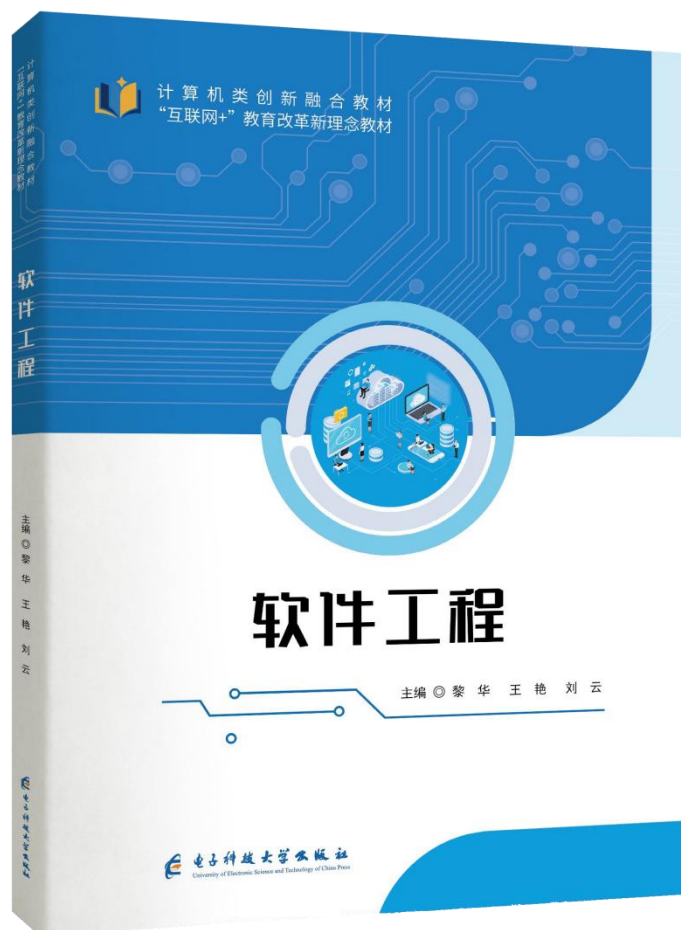


软件工程



类目：计算机类
书名：软件工程
主编：黎 华 王 艳 刘 云
出版社：电子科技大学出版社
开本：大 16 开
书号：978-7-5770-0508-9
使用层次：通用
出版时间：2025 年 7 月
定价：46.80 元
印刷方式：双色
是否有资源：是

策划编辑: 万晓桐
责任编辑: 刘 凡
封面设计: 惠清书装

计算机类创新融合教材
“互联网+”教育改革新理念教材



计算机类创新融合教材
“互联网+”教育改革新理念教材

软件工程



软件工程

主编◎黎华 王艳 刘云

软件工程

主编◎黎华 王艳 刘云



电子科技大学出版社

电子科技大学出版社
University of Electronic Science and Technology of China Press

从



计算机类创新融合教材
“互联网+”教育改革新理念教材



软件工程

主 审 ◎ 王春晓

主 编 ◎ 黎 华 王 艳 刘 云

副主编 ◎ 孙 冲 娜日斯 李旭光

顾云峰 孙伟伟 李亚红

傅 纲 李千目



电子科技大学出版社
University of Electronic Science and Technology of China Press

· 成都 ·

图书在版编目(CIP)数据

软件工程 / 黎华, 王艳, 刘云主编. —成都: 电子科技大学出版社, 2025. 7
ISBN 978-7-5770-0508-9

I. ①软… II. ①黎… ②王… ③刘… III. ①软件工程 IV. ①TP311.5

中国国家版本馆 CIP 数据核字(2023)第 161619 号

软件工程

RUANJIAN GONGCHENG

黎 华 王 艳 刘 云 主 编

策划编辑 万晓桐
责任编辑 刘 凡
责任校对 魏 彬
责任印制 梁 硕

出 版 电子科技大学出版社
成都市一环路东一段 159 号电子信息产业大厦九楼 邮编 610051
主 页 www.uestcp.com.cn
服务电话 028-83203399
邮购电话 028-83201495

印 刷 涿州汇美亿浓印刷有限公司
成品尺寸 210mm×285mm
印 张 11.5
字 数 340 千字
版 次 2025 年 7 月第 1 版
印 次 2025 年 7 月第 1 次印刷
书 号 ISBN 978-7-5770-0508-9
定 价 46.80 元

版权所有,侵权必究



前言

软件工程作为一级学科,如今已成为一个非常活跃的研究领域。软件工程是指导计算机软件开发与维护的工程学科,它采用工程的概念、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好技术方法结合起来,以便经济地开发高质量的软件并有效地维护它。严格遵循软件工程的方法,可以大大提高软件的开发效率和成功率,减少软件开发和维护中的问题。

与时俱进,是软件工程专业教材不断发展的需要。本书就是编者在教学研究、教学改革和软件工程丰富实践的基础上形成的新的知识结晶。本书针对计算机技术和其他相关专业学生的发展需求,系统、全面地介绍了软件工程的概念、原理、方法及其应用,详细介绍了软件生存周期、传统软件工程、面向对象软件工程、软件过程工程和软件体系结构设计的思想与实现方法,力图反映软件工程领域的最新发展,具有较强的系统性和可读性。本书内容包括:软件工程概论、软件过程、需求分析、软件设计、面向对象设计、面向对象软件开发过程模型、软件质量与软件测试、软件项目管理等。

作者在编写本书的过程中参考了国内外有关软件工程的专著、教材和论文,详见书后的主要参考文献。在此,向所有作者表示感谢。

由于软件工程近年来快速发展以及作者水平有限,书中难免存在错误和不足之处,敬请读者提出宝贵意见和建议。

目 录

第 1 章	软件工程概论	1
1.1	软件及软件危机	2
1.2	软件工程	5
1.3	软件工程的基本原则	9
	思考与练习	10
第 2 章	软件过程	11
2.1	软件开发模型	12
2.2	敏捷开发	15
	思考与练习	20
第 3 章	需求分析	21
3.1	概述	22
3.2	需求获取技术	28
3.3	需求分析方法	35
3.4	结构化分析方法	39
3.5	需求描述与评审	50
3.6	需求有效性验证	53
	思考与练习	54
第 4 章	软件设计	55
4.1	软件设计概述	56
4.2	软件设计的过程	64

4.3 软件设计的内容	68
4.4 结构化设计的方法	76
思考与练习	87

第5章 面向对象设计 89

5.1 常见的软件工程方法三论	90
5.2 结构化方法与面向对象方法的比较	91
5.3 面向对象的基本概念	94
5.4 面向对象方法的总结	97
5.5 面向对象建模(UML)	98
思考与练习	100

第6章 面向对象软件开发过程模型..... 101

6.1 迭代开发与瀑布式开发	102
6.2 迭代开发与快速原型法	103
6.3 迭代的适用范围	104
6.4 Rational 统一过程模型	104
思考与练习	110

第7章 软件质量与软件测试 111

7.1 质量与全面质量管理	112
7.2 软件质量模型	115
7.3 软件质量过程管理	124
7.4 软件测试	135
思考与练习	145

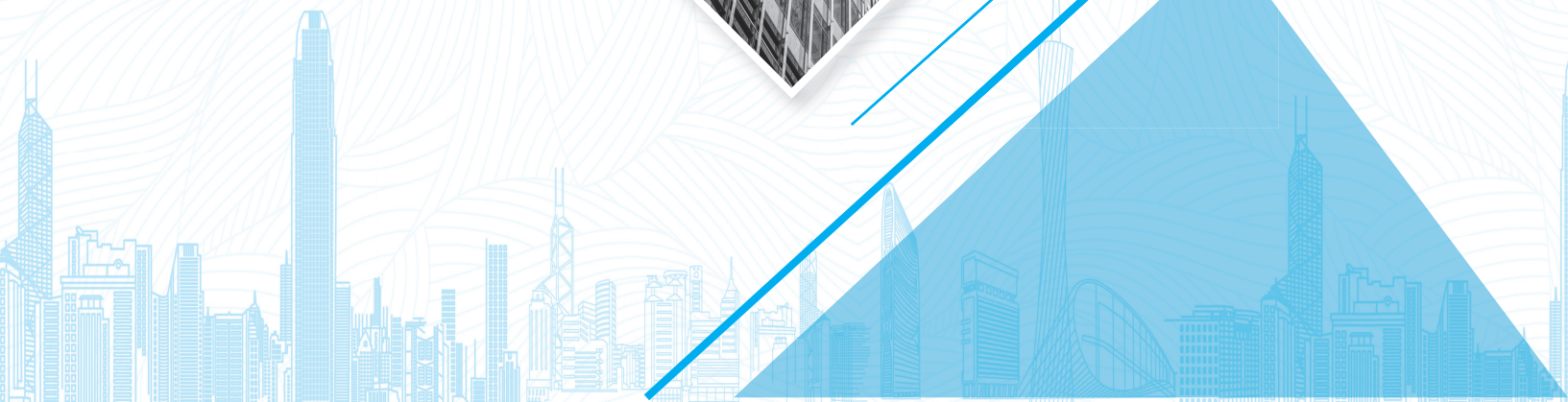
第8章 软件项目管理 147

8.1 项目管理的基本概念	148
8.2 进度管理	152
8.3 成本管理	161
8.4 风险管理	170
思考与练习	174

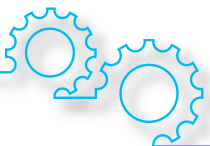
参考文献 175

第 1 章

软件工程概论



21 世纪是高度依赖计算机信息系统的时代,面对大量的计算机应用需求,怎样才能更有效地开发出各种不同类型的软件,是软件开发技术与软件工程所要解决的问题。软件工程是应用计算机科学、管理学、数学、项目管理、质量管理、软件人类工程学及系统工程的原则、方法创建的学科,它对指导软件开发、质量控制以及开发过程的管理起着重要的作用。本章将概括地介绍软件的基本概念与特点,软件工程学科的诞生背景与形成过程,软件工程学研究的内容与对象等,使读者对软件工程与软件开发技术有所认识。



1.1 软件及软件危机

随着计算机应用领域的扩大,软件规模也越来越大,复杂程度不断增加,使得软件生产的质量、周期、成本难以预测和控制,出现了软件危机。软件工程正是为了解决软件危机而提出的,其目的是改善软件生产的质量,提高软件的生产效率。经过几十年的实践与探索,软件工程正在逐步发展成为一门成熟的专业学科,在软件产业的发展中起到重要的技术保障和促进作用。

1.1.1 软件及其特性

软件是计算机系统的思维中枢,是软件产业的核心。作为信息技术的灵魂,计算机软件在现代社会中起着极其重要的作用。

1. 软件

计算机软件是由计算机程序的发展而形成的一个概念。它是与计算机系统上的程序、规程、规则及其文档和数据的统称。软件由两部分组成:一是计算机可执行的程序和有关的数据,二是与软件开发、运行、维护、使用 and 培训有关的文档。

程序是按事先设计的功能和性能要求执行的语句序列。数据是程序所处理信息的数据结构。文档则是与程序开发、维护和使用相关的各种图文资料,如各种规格说明书、设计说明书、用户手册等。在文档中记录着软件开发的活动和阶段成果。

2. 软件的特点

软件是一种逻辑产品而不是实物产品,软件功能的发挥依赖于硬件和软件的运行环境,没有计算机硬件的支持,软件毫无实用价值。若要对软件有一个全面而正确的理解,则我们应从软件的本质、软件的生产等方面剖析软件的特征。

(1) 软件固有的特性

① 复杂性。软件是一个庞大的逻辑系统。一方面在软件中要客观地体现人类社会的事务,反映业务流程的自然规律;另一方面在软件中还要集成多种多样的功能,以满足用户在激烈的竞争中对大量

信息及时处理、传输、存储等方面的需求,这就使得软件变得十分复杂。

②抽象性。软件是人们经过思考后加工出来的产品,一般存储在内存、磁盘、光盘、硬盘等载体上,我们无法观察到它的具体形态,这就导致了软件开发不仅工作量难以估计,进度难以控制,而且质量也难以把握。

③依赖性。软件必须和运行软件的机器(硬件)保持一致,软件的开发和运行往往受到计算机硬件的限制,对计算机系统有着不同程度的依赖性。软件与计算机硬件的这种密切相关性与依赖性,是一般产品所没有的。为了减少这种依赖性,在软件开发中提出了软件的可移植性问题。

④软件使用特性。软件的价值在于应用。软件产品不会因多次反复使用而磨损老化,一个久经考验的优质软件,可以长期使用。由于用户在选择新机型时,通常会提出兼容性要求,所以一个成熟的软件应当可以在不同型号的计算机上运行。

(2) 软件生产特性

①软件开发特性。由于软件固有的特性,软件的开发不仅具有技术复杂性,还有管理复杂性。技术复杂性体现在软件提供的功能比一般硬件产品提供的功能多,而且功能的实现具有多样性,需要在各种实现方案中做出选择,更有实现算法上的优化带来的不同,而实现上的差异会带来使用上的差别。管理复杂性表现在:第一,软件产品的能见度低(包括如何使用文档表示的概念能见度),要看到软件开发进度比看到有形产品的开发进度困难得多;第二,软件结构的合理性差,结构不合理使软件管理复杂性随软件规模增大而呈指数级增长。因此,领导一个庞大的项目组织进行规模化生产并非易事,软件开发比硬件开发更依赖于开发人员的团队精神、智力水平和对开发人员的组织与管理。

