劳,评审效率和效果都会降低,同时还会延误其他工作。

评审时间一旦超过一个小时,加上往往会议室都会比较闷,大家的注意力和精神都会无法集中,因此一个小时以上的评审,甚至会议,往往都是无效的,这也就是为什么连人大

议上都会有人打瞌睡睡觉的原因。

2. 评审过程中对工件进行修改

评审的目的是为了找到错误，而不是修改错误，因此评审时要最大程度的利用评审的有效时间进行错误的查找和分析，而不要用这个时间对开发工件进行修改，哪怕是个别文字的修改，也会耽误参加评审人员的时间，降低整个评审的效率。

这种错误的错误等级最高，几乎所有的评审中，都会发生这类错误，因此要格外注意。

3. 评审准备工作不足

这也是造成评审会时间过长的一个根本原因，因为很多人在参加评审之前根本没有给安排固定的时间进行文档的审核，因此，就会出现评审会开始时评审人员都在埋头看文档，或者匆忙间找一些措辞和小的错误来掩饰自己此前没有时间检查工件的实情。于是，主要开发者不得不重新对工件进行详细介绍，这一般要花费 10 ~ 20 分钟的时间，使得评审会很容易就会超过 1 个小时的有效时间范围。

这个错误的错误级别也是最高的，基本上国内很少有软件公司给参加评审的人员以专门的准备时间，因此也需要特别慎重。

4. 评审准备是否算工作量

另外，很多公司并没有认识到评审是一项重要的工作，因此，没有给参加评审者以时间来进行工件审核，谁在这上面花费了时间就会影响自己正常任务的完成，所以，很多参加评审的人就只能临时抱佛脚，评审会上现看现说，于是，评审也就流于形式了。

国外某些公司在这方面做得很不错，这一点是值得我们学习的。

电信集团公司当年的评审会，都是需要提前做准备的，尤其是招投标的评审，还会专门安排人员进行封闭审核分析，这当然是因为这种评审是十分必要的。

在笔者创建的绩效管理模型中，评审是做了绩点计算的，同时定义一次评审的工作量定为一点基础绩点，因此，这也那个绩效管理模型的基础评价指标之一。

8.6.5 小结

评审是软件开发过程中的一项重要活动，因为在咨询工作中发现很多公司都存在一些概念甚至操作上的严重问题，因此把评审单独以一节的内容写出来作为提示，以减少人们在开发中遇到的常识性错误。

8.7 配置管理工具如何用

我们在网络上能够搜索到的关于配置管理工具的相关内容大体分类如下：

- 什么是配置管理？
- 有哪些配置管理工具？
- 配置管理工具之间的比拼。

- 如何选择配置管理工具?
- 配置管理计划如何制定执行。

本节不是为了介绍配置管理工具的一般性使用方法或者上述内容,本节的目的在于配置管理工具和常规文档以及建模工具,甚至开发编码工具之间的配合问题,如何通过这个来验证配置管理工具中的信息的真实性,以及如何防治软件开发中的弄虚作假问题。

8.7.1 配置管理工具的使用方法

1. 起因

在2001年笔者参加的CMM的最终评审中,从印度来的主任评估师要求分析设计组成员在对话结束后提供一份设计变更的请求资料和设计变更的结果。

虽然我们做过设计变更,但却是一个小的工具产品的开发,只有为数不多的几次设计变更,因为整个开发周期也只有3个月的时间。而在这几次变更中,我们采用的是会谈商议的方式,并没有采用邮件发送方式或者其他变更文档,只用lotus notes这个内部OA系统发送了内部变更申请和批复邮件,主任评估师让我们打印相关的邮件内容供他审核。

在我们离开对话现场15分钟后,做了两封邮件(因为有一个人是notes高手),即使是标定时间和日期都做的完全符合“实际”情况。于是,这唯一的弱点就通过了主任评估师的审核,使得分析设计组的全部工作达标。

这样的一个案例问题同样在很多公司中上演过,在很多评审过程中发生过,只是类型略有差异,造假的内容各有不同而已。

首先,作为一名非开发者,应考虑如何避免被欺骗?无论是企业的管理者,还是所谓的主任评估师/ISO预审员或者相关角色,甚至是招标审核方,或者是客户/用户方,都需要了解技术人员做的是什,其真实性如何。

其次,对于技术人员来说,应考虑如何有效地利用配置管理工具,如何让企业的管理者知道自己没有浪费时间,如何让客户/用户知道自己的辛苦,知道自己是怎样找到一条捷径来实现用户所需要功能的——很多时候代码的多次重构,都是从一条条弯路上寻找更快捷到达目的地的过程。中间这些弯路是不可避免的,但是,非开发人员只能注重结果,他们是不太关注中间的过程和艰辛的。那么,这些问题如何解决呢?

2. 解决办法

针对这种现象,作为技术人员一方面要客观地体现技术和研发过程经历的艰难,另一方面更要实事求是地做事。所有阅读本书的读者都可以看到本书有一个与众不同的修订历史记录,该历史记录记录了笔者本人从开始撰写各个独立的章节到后面的合并以及第一次的正式出版,然后又进行了新的章节的撰写并将这些完成的内容再次与整本书合并。这个修订历史列表同样应用于2002年底中国电信MSS系统规范文档中,该文档的修订历史列表的部分内容如图8-14和图8-15所示。

版权声明: [REDACTED] 有限责任公司, 2002			
文档状态			
文档状态	☐ 草稿	☒ 正式	
文档评审人员	评审组组长: [REDACTED]		
	评审组成员: [REDACTED]		
评审时间	2002年12月13日星期五 9:00-12:00		
修订文档历史记录			
日期	版本	说明	作者
2002-10-12	1.0.0	创建省级规范文档	杨帆、赵雷
2002-10-12	1.0.1	重新排布文档格式	白慧冬
2002-10-14	1.1.0	新增部分模块的说明, 机构设置, 编码办法	杨帆
2002-10-14	1.1.1	修改了文档的格式和插图	赵雷、白慧冬
2002-10-15	1.1.2	修改了全部文档的插图	白慧冬
2002-10-17	1.2.0	依据省级规范和调研信息, 重新整理全部文档的格式和内容, 名称修改为业务规范, 并重新创建整理出《中国电信集团建设项目管理信息系统业务规范》	白慧冬、赵雷
2002-10-18	1.2.1	开始对规范的具体业务内容进行逐项调讨论	白慧冬、赵雷
2002-10-18	1.2.2	完成年度计划管理、网络规划与建设方案管理	白慧冬
2002-10-18	1.2.3	完成项目可研阶段、项目立项阶段	赵雷
2002-10-20	1.2.4	修改了部分内容和文档结构	白慧冬
2002-10-21	1.2.5	与计划投资处、工程建设处进行了相关业务流程的交流, 并完成了相关部分的文档描述	白慧冬、赵雷
2002-10-24	1.2.6	完成了工程实施过程和商务管理的内容阐述	白慧冬
2002-10-24	1.2.7	完成了设计阶段、批次资金计划、付款管理、竣工交付四个部分的内容阐述	赵雷
2002-10-24	1.3.0	初步完成合并整个文档的内容	白慧冬、赵雷
2002-10-25	1.3.1	修改了原文档中的部分内容	白慧冬、赵雷
2002-10-28	1.3.2	修改文档结构, 添加项目属性描述/汇总部分	白慧冬
2002-10-29	1.3.3	完成项目属性部分	赵雷
2002-10-30	1.3.4	完成附录中的项目编号办法部分	赵雷
2002-10-30	1.3.5	修改了文中的部分措辞和格式	赵雷、白慧冬
2002-10-31	1.3.6	添加了商务处提供的价格信息表附件	白慧冬

图 8-14 中国电信 MSS 系统规范文档的修订列表起始部分

在开发过程中, 可以把文档中这些修订说明部分的内容复制到配置管理工具的 comment 中, 两者对应就可以形成全部文档的可靠性验证了。

在审核时, 审核人员可以查阅文档修订记录中关键条目的说明是否与配置管理工具中该文档的 comment 内容一致, 如果配置管理工具中的版本数量过少, 或者 comment 内容对应的修订时间与文档中的修订时间不同, 那就可以认定该文档有造假。如果当年印度那位头发全白的主任评估师去打开配置库, 进行这样的验证性查找, 或者委托他的助手进行这样的工作, 那么当时的这个问题就会被发现。

对于技术人员来说, 拿出这样的一份文档足以证明其劳动的艰辛, 在中国电信集团的那次终审中, 所有参加评审的领导发表意见时的第一句话都会说, 可以看到四川设计院在进行该项目的工作中付出了很多的劳动, 很辛苦, 等等。

2002-12-09	1.7.2	修改流程图中的名称与实际名称不同的问题	白慧冬
2002-12-10	1.8.0	修改了全部文档的格式和部分用词, 填写了一些属性说明	白慧冬
2002-12-10	1.8.1	修改了很少的词	白慧冬
2002-12-15	1.9.0	根据评审意见进行初步修改	白慧冬、杨帆
2002-12-16	1.9.1	将业务规范名称修改为业务需求	白慧冬
2002-12-17	1.9.2	添加完成了统计分析报表部分的内容	白慧冬、杨帆
2002-12-18	1.9.3	修改需求文档中的流程图	白慧冬
2002-12-19	1.9.4	修改了文档中的部分内容	白慧冬
2002-12-23	1.9.5	开始整理专业分类	白慧冬
2002-12-24	2.0.0	根据评审意见进行修改	白慧冬、杨帆
2002-12-26	2.0.1	修改了一些措辞, 补充了一些名词的解释	白慧冬
2002-12-27	2.0.2	修改了一些措辞和格式上用词不同的问题	白慧冬
2002-12-30	2.0.3	修改了一些用词和项目的标识版本内容	白慧冬

图 8-15 中国电信 MSS 系统规范文档的修订列表结束部分

另外, 这部分内容对应于可度量绩效管理模型中开发工作绩效的内容, 其中的配置库中的绩效计算基础数据就是从这里通过这样的方法可以获取到的。

3. 问题

有人可能会问, 文档的修订说明是否必须和配置库的 comment 一一对应。

笔者的回答是, 不需要一一对应, 但是, 凡是配置库中有的 comment 都应该出现在文档的修订说明中。也就是说, 配置库中的版本数量应该少于或者远远少于文档的实际修订次数。

这是因为, 从配置库中取出来的文档可能需要一定时间的修订后才会进行一次提交, 才会形成一条 comment 内容。另外, 技术人员也有可能把文档带回家修订, 或者在与客户洽谈时等非开发现场进行修订, 这个时候, 文档的修订次数就会增加, 每次增加都应该增加一条修订说明, 而不要等提交到配置库时才产生一条修订说明。

如果两者数量完全一样, 应该怀疑该文档内容和修订内容的真实性了。

有人还会问, 那代码或其他的操作?

对于代码和 UML 模型的修订过程, 可以在代码上形成版本号, 因为代码的开发很少离开工作现场, 所以, 我们可以做到每一次的修订都与配置库关联进行, 填写对应的 comment 内容。

另外, 代码和模型文件的修订内容可以很具体, 一般一次任务可以进行如下说明:

首先, check out 的时候写明本次任务的内容在 comment 内; 其次, 未完成的任务提交要在 comment 内写明本次提交的修订内容, 同时标注任务未完成; 最后, 完成后的提交, 要在 comment 内写明本次任务修订的内容全部完成, 或者哪些内容因为什么样的变更而暂时不考虑完成。

这样基本上就完成了对代码和模型文件的全面配置管理工作。

8.7.2 小结

配置管理工具是软件工程活动的核心工具之一，其重要性是每一个参加过实际项目开发的人都能体会到的。但是，如何用好配置管理工具则是大多数技术人员都很迷惑的事情。采用本节建议的修订列表方法不会浪费较多的时间，毕竟每次工作结束都可以用几秒钟的时间来完成这样一段文字，而更多的就是复制与粘贴操作，不会增加过多的工作量，却可以获得相当好的效果。

总之，这样做的好处如下：

- 1) 应对客户/用户大量需求变更的有效手段——任何人看到一个这样的修订列表，都会感慨技术人员付出了多少劳动，是多么辛苦，还可以促进苛刻要求的协商解决。
- 2) 同样让领导感受到我们的工作量——和对客户一样。
- 3) 真正实现配置库的有效利用，技术人员也能感受到自己进行工件开发的充实感。
- 4) 实现工作量统计的数据基础——关联可度量绩效管理模型。
- 5) 有利于出现问题的倒查和分析。

也许有一天，当我们想做总结时，将所有自己编写的修订列表取出来，那么工作总结也就自动生成了。

8.8 软件人员的分化

提到软件开发人员，都需要考虑一下自己的定位问题和发展方向问题。当然，这里面也包括笔者本人，几乎是每时每刻都在考虑自己的定位和发展方向问题。

即使在这篇文章随本书第一版发表6年后的今天看来，这个趋势仍然是存在的，仍然在进行中，一些迹象在实际中已有所体现。

8.8.1 人员分化原因

软件开发人员一般分为如下几类：需求调研人员、分析设计人员、编码实施人员、测试人员、项目经理。

其实上面这5种人都代表着软件人员工作的不同层面和相应的发展方向。

国外的软件从业人员随着建模工具与集成开发环境的不断结合，已经开始逐渐走向分化，尤其是设计和编码的分化是对整个软件从业人员影响最大的，这是因为在传统的软件从业人员划分中，设计和编码是一体的，不可分割的。

但是，随着建模工具的逐渐平民化，也将意味着设计人员范围的扩大，在我国将有更多传统意义上的程序员逐渐开始以设计为主的职业生涯，这是符合软件工程化的大趋势的。

工程化要求的就是职能明晰、分工合作，这将无形中推动软件的细分工，不仅仅是从最早的需求、分析、设计实现的分化，人们期待已久的设计与编码的分离也将势在必行。

这个关键的分离将是推进软件工程化在开发过程中的最后步骤，也是难度最大的一个步骤。

设计到编码的分离将产生如下两种关键性的影响：

- 1) 将对设计人员的个人素质和技能提出更高的要求。
- 2) 将降低对编码人员的基本要求，逐渐将纯粹的只能做低层次编码实现的人员彻底推向软件蓝领的状态下。

8.8.2 需求人员的分离

需求人员中分离出来的第一个部分就是行业专家，其次是需求调研人员。下面先从行业专家这个角色谈起。

1. 行业专家

最早从程序员中分离出来的就是需求人员，需求人员中一部分后来成为了行业专家，这些行业专家不一定是在这个行业中工作，但是，他们对于这个行业的工作内容和操作步骤往往比在这个行业内工作的人员还要熟悉，还要了如指掌。

同时，这些人还必须掌握计算机相关知识，在软件行业内也要占有一席之地。可以这样打个比方：如果这些人完全抛弃了行业知识，他们同样可以通过自己掌握的计算机相关知识成为软件行业中的一名普通程序员，不要求他们必须成为天才般的高手，但是他们必须有着丰富的计算机领域知识和编程知识。

他们在软件行业内可能根本称不上高手，但是在他们所了解的行业内，这个行业的从业人员应该对他们十分信服，相信他们的能力，相信或者说崇拜其在计算机领域内的造诣，只有这样，他们说出来的话才具有说服力。

2. 需求调研人员

而没有成为行业专家的那部分人，应该掌握如下几项必备技能：行业知识、软件建模知识、熟练使用软件建模工具、熟悉两种以上的计算机编程语言。

(1) 行业知识

保证需求调研人员对该行业内相关操作习惯的理解，保证其能够理解该行业从业人员的表达方式，并可以通过建模的方式将这些内容转换成模型。

(2) 软件建模知识

这是保证需求调研人员将行业知识转换成计算机模型的基础知识条件。这也是保证设计人员能够熟练使用软件建模工具的基础。

(3) 熟练使用软件建模工具

这是保证需求调研人员将行业知识转换成计算机模型的基础技能条件。将保证需求调研人员可以高效顺利地将行业知识表达成模型。

(4) 熟悉两种以上的计算机编程语言

这是计算机领域知识的要求，要求需求调研人员能够根据情况选择适合的表达方式，让

分析设计人员领会并指导下一步的分析设计工作。也是避免需求提出的过高过于抽象而无法实现的基础条件。

3. 需求人员的工作内容

需求人员的工作内容就是将用户的业务需求转换成软件需求,并通过建模的方式表达,以便于在后续的工作中能够顺利推动工作的开展。关于需求人员如何工作的内容已经在本书第2章中有详细的阐述。

8.8.3 设计人员的分离

1. 纯设计人员的形成

这样的划分将对原先定义的程序员产生非常重大的影响,一部分人将由于其技能水平较高、经验较为丰富,而提升成为以设计为主的设计实现人员。

他们主要的职责是,将分析模型转化为设计模型,对设计模型中较为复杂的实现和算法提出有效的解决办法。同时,进行一些必要的关键代码的编写工作(关键核心微代码的开发),与编码实施人员配合,指导编码实施人员完成后续编码工作。

