# 第一篇 理论部分

# 第1章 软件工程与软件测试

本章介绍了软件的发展历史、当前流行的软件过程模型等理论知识，为读者学习本书后续内容作必要准备。

## 1.1 软　　件

软件是一系列按照特定顺序组织的计算机数据和指令的集合。一般来讲，软件分为系统软件、应用软件等。其中系统软件为计算机使用提供最基本的功能，但是并不针对某一特定的应用领域；而应用软件则恰好相反，不同的应用软件根据用户和所服务的领域提供不同的功能。

一般认为，软件包括如下内容。

① 运行时，能够提供所要求功能和性能的指令或计算机程序的集合。

② 程序能够满意地处理信息的数据结构。

③ 描述程序功能需求以及程序如何操作和使用所要求的文档。

### 1.1.1 软件发展史

软件的发展经历了如下几个阶段。

20 世纪 50 年代初期至 60 年代中期是软件发展的第一阶段，称为程序设计阶段。此时硬件已经通用化，而软件的生产却逐渐个体化。软件产品为专用软件，规模较小，功能单一，开发者即使用者，软件只有程序，无文档。软件设计在人们的头脑中完成，形成了“软件等于程序”的错误观念。

第二阶段是从 20 世纪 60 年代中期开始到 70 年代末期结束，称为程序系统阶段。此时随着多道程序设计技术、多用户系统、人机交互式技术、实时系统和第一代数据库管理系统的出现，出现了专门从事软件开发的“软件作坊”。但软件技术和管理水平相对落后，导致 “软件危机”出现。软件危机主要表现在以下几个方面。

① 软件项目无法按期完成，超出经费预算，软件质量难以控制。

② 开发人员和开发过程之间约定不严密，相应的管理不规范，文档书写不完整，使得软件维护费用高。

③ 缺乏严密有效的质量检测手段，交付给用户的软件质量差，在运行中出现许多问题，甚至带来严重的后果。

④ 系统更新换代难度大。

第三阶段称为软件工程阶段，从 20 世纪 70 年代中期开始到 80 年代中期结束，由于微处理器的出现，分布式系统得以广泛应用，使得计算机真正成为大众化的东西。以软件的产品化、系列化、工程化和标准化为特征的软件产业发展起来，软件开发有了可以遵循的软件工程化的设计准则、方法和标准。1968 年，北大西洋公约组织的计算机科学家在德国召开国际会议，讨论软件危机问题，正式提出并使用“软件工程”概念，标志软件工程学的诞生。

软件工程学涉及与生产软件相关的所有活动，包括计算机科学、管理学、经济学、心理学等，其研究的主要内容是如何应用科学的理论和工程上的技术来指导软件的开发，从而达到以较少的投资获得高质量软件的最终目标。

第四阶段是从 20 世纪 80 年代中期至今，客户端/服务器（C/S）体系结构，特别是 Web 技术和网络分布式对象技术飞速发展，导致软件体系结构向更加灵活的多层分布式结构演变，CORBA、EJB、COM/DCOM 三大分布式的对象模型技术相继出现。

2006 年，出现了面向服务的架构（Service-Oriented Architecture，SOA）。作为下一代软件架构，SOA 是“抽象、松散耦合和粗粒度”的软件架构，能够根据需求通过网络对松散耦合的粗粒度应用组件进行分布式部署、组合和使用，主要用于解决传统对象模型中无法解决的异构和耦合问题。

至此，软件发展经历了从 Mainframe 结构、Client/Server 结构、B/S 多层分布式结构到 SOA 的演变过程，整个软件系统变得越来越分散、越来越开放、越来越强调互操作性。

## 1.1.2　软件生命周期

同任何事物一样，一个软件产品或软件系统也要经历孕育、诞生、成长、成熟、衰亡等阶段，这个过程一般被称为软件生命周期。通过将整个软件生存周期划分为若干阶段，使得每个阶段有明确的任务，使规模大、结构复杂和管理复杂的软件开发变得容易控制和管理。通常，软件生存周期包括可行性分析与开发项计划、需求分析、设计（概要设计和详细设计）、编码、测试、维护升级到废弃等阶段，如图 1.1 所示。