②软件产品形式的特性。软件产品的开发成本高昂而生产成本极低。硬件产品试制成功之后,批量生产需要建设生产线,投入大量的人力、物力和资金,生产过程中还要对产品进行质量控制,对每件产品进行严格的检验。然而,软件是把人的知识与技术转化为信息的逻辑产品,开发成功之后,只需对原版软件进行复制即可,大量人力、物力、资金的投入和质量控制、软件产品检验都是在软件开发中进行的。由于软件的复制非常容易,软件的知识产权保护就显得极为重要。

③软件维护特性。软件在运行过程中的维护工作比硬件复杂得多。首先,软件投入运行后,总会存在缺陷甚至暴露出潜伏的错误,需要进行“纠错性维护”;其次,用户可能要求完善软件功能,对软件产品进行修改,进行“完善性维护”。当支撑软件产品运行的硬件或软件环境改变时,也需要对软件产品进行修改,进行“适应性维护”。软件的缺陷或错误属于逻辑性的,因此不需要更换某种备件,而是修改程序,纠正逻辑错误,提高性能,增强适应性。当软件产品规模庞大、内部的逻辑关系复杂时,经常会发生纠正一个错误而产生新的错误的情况,因此,软件产品的维护要比硬件产品的维护工作量大而且复杂。

1.1.2 软件危机

20 世纪六七十年代,由于软件规模的扩大、功能的增加和复杂性的增强,在一定时间内仅依靠少数人开发一个软件变得越来越困难。在软件开发中经常会出现时间延迟、预算超支、质量得不到保证、移植性差等问题,甚至有的项目在耗费了大量人力、财力后,由于离目标相差甚远而宣布失败。这种情况使人们认识到“软件危机”的存在。

1. 软件危机的突出表现

(1) 软件生产率低。软件生产率提高的速度远远跟不上计算机应用迅速普及和深入的趋势。落后的生产方式与开发人员的匮乏,使得软件产品的供需差距不断扩大。由于缺乏系统有效的方法,软件开发的知识、经验和相关数据难以积累与复用。另外,低水平的重复开发过程浪费了大量的人力、物力、财力和时间。人们为不能充分发挥计算机硬件性能而苦恼。

(2) 软件产品常常与用户要求不一致。开发人员与用户之间的信息交流往往存在障碍,除了知识背景的差异,缺少合适的交流方法及需求描述工具也是一个重要原因。这使得获取的需求经常存在二义性、遗漏,甚至是错误。由于开发人员对用户需求的理解与用户的本意有所差异,因此开发中后期需求与现实之间的矛盾常集中暴露。

(3) 软件规模的增长,带来了复杂度的提高。由于缺乏有效的软件开发方法和工具的支持,过分依靠程序设计人员在软件开发过程中的技巧和创造性,所以,软件的可靠性往往随着软件规模的增长而下降,质量保障越来越困难。

(4) 不可维护性突出。软件的局限性和欠灵活性,不仅使错误非常难改正,而且不能适应新的硬件环境,也很难根据需要增加一些新的功能。整个软件维护过程除了程序之外,没有适当的文档资料可供参考。

(5) 软件文档不完整、不一致。软件文档是计算机软件的重要组成部分,在开发过程中,管理人员需要使用这些文档资料来管理软件项目;技术人员则需要利用文档资料进行信息交流;用户也需要通过文档来认识软件,对软件进行验收。但是,由于软件项目管理工作不规范,软件文档往往不完整、不一致,这给软件的开发、交流、管理、维护等都带来了困难。

2. 产生软件危机的原因

软件危机是指计算机软件的开发和维护过程中所遇到的一系列严重问题。这些问题不仅局限于那些“不能正确完成功能”的软件,还包括如何开发软件,如何维护大量已有软件,如何使软件开发速度与对软件需求增长相适应等问题。产生软件危机的原因主要有以下几点。

(1) 软件独有的特点给开发和维护带来困难。由于软件的抽象性、复杂性与不可预见性,软件在运行之前,开发过程的进展情况较难衡量,软件的错误发现较晚,软件的质量也较难评价,因此,管理和控制软件开发过程相当困难。此外,软件错误具有隐蔽性,往往在运行很长时间之后仍需要改错。这在客观上使得软件维护较为困难。

(2) 软件人员的错误认识。相当多的软件专业人员对软件开发和维护的认识还有不少错误。例如,软件开发就是编写程序,忽视软件需求分析的重要性,轻视文档的作用,轻视软件维护等。这些错误认识加重了软件危机的影响。

(3) 软件开发工具自动化程度低。尽管软件开发工具已经有了很大的进步,但直到今天,软件开发仍然离不开工程人员的个人创造与手工操作,软件生产仍不可能像硬件设备的生产那样,达到高度自动化。这样不仅浪费了大量的财力、物力和宝贵的人力资源,难以避免低水平的重复性劳动,而且软件的质量也难以保证。此外,软件生产工程化管理程度低,致使软件项目管理混乱,难以保障软件开发进度按计划进行。



1.2 软件工程

为了克服“软件危机”,1968年Fritz Bauer首先提出“软件工程”的概念,试图用工程的方法和管理手段,将软件开发纳入工程化的轨道,以便开发出成本低、功能强、可靠性高的软件产品。几十年来,人们一直在努力探索克服软件危机的途径。

1.2.1 软件工程的形成与发展

自1968年提出软件工程这一概念以来,人们一直在寻求更先进的软件开发的方法与技术。当出现一种先进的方法与技术时,软件危机就会得到一定程度的缓解。然而,这种进步又促使人们把更多、更复杂的问题交给计算机去解决,于是又需要探索更先进的方法与技术。几十年来,软件工程研究的范围和内容也随着软件技术的发展不断变化和拓展。软件工程的发展经历了以下3个阶段。

第一阶段:20世纪70年代,为了解决软件项目失败率高、错误率高以及软件维护任务重等问题,人们提出软件生产工程化的思想,希望使软件生产走上正规化的道路,努力克服软件危机。人们发现将传统工程学的原理、技术和方法应用于软件开发,可以使软件生产规范化。它有利于组织软件生产,提高开发质量,降低成本和控制进度。随后,人们又提出软件生命周期的概念,将软件开发过程划分为不同阶段(需求分析、概要与详细设计、编程、测试、维护等),以适应更加复杂的应用。人们还将计算机科学和数学用于构造模型与算法上,围绕软件项目开展了有关开发模型、方法以及支持工具的研究,研发了多种开发模型、方法与多种软件开发工具(编辑、编译、跟踪、排错、源程序分析、反汇编、反编译等),并围绕项目管理提出了费用估算、文档评审等管理方法和工具,基本形成了软件工程的观念、框架、方法和手段,成为软件工程的第一代——传统软件工程时代。

第二阶段:20世纪80年代,面向对象的方法与技术受到了广泛的重视,Smalltalk-80的出现标志着面向对象的程序设计进入实用和成熟阶段。20世纪80年代末逐步发展起来的面向对象的分析与设计方法,形成了完整的面向对象技术体系,使计算机系统的生命周期更长,适应更大规模、更广泛的应用。这时,进一步提高软件生产率、保证软件质量就成为软件工程追求的更高目标。软件生产开始进入以过程为中心的第二阶段。这个时期人们认识到,应根据软件生命周期的总费用及总价值来确定软件开发方案。在重视发展软件开发技术的同时,人们提出软件能力成熟度模型、个体软件过程、群组软件过程等概念。在软件定量研究方面提出了软件工作量估计COCOMO模型等。软件开发过程从目标管理转向过程管理,形成了软件工程的第二代——过程软件工程时代。

第三阶段:进入20世纪90年代以后,软件开发技术的主要处理对象为网络计算和支持多媒体信息的互联网应用。为了适应超越企业规模、资源共享、群组协同工作的需要,企业需要开发大量的分布式处理系统。这一时期软件工程的目不仅在于提高个人生产率,而且通过支持跨地区、跨部门、跨时空的群组共享信息,协同工作来提高群组、集团的整体生产效率。因为整体性软件系统难以更改、难以适应变化,所以提倡基于构件的开发方法,即部件互联集成。同时,人们认识到计算机软件开发领域的特殊性,不仅要重视软件开发方法和技术的研究,更要重视总结和发展包括软件体系结构、软件设计模

式、互操作性、标准化、协议等领域的复用经验。软件复用和软件构件技术正逐步成为主流软件技术，软件工程也由此进入了新的发展阶段——构件软件工程时代。

1.2.2 软件工程的基本概念

软件工程这一概念已提出几十年，对软件工程的理解是不断深入的。作为一门新兴的交叉性学科，它所研究的对象、适用范围和所包含的内容都在不断发展和变化。

1. 软件工程的定义

最初，软件工程被定义为：“为了经济地获得可靠的、能在实际机器上高效运行的软件，而建立和使用的健全的工程原则。”这个定义虽然没有提到软件质量的技术层面，也没有直接谈到用户满意程度或要求按时交付产品等问题，但人们已经认识到借鉴和吸收对各种工程项目开发的经验，对软件的开发无疑是有益的。

软件工程是指导计算机软件开发和维护的工程学科。它强调按照软件产品的生产特性，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最好的技术结合起来，以便经济地开发出高质量的软件并有效地维护它。

由于引入了软件工程的思想，把其他工程技术和开发领域中行之有效的知识和方法运用到软件开发工作中来，提出了按工程化的原则和方法组织软件开发工作的解决思路 and 具体方法，在一定程度上缓解了“软件危机”。

2. 软件工程的目标

软件工程的目标是基于软件项目目标的成功实现而提出的，主要体现在以下几方面：

- ①软件开发成本较低。
- ②软件功能能够满足用户的需求。
- ③软件性能较高。
- ④软件可靠性高。
- ⑤软件易于使用、维护和移植。
- ⑥能按时完成开发任务，并及时交付使用。

在实际开发中，企图让以上几个质量目标同时达到理想的程度往往是不现实的。软件工程目标之间存在的相互关系如图 1-1 所示。从图中可以看出：有些目标之间是相互补充的，如易于维护和高可靠性之间、低开发成本与按时交付之间；有些目标是彼此相互冲突的，如果只考虑降低开发成本，很可能同时也降低了软件的可靠性，如果一味追求提高软件的性能，可能造成开发出的软件对硬件的依赖性较强，从而影响到软件的可移植性；不同的应用对软件质量的要求不同，如对实时系统来说，其可靠性和效率比较重要；对生命周期较长的软件来说，其可移植性、可维护性比较重要。

软件工程的首要问题是软件质量。软件工程的目的是在以上目标的冲突之间取得一定程度的平衡。因此，在涉及平衡软件工程目标这个问题的时候，软件的质量应该放在最重要的位置加以考虑。软件质量可用功能性、可靠性、可用性、效率、可维护性和可移植性等特性来评价。功能性是指软件所实现的功能能够达到它的设计规范和满足用户需求的程度；可靠性是指在规定的条件和时间内，软件能够正常维持其工作的能力；可用性是指软件帮助使用者完成预期目标的能力；效率是指在规定的条件下用软件实现某种功能所需要的计算机资源的有效性；可维护性是指当环境改变或软件运行发生故

障时,为了使其恢复正常运行所做努力的程度;可移植性是指软件从某一环境转移到另一环境时所做努力的程度。在不同类型的应用系统对软件的质量要求是不同的。

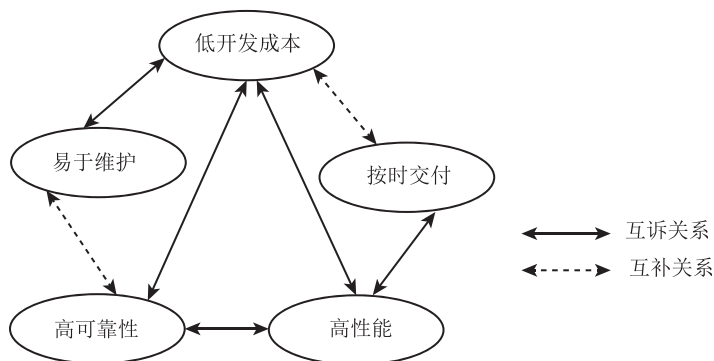


图 1-1 软件工程目标之间的关系

3. 软件工程知识体系及知识域

软件工程作为一门学科,对其核心的知识体系已经达成了一定的共识。2005 年 9 月,ISO/IEC JTC1/SC7 正式发布为国际标准,即 ISO/IEC TR19759—2005 软件工程知识体系指南(SWEBOK)。SWEBOK 将软件工程知识体系划分为 10 个知识域,分为两类过程:一类是开发与维护过程,包括软件需求、软件设计、软件构造、软件测试和软件维护;另一类是支持过程,包括软件配置管理、软件工程管理、软件工程过程、软件工程工具与方法、软件质量。每个知识域还可进一步分解为若干个论题,在论题描述中引用有关知识的参考文献,形成一个多级层次结构,以此确定软件工程知识体系的内容和边界。每个知识域的具体内容如表 1-1 所示。有关知识涉及的相关学科包括计算机工程、计算机科学、管理学、数学、项目管理、质量管理、软件人类工程学、系统工程等。