其工作内容的具体描述参见本书3.4节和第4章中的详细描述。当然,对于一些公司,这些人员可能必须承担是第3章和第4章中的全部工作内容。

2. 设计人员的技能要求

除了基本的计算机相关知识外,设计人员必须掌握的几项基本技能是,软件建模知识、熟练使用软件建模工具、高等数学/物理学知识、精通一种并熟悉两种以上的编码实现语言。

3. 掌握这些技能的原因

(1) 软件建模知识

保证设计人员可以熟练地通过模型来描绘出软件的实现状况。这是保证设计人员能够熟练使用软件建模工具的基础。

(2) 熟练使用软件建模工具

保证设计人员可以通过建模工具表达出自己的设计思想和设计理念,通过图形和注释让编码实施人员理解自己的设计思想,可以根据自己的设计来实现设计模型中所要表达的含义,可以推动该项目的持续前进。

(3) 高等数学/物理学知识

保证设计人员有着较好的逻辑思维能力和相关的知识,可以进行创造性的编码,有能力编写功能实现的部分微代码,编写高难度算法实现的微代码,这部分代码是一般软件蓝领人员无法编写的。

(4) 精通一种并熟悉两种以上的编码实现语言

保证设计人员对计算机语言有着较为深入的理解,基本上做到自己构建的数学模型和高难度算法自己可以进行编码实现。有能力完成软件蓝领人员无法实现的高难度算法的微代码

编写工作，而不是仅仅将设计停留在理论层面，避免造成设计与实现的分离。

8.8.4 编码人员的分离

1. 编码人员的形成

编码人员中的一部分是从原来的程序员白领逐渐降低到蓝领的，这些人的组成一般是这样的：原来程序员/设计人员中水平较低无法承担独立模块设计开发工作的部分人员，一些没有受过高等教育缺乏必要的高等数学知识和深入建模知识的人员。

与第2章2.2.4小节中的观点相同，这样的划分是大势所趋，形势是不可逆转的。如果我们不积极转变，而继续走下去，迟早有一天，可能会像印度那样，中国的程序员几乎全部沦为外国设计人员手下的蓝领工人了。

除了具有基础的计算机理论知识外，编码人员必须具备的几项基本技能是：软件建模基本知识、熟悉常规算法、掌握多种编程语言。

2. 具备这些技能的原因

(1) 软件建模基本知识

这是保证编码人员可以看懂基于UML开发的设计模型，并对导出的代码进行实现的基本保障之一。

(2) 熟悉常规算法

要求编码人员可以熟练地调用实现常规算法，对导出的代码通过其中注释的说明实现功能的代码编写工作。

(3) 掌握多种编程语言

如果编码人员只熟悉一两种语言的话，可能很难胜任该项工作，所以，编码实施人员要掌握多种编程语言并根据具体项目的要求进行低层的编码实现工作。

3. 关于编码工人（软件蓝领）和大学毕业生

上面已提到了对编码人员的要求，在本书的2.2.4小节中也曾提到过大学毕业生的问题，这是网上争论的一个热点：很多程序员都认为蓝领工人是对程序员职业的一种轻视和侮辱，其实，笔者对这个问题也考虑了很久，自从2000年网上开始争论印度模式的时候，笔者也在思考：如果不通过编码，如何才能保证自己的设计真正可以有效地转换成编码呢？的确很难，或者说，这几乎是不可能的，而且不熟悉某种语言的人是绝对不可能做这种语言所开发项目的设计工作的，因为设计和语言的关系实在是密不可分的，这个道理非常明显。

笔者经过几年的研究和实践，加上对国外一些设计模式的了解（包括日本、美国），认为：

大学毕业生在成为真正的软件工程师之前，需要参与一段时间的编码工作，其后的工作层面和蓝领软件工人是完全不同的。

他们到应该有半年最低层次的编码工作实践如果哪位大学毕业生在校期间已经参与了相当数量的编码工作，对程序设计语言已经有了相当深入的理解，那么这项工作可以省略），此后的两年时间里，他们应该主要参与的是微代码的编写和协助分析设计的工作。他们在编

写这些代码的过程中要侧重于对语言的理解和认识，侧重于与数学理论的结合，而不是简单地进行编码就可以了。

经过近3年的开发经历后，这些人逐渐成熟起来，并且应该能比较独立的参与分析设计工作——当然，话分两头，物有极端，肯定会有一些人成长的比较迅速，也许只需要1~2年的时间就能做到，但同样也会有一些人成长得比较慢，他们可能需要更长的时间，或者干脆降下来从事最低层次的编码工作了。

可以告诉大家，笔者也在进行同样的试验，当然，关于软件蓝领工人的试验目前笔者还没有相应的环境，不过，逐渐会有这样的测试进入到实际的工程过程中的。

8.8.5 关于测试人员

1. 测试与编码

在需求、设计和编码人员众多的中国，现在最缺少的软件行业的从业人员是什么？答案应该就是测试人员。尤其是高素质的测试人员，在中国实在是太少了。我国开发人员与测试人员的比例几乎可以达到十几比一。

测试人员是唯一与程序员在产生之初就站在对立面上。他们的职责就是为程序员所编制的程序提供可靠的检查检验机制，以保证程序员提交的程序能稳定可靠性地运行。

但是，由于测试人员与程序员最大的不同在于：程序员是创造者，测试人员是保障审核者。所以，更多的人愿意从事程序员这个职业，而不是测试和保障。

因为创新是每一个人内心所渴望的，渴望通过创新来证明自己的能力。而每个人都会有一种直接的感觉：测试属于后勤类性的工作。在国外也很相似，几乎只能听到某某天才程序员编写了多少著名软件，却从来没有听到过某某天才测试人员为这些著名软件检查出多少的Bug，才促使这个软件成名的，或者因此而名闻天下的。

2. 我国的现状

在网上经常可以看到这样的一些帖子，诸如：

主 题：测试工程师与软件工程师的区别

作 者：cuitdj（淡淡一笑）

回复次数：99

发表时间：2003-1-28 10:35:39

小弟在大学就一心在为做软件工程师而努力，并幻想将来能有自己创业。

可现在却阴错阳差的做了测试工程师。

大家能谈谈它俩在将来发展上的不同吗？

做测试工程师的职业规划该是什么？

靠什么来自己创业呢？

主 题：做测试做的真郁闷，我应该怎么办？

作 者：likecola（胖石头）

回复次数：31

发表时间：2003-8-1 15:23:53

在公司里就我一人是专职的 QA，程序员做好的东西我就跑一遍，发现有 bug 就跟踪这个 bug，最后直到他给修改掉。

这样下去我觉得根本没有提高，我应该怎么办？

主 题：如何从测试中发掘和培养乐趣？

作 者：lambo（YEH）

回复次数：7

发表时间：2003-7-26 11:49:32

大家好，先说一下隅的情况，隅在深圳，刚毕业，正准备去做测试，以前在公司做了半年的 coder，对测试也不是很了解，只是在网搜了很多有关测试的资料。隅喜欢做些软件系统底层的、具有挑战的、能让你感觉到自由的无限乐趣的东西，但隅不知道能不能从测试中寻找到此种乐趣或者培养某种乐趣。可以像某大虾说的那样吗：“无论你想成为 hacker 还是 cracker，测试工作都是你最好的选择。不要把测试只局限在找 Bug，去创造条件产生 bug 不是更有挑战性吗？”

正在郁闷，因为就中国国情来说，软件测试是没有受到足够的重视。

急盼回答，谢谢。

这里还有一些很典型的回复内容：

“从事了近3年的测试工作，但停留在黑盒测试，虽然是公司质管组的组长，但感觉离真正的专家还有很大的差距，希望能和大家共同进步”。

“在普遍的大环境下，没有人愿意做测试；公司觉得养不起那么多测试人员；开发人员不遵守规范，提交的代码质量不高。但是测试工作是重要的，优秀的测试人员是难得的。测试是软件提交的出口，又是意见反馈的入口，需要人知识面广，认真负责。测试工程师需要对系统有较全面地了解，而开发者可能只对自己的模块理解的深入。好的测试工程师要能够帮助开发人员定位问题，对于一些不易重复出现的错误找到规律。好的测试工程师能够对代码进行一定的走读分析，将错误尽可能地在项目测试的早期阶段发现。测试工程师要对 Script 语言有一定的了解，编写测试 Scenario 和测试用例，将常规的测试项目

自动化来保证回归测试的有效性。自动测试工具很多，有 XRunner/Winrunner、TestMaster，但都很贵，要 100 000 元数量级。所以非图形界面的可以用 Shell/Tcl/Perl 来自己写。”

大家好！我想说说我工作一年多来的感受，我也是美其名曰做了一年多的测试，可是实际情况呢：开始的一年，由于刚从学校毕业，懵懵懂懂地进了一家小公司，做了一年的打杂兼测试（开始倒是万丈雄心地想干编程，但是我的主管有性别歧视倾向，苦口婆心劝说一番，我也就放弃了）。所谓的测试其实就是手工采用公司推出的软件产品，再把问题反馈给研发部，有很多很多的问题属于编程人员自己根本没有调试通过，一年就在这样的日子中度过了。我倒是利用休息时间瞎学了一顿 Rational robot、winrunner，可是网上资料又少，身边没有可以请教的人，^_^大家可想而知这水平也就好不到哪里去了！

.....

唉！我都不知道出路在哪里了??????

我也试着写测试计划、测试用例、引入自动化测试，可是这一切都是我自己一个人瞎鼓弄，没有可以商量、请教的人，到底该怎样做、这样写对不对，没有人能告诉我！你们能告诉我吗？我该怎么办？

我也试着换工作、跳槽，可是那些公司的测试工作都是刚起步，情况都差不了多少；换来换去都一个样。

有谁能告诉我哪家公司在测试方面比较正规，我多么希望能有一个师父给我指点指点呀！！

我和你是一样的遭遇：刚毕业时做技术支持，同时兼做测试，跟你一样，只是操作公司的产品，再把问题反馈给开发人员，也是这样大约过了1年。我的志向是做一名优秀的程序员，所以工作时都不是很认真，平时比较关注的是软件开发方面的知识，而且后来一直在找程序员的工作，可是程序员的要求都比较高，基本上没有2~3年的经验是找不到工作的。后来，我找了测试工作（因为几乎所有的公司打出来的招聘广告对测试人员的要求都不高），一直做到现在，工作的时候没有人指导，一切都是靠自己去学习 and 理解，去摸索测试工作该怎么做，买了不少的书，查了不少的资料，自己也学到了一些东西，但是距离合格的软件测试工程师还有很大距离。这么长时间的磨炼，我对测试产生了兴趣，我在不断地收集测试方面的资料，同时我也在关注软件开发方面的信息，因为我的志向没有改变：软件开发人员（只不过是测试工具的开发人员）。

二位的一席良言一语道破了压抑在我心中许久的苦楚!!!!!! 我原来想做程序员,但是毕业后第一家公司的经历与二位一样,默默奉献,无人问津!跳槽时由于没有开发经验屡屡遭人冷落,现在的工作还是测试。我不是对测试有成见,我相信:能发现一个错误并不比能改正一个错误容易,一名优秀的测试人员与优秀的开发人员同样重要。但是,正如 yanghongxia (杨红霞) 所说,我希望做一个白盒测试人员,写测试计划、编写测试 Scenario 和测试用例,将常规的测试项目自动化来保证回归测试的有效性。而现在的工作只是黑盒测试(即功能性测试),手工采用公司推出的软件产品,再把问题反馈给研发部。这样的工作不知什么时候是个头!

另外,我觉得目前国内软件业真的不重视测试环节。每个公司在这方面的兴趣不大,测试高手比软件开发高手难找多了。比如,如果你在一个很好的公司做开发,有很多高手带你,你很快就能上手,用 2~3 年的时间就可以成为精英。而测试方面呢?国内好的软件公司许多都不重视测试的。可以想象在这样的环境中,有了问题找谁请教呢?哪有高手可以带你,只有自己在黑暗中摸索吧,苦闷!

这些帖子和回复都表明了一点:群体对测试工作的不认可。

也可以说是:群体对测试工作的漠视和轻视。

但是,测试工作又是那样的重要,离开了测试,软件的质量也就失去了保障。一些人称软件测试是与软件开发同等重要的事情,在 XP 这样著名的开发过程中还倡导测试先行。在我国,虽然几乎每个软件公司都有独立/相对独立的测试部门,但是每个公司的测试部门都没有受到很高的重视。设备简陋,缺乏足够的测试系统工具和人员培训是国内测试人员所面临的主要问题。

而且,在很多公司招聘员工的时候,同样对于同一层次的被招聘人员,开发人员的基本待遇要比测试人员高出 10%~20%。

由于这样的重视程度,加上较低的资金投入,造成我国的软件测试,大部分还停留在基本的功能性测试和操作测试上,并没有或者还不能真正深入到软件内核部分的性能测试以及其他的扩展性测试。这些都造成了开发的软件不得不在安装前后不断地由用户直接提出修改意见,直接影响了软件产品的最初形象。

3. 针对测试的建议

建议有能力的公司提高自己的测试能力,否则,最终软件产品将直接影响公司的形象,就像 Borland 当年发布的 Borland C++ 4.0 那样,几乎把 Borland 全部毁掉。

建议你的公司购买一些测试专用机器,通过组合和拼装后基本上就可以形成所有测试所需要的环境。

建议购买几种测试专用软件。对你的软件做一些性能和资源占用等方面的测试,这样,也可以提高贵公司的外在形象,显示出公司的质量控制与保障能力还是比较强的。在软件的功能、操作性都达到要求的基础上,用户将会越来越看重性能方面的指标了,如果

公司拿不出这些指标，那么用户该如何看待你的公司呢？

当然，如果你的公司不打算有大的发展，那么，也要有一些测试方面的措施，至少保障拿出去的产品不至于不能运行。

至于更多关于测试的话题，就无法继续在本书中撰写了，也许笔者将来会考虑写一些关于测试方面的内容，不过，毕竟笔者既不是天才，也不是全才，所以，还是走比较踏实的道路，循序渐进。

8.8.6 小结

笔者在写这一节内容时心情并不是很好，应该说是很差。这节内容中没有对项目经理这类的管理角色进行评论。笔者认为项目经理、管理人员、文档工程师等都是必需的角色，应该有人专职或兼职从事这类工作，尤其是项目经理这个位置是必不可少的。

但是，也决不可片面夸大这些角色而影响了众多技术人员对公司的信心。笔者曾遇到过类似的问题，在一家公司，当项目结束时，很多较有水平的技术人员纷纷跳槽，笔者也是其中一员。这是一个公司的悲哀，也是软件业的悲哀。

片面地夸大管理的效果，最终将导致技术的没落。

片面地夸大任何一个部分都是不对的，在适当的时候强调适当的角色，同时也不要忘记给从事现有角色的人以信心和鼓励。

8.9 Use Case 阐述模型化问题及延伸

笔者 2008 年开始与西安楚凡科技公司进行合作，随后在应用中发现 Trufun 工具进行 Rose 模型导入时会出现如下问题：

1) 用例图中的关系全部丢失，变成了两条连接线。比如，原有的 extend 关系消失了，成为两条无箭头的连接线。

2) 设计模型中的时序图没有转换过来，直接丢失了。这个时序图是直接关联到分析/设计类出现的地方，同时也是直接从需求可以映射出来的部分。

除此之外，Trufun 工具跟随标准十分紧密，这也同时丧失了给技术人员自助选择的方式。因为 UML 本身是一种普适性的模型化语言，并不是每一个行业任何一种软件都要全部应用的，一般只需要使用其中的一部分进行建模开发就足够了。

更多关于 Trufun 工具的内容可以直接公司网站上查看：www.Trufun.net。

于是，在 2008 年年底笔者针对当前 UML 工具的问题和全程建模方法论的需要，向西安楚凡科技公司提出了一种如图 8-16 所示的 Use Case 阐述模型化方式（但是因为一些其他原因他们没有继续进行相关的研发），这样就解决了原来全程建模过程中唯一必须文档进行编写开发的工作内容。但是由于目前还没有建模工具可以支持这样的功能，所以，只能在本书中给出介绍。

用户可以输入用例阐述的附加属性信息，包括：

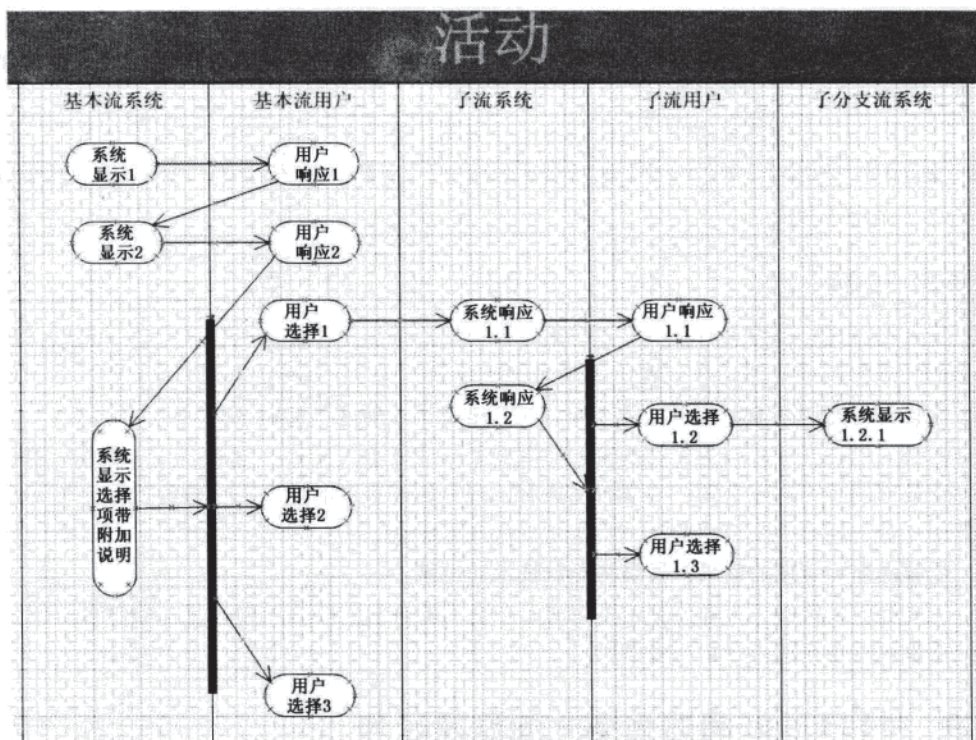


图 8-16 Use Case 阐述模型化示意图

- 1) 简要说明。
- 2) 备选流说明。
- 3) 异常流说明。
- 4) 特殊需求。
- 5) 前置条件。
- 6) 后置条件。
- 7) 扩展点。

其中 2、3、5、6 这 4 项可以扩展或为多项。

其中 5、6 两项需要直接关联其他用例或者其他用例的子流、分支流。一般来说，A 是 B 的前置条件，如果两者是 include 关系，则 B 一定是 A 的后置条件，如果是 extend 关系，则 B 未必是 A 的后置条件，这一点需要锁定关系。

这些部分第一阶段先按照文本模式增加，以后可以考虑与用例间关系/类图的对应关系关联。

整个用例阐述，根据用户需要自动生成基本流和子流的泳道，如果需要生成子分支流，则根据用户选择设定用例阐述模型属性的层级，最高可以到 6 层。但是当用户要生成

第4层的时候,就要显示建议:该用例需要考虑拆分或者描述上需要修改。这样就可以实现用例阐述的模型化驱动开发了。

需要进行交互建模时,可以通过模板导出基本的用例阐述文档,然后再进行后续的交互建模的开发和界面设计工作。

通过上述开发方式可以直接将 UML 工具与文档内容关联起来进行操作。

由于本书的目的不在于工具的开发,所以其他关于 UML 工具改进的功能建议和形态这里就不再进行更多地阐述了。

8.10 最简化开发过程

在2009年元旦过后的一次培训中,参加培训的大部分是高校教师和研究所的人员,他们在培训中一直希望能够提出一个最简化的开发过程模型以方便他们的使用。本节将提供一个最简化开发过程的模型,详细描述各个关键过程的组合关联关系和用途。

无论是较小的开发团队,还是个人开发软件,最基本的软件开发模型都应该包括如下几个关键过程:

- 项目计划;
- 配置管理;
- 需求管理;
- 风险管理;
- 团队会议。

这里提到的几个关键过程也属于传统概念的简化模型,不是完全的照搬版本,因为那样会占用大量的时间,而且并不适用于小型团队。下面将分别针对这几个关键过程进行介绍和特例说明。

1. 项目计划

这里所指的项目计划的内容可以参考8.5节的内容,主要包括:

- 1) 利用 Project 等工具进行整体计划的准备。
- 2) Excel 或者其他表格工具进行每周任务的选择和分配。
- 3) 周末通过团队会议方式进行任务完成情况的确定。

2. 配置管理

进行正常的版本控制和管理即可,不需要考虑诸如软件工具之类的管理。另外,不需要安排专人进行工具的管理,使用一个简单常用的配置管理工具即可。

3. 需求管理

不需要使用其他的需求管理工具,只需要使用 Excel 等类似的表格工具进行需求管理的标记和跟踪即可,每次在周工作例会上进行状态变更和分析。

注意:

这里的表格文件和项目计划的表格文件是同一个。

4. 风险管理

不需要使用专用的风险管理工具,在进行项目计划和需求管理的表格文件中进行风险管理即可。每周进行一次评估,然后针对可能发生的风险和已经发生的风险讨论如何解决。

5. 团队会议

这里的团队会议包括以下两个方面的内容:

第一是,随时举行的评讨论会。

第二是,每周的周工作总结和任务安排会。

(1) 随时举行的评讨论会

对于一个小型团队,评讨论会的方式就是2~3个人在一起针对某一个已经发生或者即将发生的问题讨论确定解决方案,会将讨论结果发送给所有的人了解具体内容——在一个小型团队中应该尽可能地保证所有信息的公开,这是对每一个团队成员的尊重,让每一位成员掌握实际情况,增加他们的责任感。

更详细的内容可以参考8.6节的内容。

(2) 周工作例会

每周的工作总结和任务安排会议中需要讨论如下3项内容:

第一项,本周需求的完成情况。

因为每周可能完成的内容都不会很多,所以,这项工作一般在10分钟以内就可以结束。会议涉及需要深入讨论的内容一律单独开会讨论,工作例会讨论的就是本周需求是否完成和完成时间如何。

第二项,项目计划进展情况。

根据上周的任务安排,确定已经完成的工作内容、没有完成的工作内容以及延期完成的工作内容,并修改周工作任务和计划表。因为在周内已经完成的任务,早已在完成时对表格做了标记,所以,这项任务一般也在10分钟内即可完成。

第三项,风险的发生和处理。

风险分析的内容包括:

- 针对刚结束的一周风险的发生进行评估和解决方法的确定,确定的解决方法是需要单独安排任务进行的。
- 针对下周可能发生的风险进行预计和分析,标记出风险的列表,确定可能的解决方法。

以上风险最多只需要分析出可能发生的前10个风险,重点分析前5个风险的发生和处理即可。通常,对于一个小团队来说这个过程只需要10~15分钟即可完成。

另外,也可以考虑采用日例会的形式,那么这个会议就是一个站立会议,每次会议不超过10分钟,在每天早晨成员到达工作现场后进行。

附录 A

全程建模方法论[⊖]