表 1-1 软件工程知识体系指南的内容

知识域	子知识域
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、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、软件设计法、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、测试级别、测试技术、与测试相关的度量、测试过程
软件维护	软件维护基础、软件维护关键问题、维护过程、维护技术
软件配置管理	软件配置过程管理、软件配置标识、软件配置控制、软件配置状态报告、软件配置审计、软件发行管理和交付
软件工程管理	项目启动和范围定义、软件项目计划、软件项目实施、评审与评价、项目收尾、软件工程度量
软件工程过程	过程定义、过程实践与变更、过程评估、过程和产品度量
软件工程工具与方法	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

软件工程知识体系中涉及的主要技术要素包括软件开发方法、软件开发工具和软件过程。

(1) 软件开发方法

软件开发方法是在工作步骤、软件描述的文件格式、软件的评价标准等方面做出规定。它主要解决什么时候做什么,以及怎样做的问题,是软件工程最核心的研究内容。实践表明,在开发的早期阶段多做努力,就会使后来的测试和维护阶段缩短,从而使费用大大减少。因此,针对分析和设计阶段的软件开发方法特别受到重视。目前,人们提出了结构化方法、面向数据结构方法、原型化方法、面向对象的方法、形式化方法等多种实用有效的软件开发方法,利用这些方法确实开发出不少成功的系统,但各种开发方法具有一定的适用范围,所以选择正确的开发方法是非常重要的。

针对软件开发方法的评价一般是通过以下 4 个方面来进行。

①技术特征:支持各种技术概念的方法特征,如层次性、抽象性(包括数据抽象和过程抽象)、并行性、安全性、正确性等。

②使用特征:具体开发时的有关特征,如易理解性、易转移性、易复用性、工具的支持、使用的广度、易修改性、对正确性的支持等。

③管理特征:对软件开发活动管理的能力方面的特征,如易管理性、支持协同工作的程度、中间阶段的确定、工作产物、配置管理、阶段结束准则和代价等。

④经济特征:对软件机构产生的质量和生产力方面的可见效益,如分析活动的局部效益、整个生命周期效益、获得该开发方法的代价、使用它和管理它的代价等。

下面重点介绍一些常用的软件开发方法。

①结构化方法。结构化方法是传统的基于软件生命周期的软件工程方法,自 20 世纪 70 年代产生以来,获得了极有成效的软件项目应用。结构化方法是以软件功能为目标来进行软件构建的,包括结构化分析、结构化设计、结构化实现、结构化维护等内容。这种方法主要通过数据流模型来描述软件的数据加工过程,并可以通过数据流模型,由对软件的分析过渡到对软件的结构设计。

②JSD 方法。JSD 方法主要用在软件设计上,1983 年由法国学者 Jackson 提出。它以软件中的数据结构为基本依据来进行软件结构与程序算法设计,是对结构化软件设计方法的有效补充。在以数据处理为主要内容的软件系统开发中,JSD 方法具有比较突出的设计建模优势。

③面向对象方法。面向对象方法是从现实世界中客观存在的事物出发来构造软件,包括面向对象分析、面向对象设计、面向对象实现、面向对象维护等内容。一个软件开发出来是为了解决某些问题的,这些问题所涉及的业务范围被称作该软件的问题域。面向对象强调以问题域中的事物为中心来思考问题、认识问题,并根据这些事物的本质特征,把它抽象地表示为系统中的对象,作为系统的基本构成单位。确定问题域中的对象成分及其关系,建立软件系统对象模型,是面向对象分析与设计过程中的核心内容。自 20 世纪 80 年代以来,人们提出了许多有关面向对象的方法,其中,由 Booch、Rumbaugh、Jacobson 等人提出的一系列面向对象方法成为主流方法,并被结合为统一建模语言(UML),成为面向对象方法中的公认标准。

(2) 软件开发工具

软件开发工具是指用来辅助软件开发、维护和管理软件的工具。现代软件工程方法得以实施的重要保证是软件开发工具和环境。软件开发工具使软件在提高开发效率、提升工程质量、减少软件开发对人的依赖性等多方面得到改善。软件开发工具与软件开发方法有着密切的关系,软件开发工具是软件方法在计算机上的具体实现。

软件开发环境是方法与工具的结合,以及配套软件的有机组合。该环境旨在通过环境信息库和通信机制实现工具的集成,从而为软件生命周期中某些过程的自动化提供更有力的支持。集成机制主要实现工具的集成,使之能够系统、有效地支持软件开发。

(3) 软件过程

尽管有软件开发工具与工程化方法,但这并不能使软件产品生产完全自动化,它们还需要合适的软件过程才能真正发挥作用。软件过程是指生产满足需求且达到工程目标的软件产品所涉及的一系列相关活动,它覆盖了需求分析、系统设计、实施以及支持维护等各个阶段。这一系列活动就是软件开发过程中开发机构需要制订的工作步骤。

软件过程有各种分类方法。按性质划分,软件过程可概括为基本过程类、支持过程和组织过程类;按特征划分,有管理过程、开发过程与综合过程;按人员的工作内容来分类,有获取过程、供应过程、开发过程、运作过程、维护过程、管理过程与支持过程。软件过程研究的对象涉及从事软件活动的所有人。提高软件的生产率和质量,其关键在于管理和支持能力。所以,软件过程特别重视管理活动和支持活动。



1.3 软件工程的基本原则

为了保证在软件项目中能够有效地贯彻与正确使用软件工程规程,需要有一定的原则来对软件项目加以约束。经过长期的实践,著名软件工程专家 B. W. Boehm 提出了以下 7 条软件工程的基本原则。

1. 采用分阶段的生命周期计划,以实现项目的严格管理

软件项目的开展,需要计划在先,实施在后。统计资料表明,有 50% 以上的失败项目是由于计划不周而造成的。在软件开发与维护的漫长生命周期中,需要完成许多性质各异的工作,这意味着,应该把软件生命周期划分为若干个阶段,并相应地制定出切实可行的计划,然后严格按照计划对软件的开发与维护进行管理。

2. 坚持进行阶段评审,以确保软件产品质量

软件的质量保证工作贯穿软件开发的各个阶段。实践表明,软件的大部分错误是编程之前造成的。根据 B. W. Boehm 的统计,设计错误占软件错误的 63%,编码错误仅占 37%。软件中的错误发现与纠正得越晚,所需要付出的代价也越高。因此,在每个阶段都要进行严格的评审,尽早地发现软件中的错误,通过对软件质量实施过程监控,以确保软件在每个阶段都能够有较高的质量。

3. 实行严格的产品控制,以适应软件规格的变更

在软件开发过程中不应随意改变需求,因为改变一项需求需要付出较高的代价。但在软件开发过程中改变需求又是难免的,只能依靠科学的产品控制技术来满足这种要求。也就是说,当改变需求时,为了保持软件各个配置成分的一致性,必须实现严格的产品控制,其中主要是实行基准配置管理。所谓基准配置,是指经过阶段评审或软件配置成分(各个阶段产生的文档或程序代码)。基准配置管理也称为变动控制,是指一切有关修改软件的建议,特别是涉及对基准配置的修改建议,都必须严格按照规程进行评审,获得批准以后,才能实施修改,绝对不能随意修改软件。严格实行产品控制,能够对软件的规格进行跟踪记录,使软件产品的各项配置成分保持一致性,由此来适应软件的需求变更。

4. 采用现代程序设计技术

采用先进的软件开发和维护技术,不仅能够提高软件开发和维护效率,而且可以提高软件产品的质量,降低开发成本,缩短开发时间,增加软件的使用寿命。例如,构件架构系统的特点是通过创建比“类”更加抽象、更具有通用性的基本构件,可以使软件开发如同可插入的零件一样装配,这样的软件不仅开发容易,维护方便,而且可以根据用户的特定需求方便地进行改装。

5. 软件成果能清楚地审查

软件成果是指软件开发各个阶段产生的一系列文档、代码、资源数据等,是对软件开发给出评价的基本依据。由于软件产品的生产过程可见性差,没有明显的生产过程,工作进展难以准确度量和控制,所以软件产品的开发过程比其他产品的开发过程更难于评价和管理。为了提高软件开发过程的可见性,更好地进行管理,就要根据软件开发项目的总目标及完成期限,规定开发组织的责任和产品标准,从而使得所得到的结果能够清楚地审查。

6. 开发小组人员应该少而精

开发小组人员的素质和数量是影响软件产品质量和开发效率的重要因素。高素质人员的开发效率和质量是低素质的人员的几倍甚至几十倍。此外,随着开发小组人员数目的增加,因为交流讨论而造成的通信开销也会急剧增加。如果小组中有 N 个成员,可能的通信路径就有 $N(N-1)/2$ 条,这势必影响人员之间的相互协作与工作质量,因此,组成少而精的开发小组非常重要。

7. 承认不断改进软件工程实践的必要性

在遵循上述 6 条基本原理的基础上,还应该注意不断改进软件工程实践的必要性。这说明软件工程在实际应用中,不仅要积极主动地采用新的软件技术,而且要注意不断地总结经验。例如,收集进度和资源消耗数据,收集软件出错类型和问题报告数据等,这些数据不仅可以用来评价新的软件技术的效果,而且可以用来指明必须着重开发的软件工具和应该优先研究的技术。



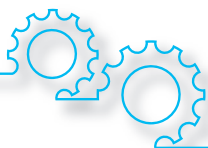
思考与练习

1. 什么是软件危机? 它和软件工程有什么关系?
2. 简述软件和软件工程的定义以及软件工程的形成过程。
3. 软件工程的目标是什么? 如何解决多目标之间的矛盾?
4. 在软件开发中,软件开发工具有什么作用?
5. 什么是软件支持过程? 它与软件工程方法学有何关系?
6. 在软件工程知识体系中,软件工程划分为哪些知识域?
7. B. W. Boehm 提出的软件工程基本原则的作用是什么?

第 2 章

软件过程





2.1 软件开发模型

软件开发模型(software development model)是指软件开发全部过程、活动和任务的结构框架。软件开发由需求分析、系统设计、编码、测试和维护等阶段构成。不同类型的软件项目可以采取不同的软件开发模型。典型的软件开发模型有边做边改模型、瀑布模型、快速原型模型、增量模型、螺旋模型等。

2.1.1 边做边改模型

在这个模型中,开发人员拿到项目立即根据需求编写程序,调试通过后生成软件的第一个版本。在提供给用户使用后,如果程序出现错误,或者用户提出新的要求,开发人员再修改代码,直到用户满意为止。

这种模型适合于小程序(不是项目,规模在几百行代码以内)的开发。开发者具有比较丰富的经验,可以根据用户的需求立即开始编程。但是,因为需求没有经过严格的定义,开发进程非常容易受到用户需求变化的影响,最后开发过程将变得不可控而导致最后的失败。

2.1.2 瀑布模型

瀑布模型是由温斯顿·罗伊斯(Winston Royce)在1970年提出的,如图2-1所示。该模型给出了固定的顺序,将软件生存周期活动从上一个阶段向下一个阶段逐级过渡,如同瀑布一样最终得到所开发的软件产品,投入使用。瀑布模型将软件生命周期划分为制定计划、需求分析、软件设计、程序编写、软件测试和运行维护六个基本活动,并且规定了它们自上而下、相互衔接的固定次序。

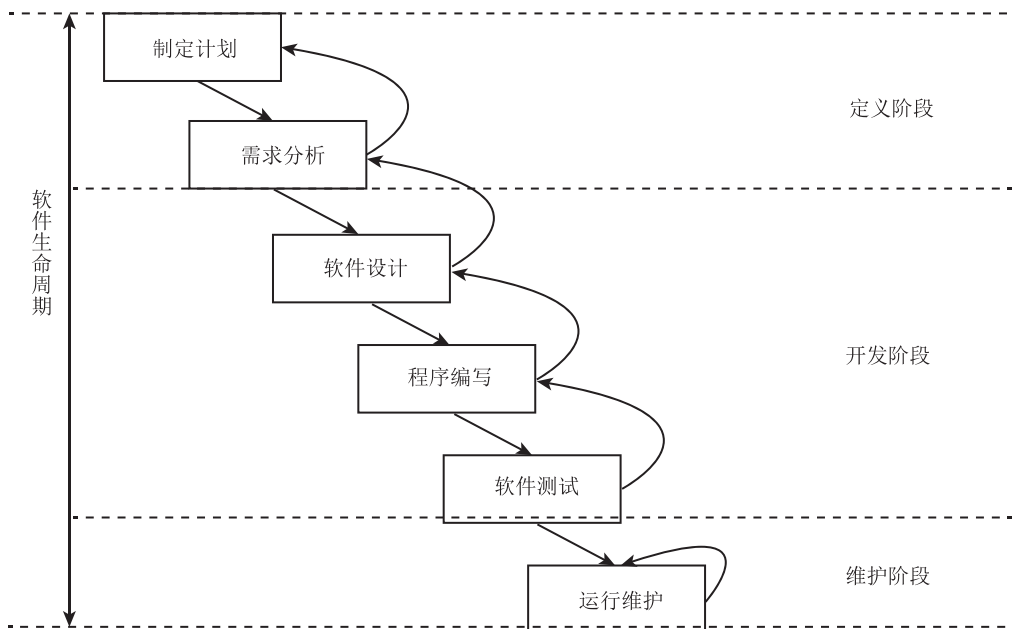


图 2-1 瀑布模型

在瀑布模型中软件开发的各项活动严格按照线性方式进行,当前活动接受上一项活动的工作结果,完成所需的工作内容,每一个阶段都需要设置“里程碑计划”,在每一个阶段的结束都需要进行相应的项目评审,来决定是否可以开展下一阶段工作。用户只能在项目实施的最后阶段才能够真正见到开发成果。