工具 EA 描述。

本附录内容是通过 EA 工具进行的全程建模方法论的整体描述，表示除了 Rose 之外，本书中的所有模型都可以通过任何 UML 工具进行描述。

[⊖] 此模型文件在本书附带光盘中。

全程建模整体视图如图 A-1 所示。

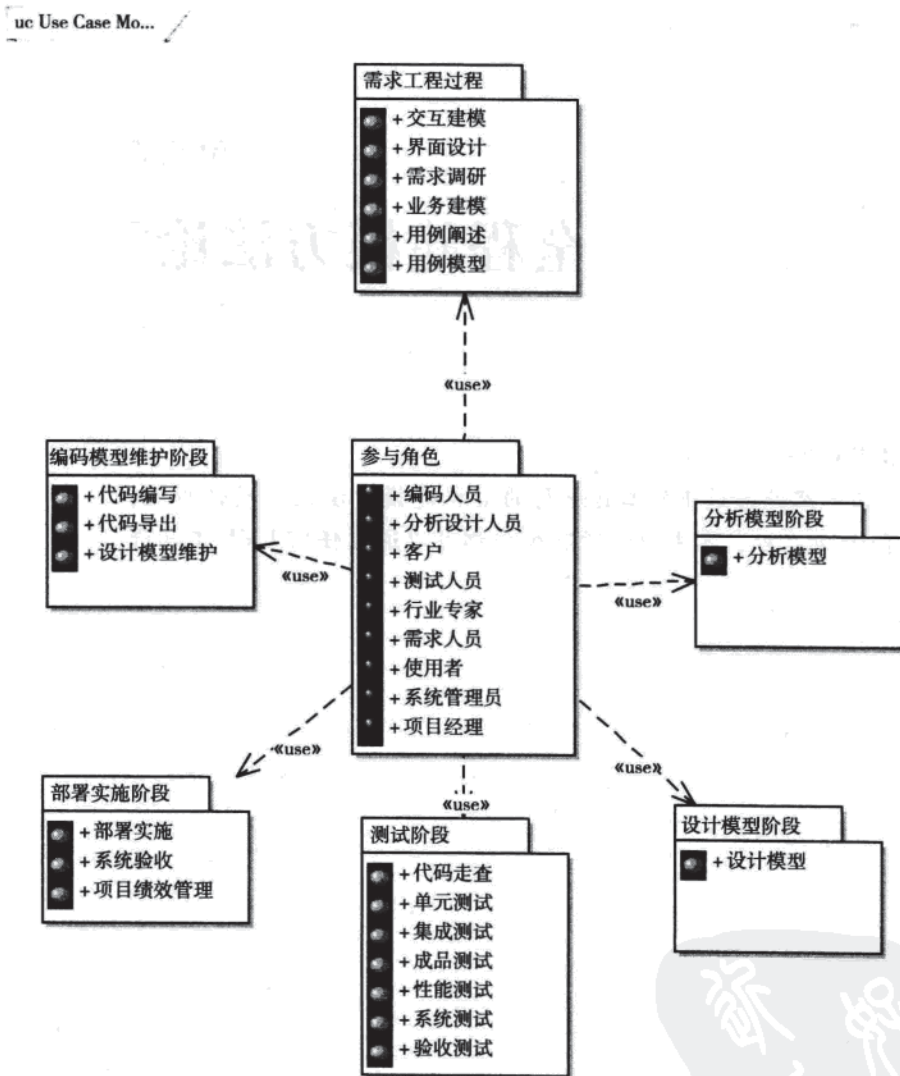


图 A-1

需求工程图如图 A-2 所示。

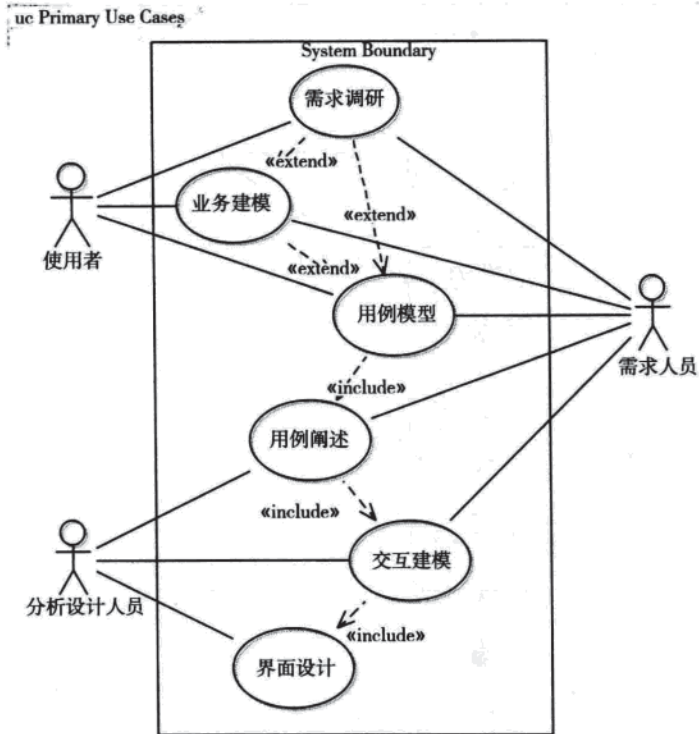


图 A-2

分析模型阶段视图如图 A-3 和图 A-4 所示。

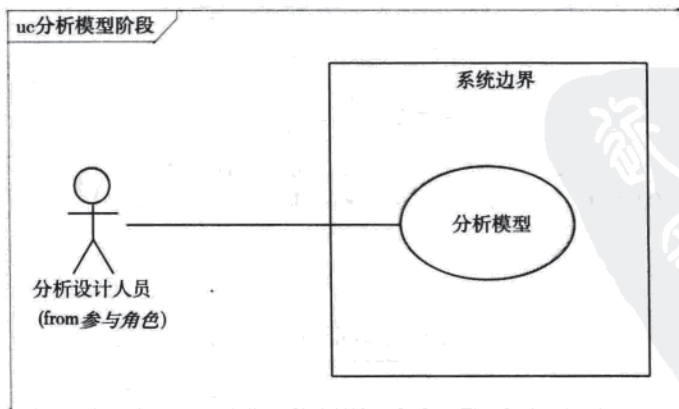


图 A-3

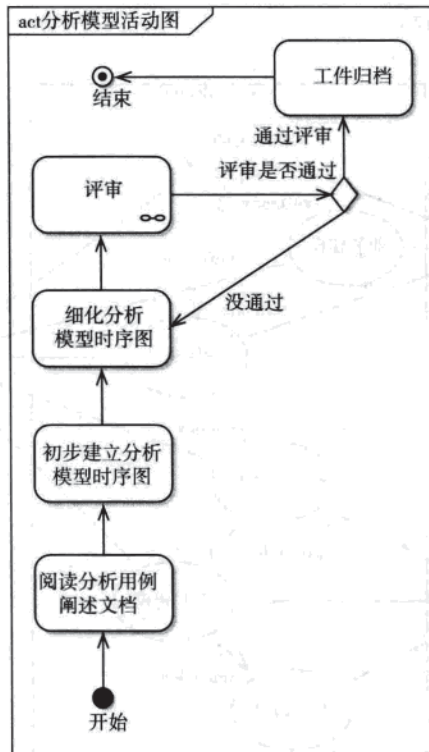


图 A-4

设计模型阶段视图如图 A-5 所示。

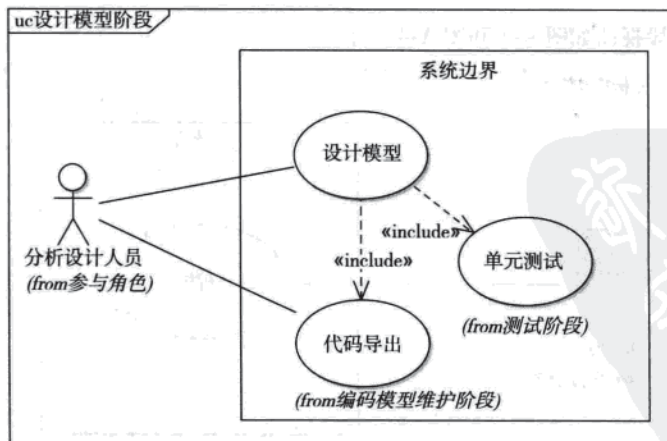


图 A-5

编码和模型维护阶段视图如图 A-6 所示。

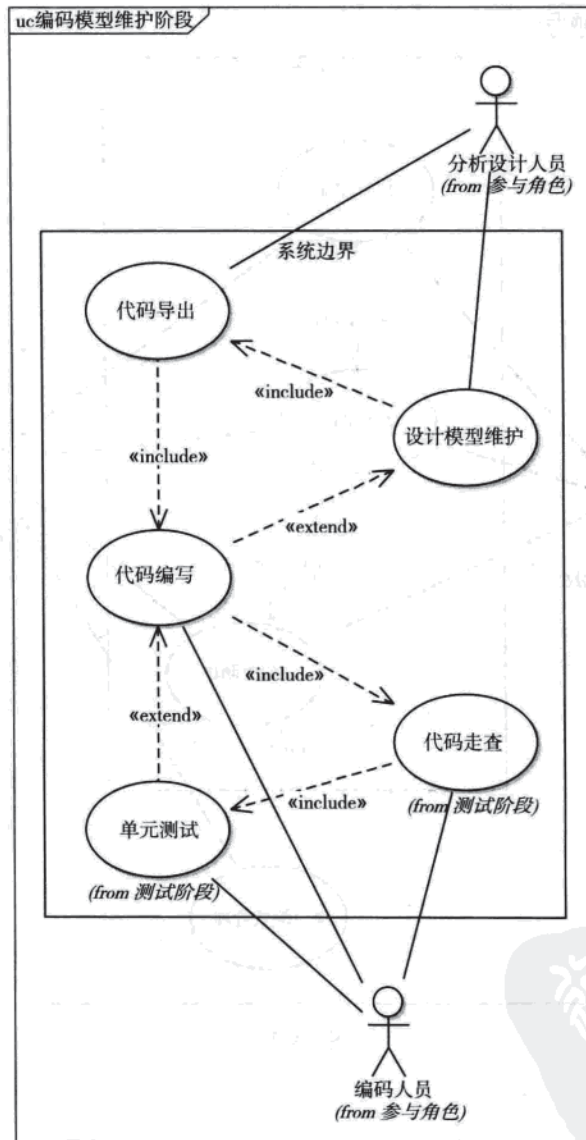


图 A-6

部署实施阶段视图如图 A-7 所示。

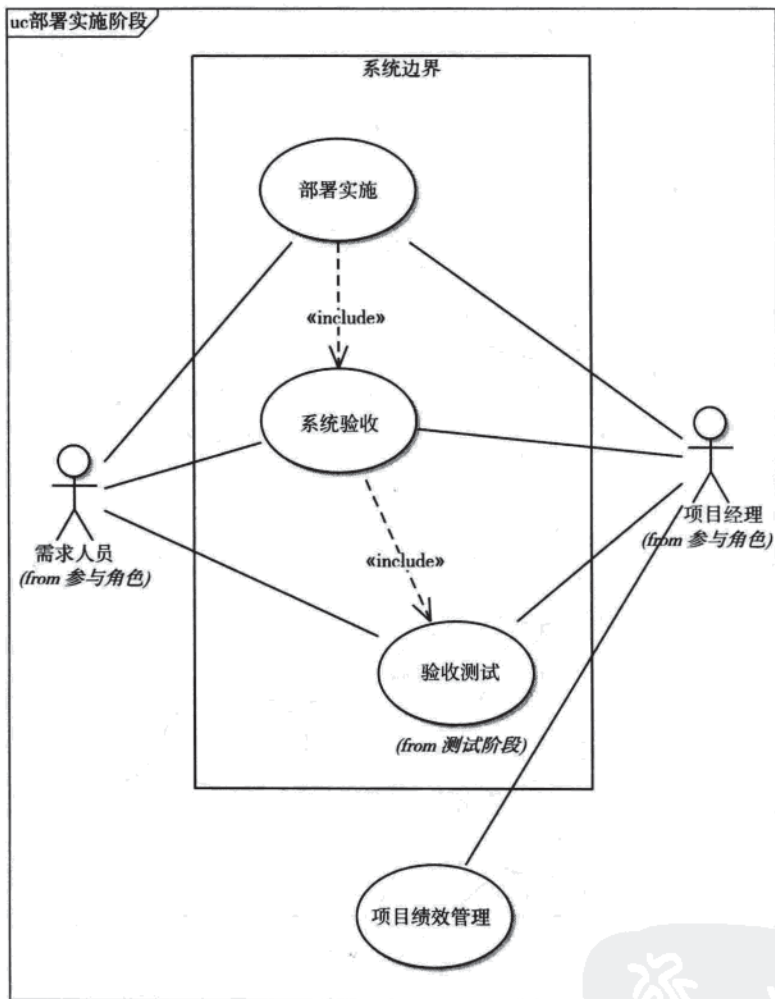


图 A-7

测试阶段如图 A-8 所示。

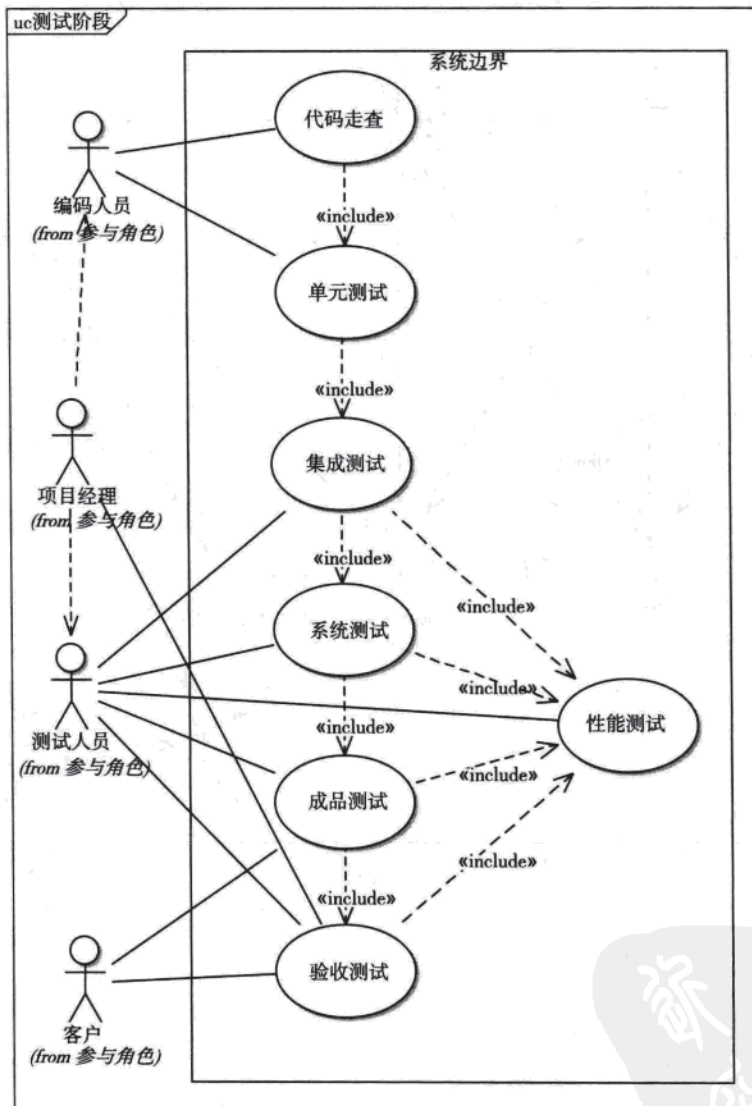


图 A-8

参与者的角色间关系如图 A-9 所示。

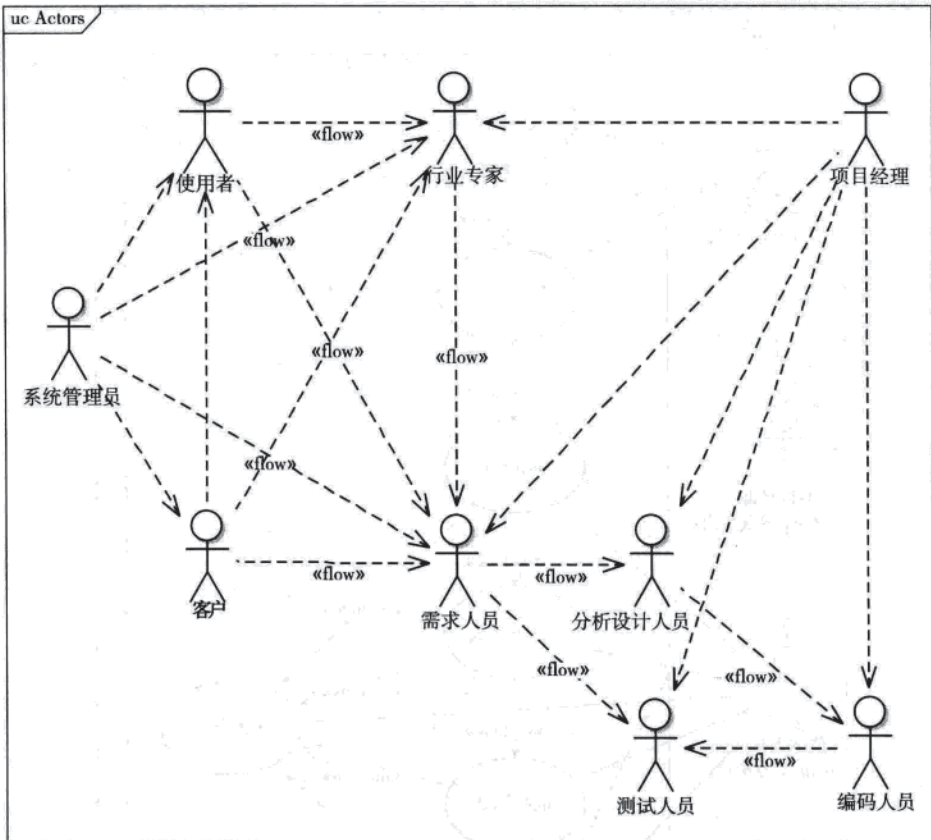


图 A-9

附录 B

幼儿教育系统[⊖]

基于 TruFun 的培训实例。

这是 2009 年元旦后与西安楚凡科技合作的第一个培训案例，培训中有学员通过头脑风暴法最终选定了这个案例作为培训案例，案例的名称为：幼儿教育系统。

⊖ 此模型文件在本书附带光盘中。

歡迎免費
PDG

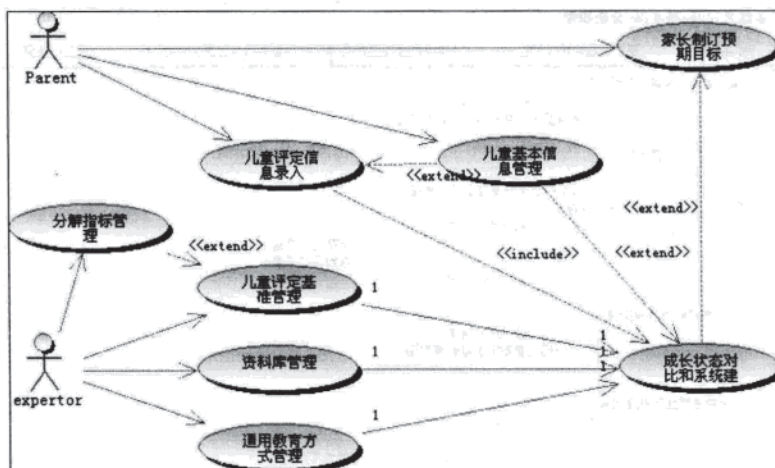


图 B-3

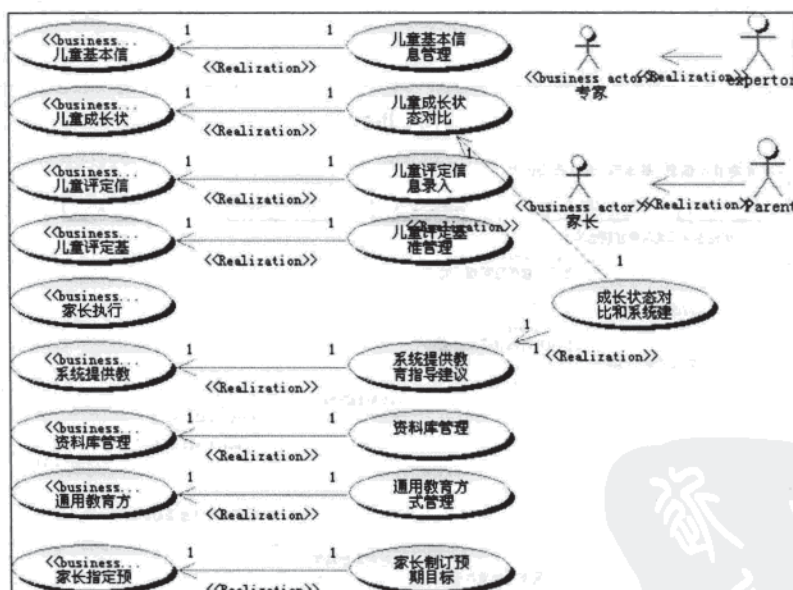


图 B-4

儿童基本信息管理分析模型时序图基本流如图 B-5 所示。

儿童基本信息评定设计模型，B/S 模式设计示例如图 B-6 所示。

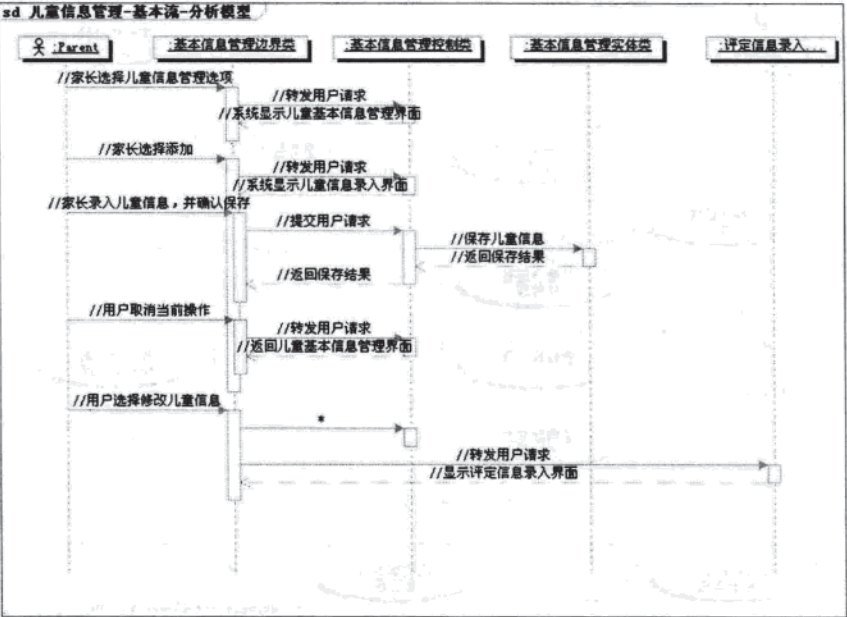


图 B-5

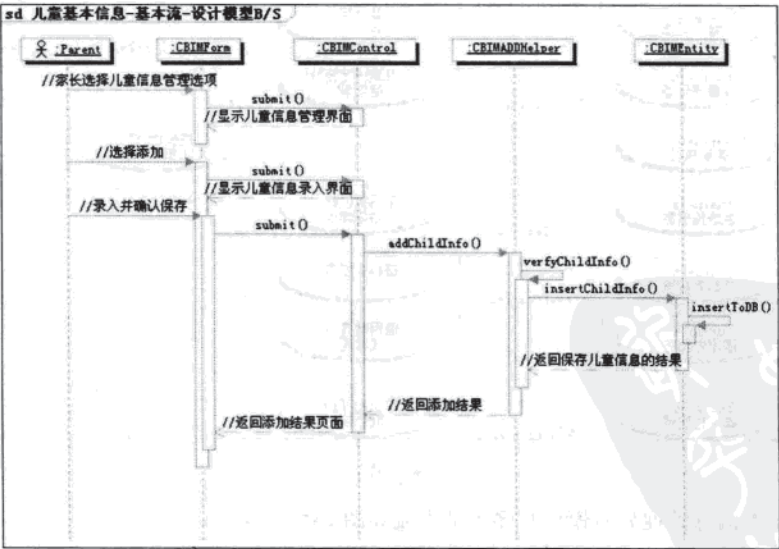


图 B-6

C/S 系统设计模型示例如图 B-7 所示。

左侧模型树如图 B-8 所示。

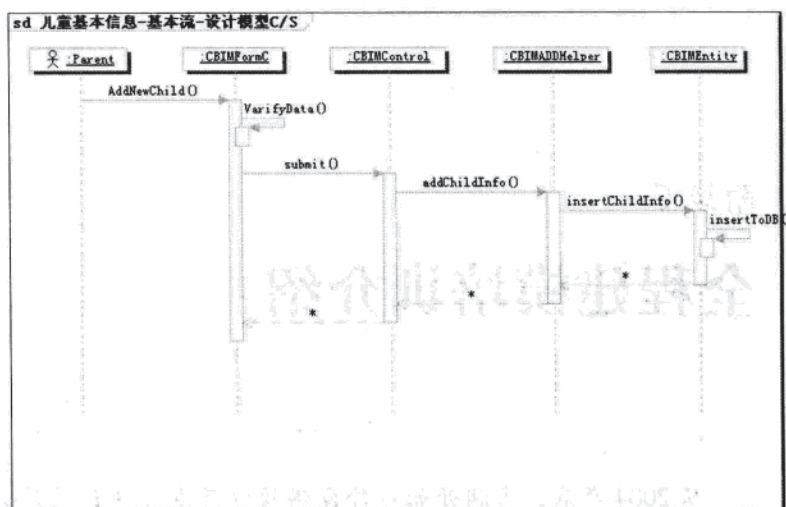


图 B-7

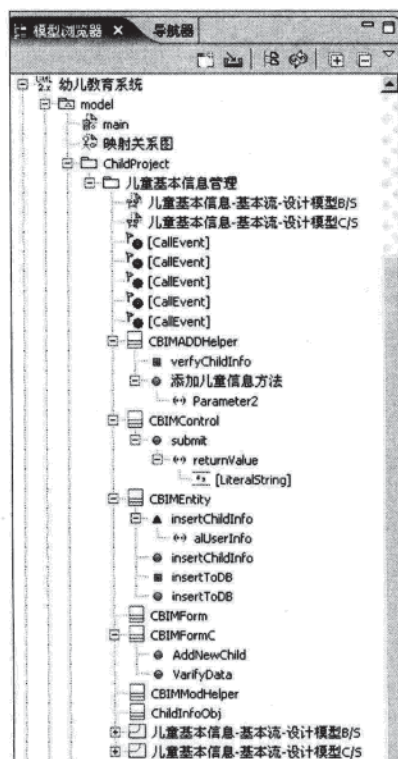
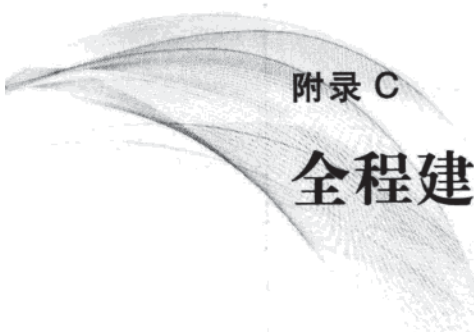


图 B-8



附录 C

全程建模培训介绍

从 2004 年底，青润开始对外提供软件开发方法论的培训。下面介绍全程建模培训的内容和培训方式。



1. 完全客户方实例教学法

参加过全程建模培训的朋友都知道,青润在培训过程中会采用1~2个全过程实践的培训教学案例,这个教学方法有如下特点:

1) 青润是不事先准备案例,所有的这类案例都是参与培训的学员或者接受培训的公司、企业现场提出的。培训案例可能是参加培训的学员/公司以前做过的项目或正在进行的项目,也可能是将来要进行的项目,这些都不作限制。

2) 青润与学员们一起完成这个培训案例。主要由青润来引导分析,学员动手做,在遇到问题的时候青润会参与分析。

3) 这个培训案例会从需求开始,一直做到代码导出为止。也就是说,培训结束的时候,所有参加培训的学员都能看到从自己开发的设计模型中导出的代码。

4) 培训案例一般只有1个,毕竟只有5天的培训时间。2005年初采用了两个案例,因为当时参加培训的学员分别采用了Java和VB两种开发语言。

5) 通过案例的全过程操作,每一个学员都清楚自己在应用全程建模方法的各个阶段可以如何操作,遇到问题时,可以如何分析解决。

注意,这里都是可以,而不是应该。对于一个完整的过程论来说,需要应对的是千变万化的用户需求和过程需求,因此并不是所有的步骤都是必需的,而任何操作都应该根据实际情况判断来选择可行的操作方式,所以这里是可能,而不是应该。

青润称自己的这种培训方式为:完全客户方实例教学法。

这个名词不一定准确,但是,笔者认为这样做才能起到真正的效果,否则,任何人都可以在准备的案例中作假。

不知国内是否还有人或者培训公司的培训会像青润这样做,至今,青润这样做培训已经有6年之久了。

2. 咨询式培训

这样做培训的原因如下:

青润本人是做技术的,做技术的人都有一个特点,不喜欢大量的重复,因此采用这种培训方式,可以使得每次培训的内容都因为项目的不同而产生不同的感受,同时,学员的不同也会在不同的阶段为作为讲师的青润提出各种不同的问题和实践操作现象。于是,一次培训也就成了一次咨询,每次遇到的问题都可能雷同,却又不完全相同。这一方面提高了青润对培训工作的兴趣,同时完全实际的项目培训方式也让学员能够从尽可能多地角度来体验全程建模方法论的应用方式,而青润也在其中积累了更多的需要总结和注意的事项。

因此,这样的培训就成了一次咨询式培训。

3. 培训前的准备

青润在每次培训前,从来不备课,但是必须做一件事情,那就是了解参加培训成员的项目经历和个人基本情况,包括求学经历、技术技能、曾经参与过的项目等信息。

4. 已完成的培训案例

序 号	完成时间	课 程 名 称	委 托 方	培 训 时 间	学 生 人 数	联系人/职务/联系电话
8	2009.1	全程建模培训	西安楚凡科技	4 天	9	
7	2008.04	EA 工 具 和 UML 建模培训	EA 中国总代	2 天	18	南京西门子公司
6	2006.10	全程建模技术 及软件工程过程 及方法培训	亚嵌	5 天半	20	总参某部通机局
5	2006.1	全程建模技术 培训	CSDN 网友自 发组织	3 天, 每天 9 小时	19 人	这是第一次对外 公开课, 3 人从济南 到北京来参加培训, 一人从成都到北京 来参加培训
4	2004.8	全程建模技术 培训	浙江天正信息 科技有限公司 (原浙江省计算 技术研究所)	5 天, 每天 6 小时	20 人	
3	2004.6	开发过程、Ja- va 语言、架构模 式、配置管理等 知识	四川经源通科 技发展有限公司	4 次, 每次 3 个小时	5 人	
2	2003.11	OOAD 与 UML 课程	北航软件学院 (研究生必修课 程)	四周, 每周日 3 个小时	110 人	
1	2002.9	建模技术、开 发过程、架构模 式、Java 语言	四川通信科研 规划设计院	6 天, 每天 3 个小时	11 人	

5. 相关链接

部分曾经完成的培训链接如下。

[全程建模] 2006 年全程建模培训的总结, 网址为:

<http://blog.csdn.net/qingrun/archive/2006/01/15/579987.aspx>。

2006 年 1 月全程建模培训:

<http://blog.csdn.net/qingrun/Gallery/172074.aspx>

[全程建模] 新一期全程建模培训顺利完成:

<http://blog.csdn.net/qingrun/archive/2006/10/14/1334372.aspx>

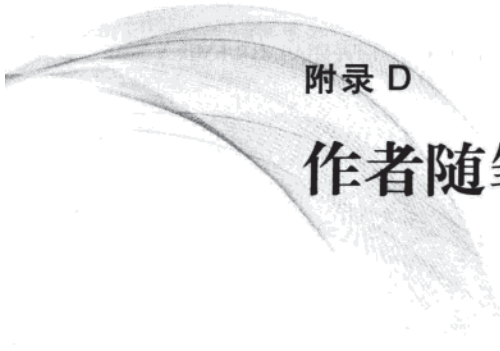
[全程建模] 2008 年的全程建模培训顺利完成通告:

<http://blog.csdn.net/qingrun/archive/2009/01/06/3716712.aspx>

2008 年的全程建模培训照片:

<http://album.hi.csdn.net/views/guestalbum/473713?d=MjAwOS0xLTggMjA6MDU6Mjk=>





附录 D

作者随笔一



青人来自崇山间，
润心爱在百丈崖。
意如纸雪斩污处，
愿胜天下唯文雅。

这首诗，我们暂时称之为诗，暂且不论这首诗是否押韵，也不管它是否符合绝句的平仄。但是这就是青润在初中三年级的那个寒冬写成的一首铭己至今的诗。大家可以注意一下这首诗每句话的头四个字！

但是，完成这首诗之后，青润却没有想到，这首诗却已经验证了青润人生这前二十七年的经历。关于诗里写的是什麼，每人会有自己的理解，这里笔者只做一个简短的说明：前两句写出了笔者出生在太行山脚下，后两句写的是笔者想要做的事情和想法。其他的，这里还不想多做什么说明。

“青润”，很多人都问过我，青润这两个字是怎么来的？有什么含义？

因为看着我的原名，好像与此一点关系都没有，看着我这个人也是既没有相貌也没有身材，好像也不那么青润似的。我们不妨先来看一下“青润”这个词的汉语意思（以下内容是笔者找到的一些对这个词语的解释）：

有人用这个词形容茶：

“极品铁观音：安溪极品铁观音乃乌龙茶中之珍品，

.....

成品条索紧结匀整，色泽砂绿青润

.....

享有‘活、甘、清、香’四绝之称。”

也有人用它形容石：

“.....

湖石形象玲珑秀丽、色泽青润，石型高大成峰，并具有瘦、皱、透、漏

.....”

在佛教经文中用它形容草：

“佛说大乘菩萨藏正法经卷第九

.....

自生异草青润柔软右旋宛转。如迦邻那宝。

.....”

因此，青润这个词是用于形容一些物品的颜色鲜活青翠，湿润优美的样子。但是，这些都与青润的这个笔名没有任何直接的关系。说实话，我倒真的希望自己能活得像这个笔名那样鲜润自在，而且在很多事情上，青润非常赞同老子的观点：顺其自然，无欲无妄。可是，人生却不是这样，所以，青润平时操行的却是孔孟的理论：明知不可为而为之。

笔者的这个笔名起源于初二一堂语文课，当时，写了一首诗，可能也是小子的第一首

诗，笔者留下了这样的一个笔名“青润”，这两个字来源于中国两个著名的数学家的名字，估计到这里，我就不用多说了。

1976年12月，我出生于一个普通工人家庭。虽然一路走下来，成绩一直都很好，但是没有林锐先生那样的好运气。在高三那一年就受到了有生以来最大的打击，笔者把本来应该拥有的机会毁掉了。

在那一年的全国物理奥赛上，一道11分的光学成像试题的标准答案给错了，这个错误的正确答案，其实当时我只用了五分钟就做出来了。结果，笔者多想了一点，就感觉不对，于是修改了成为错误的正确答案。这一改，让青润丢掉了11分，从本来至少可以是河南省第5名的成绩一下子跌到了第28名。不仅失去了参加决赛的权利，连保送的资格都没有了。初赛20天后，我给当时北京大学的苏幼生老师写了封信，连带我的实验和理论推理结果一并寄给了他，一心希望他能给我争取到一个参加决赛的机会。但是，他的回信肯定了我的正确，却无法给我这个机会。至今，我还怀着一份心痛珍藏着这位至今未曾谋面的苏幼生老师的那封信。

随后的高考，我浪费了原始分80多分进入了这个我后来又恨又悔但又无可奈何，还有些留恋怀念的学校——沈阳工业大学。

当时，我几乎彻底疯掉了。上了大学以后，我仍不肯服输，一心投入到自己感兴趣的科目中。课堂上的事情课堂上完成，课后，我就几乎没有学过与本专业有关的内容了。

这时候，我重新找到了小学时的密友——计算机。于是，把几乎所有的精力都投入到相关知识的学习中了。

为什么说是小学时的密友？因为最早，我是在小学四年级开始接触计算机，同时也开始学习计算机语言的。那时，正好学校来了一个计算机老师，很年轻，只比我大八岁，名叫吴守海。但是，吴老师很会吸引我们，很会表述自己的观点，通过一些很小的技巧让我完全沉浸到计算机的海洋中了。我成为月山铁路子弟小学跟随他学习时间最长的学生，整整两年……直到他因为他母亲治病而离开月山到郑州铁三小教学为止。后来，我还去郑州看望过他两次。不过，大学毕业后，因为一直忙于生计，到现在也就没有再找到时间去看望他了。

而上初中和高中的时候，因为学校的环境有限，我没有再接受过什么计算机教育。于是，初中的时候，我开始在三个竞赛方向上努力：数学、物理、化学。

初一下学期，因为我父亲到了新乡工作，我便以月山铁路子弟中学第一名的成绩进入了新乡铁路子弟第一中学学习。这个学期第一次全年级考试，我就取得了数学的满分，就这样我的数学成绩（几何和代数）的满分一直保持到初三。因此，放假前，班主任（当时任我所在班级的数学老师）就让我好好准备一下，争取下学期进入竞赛小组。其实，假期的时候，我也不知道应该学些什么。就这样，第二学期开学后的全年级数学竞赛选拔赛，一共八次考试，最后我以总分250分的成绩成为第一名。因为这个分数很有特色，所以，我一直清晰地记忆至今。

就这样初中的最后两年，我就过着以数学竞赛为主，同时参加了物理和化学竞赛的生活。初二结束那年暑假，我在参加郑州铁路局暑期数学夏令营的时候，因为为了维护学校的荣誉和名声而说了几句真话，由此，受到了几乎所有同学的孤立，同时尽可能地给我难看。因为我揭发的那个人是教师子弟，家里经济条件也很好。他经常买一些东西给几个关系比较好的人吃，而且上课的时候总是在玩，因此被其他学校的老师点名批评我所在的学校很多次。而这些事情在辅导老师到达后，大多数人沉默，犯事的几个人说，一切都很好……

两年以后，在高一的一次竞赛前，当时参加过夏令营的一个同学对我说：对不起，小白，当时真的对不起你。听到这句话，我的眼泪一下子就涌了下来。

那次，我在郑州铁路局主办的夏令营后的初中数学竞赛中成绩不是很好，一个同学后来在和大家聊天提到这件事情的时候说，如果当时不是那件事情，我们学校肯定要增加一个一等奖的。

希望大家不要认为我这个人记仇，但是，很多事情的确是无法轻易释怀的。人这一辈子总是要记住一些，也会遗忘一些。不过，我提到这个经历也只是希望可以让同龄人和已经做父母的人考虑一下自己的孩子要走什么样的道路。也告诉大家，不要把小孩子看得太简单了，有些时候他们也许比我们还要复杂，让人免不了感受到痛苦和压抑。

高中的时候，因为数学老师不再组织数学竞赛相关的集中训练，我就开始逐渐把我的重点转移到了物理竞赛上。两年的时间里，依次拿到了一个郑州铁路局物理奥赛一等奖和那个全国物理奥赛河南赛区一等奖——也就是那个改变了我一生的竞赛。

平明看日黄沙尽，不知此生身何在？

大学期间，我很快成为了老师眼中的焦点，被人称为材料系软件第一高手。记得在大二上学期参加的一次校级计算机竞赛中，我的程序设计拿了全校第二名，第一名是当时电子系94级的一个校友。后来还在大学生数学模型竞赛中为学校取得了第一个最高奖，也是当时学校参加此项竞赛以来惟一的一个国家二等奖。那次竞赛的算法和程序实现都是我一个人提出并实现的，如果不是时间缘故我最后实在无法完成后续的程序开发，我想应该可以获得更好的成绩。

毕业设计期间被一个老师带到中科院××研究所去参加实际的研究性项目。在那里，我本以为我找到了一条可以再次达到我目标的途径。我在毕业设计中的一个额外的实验中发现了一个材料摩擦学中的一种新的临界现象。在与学校几个物理老师讨论后，他们都认为这将是一个非常巨大的课题，但是因为我毕业设计的时间限制，我没有机会也没有时间继续做下去，这个方向也不知道几十几百年之后才可能有人继续重复我走过的那条路。因为这些新的发现和试验的开创性，我的毕业设计和答辩，很轻松地拿到了双A。要求在十分钟内完成的毕业答辩，我的讲述就整整占用了二十五分钟之久，而所有的评审老师都听得聚精会神，没有人提出异议，包括系里那位严厉得令人生畏的老教授。

后来，我离开了沈阳，永远离开了我的专业方向，也离开了物理学方向。这是因为毕业前在我和一位两院院士的一次面对面的谈话中，这位七十多岁的老院士对我说：“你说你这

好那好，都没有用！如果有一个人过来告诉我说，他的父亲是个部长，能给我拿来一个两千万的项目，你说，我要你还是要他？”

听完老院士这“语重心长”的一段话后，我一下子明白了，全都明白了。然后，我很有礼貌地向老院士告辞，感谢他给了我一次直接和他面谈的机会。随后，我就走了，离开了中科院，离开了沈阳，离开了东北，也远离了我的梦，带走的只有一颗破碎的心和被摧毁的理想。然后，我做出了我这一生中的第一个重大决定：中国国内的学位再也不读了！

从此，我开始了我的程序员生涯，凭借着良好的基础和背景（我曾经在上大学的时候帮一个硕士修改了她的实现算法，使得本来一次计算要三昼夜才能完成的计算，减少到每次只需要不到四个小时。加上我在此之前还独立开发过几个校园内流行的小软件，包括1997年底一个在DOS下实现的配乐教学课件），很方便地就屏蔽了专业的差异，在上海的一家小公司找到了我的第一份程序员工作。后来在2000年5月辞职后就进入了香港联想深圳研发中心，后来，在一个大学同学的劝说下到托普集团中央研究院工作了将近两年。期间参加了托普集团中央研究院技术研发中心在2001年11月进行的CMM3级评估，当时我是以分析设计组组长的身份参加的面谈。一年多以前来到了现在这家公司，开始将所有精力投入到电信行业的软件开发过程中。我再次进入了国企，希望国企不要让我再次失望！

此间胜负平常事，是非成败如何评？

很多认识的和不认识的朋友都觉得我做程序员是很成功的。我没有和他们说过什么，因为争辩一些过去发生的而且也无法改变的事情是没有什么意义的。虽然，我心中总有着隐隐的痛。

其实，我个人认为我作为程序员最成功的地方的确有一点，不过，也仅仅只有这一点：经常总结！

可能因为自小文笔就比较好，所以经常总结一些经验和教训，不断地提高自己。上大学的时候虽然是纯工科，但是也经常有一些短小的篇幅见诸报纸的字里行间。笔者到现在还在天极网有一个个人专栏，不过，也已经有快两年没有什么更新，连看都没时间去看了。