在瀑布模型中,保障项目成功的前提是用户的需求已经进行了明确的定义,同时在每阶段评审中都保证没有偏离用户需求;否则,误差将逐阶段放大,最后导致项目失败。

2.1.3 快速原型模型

软件项目失败的主要原因之一是不能够准确地定义用户的需求,这种不准确一方面是因为项目参与者之间沟通失败,另一方面是因为不论是用户还是开发者都无法利用已有的文档模型对项目需求进行准确的定义。快速原型模型则是从后面这个角度来解决需求不能准确定义的问题,如图 2-2 所示。

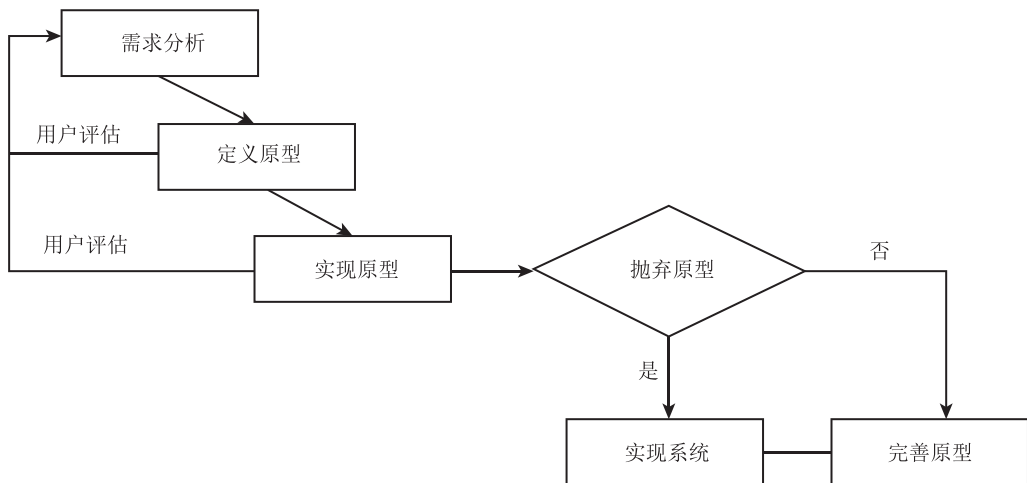


图 2-2 快速原型模型

快速原型模型的第一步是构造一个快速原型,实现用户与系统的交互。用户对原型进行评价,进一步细化待开发软件的需求,开发人员通过逐步调整原型使其满足客户的要求;第二步则是在第一步的基础上开发出客户满意的软件产品。

原型系统在构建时不需要去关注内部处理的过程细节,需要关注的是系统的输入和输出,通过对系统输入、输出的观察,用户能够直观地提出修改建议。原型系统在最后实际开发过程中可以被抛弃,也可以继续完成其内部处理过程。

快速原型模型适合于小型项目的开发,原因在于小型项目可以快速地构建原型并快速进行修改。中大型项目一般采用增量模型或者螺旋模型。

2.1.4 增量模型

无论是瀑布模型还是快速原型模型都试图在软件开发阶段开始之前明确所有的用户需求。在实际软件项目中特别是在中大型软件项目中,随着项目周期的延长,需求发生变化的频度也会变高。原因在于一方面随着用户与项目组成员接触的时间增加,需求表达的清晰度会随之提高,原先含糊的、细节性的一些需求会逐步被发现;另一方面用户本身的业务也可能发生变化,随之而来的就是需求变化,

对于后一种情况,如果不是核心业务变化,项目组需要根据需求变化而进行变化。

在增量模型中,软件被作为一系列的增量构件来进行设计、实现、集成和测试,在各个阶段并不交付一个可运行的完整产品,但第一个构件一定是系统核心功能构件,之后所增加的每一个构件都是在核心功能的基础上完成的,是核心功能的扩展。在增量模型中允许各个构件之间的开发重叠,这样有利于项目组的资源调度和使用。此外,使用增量模型可以将需求定义分散到项目开发过程中,可以避免过早确定用户需求所带来的项目风险。增量模型如图 2-3 所示。

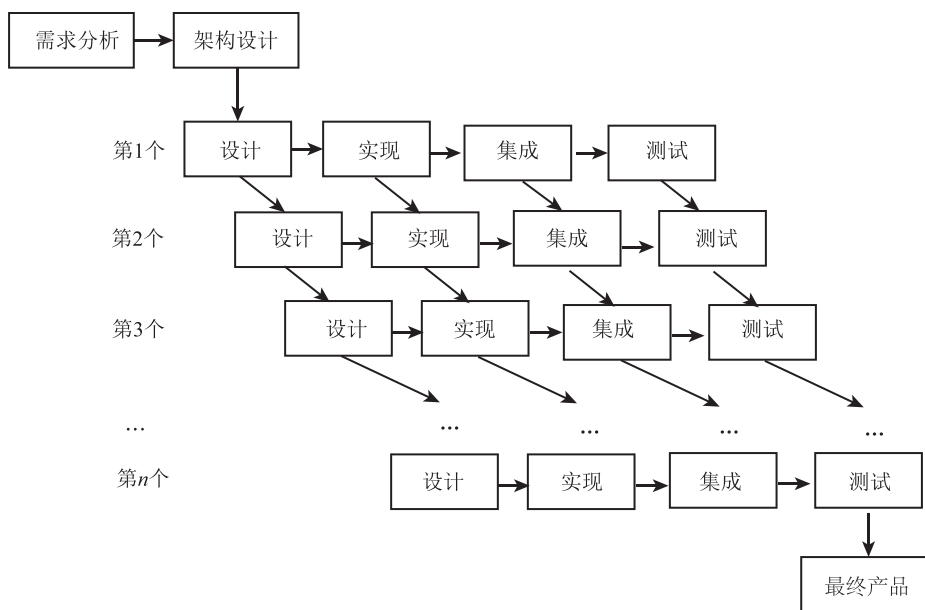


图 2-3 增量模型

因为增量模型是一个逐步开发完善的过程,软件构件是在研发过程中逐步设计加入的,因此要求系统设计必须是一个开放性的架构,要求在系统初期设计时具有高度的扩展性,这对项目组而言是一项比较有难度的工作,增量模型在系统设计不完备的情况下有可能会退化为边做边改模型。

2.1.5 螺旋模型

螺旋模型是瀑布模型和快速原型模型的组合,如图 2-4 所示。螺旋模型定义了如下四项活动。

- (1)制订计划:确定软件目标,选定实施方案,弄清项目开发的限制条件。
- (2)风险分析:分析评估所选方案,考虑如何识别和消除风险。
- (3)实施工程:实施软件开发和验证。
- (4)客户评估:评价开发工作,提出修正建议,制定下一步计划。

螺旋模型通过风险驱动,在制定计划后启动风险评估,从风险角度分析方案的开发策略,排除各种潜在的风险,有时需要通过建造原型来完成。如果某些风险不能排除,该方案立即终止,否则启动下一个开发步骤。

螺旋模型与增量模型一样,对需求的明确贯穿在整个项目的实施过程当中,但存在以下区别。

(1)增量模型的需求是在每一次增量之前进行部分的明确,而螺旋模型在每一次循环当中都是对整体需求的再一次明确。

(2)增量模型的每一次增量所提供的都是部分产品,只有在增量结束时才能够得到完整的产品,而

(3)增量模型在开发过程中允许各个构件开发并行,而螺旋模型只能是串行的。

螺旋模型强调风险分析,要求客户接受和相信这种分析,并做出相关反应是不容易的。同时,如果执行风险分析将大大影响项目的利润,那么进行风险分析毫无意义,因此,螺旋模型只适合于大规模软件项目。

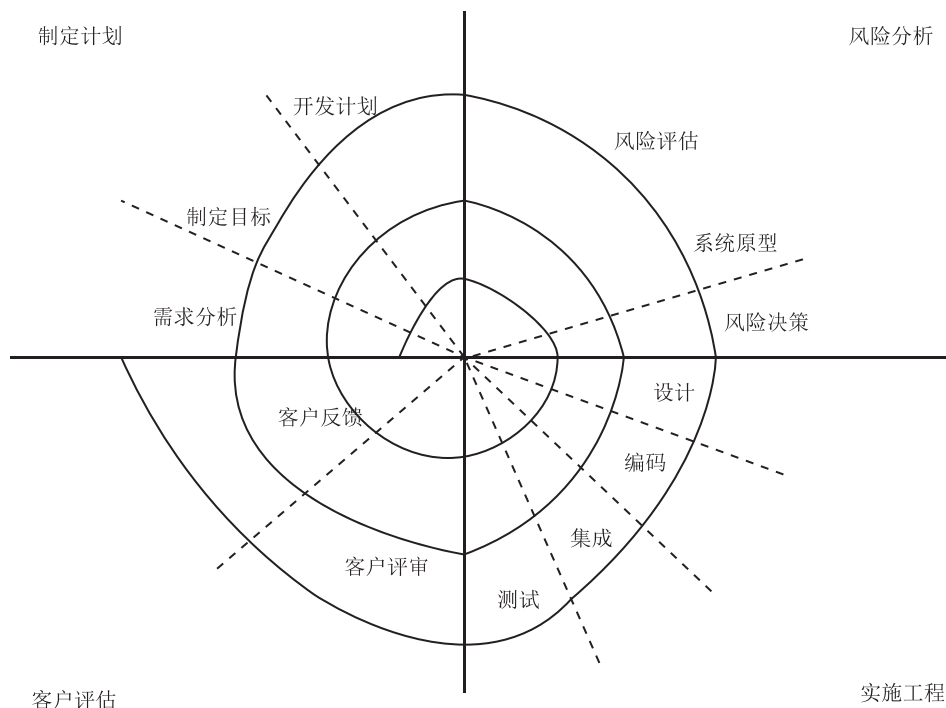
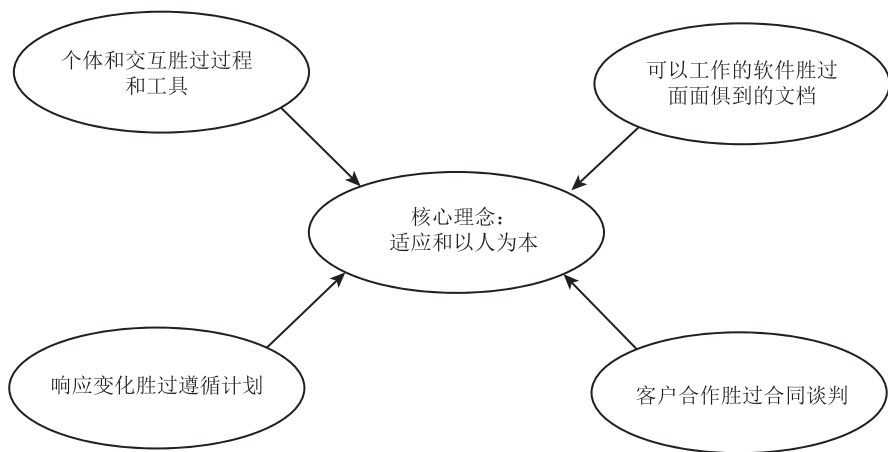


图 2-4 螺旋模型

2.2 敏捷开发

(1)个体和交互胜过过程和工具:人是软件项目获得成功最为重要的因素,合作、沟通以及交互能力要比单纯的软件编程能力更为重要,团队的构建(包括个体、交互等)要比项目环境(包括过程、工具)的构建更为重要。当然,合适的过程和工具对于成功来说非常重要,但不应过分夸大过程和工具的作用。

(2)可以工作的软件胜过面面俱到的文档:与重文档驱动型的软件开发不同,敏捷方法更强调快速交付可使用的软件版本。这一点和螺旋模型类似。通过多次交付的软件版本迭代,使软件能够尽量贴合用户需求。需要注意的是,敏捷方法并不排斥文档,在敏捷开发中直到迫切需要并且意义重大时才进行文档编制,编制的内部文档应尽量短小且主题突出。



(3)客户合作胜过合同谈判:与客户是双赢的关系而不是输赢的关系。客户的参与不仅是在需求定义和最终的项目验收维护阶段,而是项目的全过程参与。客户关系的改善不仅能够加快项目的开发进度,同时也能够项目早期尽快发现错误。

(4)响应变化胜过遵循计划:世界上唯一不变的是变化。在项目过程中需求变更、人员变动都有可能影响计划的执行,与其坚守计划不变不如主动响应变化。在敏捷方法中,长期任务被分解为若干短周期任务,以避免长期计划因变化而导致失败。