另外，就是笔者的废话比较多，反过来想想也是，如果废话不多，怎么可能写成书呢？书的最大的一个特征就是：你往往必须用大量的废话来描述一个本来已经非常简单的道理或者过程。因为，只有这样你才能让尽可能多的人理解你所说的是什么。

人生境遇殊难求，此生分做两生用！

人这一辈子只有一次机会，前一段时间，我给我的女友说：如果我有下辈子，我一定不再这么辛苦得活着。我一定会选择去下棋，下一辈子棋……

这辈子我最大的业余爱好就是下棋。但是，我却没有给我自己机会去深入学习研究它，因为我总是认为我还有更重要的事情要去做。

好了，要说的废话基本上可以告一段落了。至于这篇文字算是什么呢？我自己也说不清楚，只知道我还没有到写自传的时候。不过，我不像林锐先生那样害怕别人说他已经要盖棺定论了。反而我觉得，一个人到了一定的时候是需要总结一下的，为自己划一个阶段找一个

方向。我有记日记的习惯，所以，也许将来整理自传的时候会比较容易一些吧。反过来想想，我也一直没有机会把一个完整的自我展现出来，让世人评述，因为我本人也在不断地变化，一切顺其自然，却又争之于社会。

曾记得我在上海的时候在新浪程序员论坛上发表过一篇文章，后来成了精品，也是我作为新浪网友的惟一的一篇精品。这篇文章后来被十几个网站转载，这里就用这段文字来作为这篇随笔的结束语，与大家怀念我们的过去，共勉我们的现状，争取我们的未来：

我，是一个普普通通的程序员。普通的，扔到人群里根本就找不到的那种。而且，既没有身高也没有相貌！

但是，我崇信着“诚恳待人，个性永存”这个宗旨。我在生活中是这样，在网络里也是如此，期待着能有一天看到我的成功。

也许，我读的古书太多了，个性中充满着传统中国人的特点，却又因为看过太多的文艺复兴类作品而隐藏着叛逆的性格。终于，我在1999年7月毕业了，离开了那个让我恨、让我恼、让我烦，而又让我留恋、让我爱的学校！

在我的选择中，我没有选择那种稳定的单位，却为着个人的一点奢望而选择了漂泊。虽然我知道，这样会让那个我曾经喜欢过的女孩永远也不可能接受我，可我，还是一往无前地走下去！因为，我不甘心，不甘心在那种企业里按照很早以前就已经规划好的路子走下去，技术员、助工、工程师、高工……也许在你退休那一年，你可以侥幸获得高工的身份，但，那时，你已经快六十岁了！一辈子已经终结了！（我心里大声疾呼！）

我渴望稳定，渴望一种稳定的生活，这是每一个传统中国人固有的思维。但是，我却不得不选择漂泊，选择那种可能根本看不到未来的生活方式。因为……

因此，在网上，我想实现我的梦想。我选择了真诚。我用我的真实身份来注册网络上的所有信息，我用我真实的名字登录网站，登录在OICQ上。并告诉每一个愿意与我交朋友的人：“我的资料都是真实的。”我的这个名字是我已经用了十多年的笔名，在实际生活中我也在使用她。

与此同时，我得到了很多朋友的支持，他们也回报给我以真诚。告诉了我他们的真实姓名和身份，我们成了真正的朋友，有些还是好朋友！

不过，人生总不会那么一帆风顺。在你付出的同时，往往也会招来别人的嘲笑。有些人笑话我说：你，太幼稚了！你，太傻了！——我，无言！在无言的背后，我默默地将他们那笑得一颤一颤的头像删除了。但，却意外地发现，那只是将他们放在了另外的一个文件夹里，他们依旧在那里笑呀笑呀！我又找到了那个文件夹将他们彻底地从我的硬盘上删除了。但，他们却依旧在网络上笑呀笑呀，笑得我心里发寒！——我沉默了，我再一次无言了……

附录 E

作者随笔二——毕业十年记

新
知
识
PDG

青人来自崇山间，
润心爱在百丈崖。
意如纸雪斩污处，
愿胜天下唯文雅。

仍然是这样的一首诗开始，但心境早已和六年前大相径庭了，毕竟年已三十有四，却一无所成，除了留下了一些文字和一些已经过期的项目和代码可以回忆，还有什么？

2009年离开了CSDN论坛，因为对新制度和管理体系的不满，10年在CSDN论坛，当了9年的版主，最后给兄弟们留下一点回忆的地方我也无法保留了。现在偶尔漂流于newsmth的论坛上，发发牢骚，看看joke，选点卖点已经闲置的二手配件，更主要的原因是曾经心态对抗的两篇文字在当时初创的thinker板块被长期置底似乎长达半年以上，那个开创thinker板块的哥们还想帮我引荐几位做心理学研究的朋友一同推动全面深入的软件开发心理学的文字的后续撰写。

2010年初，终于横下心来注册了一个公司，结束了我接近4年完全无业的生活方式，一方面是因为经济危机让我最近两年损失了不少于三场培训，很多合作公司也都陷入了资金困境，我也失去了经济来源，另一方面是终于找到了这几年投入了大量精力的技术有了产品出口，我申请了自己的专利，和弟兄们一起挂起了自己的招牌，正式开始打天下了。

曙光乍现无极时，乌云转瞬遮天日。

在2004年本书第1版出版后，我突然发现我在电信设计院的工作已经到了最后的时日。一天，分管软研本来对我一直青睐有加的副总突然给我打来一个电话，说：“小白，听说你在集团说了，你可以让四川院拿到项目，也可以让四川院一个项目也拿不到？”我立刻明白，有人在背后下了黑手！

当时，我已经正式借调在电信集团网络发展部商务处工作三个多月了，领导对我非常好，不到半个月，我就开始独立办案，而且曾经在一个星期内因为谈判策略得当被处长两次表扬。安保部的人员曾经惊讶于第二天就拿到了他们昨天才把审批下来的批文交给我要求购买的笔记本。

离开集团前，一个对我非常好的副处偷偷拉着我对我说：“小白，回去是不是有什么事情？如果需要集团出面，我们给你说去。”我当时不想把事情闹大，虽然知道是被人黑了，但还是决定忍耐，于是对她说：“谢谢您了，没什么大事，真的，就是院里需要我回去承接一些项目。”当年只要在电信集团借调满一年的员工都会被直接留在电信集团作为正式员工任职，我走后华信设计院和广东设计院的各一个哥们留在了集团任职，我也很清楚离开和留下分别意味着什么。

在正式离开集团的那天中午，我们处长把我叫到他面前，问我：“那个项目四川院能不能做？”我知道他是担心我走之后，四川院技术力量能否做好这个项目。当时这个项目的合同还没有签署，而且本来就是我从广东某电信器材公司手中抢过来的二期建设合同，只要我说一个不字，这个50万的合同肯定就转手了。我当时没有任何犹豫地对他说：“您放心，四

川院肯定有能力做好这个项目，如果做得不好，您给我电话，我给您几个人的名字，您直接让四川院派他们来北京做。”我很清楚当时我那支团队的技术力量，因为其中一多半人都是和我合作了两年多的弟兄，其中两个还是跟着我从上海回到成都的，还有五个人是在成都的时候就跟着我做项目的弟兄。

我坚持，好的程序员，不应该因为感情用事而行欺骗之道！

苦甘相随寻常事，我自横心我自定。

2005年8月份我再次进入了中科院系统，这次我把希望寄托于以为海龟外籍的研究员身上，因为他做的是算法，是我最感兴趣的东西，也是吸引我进入软件行业的本原。为此，我放弃了另外一家做智能网管产品的公司给我开出的比他多50%的薪金+美国LI签证+公司股份+第二年整个中国区公司交给我的条件。

风波淡卷举眉愁，双击并举待择端。云开初现复遮日，淡抹云霞旭续出。这是曾经应邀撰写的《[技术讨论]第一日——启动的摸索与新技术的对话》(<http://blog.csdn.net/qingrun/archive/2005/12/21/558476.aspx>)的开篇之辞。

没想到，风云突变就在转瞬间，2006年4月在一切已经进入轨道之后，突然间，老板喊我过去对我说，因为你一月份提前回家了一个星期，所以，我要扣除你一个星期的工资。但是实际扣除的工资是1/3之多。诸如此类种种言而无信的行为随之发生，于是，我于无奈中提出了辞职和索赔的要求。

在这个过程中，我进行了录音笔录音的方式，其实没有录到什么关键信息，但是，这个研究员曾经看到过我处理事情的严密性，于是都信以为真，补偿了我一个月工资，还补发了前面无理扣发的工资。

他后来又挽留我两次，希望我能继续跟他干，我都拒绝了，我对他说：“因为你的言而无信，我不能再陪你浪费生命了。”

最终，他破天荒地在所外面的蒙骨乡请我吃了送别饭，这顿饭让所有的弟兄震惊，因为以前弟兄的送别饭都是在类似成都小吃这样三五十块钱就能搞定的小饭馆里面进行的。

可是，我的梦却再次破碎了！我的损失比他大得多。

我留下的是给所里弟兄们的好印象，我对弟兄们的照顾和关心，做事的细心和严谨。这也是我第1次作为弱势项目经理进行的工作阶段。

后来，我在blog中连续撰写了5篇如何与纯技术的老板合作的文字，依次如下：

1) [企业管理] 如何与一个纯技术的老板合作第一篇：

<http://blog.csdn.net/qingrun/archive/2006/07/05/880452.aspx>

2) [企业管理] 如何与一个纯技术的老板合作第二篇：

<http://blog.csdn.net/qingrun/archive/2006/07/18/935575.aspx>

3) [企业管理] 如何与一个纯技术的老板合作第三篇——应答与响应之补充：

<http://blog.csdn.net/qingrun/archive/2006/08/02/1009940.aspx>

4) [企业管理] 如何与一个纯技术的老板合作第四篇——技术型企业的管理是什么：

——<http://blog.csdn.net/qingrun/archive/2006/08/05/1023914.aspx>

5) [企业管理] 如何与一个纯技术的老板合作第五篇——当初我为什么离开电信还会选择进入中科院:

<http://blog.csdn.net/qingrun/archive/2006/11/17/1389702.aspx>

在那段时间,我对管理进行了很长时间的思考和反思,这五篇内容以及我的《交换编程方法论》和后来的《可度量绩效管理模型》等都是在那段时间内完成的。

磨砺苦寒何处来,宝剑梅花希自喻。

此后的接近四年的时间中,青润通过朋友介绍的培训和自己的技术基本上维持了生活状态,于是就没有继续寻找公司任职的机会。期间有公司邀请青润到深圳或武汉就职,前者因为邀请者的地位不对等,无法产生邀请的实质尊重,后者因为产品方向不够吸引青润,所以均被青润所拒绝。此后的四年间其中两年的收入都不到2万元人民币,此外还要支付每年2.4万元的房租。

在内心深处,青润是不想自己开公司的,一直都希望能找到一个好老板,做一些事情,但是如果找不到,甘愿用处于贫困底线的状况来推动自己有兴趣的事情的进展。

很多朋友鼓励青润说,是金子总会发光。但是事实上,很多金子还是会被埋没一生的。毕竟人的生命是有限的,有限的生命时间内不可能与时间上无限的社会发展相对抗,否则沉沙生时对每一个人都是可能的,至于能否光耀身后其实对于其本人已经没有任何实际意义了。

世事洞明皆学问,人情练达即文章。

2010年的某天在smth上看到了一篇文章《读史的三个境界》,看后,大呼过瘾,于是撰文:[软件人生]读史的三个境界看软件业(<http://blog.csdn.net/qingrun/archive/2010/03/05/5345142.aspx>)。

读史的第一个境界是权术。很多人可能看不起试图从历史中学习权术,其实懂得权术的人才是成熟的人。

有评论这样说,达到读史第一境界的人,他的思想已经进入不知道善恶的状态。

读史的第二个境界是天下大势。如果说达到第一境界的已经是人中俊杰,那么达到第二境界可谓凤毛麟角,因为只有把历史读通才能达到这个境界。

有评论这样说,达到读史第二境界的人,他的思想已经进入了不知道对错的状态。

读史的第三个境界是纯粹为了历史。

有评论这样说,达到读史第三境界的人,他的思想已经进入了不知道真假的状况。不知道善恶,不知道对错,不知道真假,一问三不知才是真正的历史专家啊!

中国的软件业已经发展20年了,这20年间我们看到了国外软件技术的更新,多次的开发方法和表现手段的革命,直到近几年国外大企业玩概念拼商业,搞名词给我们带来的虚假和真实……

青润的多篇文章提到:不要盲目的崇信国外的词汇,看清楚什么是真伪。诸如:

[企业管理] 中国软件业缺什么:

<http://blog.csdn.net/qingrun/archive/2006/08/31/1149335.aspx>

[全程建模] 关于 SOA 与新旧概念的一点看法:

<http://blog.csdn.net/qingrun/archive/2006/08/28/1131316.aspx>

[建模技术] 精通它,才能毁灭它!——谈 UML 与科学和工程的关系:

<http://blog.csdn.net/qingrun/archive/2005/03/15/320157.aspx>

现阶段,我们不要奢求中国出现大师级别的人物,我们的状况还是为了解决问题,为了不断积累,毕竟我们的时间还很短,我们的积累远远不充足。也可以说,现阶段敢于在自己头上挂上软工大师之类头衔的人物,我们可以一体打倒。

风平浪静疑无物,探绝挖尽始见真。

中国的软件业已经走过了 20 年,道路漫漫而悠长,我辈不求真为圣,但愿潜心筑根基!到这里留下一个 5 年或者 10 年后的期盼,希望本书下一个版本能够有朋友与我合著,而不再是我一个人撰写,也期盼中国软件业能真正地成为体系形态的那一天。共同努力吧。

数字资源
PDG

附录 F

文档信息和文档修订历史

本附录提供了有关文档的元数据，包括文档的创建、修改和删除历史记录。这些信息对于文档的管理和追踪非常重要。附录 F 详细列出了文档的元数据，包括文档的创建、修改和删除历史记录。这些信息对于文档的管理和追踪非常重要。

附录 F 详细列出了文档的元数据，包括文档的创建、修改和删除历史记录。这些信息对于文档的管理和追踪非常重要。

附录 F 文档信息和文档修订历史

附录 F 详细列出了文档的元数据，包括文档的创建、修改和删除历史记录。这些信息对于文档的管理和追踪非常重要。

文档名称	创建时间	修改时间	删除时间
文档 1	2023-01-01	2023-01-01	2023-01-01
文档 2	2023-01-01	2023-01-01	2023-01-01
文档 3	2023-01-01	2023-01-01	2023-01-01
文档 4	2023-01-01	2023-01-01	2023-01-01
文档 5	2023-01-01	2023-01-01	2023-01-01

附录 F 详细列出了文档的元数据，包括文档的创建、修改和删除历史记录。这些信息对于文档的管理和追踪非常重要。

文档名称	创建时间	修改时间	删除时间
文档 1	2023-01-01	2023-01-01	2023-01-01
文档 2	2023-01-01	2023-01-01	2023-01-01
文档 3	2023-01-01	2023-01-01	2023-01-01
文档 4	2023-01-01	2023-01-01	2023-01-01
文档 5	2023-01-01	2023-01-01	2023-01-01

附录 F 详细列出了文档的元数据，包括文档的创建、修改和删除历史记录。这些信息对于文档的管理和追踪非常重要。

文档名称	创建时间	修改时间	删除时间
文档 1	2023-01-01	2023-01-01	2023-01-01
文档 2	2023-01-01	2023-01-01	2023-01-01
文档 3	2023-01-01	2023-01-01	2023-01-01
文档 4	2023-01-01	2023-01-01	2023-01-01
文档 5	2023-01-01	2023-01-01	2023-01-01

1. 文档信息

书名:	软件工程之全程建模实现
版本:	5.8.8
作者:	青润
单位:	北京汇科致安科技有限公司
创建日期:	2003-07-22
上次更新日期:	2010-08-06
使用者:	项目经理、系统分析员、分析设计人员、测试经理、编码实施/测试人员、客户方高级技术人员、行业专家、学习过 UML 基础知识需要了解 UML 建模方式在软件工程过程中具体实现方式的人
类别名:	软件工程
标题:	全程建模实现
分类:	书稿

2. 文档修订历史

《基于 Rose 的全程建模》文档修订历史记录

日 期	版 本	说 明	作 者
2002-8-28	1.0.0	创建初始文档	青润
2002-9-1	1.1.0	完成了文档的初稿,并作了全文格式的修正	青润
2002-9-3	1.1.1	修改了文中的部分问题,补充了分析模型的一些内容	青润
2002-9-4	1.1.2	修改了文章的小标题和部分内容	青润
2003-7-23	1.1.3	在文档合并的时候修改了部分内容	青润