敏捷方法强调开发团队与用户之间的紧密协作、面对面的沟通、频繁交付新的软件版本、紧凑而自我组织型的团队等,也更注重人的作用。具体实施过程中遵循以下 12 条准则。

- (1)最优先要做的是通过尽早、持续交付有价值的软件来使客户满意。
- (2)即使到了开发的后期,也欢迎改变需求,敏捷过程利用变化来为客户创造竞争优势。
- (3)经常性交付可以工作的软件,交付的间隔可以从几个星期到几个月,交付的时间间隔越短越好。
- (4)在整个项目开发期间,业务人员和开发人员必须天天都工作在一起。
- (5)围绕被激励起来的个体来构建项目,给他们提供所需的环境和支持,并且信任他们能够完成工作。
- (6)在团队内部,最有效果并且富有效率地传递信息的方法,就是面对面交谈。
- (7)工作的软件是首要的进度度量标准。
- (8)敏捷过程提倡可持续的开发速度,责任人、开发者和用户应该能够保持一个长期的、恒定的开发速度。
- (9)不断地关注优秀设计的技能和好的设计会增强敏捷能力。
- (10)简单——使未完成的工作最大化的艺术——是根本的。
- (11)最好的构架、需求和设计出自组织的团队。
- (12)每隔一段时间,团队会针对如何才能更有效地工作进行反省,然后相应地调整。

2.2.1 极限编程 XP

极限编程(图 2-6)是敏捷方法的代表之一。极限编程是一种方法论,它解决软件开发方法中的约束问题,不解决项目投资管理、财务、执行、市场和销售的问题,可以对任何规模的团队起作用,适合于模糊或者快速变化的需求。

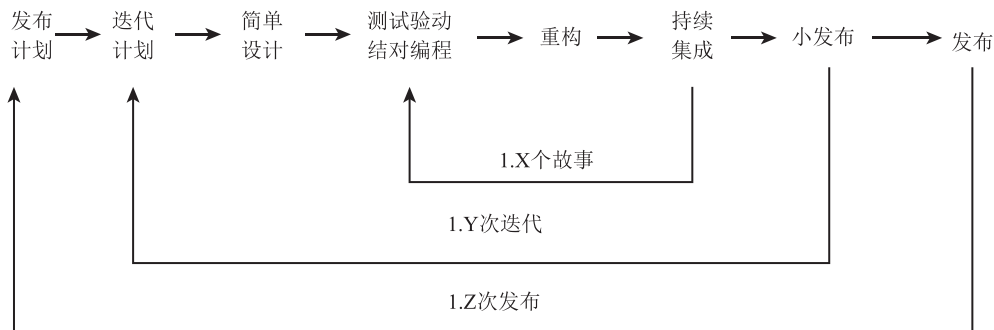


图 2-6 极限编程

作为一种方法论,极限编程的价值观体现在以下几方面。

- 沟通:强调项目成员(包括设计人员、开发人员、客户)之间的有效沟通。
- 简单:尽量保持简单,只要完成当前工作需要即可,而不需要去考虑未来可能的扩展。
- 反馈:不断地对已发布软件状态进行反馈,达到迅速沟通、编码、测试和发布的目的。
- 勇气:勇于放弃和重构代码,敢于所有人拥有代码,敢于所有工作做到极致。
- 尊重:可能存在岗位和职责的划分,但每个人对团队的贡献都应该得到尊重,不存在孰轻孰重的问题。

极限编程的实践准则在第二版中被分为基本实践和扩展实践,扩展实践是要求团队成员在有一定训练的基础上开展,比如共享代码(有的书上叫集体拥有代码),极限编程第二版的主要实践准则如下。

(1)坐在一起

在大到足够容纳整个团队的开放空间中开展工作,距离上的接近可以促进团队成员之间的沟通。

(2)完整团队

将拥有项目成功所必需的各种技能和视角的人都吸收进团队,完整团队的组成是动态的,如果一组技能或意见变得重要时,就将懂得那些技术的人吸收进团队,如果不再需要某个人,可以让他离开团队。这一点和敏捷开发以人为核心的理念并不冲突。

(3)信息工作空间

让工作空间与工作相关,任何一个对项目感兴趣的观察者都能够在走进团队工作区的短时间内就对项目运作有所了解,并能够通过近距离观察而获得更多的信息。

(4)充满活力的工作

只要在有效率的时间段内高效地工作就足够了,一周工作 40 小时。

(5)结对编程

所有产品程序的编写都由坐在同一台机器前面的两个人完成,通常一个人负责写编码,而另一个负责保证代码的正确性与可读性。结对编程是一种非正式的同级评审。它要求结对编程的两个开发人员在性格和技能上应该相互匹配。结对编程时需要注意每天结对的时间不超过 6 小时,并需要进行定期的轮换。

(6)故事

使用客户可见的功能单元进行计划。第一版当中被称为“隐喻”，非常晦涩难懂。一个功能单元的描述不再使用“需求”来表达，而是用“故事”来讲述。

(7)周计划

一次计划一周的工作。在每周开始的时候开会，在会议中回顾迄今为止的发展，让客户挑选在这周内要完成的故事，把故事分解为任务并分配下去。

(8)季度循环

一次计划一个季度的工作。每个季度根据更大的目标对团队、项目、进度和安排做一次反省，并计划下一季度的工作。

(9)10 分钟构建

在 10 分钟内自动构建整个系统和运行所有的测试。超过 10 分钟的构建，一般很少愿意使用，因此导致反馈机会的丧失。

(10)持续集成

不超过两个小时就对改变的地方进行一次集成和测试。等待越长时间的集成，花费越多，结果也越不可知。

(11)测试优先编程

在改变任何代码之前先编写一个自动化测试。测试优先编程的另一个精化是持续测试。持续测试缩短了发现错误的时间，从而缩短了改正错误的时间。

(12)共享代码

团队中的任何人都可以在任何时候改善系统的任何部分。

(13)编程规范

通过制定严格的代码规范来进行沟通，尽可能减少不必要的文档。

(14)增量设计

利用软件重构不断淘汰原有架构中腐化的代码。所谓重构，是指在不改变软件现有功能的基础上，通过调整程序代码改善软件的质量、性能，使程序的设计模式和架构更趋合理，提高软件的扩展性和可维护性。

2.2.2 Scrum

Scrum 这个名字来自橄榄球比赛，是敏捷开发中的另一个代表，其出现时间早于极限编程，在实际应用当中有将 Scrum 与极限编程相结合的案例。应用 Scrum 原则指导软件开发的过程是由“需求、分析、设计、演化和交付”等一系列框架性活动构成，每个框架活动中的一个任务称为冲刺(sprint)，一个冲刺的时间为 2~4 周，用于解决当前的问题。

Scrum 过程的全局流程如图 2-7 所示。

1. Scrum 中的三大角色

(1)产品负责人

产品负责人主要负责确定产品的功能和达到要求的标准，指定软件的发布日期和交付的内容，同时有权力接受或拒绝开发团队的工作成果。

(2)Scrum 管理员

Scrum 管理员主要负责整个 Scrum 流程在项目中的顺利实施和进行，以及清除挡在客户和开发工

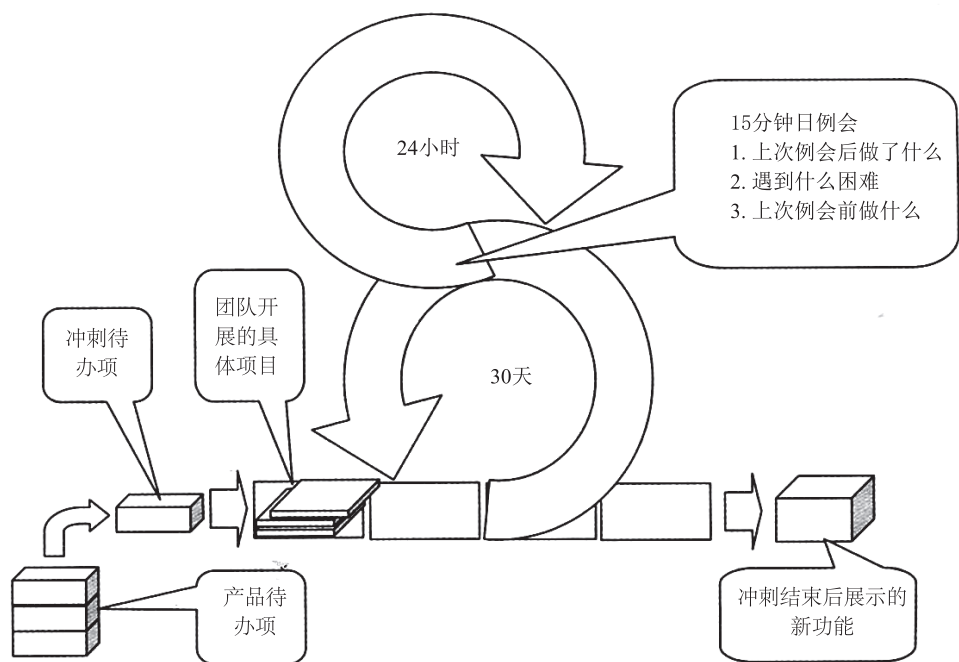


图 2-7 Scrum 过程的全局流程

作之间的沟通障碍,使得客户可以直接驱动开发。

(3) 开发团队

开发团队主要负责软件产品在 Scrum 规定流程下进行开发工作,人数控制在 5~10 人,每个成员可能负责不同的技术方面,但要求每个成员必须有很强的自我管理能力和具有一定的表达能力。成员可以采用任何工作方式,只要能达到冲刺的目标。

2. Scrum 中的三项会议

(1) 冲刺计划会议

冲刺计划会议是产品负责人和团队一起,在先前评估成果的基础上,定出冲刺目标和既定产品待办项。冲刺计划会议时,当进入短期开发阶段,选择一部分近期(1 个月)的高优先级事项作为马上要执行的任务。

(2) 站立例会

在冲刺期间,每天都会通过站立例会来进行沟通,团队成员间进行工作进度的沟通和协调,做好每日规划。

(3) 回顾会议

Scrum 中冲刺计划会议是最重要的事件,第二重要的事件就是回顾会议,因为这是团队做改进的最佳时机。如果没有回顾,就会发现团队总是犯同样的错误。

3. Scrum 的三项工作

(1) 产品待办项(product backlog)

产品待办项是指一个项目的总体目标和计划,其中排在前面的都是优先级最高的事项。

(2) 冲刺待办项

冲刺待办项是指一个冲刺的目标和计划。

(3) 燃尽图

燃尽图用于表示项目剩余的工作时间。

4. Scrum 工作过程

(1) 将整个产品待办项分解成冲刺待办项,这个冲刺待办项是按照目前的人力、物力条件可以完成的。

(2) 召开冲刺计划会议,划分、确定这个冲刺内需要完成的任务,标注任务的优先级并分配给每个成员。注意这里的任务是以小时计算的,并不是按天计算的。

(3) 进入冲刺开发周期,在这个周期内,每天需要召开站立例会。

(4) 整个冲刺周期结束,召开回顾会议,将成果向产品负责人演示,总结问题和经验。

(5) 这样周而复始,按照同样的步骤进行下一次冲刺。

2.2.3 敏捷开发的误区

1. 误区一:敏捷是“一个”过程

敏捷不是一个过程,是一类过程的统称,它们有一个共性,就是符合敏捷价值观,遵循敏捷的原则。

2. 误区二:敏捷仅是个软件过程

如果仅仅从软件过程的角度去认识敏捷、实施敏捷,则效果不会太好。敏捷相比于以前的软件工程最大的革新之处在于把人的作用提高到了过程之上,让开发人员理解并实施,体验到敏捷的好处,而不是盲目机械地实施规范。

3. 误区三:敏捷是反文档的

文档只是达成目标的一种手段,如果这种手段是低效的,那就换一种手段。可是完全抛弃了文档,怎样解决沟通的问题?文档的本质是把知识显性化。在实施敏捷的时候,需要在团队内明确哪些知识是必须显性的,这些知识可以通过文档交流;哪些知识是可以隐性的,这些知识完全可以通过口头的方式进行交流,以达到沟通的最佳效率。

4. 误区四:为了敏捷而敏捷

做什么事情都要有明确的目标,敏捷虽好,得看用户需不需要,能不能解决用户现在头疼的问题,如果不是,那就不要给自己找麻烦了。

5. 误区五:重做就是重构

重做不等于重构,很多场合这两个概念是混淆的。但是在敏捷中,重构的一个特征是必须可控的。当对系统结构进行大的调整时,如果没有测试驱动辅助的话,那么可控性就会很差,这不能叫作重构。

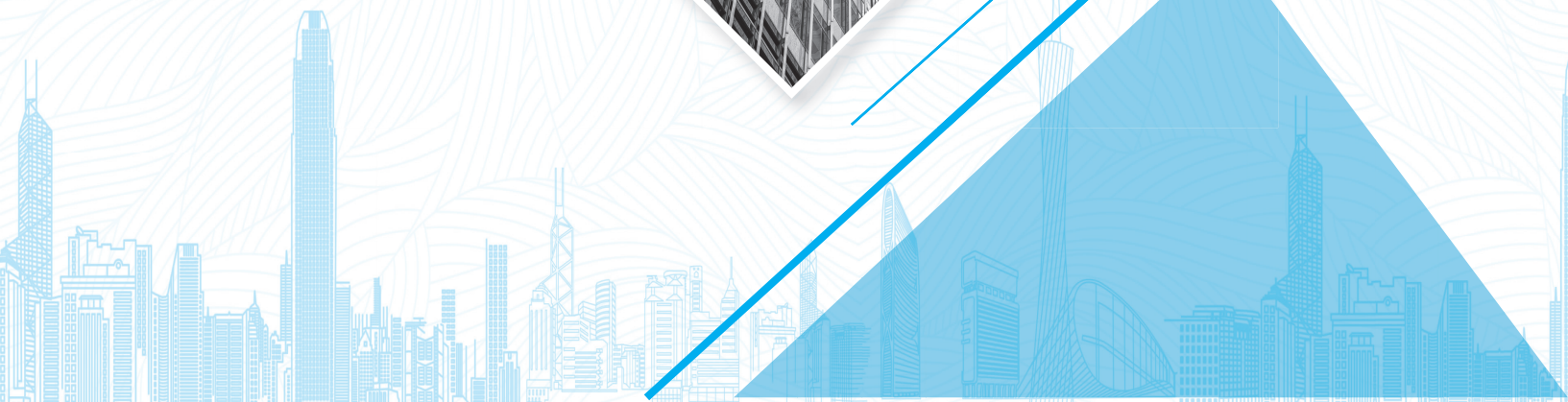


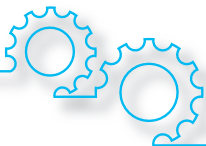
思考与练习

1. 瀑布模型有哪些特点?对于里程碑,你有哪些认识?
2. 一般认为,只有大型项目才采用螺旋模型,其原因是什么?
3. 为什么说构件复用模型是一种有利于软件按工业流程生产的过程模型?
4. 敏捷方法的价值观和原则与传统的方法有哪些联系和区别?
5. 在什么情况下建议不用敏捷方法来开发软件系统?
6. 原型化的瀑布模型具有什么特点?它与瀑布模型最大的不同是什么?