《基于 Rose 的全程建模之二——Rose 中的协作开发》文档修订历史记录

日 期	版 本	说 明	作 者
2002-10-14	1.0.0	创建文档	青润
2002-10-30	1.1.0	添加了文档与模型结合的部分	青润
2002-11-29	1.1.1	修改了部分图形	青润
2002-12-3	1.2.0	删除了配置管理部分,添加了分包特殊操作部分	青润
2003-5-9	1.2.1	修改了一些用词问题	青润
2003-7-18	1.3.0	增加了一个技巧的描述	青润

《基于 Rose 的全程建模之三——如何进行需求分析》文档修订历史记录

日 期	版 本	说 明	作 者
2002-10-14	1.0.0	创建文档	青润
2002-10-20	1.0.1	完成了 2.1 的内容	青润
2002-11-26	1.0.2	初步完成了 2.2 中的内容	青润

(续)

日 期	版 本	说 明	作 者
2002-11-27	1.0.3	修改了 2.3 中的内容	青润
2002-11-28	1.0.4	正在撰写 2.4 中的内容	青润
2002-12-4	1.0.5	修改了部分内容	青润
2002-12-5	1.0.6	修改了 2.5 中的内容和部分格式	青润
2002-12-11	1.0.7	添加了 2.3.3.1 的内容	青润
2002-12-13	1.0.8	修改了部分目录	青润
2002-12-16	1.0.9	添加了 2.2.4.3, 修改了 2.3 中的内容	青润
2002-12-26	1.0.10	添加了 2.9 的内容, 但还没有完成	青润
2002-12-31	1.0.11	添加了 2.1.9 中的内容	青润
2003-1-2	1.0.12	添加了 cmm 的部分阐述	青润
2003-1-16	1.0.13	添加完成了 2.1.9 中的主要内容	青润
2003-1-17	1.0.14	添加了一个新的 2.1.8 和 1.2.4 的内容	青润
2003-2-1	1.0.15	添加了 2.1.7 的图和表	青润
2003-2-16	1.0.16	添加了 2.2.2 中的描述	青润
2003-2-18	1.0.17	添加了 2.1.1、2.4.2 和 2.4.3 中的描述	青润
2003-2-18	1.0.18	修改了 2.3.5 和 2.4.3 中的内容	青润
2003-2-21	1.0.19	补充了 2.3.4 中的内容, 添加了 2.4.3	青润
2003-2-21	1.1.0	修改了 2.5 的目录结构	青润
2003-3-11	1.1.1	添加了 2.4.3, 修改了 2.4.2	青润
2003-3-11	1.2.0	添加了 3 界面设计的目录结构	青润
2003-3-22	1.2.1	修改了 3.1 的内容	青润
2003-3-31	1.3.2	修改了 3.1 的内容, 添加了习惯对人影响的内容	青润
2003-4-2	1.3.3	修改了 3 中的一点内容	青润
2003-4-2	1.3.4	添加了 3 中关于 RUP 的描述	青润
2003-4-4	1.3.5	添加了 3.1.2.3 界面设计的时间特点的描述	青润
2003-4-9	1.3.6	添加了 3.2 中的一些内容	青润
2003-4-9	1.4.0	添加了每个过程的参与人员建议目录和部分内容	青润
2003-4-17	1.4.1	修改了一些内容	青润
2003-4-24	1.4.2	修改了一点查找用例和主角部分的内容	青润
2003-5-23	1.4.3	修改了界面设计中关于 RUP 的内容	青润
2003-6-1	1.4.4	再次修改了界面设计中关于 RUP 的内容	青润
2003-6-3	1.4.5	完成了界面设计的步骤和参与人员建议的内容	青润
2003-6-13	1.4.6	修改了界面设计的内容	青润
2003-7-7	1.4.7	添加了用例分包的示意图	青润
2003-7-9	1.4.8	修改了一些内容	青润
2003-7-13	1.4.9	修改了 2.4 的内容	青润
2003-7-20	1.4.10	修改了一些内容和小问题	青润
2003-7-21	1.4.11	修改了如何进行交互建模中的内容	青润
2003-7-22	1.4.12	初步完成了需求阶段的全部内容	青润
2003-7-23	1.4.13	修改了一些小问题	青润

《软件开发方法之交换编程》修订历史记录

日 期	版 本	说 明	作 者
2006-7-6	0.0.1	创建了文档的整体目录和初步考虑内容	青润
2006-7-6	0.1.0	完成了整个目录的创建	青润
2006-7-7	0.1.1	开始撰写优劣分析部分的内容	青润
2006-7-9	0.1.2	开始按顺序修改内容	青润
2006-7-9	0.1.3	完成了3, 优劣分析部分的内容	青润
2006-7-18	0.1.4	基本上完成了全文的内容	青润
2006-8-11	0.2.0	根据霍泰稳的建议作了少量格式调整	青润
2006-10-23	0.2.1	补充了操作建议和操作建议分析	青润
2006-12-4	0.3.0	修订为论文样式	青润

《可度量绩效管理模型》修订历史记录

日 期	版 本	说 明	作 者
2008-3-21	0.0.1	创建了文档的整体目录和初步考虑内容	青润
2008-3-25	0.1.0	增加了薪资收入组成模型的内容	青润
2008-3-25	0.1.1	将绩效奖金分为开发工作绩效和管理工作绩效两个部分	青润
2008-4-3	0.1.2	修订了引言部分的内容	青润
2008-4-7	0.2.0	完成了全文内容的补充	青润
2008-5-3	0.3.0	补充绩效管理实现模型内容	青润
2008-5-6	0.3.1	修订了绩效管理实现模型的图	青润

全书修订历史记录

日 期	版 本	说 明	作 者
2003-7-22	4.0.0	初步创建文档的一级目录	青润
2003-7-23	4.1.0	初步创建了文档的二级目录	青润
2003-7-23	4.2.0	成功完成了全文的第一次整合	青润
2003-7-24	4.3.0	添加了参与人员建议和常见问题的目录整合	青润
2003-7-25	4.4.0	完成了第一次检查中到第2章需求工程的所有内容的查看	青润
2003-7-26	4.4.1	修改了一些问题和编号的事情	青润
2003-7-27	4.4.2	添加了对用例关系的描述(尚未完成)	青润
2003-7-28	4.4.3	基本上完成了全文内容的修改与调整	青润
2003-7-28	4.5.0	完成了对全文的第二次修订	青润
2003-7-29	4.6.0	重新修订了书名, 并修改了一些问题	青润
2003-7-30	4.6.1	添加了第1章的内容	青润
2003-7-30	4.6.2	添加了第2章、第3章的前两节内容	青润
2003-7-31	4.6.3	开始修改第4章代码模型一致性中前两节序言、概述的内容	青润
2003-7-31	4.6.4	修改了全文的内容	青润
2003-8-1	4.6.5	填写了部分章节前面的内容	青润
2003-8-2	4.6.6	修改了部分错误的引用关系	青润
2003-8-3	4.6.7	修订了全书所使用的图片索引关系	青润
2003-8-13	4.6.8	增加了第6章第6节对 Rational 在 IBM 中的顾虑的内容	青润

(续)

日 期	版 本	说 明	作 者
2003-8-14	4.6.9	增加了第6章第7节软件人员的分化的内容	青润
2003-8-14	4.6.10	初步完成了作者随笔的内容	青润
2003-8-15	4.6.11	修订了书中的一些在打印稿中发现的错误,开始对全书进行第一次校订	青润
2003-8-15	4.6.12	修改一些文中的引用关系和部分错误用词	青润
2003-8-17	4.6.13	修改了作者随笔的内容	青润
2003-8-17	4.6.14	修改了部分描述不够准确的内容	青润
2003-8-18	4.6.15	添加了第6章第8节关于 Rose 与 Together 的内容	青润
2003-8-18	4.6.16	修改了作者随笔中的一些描述	青润
2003-8-20	4.6.17	修改了书中的一些用词	青润
2003-8-22	4.6.18	增加了单位名称和用户描述	青润
2003-8-23	4.6.19	根据打印第九稿的修改内容修改了 2.3.7.2 我们要为用户考虑以前的全部内容	青润
2003-8-24	4.7.0	基本完成了打印第九稿修改内容的全部修订	青润
2003-8-25	4.7.1	再次开始全文修订工作,并完成了 3.3.2.1.2 以前内容的修订	青润
2003-8-26	4.7.2	完成了全书的修订工作	青润
2003-8-27	4.7.3	修改了关于 CMM 的一些观点	青润
2003-8-27	4.7.4	修改了几处用词错误	青润
2003-8-28	4.7.5	完成了到第2章第7节前的最终修订工作	青润
2003-8-29	4.7.6	完成了全文正文部分的最终修订工作	青润
2003-8-30	4.7.7	根据凌晨三点结束的聚会,笔者对文中内容进行了一些修改	青润
2003-8-31	4.7.8	笔者对全文进行了一些用词修改	青润
2003-8-31	4.7.9	根据新的认识修改了用户范畴的描述和文档信息的部分内容	青润
2003-9-1	4.7.10	笔者完成了对全文的最终修改工作	青润
2003-11-17	4.7.11	修改了一个错别字	青润
2003-11-19	4.7.12	修改了两个错别字	青润
2003-11-21	4.7.13	修改了六个错别字	青润
2003-11-23	4.7.14	添加了分析模型常见问题部分的内容,同时对其他内容进行了少量的修改	青润
2003-11-24	4.7.15	添加了用例对主角是否可以有操作的评论内容	青润
2003-11-26	4.8.0	根据孙编辑的要求开始对全文进行第一次修订,主要修订内容包括文字、断句、段落等	青润
2003-11-27	4.8.1	修改了需求调研部分的内容,添加了一幅图来说明软件工程实现过程的步骤	青润
2003-12-4	4.8.2	修改了第3章第3节前的内容	青润
2003-12-11	4.8.3	修改完了 3.3.2.1.6 以前的全部内容中的用词	青润
2003-12-13	4.8.4	修改了第3章第4节前的所有内容	青润
2003-12-15	4.8.5	修改了 3.4.1.3 以前的所有内容	青润
2003-12-16	4.8.6	修改了 3.4.3 以前的全部内容	青润
2003-12-17	4.8.7	进行了章节结构体系的调整	青润

(续)

日 期	版 本	说 明	作 者
2003-12-19	4.8.8	完成了第4章以前全部内容的文字修订和补充修改	青润
2003-12-20	4.8.9	修改了第4章第3节前的所有内容	青润
2003-12-21	4.8.10	修改了第4章第5节前的所有内容	青润
2003-12-22	4.8.11	修改了第5章第4节前的所有内容	青润
2003-12-23	4.8.12	修改了第6章第8节前的所有内容	青润
2003-12-24	4.8.13	完成了全书所有内容的一次修订	青润
2003-12-26	4.8.14	修改了2.3.1.2以前的错误和备注信息	青润
2004-1-14	4.8.15	按照要求修改书中对主角和用例的称呼为 Actor 和 Use Case。 没有修改引用资料中的称呼	青润
2004-1-15	4.8.16	删除了2.3.5和2.3.6两小节的内容,同时对全文的引用关系进行了调整	青润
2004-1-15	4.8.17	对全文进行了一次口语化内容的修改	青润
2004-1-28	4.8.18	修改了全文的不规范用词	青润
2004-1-29	4.9.0	删除了原有的第一章,后续章节编号的提升完成了第2章相关的全部内容	青润
2004-1-30	4.9.1	后续章节编号的提升完成了第3章相关的全部内容	青润
2004-1-30	4.9.2	完成了后续章节编号的提升和引用关系的修改	青润
2004-3-4	4.9.3	修改了几处用词问题和两处描述的细化	青润
2004-3-17	4.9.4	修改了2.4.2.5以前所有图表的索引关系	青润
2004-3-18	4.9.5	完成了全书所有的图表序号和索引关系	青润
2004-3-18	4.9.6	完成了字号的修改	青润
2004-3-18	4.9.7	修改完成了所有的目录结构	青润
2004-3-19	4.9.8	修改完成了注释说明文字的位置调整	青润
2004-3-20	4.9.9	修改完成了1.4.5以前的口语化文字	青润
2004-3-21	4.9.10	修改完成了全文的口语化文字和格式	青润
2004-4-6	4.10.0	修改完成了全文的一校稿内容	青润
2004-8-17	4.10.1	修改了少量错别字	青润
2004-8-18	4.10.2	修改了少量错别字	青润
2004-8-19	4.10.3	修改了三处用词问题	青润
2004-10-17	4.10.4	修改了一处描述不当的错误	青润
2005-1-31	4.10.5	对全书通读了一遍,自己找到了26处出版错误	青润
		根据最近两年的积累开始新的内容的修订和补充。 这次补充主要包括如下一些内容: 1) 在2006年1月1日开始的培训中,伊达提出一个观点,他认为用例阐述中的子流和分支流与设计模型中具体的设计类是应该有对应关系的 2) 业务建模内容以及业务用例模型和用例模型之间的对应关系,以及这种对应关系对系统的分阶段规划和开发的指导意义和现实意义 3) 可能考虑添加对 XDE、Together 等设计工具的支持内容,此内容待定 4) 增加每个阶段输入输出工件的介绍和说明 5) 其他需要进行的修改和补充	青润
2006-1-3	5.0.0		青润

(续)

日 期	版 本	说 明	作 者
2006-1-8	5.0.1	在 1.5.3 节中添加了对用例阐述子流和分支流与设计模型中具体设计类的对应说明	青润
2006-1-8	5.0.2	增加了感谢的内容(未完成),修改了本书组织结构的内容	青润
2006-5-29	5.1.0	根据读者反馈增加了输入输出的全部子目录和 UML 元素的全部子目录	青润
2007-7-25	5.2.0	进行全文修订,修订到 2.4.4 节	青润
2008-10-13	5.2.1	修订了两处不一致问题	青润
2009-10-1	5.3.0	修订了整体文档目录	青润
2009-12-2	5.3.1	开始撰写序章内容	青润
2009-12-2	5.3.2	撰写完成序章第一节概述的内容	青润
2009-12-2	5.3.3	补充了第六章的目录结构,对第一章第一节进行了少量文字补充修订	青润
2009-12-4	5.3.4	补充了配置库的基本目录	青润
2010-2-19	5.3.5	初步完成了软件工程概论之概述节的撰写	青润
2010-2-20	5.3.6	修订完成了软件工程概论之全程域默下的软件工程模型分类	青润
2010-2-21	5.3.7	基本完成了序章第一节内容的撰写	青润
2010-2-21	5.3.8	重新整理了概述的内容,将部分内容转移到第二节和第三节中	青润
2010-2-22	5.3.9	修订了第一节概述的三个小节,同时获得了 CSDN 网友 kongdong 的回复,同意我无偿引用其 blog 文字中的一些内容	青润
2010-2-22	5.3.10	修订了第 1 章部分内容	青润
2010-2-23	5.3.11	撰写完成了序章第 3 节软件工程方法的内容	青润
2010-2-23	5.3.12	修订了少量文字	青润
2010-2-24	5.3.13	补充修订完成了原型法的全部内容	青润
2010-2-25	5.3.14	补充修订完成了界面设计一节的全部内容	青润
2010-2-26	5.3.15	修订了书中组织方法论的下级目录结构	青润
2010-3-1	5.3.16	补充了结对编程、交换编程的内容	青润
2010-3-2	5.4.0	调整了章节目录,将组织方法论和心理学篇章放置于技巧之前	青润
2010-3-2	5.4.1	修订了界面设计的部分用词和内部结构	青润
2010-3-4	5.4.2	修订完成了结对编程的内容	青润
2010-3-4	5.4.3	进行交换编程的修订,未完成	青润
2010-3-5	5.4.4	修订了第一张第三节中如何与用户交流的内容	青润
2010-3-6	5.4.5	补充并初步完成了绩效管理算法模型	青润
2010-3-7	5.4.6	删除了过时章节对 Rational 在 IBM 中的顾虑	青润
2010-3-7	5.4.7	修订补充了软件组织方法论章节的概述	青润
2010-3-8	5.5.0	为了适应出版,进行了章节的全部目录结构的修订	青润
2010-3-8	5.5.1	修订了交换编程的内容,补充了参考文献的具体数量和来源,修订补充了前文中相关的内容	青润
2010-3-8	5.5.2	修订完成了可度量绩效管理模型的内容	青润
2010-3-10	5.5.3	补充修订了企业代码库的内容,并完成了知识库体系构建一节前三个分类的内容撰写	青润

(续)