第 3 章

需求分析





3.1 概述

什么是需求？到目前为止比较权威的是 IEEE 软件工程标准词汇表中的定义：用户解决问题或达到目标所需要的条件或权能。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要具有的条件或权能。

IEEE 公布的需求定义分别从用户和开发者的角度阐述了什么是需求，以需求文档的方式一方面反映了系统的外部行为，另一方面也反映了系统的内部特性。比较通俗的需求定义如下：需求是指明系统必须实现什么的规约，它描述了系统的行为、特性或属性，是在开发过程中对系统的约束。

需求工程是指系统分析人员通过细致的调研分析，准确地理解用户的需求，将不规范的需求陈述转化为完整的需求定义，再将需求定义写成需求规约的过程。需求工程包含需求开发和需求管理两部分。

软件需求是软件工程过程中的重要一环，是软件设计的基础，也是用户和软件工程人员之间的桥梁。简单地说，软件需求就是确定系统需要做什么。严格意义上，软件需求是系统或软件必须达到的目标与能力。软件需求在软件项目中占有重要地位，是软件设计和实现的基础。需求的改变将导致其后一系列过程的更改，因而软件需求是影响软件开发成败的关键因素。

3.1.1 需求的类型

软件需求通常有功能需求、非功能需求、领域需求等。下面对此分别进行阐述。

1. 功能需求

简单地说，功能需求描述系统所应提供的功能和服务，包括系统应该提供的服务和对输入如何响应以及特定条件下系统行为的描述。对于用户需求，用较为一般的描述给出。对于功能性的系统需求，需要详细地描述系统功能、输入和输出、异常等，这些需求是从系统的用户需求文档中摘取出来的，往往可以按许多不同的方式来描述。有时，功能需求还包括系统不应该做的事情。功能需求取决于软件的类型、软件的用户及系统的类型等。

理论上，系统的功能需求应该具有全面性和一致性。全面性即应该对用户所需要的所有服务进行描述，而一致性则指需求的描述不能前后自相矛盾。实际上，对于大型的复杂系统来说，要做到全面和一致几乎是不可能的，原因有二：其一是系统本身固有的复杂性；其二是用户和开发人员站在不同的立场上，导致他们对需求的理解有偏颇，甚至出现矛盾。有些需求在描述的时候，其中存在的矛盾并不明显，但在深入分析之后问题就会显露出来。为保证软件项目的成功，不管是在需求评审阶段，还是在随后的软件生命周期阶段，只要发现问题，都必须修正需求文档。

2. 非功能需求

作为功能需求的补充，非功能需求是指那些不直接与系统的具体功能相关的需求，但它们与系统的总体特性相关，如可靠性、响应时间、存储空间等。非功能需求定义了对系统提供的服务或功能的约束，包括时间约束、空间约束、开发过程约束及应遵循的标准等。它源于用户的限制，包括预算的约束、

机构政策、与其他软硬件系统间的互操作,以及如安全规章、隐私权保护的立法等外部因素。

功能需求关心的是系统个别特性,而非功能需求关心的是系统的整体特性,因而对于系统来说,非功能需求更关键。一个功能需求得不到满足会降低系统的能力,但一个非功能需求得不到满足则有可能使系统无法运行。

非功能需求不仅与软件系统本身有关,还与系统的开发过程有关。与开发过程相关的需求包括:对在软件过程中必须使用的质量标准的描述,设计中必须使用的 CASE 工具集的描述,以及软件过程必须遵守的原则等。

按照非功能需求的起源,可将其分为 3 大类:产品需求、机构需求、外部需求。进而还可以将其细分:产品需求对产品的行为进行描述;机构需求描述用户与开发人员所在机构的政策和规定;外部需求范围比较广,包括系统的所有外部因素和开发过程。非功能需求的分类如表 3-1 所示。

表 3-1 非功能需求的类别

非功能需求	产品需求	产品需求	
		效率需求	性能需求
			空间需求
		可靠性需求	
		可移植性需求	
	机构需求	交付需求	
		实现需求	
		标准需求	
	外部需求	互操作需求	
		道德需求	
		立法需求	隐私需求
			安全性需求

非功能需求检验起来非常困难,因为它们可能来自系统的易用性、可恢复性和对用户输入的快速反应的性能要求,同时需求描述的不详细和不确定也会给开发者带来许多困难。虽然理论上来说非功能需求能够量化,通过一些可用来评估非功能性系统特性的度量方法(如表 3-2 所示)可使其验证更加客观,但在实际操作中,对需求描述进行量化是很困难的,这种困难性体现为客户没有能力对目标需求进行量化或者有些目标(如可维护性)本身也没有度量可供使用。因此,在需求文档的目标陈述中,开发者应该明确用户对需求的优先顺序,同时也要让用户知道一些目标的模糊性和无法客观验证性。

表 3-2 非功能需求的度量方法

非功能需求	可使用的度量	度量方法
性能	对用户输入的响应时间	每秒处理的事务数
	用户/事件响应时间	屏幕刷新时间
易用性	学习 75% 的用户功能所需要的时间	培训时间
	在给定时间内,由用户引起的错误的平均值	帮助画面数

表续

非功能需求	可使用的度量	度量方法
可靠性	出错时间 错误发生率	失败平均时间
		无效的概率
		失败发生率
鲁棒性/ 健壮性	系统出错后重新启动的时间	失败之后的重启次数
		事件引起失败的百分比
		失败中数据崩溃的可能性
可移植性	目标系统数	依赖于目标的语句百分比
有效性	请求后出错的可能性	
完整性	系统出错时,允许丢失数据的最大限度	

3. 领域需求

领域需求的来源不是系统的用户,而是系统应用的领域,反映了该领域的特点。它们主要反映了应用领域的基本问题,如果这些需求得不到满足,系统就不可能正常运转。领域需求可能是功能需求,也可能是非功能需求,确定所需的领域知识。它经常采用一种应用领域中的专门语言来描述。

4. 业务需求

业务需求反映组织机构或客户对软件高层次的目标要求,这项需求是用户高层领导机构决定的,它确定了系统的目标规模和范围。

5. 用户需求

用户需求是用户使用该软件要完成的任务。

6. 系统需求

系统需求是容易被忽视的要求,通常是保证整个系统能够正常运行的辅助功能,用户一般不会意识到。

事实上,不同类型的系统需求之间的差别并不像定义中的那么明显。若用户需求是关于机密性的,则表现为非功能需求,但在实际开发时,可能导致其他功能性需求,如系统中关于用户授权的需求。

以上软件需求的分类方法根据软件类型的不同可能稍有差异。图 3-1 所示是软件需求各组成部分之间的一种常见关系。

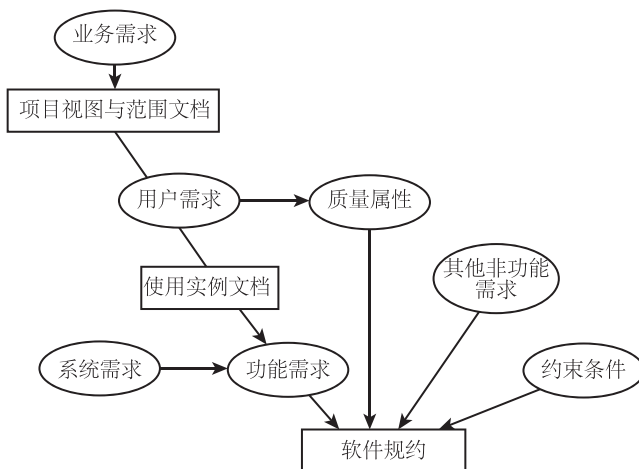


图 3-1 软件需求各组成部分之间的关系

3.1.2 需求开发目标

具体而言,需求开发主要有两个目标:第一,通过对问题及其环境的理解、分析和综合,建立分析模型;第二,完全弄清用户对软件系统的确切要求,在用户和软件开发组织之间就将要开发的软件系统达成一致协议并产生正式的需求文档。

1. 建立分析模型

一般来说,现实世界中的系统不论表面上怎样杂乱无章,总可以通过分析与归纳从中找出一些规律,再通过“抽象”建立起系统的模型。分析模型是描述软件需求的一种模型,由于各个用户往往会从不同的角度阐述他们对原始问题的理解和对目标软件的需求,因此,有必要为原始问题及其目标软件系统建立模型。这种模型一方面用于精确地记录用户对原始问题和目标软件的描述;另一方面,它也可以帮助分析人员发现用户需求中的不一致性,排除不合理的部分,挖掘潜在的用户需求。这种模型往往包含问题及其环境所涉及的信息流、处理功能、用户界面、行为模型及设计约束等,它是形成需求说明以及进行软件设计和实现的基础。

2. 正式需求文档

作为系统需求的最终成果,需求文档必须具有综合性,即必须包括所有的需求。用户和开发组织都应该很谨慎地对待需求文档,因为对于没有包括在需求文档中的需求,用户不要对它可能被最终实现抱任何希望,而一旦需求在需求文档中出现,开发组织必须实现。当然,需求也经常会发生变更,双方应互相探讨以决定取舍。

正式的需求文档应满足如下要求。

(1)具有准确性和一致性。因为它是连接计划周期和开发周期的桥梁,也是软件设计的依据,任何含混不清、前后矛盾或者一个微小的错漏,都可能导致误解或使系统出现错误,在纠正时要付出巨大的代价。

(2)无二义性。因为需求文档是沟通用户和系统分析员思想的媒介,双方要用它来表达对于需要计算机解决问题的共同理解。如果在需求说明中使用了用户不容易理解的专门术语,或用户与分析员对要求的内容可以做出不同的解释,便可能导致系统的失败。

(3)直观、易读和易于修改。应尽量采用标准的图形、表格和简单的符号来表示,使不熟悉计算机的用户也能一目了然。

鉴于需求文档的重要性,其编写也应备受重视。编写需求文档时,以下几点是应该注意的:

- (1)语句和段落尽量简短;
- (2)表达时采用主动语态;
- (3)语句要完整,且语法、标点等正确无误;
- (4)使用的术语要与词汇表中的定义保持一致;
- (5)陈述时要采用一致的样式;
- (6)避免模糊的、主观的术语,如性能“优越”;
- (7)避免使用比较性的词汇,尽量给出定量的说明,含糊的语句表达将导致需求的不可验证。

3.1.3 需求开发过程

需求工程包括需求开发和需求管理两个方面。需求管理是一种系统化方法,可用于获取、组织和记录系统需求并使客户和项目团队在系统变更需求上达成并保持一致。需求开发是一个包括创建和维持系统需求文档所必需的一切活动的过程,通常包含4个通用的高层需求工程活动:系统可行性研究、需求导出和分析、需求描述和文档编写、需求有效性验证。图3-2说明了这些活动之间的关系,也说明了在需求开发过程的各个阶段将产生哪些文档。

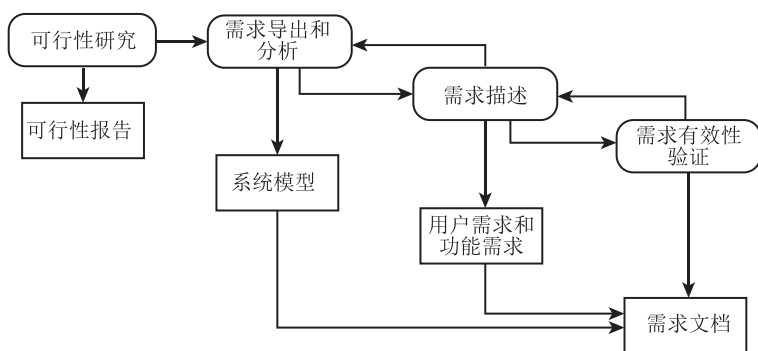


图 3-2 需求开发过程模型

(1)可行性研究。它指明现有的软件、硬件技术能否实现用户对新系统的要求,从业务角度来决定系统开发是否划算,以及在预算范围内是否能够将软件开发出来。可行性研究是比较便宜和省时的。结果就是要得出结论:该系统是否值得进行更细致的分析。

(2)需求导出和分析。这是一个通过对现有系统分析、与潜在用户和购买者讨论、进行任务分析等活动导出系统需求的过程,也可能需要开发一个或多个不同的系统模型和原型。这些都能帮助分析了解所要描述的系统。

(3)需求描述。需求描述就是把在分析活动中收集的信息以文档的形式确定下来。在这个文档中有两类需求:用户需求是从客户和最终用户角度对系统需求的抽象描述;功能需求是对系统要提供的功能的详尽描述。

(4)需求有效性验证。这个活动检查需求实现的一致性和完备性。在这个过程中,不难发现需求文档中的错误,之后必须加以改正。

当然,需求过程中的各项活动并不是严格按顺序进行的。在定义和描述期间,需求分析继续进行,不排除在整个需求工程过程中不断有新的需求出现。因此,分析、定义和描述是交替进行的。

在初始的可行性研究之后,下一个需求工程过程就是需求导出和分析。在这个活动中,软件开发技术人员要和客户及系统最终用户一起调查应用领域,即系统应该提供什么服务,系统应该具有什么样的性能以及硬件约束等。

需求获取是在问题及其最终解决方案之间架设桥梁的第一步。获取需求的一个必不可少的结果是对项目中描述的客户需求的普遍理解。一旦理解了需求,分析者、开发者和客户就能探讨、确定描述这些需求的多种解决方案。参与需求获取的人员只有在他们理解了问题之后才能开始设计系统,否则,对需求定义的任何改进,在设计上都必须大量返工。把需求获取集中在用户任务上而非用户接口上,有助于防止开发组由于草率处理设计问题而造成失误。

所有对系统需求有直接或间接影响力的人统称为项目相关人员。项目相关人员包括使用系统的最终用户和机构中其他与系统有关的人员,正在开发或维护其他相关系统的工程人员、业务经理、领域专家等。以下原因增加了系统需求导出和分析的难度。

项目相关人员通常并不真正知道他们希望计算机系统做什么。让他们清晰地表达出需要系统做什么是一件困难的事情,他们或许会提出不切实际的需求。项目相关人员用他们自己的语言表达需求,这些语言会包含很多他们所从事工作中的专业术语和专业知识。需求工程师没有客户的领域中的知识和经验,而他们又必须了解这些需求。不同的项目相关人员有不同的需求,他们可能以不同的方式

由于软件开发项目和组织文化的不同,对于需求开发没有一个简单的、公式化的途径。需求开发活动通常包括如下 14 个步骤:

- (1) 定义项目的视图和范围；
- (2) 确定用户类；
- (3) 在每个用户类中确定适当的代表；
- (4) 确定需求决策者和他们的决策过程；
- (5) 选择需求获取技术；
- (6) 运用需求获取技术对作为系统一部分的使用实例进行开发并设置优先级；
- (7) 从用户那里收集质量属性的信息和其他非功能需求；
- (8) 详细拟定使用实例并使其融合到必要的功能需求中；
- (9) 评审使用实例的描述和功能需求；
- (10) 开发分析模型用于澄清需求获取的参与者对需求的理解；
- (11) 开发并评估用户界面原型以帮助想象还未理解的需求；
- (12) 从使用实例中开发出概念测试用例；
- (13) 用测试用例来论证使用实例、功能需求、分析模型和原型；
- (14) 在继续进行设计和构造系统每一部分之前,重复(6)~(13)步。

```

graph LR
    Entry[过程入口] --> Domain[领域了解]
    Domain -- (1) --> Collect[需求收集]
    Collect -- (3) --> Classify[分类]
    Classify -- (6) --> Conflict[冲突解决]
    Conflict -- (8) --> Check[需求检查]
    Check -- (9) --> Priority[优先排序]
    Priority -- (10) --> Check
    Check -- (11) --> Domain
    Check -- (12) --> Collect
    Check -- (13) --> Priority
    Priority -- (7) --> Conflict
    Conflict --> Doc[需求文档]
    Priority -- (14) --> Doc
    Doc -- (15) --> Doc

```

过程活动包括以下内容。

(2)需求收集。这是一个与项目相关人员沟通以发现他们需求的过程。很显然,在这个活动期间能对领域有进一步的了解。

(4)冲突解决。在有多个项目相关人员参与的地方,需求将不可避免地发生冲突,这个活动就是发

现而且解决这些冲突。

(5) 优先排序。在任何一组需求中,一些需求总是会比其他的需求更重要。这个阶段包括和项目相关人员交互以发现最重要的需求。

(6) 需求检查。检查需求是否完全、是否一致以及是否与项目相关人员对系统的期待相符合。

从图 3-3 可以看出,需求导出和分析是一个重复过程,从一个活动到另一个活动会有持续不断的反馈。过程循环从领域了解开始,以需求检查结束。分析人员在每个回合中都能进一步加深对需求的理解。



3.2 需求获取

需求获取(requirements elicitation),也称为需求收集(requirements capture),它是与发现目标系统应该提供的需求相关的活动的统称。

需求开发小组的一个或多个成员与顾客组织之间的交流通常是需求获取的重要前提。为了获取客户的需求,需求小组成员必须熟悉应用领域,并使用正确的术语同客户交谈。消除术语误解问题的一个办法是建立并随时更新术语表。在需求获取时,经常使用访谈、场景(scenario,也称为情景,在面向对象分析中也叫用例)以及其他一些技术(例如,向客户组织成员发放调查表,检查客户工作使用的各种表格)。

Christel 和 Kang 指出,以下问题导致需求的获取非常困难。

(1) 范围问题。开发时经常还未定义好系统边界,或者客户/用户刻画的技术细节不清晰、不必要,系统目标的描述不简明、不全面。

(2) 理解问题。客户/用户不能完全确定需要什么,不能完全理解问题领域,与系统工程师在需求沟通方面存在歧义,甚至提出一些同其他客户/用户的需要相冲突的需求以及一些不可测试的需求。

(3) 易变问题。需求随事件发生变化。

为了克服这些问题,系统工程师必须有组织地收集需求。Sommerville 和 Sawyer 建议采用如下步骤来指导需求的获取。

(1) 针对提议的系统评估业务及技术可行性,给出需要和可行性的陈述。

(2) 确定那些能够帮助刻画需求和熟悉组织及相关业务的人员,给出参与需求获取活动的客户、用户和其他风险承担者的列表。

(3) 定义系统或产品的技术环境(如计算体系结构、操作系统),给出关于系统的技术环境的描述。

(4) 确定“领域约束”(即目标应用领域的业务环境的特征),这些约束将限制待建造系统/产品的功能或性能。需要给出关于系统/产品范围的限制性陈述,以及需求列表和应用于每个需求的领域限制。

(5) 定义一种或多种需求获取方法。

(6) 要求大量人员参与,以便能从不同视角来定义需求,并确定每个正式需求的理由。

(7) 确定有歧义的需求为原型实现的候选对象。

(8)给出使用场景,以帮助客户/用户更好地确定关键需求,提供不同运行条件下系统或产品的使用指导意见。

(9)给出原型以更好地定义需求。

以上每一项工作的成果都要由参与需求获取活动的所有人员评审确认。

3.2.1 需求获取方法

为了获取正确的需求信息,可以使用一些基本的需求获取方法和技术。

1. 建立联合分析小组

系统开始开发时,系统分析员往往对用户的业务过程和术语不熟悉,用户也不熟悉计算机的处理过程,因此在系统分析员看来,用户提供的需求信息往往是零散、片面的,需要由一个领域专家来沟通。因而,建立一个由用户、系统分析员和领域专家参加的联合分析小组,对开发人员与用户之间的交流和需求的获取将非常有用。有些学者也将这种面向联合开发小组的需求收集方法称为“便利的应用规范技术”(facilitated application specification techniques, FAST)。有人主张,在参加 FAST 小组的人员中,用户方的业务人员应该是系统开发的主体,是“演员”和“主角”;系统分析员作为高层技术人员,应成为开发工作的“导演”;其他与会开发人员是“配角”。切忌在需求获取阶段忽视用户业务人员的作用,由系统开发人员越俎代庖。

2. 客户访谈

要获取全面的用户需求,光靠联合分析小组中的用户代表是不够的,系统分析员还必须深入现场,同用户方的业务人员进行多次交流。根据用户将来使用软件产品的功能、频率、优先等级、熟练程度等方面的差异,将他们分成不同的类别,然后分别对每一类用户通过现场参观、个别座谈或小组会议等形式,了解他们对现有系统的问题和新功能等方面的看法。

客户访谈是一个直接与客户交流的过程,可以了解用户对软件的要求和建议。由于是与用户面对面交流,如果系统分析员没有做好充分准备,容易引起用户的反感,从而产生隔阂,所以分析员必须在这个过程中尽快找到与用户的“共同语言”,进行愉快的交谈。首先,必须对问题的背景和问题所在系统的环境有全面的了解;其次,尽可能了解将要会谈用户的个性特点及任务状况;最后,事先准备一些问题。在与用户交流时,应遵循循序渐进的原则,切不可急于求成,否则欲速则不达。

3. 问卷调查

所谓问卷调查法,是指开发方就用户需求中的一些个性化的、需要进一步明确的需求(或问题),通过采用向用户发问卷调查表的方式,彻底弄清项目需求的一种需求获取方法。这种方法适合于开发方和用户方都清楚项目需求的情况。因为开发方和客户方都清楚项目的需求,所以需要双方进一步沟通的需求(或问题)比较少,通过采用这种简单的问卷调查方法就能使问题得到较好的解决。

4. 问题分析与确认

用户无法在一两次交谈中,对目标软件的需求阐述清楚,也不能控制用户在回答问题过程中的自由发挥。在每次访谈之后,要及时进行整理,分析用户提供的信息,筛选掉错误的、无关的部分,整理有用的内容,以便在下一次与用户见面时确认。同时,准备下一次访谈时的进一步更细节的问题。如此循环,一般需要 2~5 次。

5. 快速原型法

通常,原型是指模拟某种产品的原始模型。在软件开发中,原型是软件早期的一个可运行的版本,它反映最终系统的部分重要特性。如果在获得一组基本需求说明后,通过快速分析构造出一个小型的软件系统,能够满足用户的基本要求,就使得用户可在试用原型系统的过程中获得亲身感受并受到启发,做出反应和评价,然后开发者根据用户的意见对原型加以改进。随着不断试验、纠错、使用、评价和修改,获得新的版本。如此周而复始,逐步减少分析和通信中的错误,弥补不足之处,进一步确定各种需求和细节,适应需求的变更,从而提高最终产品的质量。

作为开发人员和用户的交流手段,快速原型法可以获取两个层次上的需求:第一层包括设计界面,这一层的目的是确定用户界面风格及报表的版式和内容;第二层是第一层的扩展,用于模拟系统的外部特征,包括引用了数据库的交互作用及数据操作、执行系统关键区域的操作等,此时用户可以输入成组的事务数据,执行这些数据处理的模拟过程,包括出错处理。

在需求分析阶段采用快速原型法,一般可按照以下步骤进行:

- (1)利用各种分析技术和方法,生成一个简化的需求规约。
- (2)对需求规约进行必要的检查和修改后,确定原型的软件结构、用户界面和数据结构等。
- (3)在现有的工具和环境的帮助下,快速生成可运行的软件原型并进行测试、改进。
- (4)将原型提交给用户评估并征求用户的修改意见。
- (5)重复上述过程,直到原型得到用户的认可。

使用原型实现方法的优点,如表 3-3 所示。

表 3-3 原型实现方法的优缺点

编号	优点	缺点
1	开发者与用户充分交流,可以澄清模糊需求,需求定义比其他模型好得多	开发者在不熟悉的领域中不易分清主次,原型不切题
2	开发过程与用户培训过程同步	产品原型在一定程度上限制了开发人员的创新
3	为用户需求的改变提供了充分的余地	随着更改次数的增多,次要部分越来越大,“淹没”了主要部分
4	开发风险低,产品柔性好	原型过快收敛于需求集合,以致忽略了一些基本要素
5	开发费用低,时间短	资源规划和管理较为困难,随时更新文档也造成一定的麻烦
6	系统易维护,对用户更友好	只注意原型是否满意,忽略了原型环境与用户环境的差异

由于开发一个原型需要花费一定的人力、物力、财力和时间,而且用于确定需求的原型在完成使命后一般就被丢弃,因此,是否使用快速原型法必须考虑软件系统的特点、可用的开发技术和工具等方面。表 3-4 中的 6 个问题可用来帮助判断是否要选择使用快速原型法。

表 3-4 原型实现方法的选择

问题	废弃型原型法	演化型原型法	其他预备性工作
应用领域已被理解了吗?	是	是	否
问题可以被建模吗?	是	是	否
客户能够确定基本需求吗?	是	否	否
需求已被建立而且稳定吗?	否	是	是

续表

问题	废弃型原型法	演化型原型法	其他预备性工作
有模糊不清的需求吗?	是	否	是
需求中有矛盾吗?	是	否	是

先进的快速开发技术和工具是快速原型法的基础。如果为了演示一个系统功能,需要手工编写数千行甚至数万行代码,那么采用快速原型法的代价太大,就变得没有现实意义了。为了快速开发出系统原型,必须充分利用快速开发技术和复用软件构件技术。

1984 年,Boar 提出一系列选择原型法的因素,包括应用领域、应用复杂性、客户特征以及项目特征。如果是在需求分析阶段,则必须从系统结构、逻辑结构、用户特征、应用约束、项目管理和项目环境等多方面来考虑,以决定是否采用原型化方法。