日 期	版 本	说 明	作 者
2010-3-10	5.5.4	撰写完成了实用性知识库体系构建的全部内容	青润
2010-3-10	5.5.5	修订到 UseCase 模型部分的输入输出内容	青润
2010-3-11	5.5.6	撰写完成了配置管理工具如何使用的内容	青润
2010-3-11	5.5.7	撰写完成了全部内容的输入输出、UML 元素和用途的撰写	青润
2010-3-11	5.5.8	将项目管理者心理学、企业管理者心理学的内容融合到书中，完成了格式修订，待内容修订	青润
2010-3-11	5.5.9	将项目计划如何做、评审如何做的内容融合到书中，并删除了 rose 与 together 对比一节的内容，完成了格式修订，待内容修订	青润
2010-3-11	5.5.10	对内容进行了修订	青润
2010-3-12	5.5.11	对软件开发中的心态对抗问题研究一章进行了修订	青润
2010-3-12	5.5.12	新撰写了项目经理分类一节放在技巧和相关文章一章内	青润
2010-3-12	5.5.13	新撰写了管理中的权力使用分析一章	青润
2010-3-13	5.5.14	修订补充了高层管理者心态问题研究一节	青润
2010-3-13	5.5.15	将序章更新为第一章，重新进行了图表编号	青润
2010-3-14	5.5.16	修订完成第二章第三章的全部图表引用关系	青润
2010-3-15	5.5.17	修订完成了全书所有的图表和引用关联关系	青润
2010-3-15	5.5.18	修订了全书的参考文献	青润
2010-3-15	5.5.19	增加了模式的分类和模式的演变两段文字	青润
2010-3-15	5.5.20	修订了设计类的构建部分的内容	青润
2010-3-15	5.5.21	补充了附录二、附录三的内容	青润
2010-3-16	5.5.22	修订了配置管理工具如何用和项目经理分类的内容	青润
2010-3-18	5.5.23	对一些细节进行了修订：需求中 Use Case 阐述与设计类之间的关系，设计模型时序图命名	青润
2010-3-22	5.5.24	在项目经理分类中增加了一个分析案例的总结	青润
2010-3-22	5.5.25	增加了全程建模培训的介绍内容	青润
2010-3-22	5.5.26	增加了论本本主义一节	青润
2010-3-22	5.5.27	修订了所有的外部文字引用和例子的格式框	青润
2010-3-23	5.5.28	修订了 Use Case 阐述的层级关系与后续设计类和系统架构层的直接关系内容，修订了习惯对人的影响的内容	青润
2010-3-23	5.5.29	修订了论本本主义一节，补充了中国电信 ERP 实施的案例分析 and 国内 ERP 失败原因分析的内容	青润
2010-3-23	5.5.30	在主要建模模型章节补充 trufun 和 EA 工具的建模模型图	青润
2010-3-24	5.5.31	修改补充完成了 Use Case 模型中 Trufun 工具模型部分的撰写，同时完成了全书中对第 2 章图表引用的关系	青润
2010-3-24	5.5.32	修改补充完成了分析设计中 Trufun 工具模型部分的撰写，同时完成了全书中对第 3 章图表引用的关系	青润
2010-3-24	5.5.33	撰写了 Use Case 阐述模型化问题一节的内容	青润
2010-3-27	5.5.34	撰写了最简化开发过程的内容，修订了读者分类的内容	青润
2010-3-30	5.5.35	增加了 OO 原则中松耦合与高内聚的分析一节	青润
2010-3-30	5.6.0	基本上完成了全书所有图形的画布转移处理	青润
2010-3-31	5.6.1	最终修订完成了全书所有图形的画布转移处理	青润
2010-3-31	5.6.2	将第 8 章第 8 节调整为第 7 章第 4 节	青润

(续)

日 期	版 本	说 明	作 者
2010-3-31	5.6.3	对全书格式和图片大小进行了调整,同时修订了部分不恰当的用词问题,修订了我这个字的使用为笔者	青润
2010-4-1	5.6.4	对书中的部分不当用词进行了修订	青润
2010-4-1	5.6.5	调整了论本本主义到更适合的第7章	青润
2010-4-2	5.6.6	修订了第7章第3节中的部分内容和全书中其他的具体的用词问题	青润
2010-4-2	5.6.7	删除了全书中无用的空格,修订了格式	青润
2010-4-14	5.6.8	提交给出版社签最后一次审查修订了少量错别字和全部 extend 关系的图片修正	青润
2010-5-17	5.7.0	根据出版社编辑的建议,进行了修订,调整了部分用词和参考文献的统一化	青润
2010-5-20	5.7.1	根据出版社编辑查到的错误进行了修订:图表 1-7 中英文翻译,图表 3-34 和图表 3-42 各有两副图,图表 5-3 下有“OK”字样错误,第6章最后一行“这个方面”指代不明修改	青润
2010-7-22	5.8.0	拿到校样后修订完成了 2.3.6 前的全部内容	青润
2010-7-22	5.8.1	完成了 2.4.1 前的全部内容	青润
2010-7-23	5.8.2	完成了分包的目的之前的全部内容修订	青润
2010-7-24	5.8.3	补充了元 Use Case 的概念和边界定义划分方式等内容,第2章第6节以前的内容全部修订完成	青润
2010-7-24	5.8.4	完成了 3.3.7 以前的全部内容修订	青润
2010-7-25	5.8.5	完成了第7章第3节以前的全部内容修订	青润
2010-7-26	5.8.6	完成了全书第一次校样所有内容的修订	青润
2010-7-28	5.8.7	增加了附录六的随笔内容	青润
2010-8-6	5.8.8	修订了前言的内容	青润
2010-8-23	5.8.9	最终校样第1天检查完成,到第1章第1节结束	青润
2010-8-24	5.8.10	完成了第3章内容的修订	青润
2010-8-24	5.8.11	完成了 7.1.2 以前的全部内容修订	青润
2010-8-25	5.8.12	完成了全书校样的修订	青润

新平知覺

PDG

后 记

全书到了这里基本上也就画上了第一个句号，至于这个句号划得是否圆满，是否像个圆（阿 Q?），还需要各位读者去评价。笔者在这本书中主要做了一些抛砖引玉的铺垫。

笔者在第一本书出版后的 6 年多时间里又经历了多个企业的项目，这些项目规模不大，但有些接近涉密，有些技术难度较高，从传统的行业应用转移到一些专用软件系统，涉及的范围从软件扩展到硬件，从常规行业系统延伸到嵌入式软件的开发。

同时，笔者对外提供了多次技术咨询和专场培训，很多参加培训的朋友也都在各自的企业行业内进行了全程建模的实践，本书中有不少内容来自这些朋友的建议和经验总结。

笔者针对书的看法是：书上能记录下来的都是过去的东西，能够成书就说明这些东西经过了很多次的考验（至少本书中的所有内容都是这样的），相对而言比较成熟了。比如说本书中的内容大部分都是 2009 年以前笔者总结出来的经验。而不能成书的，或者说来不及成书的才是真正及时的崭新的知识。但是，新生事物必然也带来新的风险，那就是没有经过多次考验的新生事物可能具有不稳定不够安全的风险。



参考文献

- [1] Deepak Alur, John Crupi, Dan Malks. J2EE 核心模式[M]. 刘天北, 熊节等译. 北京: 机械工业出版社出版.
- [2] 陈坤森. 计算机多媒体之使用者接口设计探讨[J/OL]. 中国台湾地区的国家高速计算机中心助理研究员. http://www.nchc.gov.tw/chinese/07_publication/nchcs/Other/V8N1/p4.htm.
- [3] Rational Company. RUP 2002 中文版. 2002. 3.
- [4] 中国电子学会总部. 中国电子学会简报. 2000, 114 (1)期. 2000.
- [5] 林锐. 软件工程思想[M]. 2000.
- [6] 李维. Borland 传奇[M]. 北京: 电子工业出版社出版. 2002.
- [7] Carnegie Mellon University. Process Maturity Profile——Software CMM® CBA IPI and SPA Appraisal Results 2002 Year End Update. 2003.
- [8] Williams, L. & Kessler, R. Pair Programming Illuminated[M]. Pearson. 2002.
- [9] Constantine L. L. Constantine on People ware Constantine, Yourdon Press Computing Series, ed. E. Yourdon. Englewood Cliffs, NJ; Yourdon Press. 1995.
- [10] F. Brooks. The Mythical Man Month(人月神话)[M]. anniversary ed. Addison-Wesley. 1995.
- [11] Alistair Cockburn. Agile Software Development.
- [12] Gerald M. Weinberg. 程序开发心理学(银年纪念版)[M]. 北京: 清华大学出版社. 2003.
- [13] 尤登. 死亡之旅[M]. 周浩宇译. 北京: 电子工业出版社. 2005.
- [14] K. Beck. Embracing change with extreme programming. IEEE Computer, pages 70 - 77, Oct. 1999.
- [15] K. Beck. Extreme Programming Explained: Embrace Change. Reading[M]. Massachusetts: Addison-Wesley, 2000.
- [16] Cockburn, A. & Williams, L. The costs and benefits of pair programming, in ? Extreme Programming Examined[M]. Addison-Wesley. 2001, 14. 223 - 244.
- [17] Frank Padberg, Matthias M. M'uller, — Analyzing the Cost and Benefit of Pair Programming, Proceedings of the Ninth International Software Metrics Symposium (METRICS'03), 2003.
- [18] Laurie Williams, Richard L. Upchurch. In Support of Student Pair-Programming, SIGCSE 2001 2/01 Charlotte, NC, USA, 2001.
- [19] Laurie A. Williams, Robert R. Kessler. The Effects of Pair-Pressure and Pair-Learning on Software Engineering Education. .
- [20] Matthias M. Muller, Walter E Tichy, Case Study: Extreme Programming in a University Environment, IEEE, 2001.
- [21] Laurie Williams, Richard Upchurch. EXTREME PROGRAMMING FOR SOFTWARE ENGINEERING

- EDUCATION? 31 'st ASEE/IEEE Frontiers in Education Conference, 2001.
- [22] Jennifer Bevan, Linda Werner, Charlie McDowell. Guidelines for the Use of Pair Programming in a Freshman Programming Class. Proceedings of the 15th Conference on Software Engineering Education and Training (CSEET 02), 2002.
 - [23] Charlie McDowell, Linda Werner, Heather E. Bullock, Julian Fernald. The Impact of Pair Programming on Student Performance, Perception and Persistence. Proceedings of the 25th International Conference on Software Engineering (ICSE 03), 2003.
 - [24] Gerardo Canfora, Aniello Cimitile, Corrado Aaron Visaggio. Working in pairs as a means for design knowledge building: an empirical study. Proceedings of the 12th IEEE International Workshop on Program Comprehension (IWPC 04), 2004.
 - [25] Matthias M. Müller, Frank Padberg, An Empirical Study about the Feelgood Factor in Pair Programming, Proceedings of the 10th International Symposium on Software Metrics (METRICS 04), 2004.
 - [26] Laurie Williams, Anuja Shukla, Annie I. Antón, — An Initial Exploration of the Relationship Between Pair Programming and Brooks' Law. Proceedings of the Agile Development Conference (ADC'04), 2004.
 - [27] Hiyam Al-Kilidar, Peter Parkin, Aybüke Aurum, Ross Jeffery. Evaluation Of Effects Of PairWork On Quality Of Designs. Proceedings of the 2005 Australian Software Engineering Conference (ASWEC 05), 2005.
 - [28] Chih-wei Ho, Kelli Slaten, Laurie Williams, Sarah Berenson. Work in Progress-Unexpected Student Outcome from Collaborative Agile Software Development Practices and Paired Programming in a Software Engineering Course. 34th ASEE/IEEE Frontiers in Education Conference, 15-16, October 20 23, 2004.
 - [29] Neha Katira, Laurie Williams, Jason Osborne. Towards Increasing the Compatibility of Student Pair Programmers. ICSE'05, 625-626, May 15-21, 2005.
 - [30] Kelli M. Slaten, Maria Droujkova, Sarah B. Berenson, Laurie Williams, Lucas Layman, "Undergraduate Student Perceptions of Pair Programming and Agile Software Methodologies: Verifying a Model of Social Interaction". Proceedings of the Agile Development Conference (ADC'05), 2005.
 - [31] Lucas Layman, Laurie Williams, Jason Osborne, Sarah Berenson, Kelli Slaten, Mladen Vouk. How and Why Collaborative Software Development Impacts the Software Engineering Course. 35th ASEE/IEEE Frontiers in Education Conference, 9-14, October 19 22, 2005.
 - [32] Shaochun Xu, Vclav Rajlich. Pair Programming in Graduate Software Engineering Course Projects. 35th ASEE/IEEE Frontiers in Education Conference, 7-12, October 19 22, 2005.
 - [33] Terence C. Ahern. Work in Progress-Effect of Instructional Design and Pair Programming on Student Performance in an Introductory Programming Course. 35th ASEE/IEEE Frontiers in Education Conference, 11-12, October 19-22, 2005.
 - [34] Lucas Layman. Changing Students Perceptions: An Analysis of the Supplementary Benefits of Collaborative Software Development. Proceedings of the 19th Conference on Software Engineering Education & Training (CSEET 06), 2006.
 - [35] Shaochun Xu, Vclav Rajlich. Empirical Validation of Test-Driven Pair Programming in Game Development. Proceedings of the 5th IEEE/ACIS International Conference on Computer and Information Science and 1st IEEE/ACIS International Workshop on Component-Based Software Engineering, Software Architect-

- ture and Reuse (ICIS-COMSAR 06), 2006.
- [36] Laurie Williams, Lucas Layman, Jason Osborne, Neha Katira. Examining the Compatibility of Student Pair Programmers. Proceedings of AGILE 2006 Conference (AGILE'06), 2006.
 - [37] Jari Vanhanen & Casper Lassenius. Effects of Pair Programming at the Development Team Level: An Experiment. 2005 IEEE, 2005.
 - [38] Laurie Williams. Debunking the Nerd Stereotype with Pair Programming. Computer, 83-85, 2006.
 - [39] Kim Man Lui, Keith C. C. Chan. A Cognitive Model for Solo Programming and Pair Programming. Proceedings of the Third IEEE International Conference on Cognitive Informatics (ICCI 04), 2004.
 - [40] Joseph Chao, Gulgunes Atli. Critical Personality Traits in Successful Pair Programming. Proceedings of AGILE 2006 Conference (AGILE'06), 2006.
 - [41] Hanna Hulkko, Pekka Abrahamsson. A Multiple Case Study on the Impact of Pair Programming on Product Quality. ICSE'05, 495-504, May 15 21, 2005.

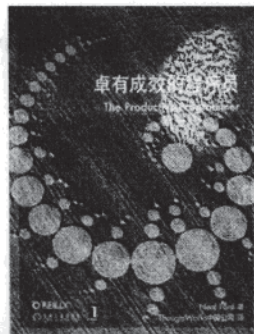




华章经典

为每一个团队提供最有价值的阅读服务

每一本书都值得您和您的团队一起阅读
分享阅读 分享成功





专业成就人生
立体服务大众

www.hzbook.com

填写读者调查表 加入华章书友会
获赠精彩技术书 参与活动和抽奖

尊敬的读者：

感谢您选择华章图书。为了聆听您的意见，以便我们能够为您提供更优秀的图书产品，敬请您抽出宝贵的时间填写本表，并按底部的地址邮寄给我们（您也可通过www.hzbook.com填写本表）。您将加入我们的“华章书友会”，及时获得新书资讯，免费参加书友会活动。我们将定期选出若干名热心读者，免费赠送我们出版的图书。请一定填写书名书号并留全您的联系信息，以便我们联络您，谢谢！

书名：

书号：7-111-()

姓名：	性别： ☐ 男 ☐ 女	年龄：	职业：
通信地址：		E-mail：	
电话：	手机：	邮编：	

1. 您是如何获知本书的：

☐ 朋友推荐 ☐ 书店 ☐ 图书目录 ☐ 杂志、报纸、网络等 ☐ 其他

2. 您从哪里购买本书：

☐ 新华书店 ☐ 计算机专业书店 ☐ 网上书店 ☐ 其他

3. 您对本书的评价是：

技术内容	☐ 很好	☐ 一般	☐ 较差	☐ 理由_____
文字质量	☐ 很好	☐ 一般	☐ 较差	☐ 理由_____
版式封面	☐ 很好	☐ 一般	☐ 较差	☐ 理由_____
印装质量	☐ 很好	☐ 一般	☐ 较差	☐ 理由_____
图书定价	☐ 太高	☐ 合适	☐ 较低	☐ 理由_____

4. 您希望我们的图书在哪些方面进行改进？

5. 您最希望我们出版哪方面的图书？如果有英文版请写出书名。

6. 您有没有写作或翻译技术图书的想法？

☐ 是，我的计划是_____ ☐ 否

7. 您希望获取图书信息的形式：

☐ 邮件 ☐ 信函 ☐ 短信 ☐ 其他_____

请寄：北京市西城区百万庄南街1号 机械工业出版社 华章公司 计算机图书策划部收
邮编：100037 电话：(010) 88379512 传真：(010) 68311602 E-mail: hzsj@hzbook.com