(1)系统结构。联机事务处理系统、相互关联的应用系统适合于用原型化方法,而批处理、批修改等结构不适合用原型化方法。

(2)逻辑结构。有逻辑结构的系统,如操作支持系统、管理信息系统、记录管理系统等适合于用原型化方法,而基于大量算法的系统则不适合用原型化方法。

(3)用户特征。不满足于预先做系统定义说明、愿意为定义和修改原型投资、不易肯定详细需求、愿意承担决策的责任、准备积极参与的用户是适合使用原型化方法的用户。

(4)应用约束。对已经运行系统的补充,不能使用原型化方法。

(5)项目管理。只有项目负责人愿意使用原型化方法时,才适合使用原型化的方法。

(6)项目环境。需求说明技术应该根据每个项目的实际环境来选择。

当系统规模很大、要求复杂、系统服务不清晰时,在需求分析阶段先开发一个系统原型是很值得的,特别是当性能要求较高时,在系统原型上先做一些试验也是很必要的。

为了有效实现软件原型,必须快速开发原型,以使得客户可以评估其结果并及时变更。可以使用 3 种方法和工具来实现快速原型。

(1)第四代技术(4GT)。第四代技术包含广泛的数据库查询和报表语言、程序和应用生成器,以及其他很高级的非过程语言。4GT 使软件工程师能快速生成可执行代码,因此它们是理想的快速原型实现工具。

(2)可复用软件构件。结合原型实现方法和程序,构件复用只能在一个数据库系统已经被开发以便存在可以被分类和检索的构件的情况下,才可以有效地工作。特殊的是,现有的软件产品可被用做“新的、改进的”来替代产品原型,这在某种意义上也是一种软件原型实现的复用形式。

(3)形式化规约和原型实现环境。过去 20 年中,已经开发了一系列的形式化规约语言和工具来替代自然语言规约技术。现在,正在继续开发交互式环境,使得分析员能够交互地创建基于语言的系统或者软件规约;激活自动工具把基于语言的规约翻译成可执行代码,使得客户使用原型可执行代码满足(达到)去精化形式化需求。

3.2.2 分析人员与用户的合作关系

深入实际是一项能了解社会和机构需求的观察技术,分析人员把自己放在待建系统的工作环境中,观察、记录参与者的实际任务。深入实际的价值是它能帮助开发人员发现隐性的系统需求,这些需

求是实际存在的,但却是非规范化过程获得的。

优秀的软件产品是建立在高质量的需求基础之上的,而高质量的需求来源于客户与分析人员之间有效的交流与合作。通常,分析人员与客户间的关系反而会成为一种对立关系,双方的管理者都只考虑自己的利益而忽视用户提供的需求,从而产生摩擦。在这种情况下对双方都没有益处。

其实大家都想是开发出一个既能实现商业价值,又能满足用户需要,还能使开发者感到满意的优秀的软件产品。只有当双方参与者都明白要成功自己需要做什么,同时也知道要成功用户需要什么时,才能建立起恰当的合作关系。

下面列出 9 条在需求获取过程中,客户与分析人员、开发人员交流时的要求。

(1)要求分析人员使用符合客户语言习惯的表达。需求讨论集中于业务需要和任务,故要使用业务术语,客户可将其教给分析人员,而客户不一定要懂得计算机的行业术语。

(2)要求分析人员了解客户的业务及目标。通过与用户交流来获取用户需求,分析人员才能更好地了解客户的业务,以及提出使产品更好地满足客户需要的方法。开发人员和分析人员可以亲自去观察客户是怎样工作的。如果新开发系统是用来替代旧系统,那么开发人员应使用一下旧系统,这将有利于他们明白旧系统是怎样工作的,系统的工作流程及可改进之处。

(3)要求分析人员编写软件需求规约。分析人员要把从客户那里获得的所有信息进行整理,分析业务需求及规范、功能需求、质量目标、解决方法和其他信息。通过这些分析得到一份软件需求规约。要评审编写出的规约,确保它们准确而完整地表达了客户的需求。一份高质量的软件需求规约有助于开发人员开发出真正满足客户需要的产品。

(4)要求得到需求工作结果的解释说明。分析人员采用了多种图表作为文字性软件需求规约的补充时,客户很可能对此并不熟悉,可以要求分析人员解释说明每张图表的作用或其他的需求工作结果和符号的意义,及怎样检查图表有无错误或不一致等。

(5)要求开发人员尊重客户的意见。如果用户与开发人员之间不能相互理解,那么关于需求的讨论将会出现障碍。共同合作能使大家“兼听则明”。参与需求开发过程的客户有权要求开发人员尊重他们,并珍惜他们为项目成功所付出的时间。同样,客户也应对开发人员为项目成功这一共同目标所做出的努力表示尊重与感激。

(6)要求开发人员对需求及产品实施提供建议。通常,客户所说的“需求”是一种实际可能实施的解决方案,分析人员将尽力从这些解决方案中了解客户真正的业务及其需求,同时还应找出已有系统不适合当前业务之处,以确保产品不会无效或低效。在彻底弄清业务领域内的事情后,分析人员提出改进方法。有经验且富有创造力的分析人员还能提出增加一些用户并未发现的具有价值的系统特性。

(7)描述产品易使用的特性。客户可以要求分析人员在实现功能需求的同时还要注重软件使用的便利性,因为这些易用特性或质量属性能使用户更准确、高效地完成任务。例如,客户有时要求产品要“友好”“健壮”“高效率”,但这对于开发人员来说,太主观且并无实用价值。正确的特性应是:分析人员通过询问和调查了解客户所要的友好、健壮、高效率所包含的具体特性。

(8)调整需求,允许重用已有的软件构件。需求通常要有一定的灵活性。分析人员发现已有的某个软件构件与客户描述的需求相符,在这种情况下,分析人员应提供一些修改需求的选择,以便开发人员能够在新系统开发中重用一些已有的软件。如果有可重用的机会出现,同时客户又能调整自己的需求说明,那就能降低成本和节省时间,这时就不必严格按原有的需求说明开发。

(9)获得满足客户功能和质量要求的系统。每个人都希望项目获得成功,但这不仅要求客户要清

楚地告知开发人员关于系统“做什么”所需的所有信息,而且还要求开发人员能通过交流了解清楚取舍与限制。一定要明确说明客户的假设和潜在的期望,否则开发人员开发出的产品很可能无法让客户满意。

同时,在软件需求获取过程中,客户有下列义务。

(1)给分析人员讲解自己的业务。分析人员要根据客户讲解的业务概念及专业术语进行产品研发,但不应要求分析人员成为该领域的专家。不要期望分析人员能把握业务的细微与潜在之处,他们不一定知道那些对于客户来说是理所当然的“常识”。

(2)抽出时间清楚地说明并完善自己的需求。客户经常在最忙的时候参与需求开发。但无论如何,客户都有义务抽出时间参与“头脑风暴”会议,接受采访或其他获取需求的活动。有时分析人员可能自以为明白了客户的需求,而过后发现还需要客户进一步地讲解,这时,请耐心一些对待需求和需求的精化工作过程中的反复,因为它是人们交流中经常发生的现象,而且这对软件产品的成功研发极为重要。

(3)准确而详细地说明需求。编写一份清晰、准确的需求文档是有一点困难的。由于处理细节问题不但麻烦而且又耗时,故很容易留下模糊不清的需求。但是,在开发过程中,必须得解决这些模糊性和不准确性的问题,而客户恰是解决这些问题的最佳人选。在需求规约中暂时加上待确定(to be determined, TBD,也可采用汉语拼音略写为 DQD)标志是个不错的办法,用该标志可指明那些需要进一步探讨、分析或增加信息的地方。不过,有时也可能因为某个特殊需求难以解决或没有人愿意处理它而标注上 TBD 标志。尽量将每项需求的内容阐述清楚,以便分析人员能准确地将其写在软件需求规约中。

(4)及时做出决定。正如一位建筑师修建房屋,分析人员将会要求客户做出一些选择和决定,这些决定包括来自多个用户提出的处理方法或在质量特性冲突和信息准确度中选择折中方案等。有权做出决定的客户必须积极地对待这一切,尽快做处理、做决定,因为开发人员通常只有等客户做出了决定才能行动,而这种等待会延误项目的进展。

(5)尊重开发人员的需求可行性及成本评估。所有的软件功能都有其成本价格,开发人员最适合预算这些成本。客户预期的某些产品特性可能在技术上行不通,或者实现它要付出超出预算的代价。而某些需求试图在操作环境中达到不可能实现的性能或试图获取一些根本得不到的数据,开发人员会对此做出负面的证明或提出操作实现的可行性的需求。例如,要求某个行为在“瞬间”发生是不可行的,但考虑另一种更具体的时间需求说法(如在 50 ms 以内),这就可以实现了。

(6)划分需求优先级别。大多数项目没有足够的时间或资源来实现功能性的每个细节。决定哪些特性是必要的,哪些是次要的,哪些是好的,哪些是需求开发的主要部分,只能由客户来负责设定需求优先级,因为开发者无法完全按照客户的观点决定需求优先级。开发者可为客户确定优先级提供有关每个需求的花费和风险的信息,在时间和资源限制下,关于所需特性能否完成或完成多少应该尊重开发人员的意见。尽管没有人愿意看到自己所期望的需求在项目中未被实现,但毕竟是要面对实际情况的。业务决策有时不得不依据优先级来缩小项目范围、延长工期、增加资源或在质量上寻找折中方法。

(7)评审需求文档和原型。无论是正式的还是非正式的方式,对需求文档进行评审都会对软件质量的提高有所帮助。让客户参与评审才能真正鉴别需求文档是否完整、正确地说明了期望的必要特性。评审也给客户代表提供一个机会,给需求分析人员带来反馈信息从而改进他们的工作。如果客户认为编写的需求文档不够准确,就有义务尽早告诉分析人员并提供改进建议。客户通过阅读需求规约,很难想象实际的软件是什么样子的。更好的方法是先为产品开发一个原型,这样客户就能提供更

有价值的反馈信息给开发人员,帮助他们更好地理解需求。

(8)需求出现变更要及时联系。需求不断地变更容易给在计划内完成高质量产品带来严重的负面影响。变更是不可避免的,但在开发周期中变更出现得越晚,其影响越大。需求不断地变更不仅会导致代价极高的返工,而且工期也会被迫延误,特别是在大体结构已完成后又需要增加新特性时。因此一旦发现需要变更时,必须立即通知分析人员。

(9)应遵照开发人员组织处理需求变更的过程。为了将变更带来的负面影响降到最低,所有的参与者必须遵照项目的变更控制过程。这要求不放弃所有提出的变更,并对每项变更进行分析、综合考虑,最后做出合适的决策,确定将必要的变更引入项目中。

(10)尊重开发人员采用的需求工程过程。软件开发过程中最具挑战性的莫过于收集需求并确定其正确性。分析人员采用的方法有其合理性,也许客户认为需求过程不太划算,但请相信在需求开发上消耗的时间是“很有价值”的。客户如果能理解并支持分析人员为收集、编写需求文档和确保其质量所采用的技术,那么整个开发过程将会非常顺利。

(11)系统分析人员在开发过程中可能会遇到这样的问题,一些忙碌的客户可能不愿意积极参与需求过程,而缺少客户参与很可能导致开发出不理想的产品,故一定要确保需求开发中的主要参与者都了解并履行他们的义务。如果遇到分歧,双方可以通过协商以达成对各自义务的相互理解,这样能减少工作中的摩擦。

3.2.3 需求获取的重要性

需求获取可能是软件开发中最困难、最关键、最易出错且最需要交流的过程。需求获取只有通过客户与开发人员的有效合作才能成功。分析者必须建立一个产品对问题进行彻底探讨的环境。为了方便清晰地进行交流,需要列出重要的小组。对需求问题的全面考察需要一种技术,使用这种技术不但考虑了问题的功能需求方面,还可讨论了项目的非功能需求。

需求获取是一个需要高度合作的活动,并不是客户需求的简单拷贝。分析人员必须通过客户所提出的表面需求获取他们的真正需求。询问一个可扩充的问题将有助于理解客户目前的业务过程,并且知道新系统如何帮助或改进他们的工作。

需求获取利用了所有可用的信息来源,这些信息描述了问题域或在软件解决方案中合理的特性。研究表明:比起不成功的项目,一个成功的项目是开发人员和客户之间采用了更多的交流方式。与单个客户或潜在的用户组一起座谈,对于业务软件包或信息管理系统的应用来说是一种传统的需求来源。

在每一次座谈之后,记下所讨论的条目,并请参与讨论的用户评论并更正。及早并经常进行座谈是需求获取成功的一个关键途径,因为只有提供需求的人才能确定开发人员是否真正获取需求。进行深入收集和分析以最大程度地消除冲突或不一致性,尽量理解客户用于表述他们需求的思维过程。充分研究客户执行任务时做出决策的过程,并提取出潜在的逻辑关系。流程图和决策树是描述这些逻辑决策的好方法。

当进行需求获取时,应避免受不成熟细节的影响。在对契合的客户任务取得共识之前,客户能很容易地在一个报表或对话框中列出每一项的精确设计。如果这些细节都作为需求记录下来,它们会给随后的设计过程带来不必要的限制。应确保客户参与者将注意力集中在与所讨论的话题适合的抽象层上。