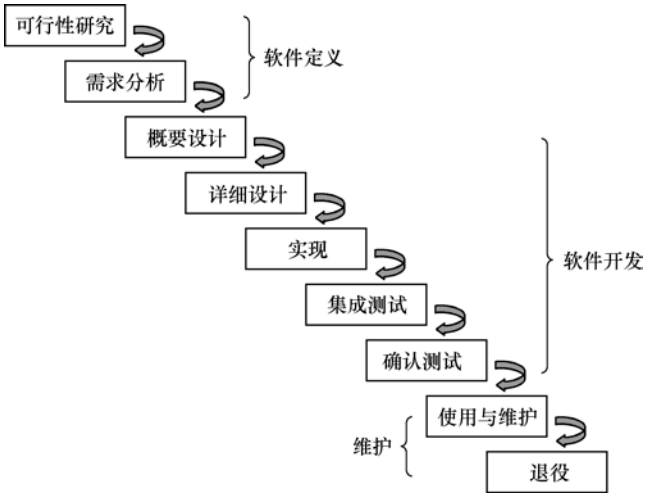


图 1.1　软件生命周期的若干阶段

软件生命周期具有如下几个阶段。

（1）问题的定义及规划

此阶段由软件开发方与需求方共同讨论，主要确定软件的开发目标及其可行性。

（2）需求分析

在确定软件开发可行的情况下，对软件需要实现的各个功能进行详细分析。需求分析作为一个很重要的阶段，在整个软件开发过程中需要根据变更计划不断变化和深入。

（3）软件设计

此阶段主要根据需求分析的结果，对整个软件系统进行设计，如系统框架设计，数据库设计等。

（4）程序编码

程序编码阶段的工作是将软件设计的结果转换成计算机可运行的程序代码。

（5）软件测试

在软件设计完成后要经过严密的测试，以发现软件在整个设计过程中存在的问题并加以纠正。整个测试过程分单元测试、组装测试以及系统测试等阶段，需要建立详细的测试计划并严格按照测试计划进行测试。

（6）运行维护

软件维护是指软件开发完成并投入使用后，由于多方面的原因，软件不能继续适应用户的要求，要延续软件的使用寿命，就必须对软件进行维护。

# 1.2 软 件 缺 陷

## 1.2.1 软件缺陷案例

让我们回首一些臭名昭著的软件缺陷案例，它们都是由于软件测试不充分导致的严重问题。

1963 年，用 FORTRAN 程序设计语言编写的飞行控制软件中的循环语句

```
DO 5 I=1, 3
```

误写为

```
DO 5 I=1.3
```

由于“,”错成“.”，结果一点之差竟造成极为严重的后果，致使美国首次金星探测飞行失败，造成价值约 1 000 多万美元的损失。

1963 年至 1966 年开发的 IBM 360 机操作系统，在开发期中每年花费 5 000 万美元，总共投入的工作量为 5 000 人/年，编写了 100 万行源程序。如此多的开销，却没有得到开发成果。项目负责人 F.P. Brooks 根据这次失败的研发总结出版了《神秘的人月》，成为软件技术发展过程中一个重要的历史性标志。

1979 年，新西兰航空公司的一架客机因计算机控制的自动飞行系统发生故障撞在阿尔卑斯山上，导致机上 257 名乘客全部遇难。

1983 年，美国科罗拉多河水泛滥，但由于计算机对天气形势预测有误，水库未能及时泄洪，造成严重的经济损失和人员伤亡。

1990 年 1 月 15 日，通信中转系统软件发生故障，导致主干远程网大规模崩溃，使数以千计的电信运营公司损失惨重。

1992 年 10 月 26 日，伦敦救护中心的计算机辅助发送系统刚启动就崩溃了，导致这个全世界最大的、每天要接运 5 000 多病人的救护机构全部瘫痪。

1996 年 6 月 4 日，欧洲空间局的阿丽亚娜火箭，发射后 37s 爆炸，损失 6 亿美元。原因在于 Ada 语言编写的一段程序中，将一个 64 位浮点整数转换为 16 位有符号整数时，产生溢出，导致系统崩溃。

临近 2000 年时，计算机业界一片恐慌，这就是著名的“千年虫”问题。其原因是在 20 世纪 70 年代，由于计算机硬件资源很珍贵，程序员为节约内存资源和硬盘空间，在存储日期数据时，只保留年份的后两位，如“1980”被存储为“80”。当 2000 年到来时，问题出现了，计算机无法分清“00”是指“2000 年”还是“1000 年”。例如银行存款的软件在计算利息时，本应该用现在的日期“2000 年某月某日”减去当时存款的日期，但是，由于“千年虫”的问题，结果用“1000 年某月某日”减去当时存款的日期，存款年数就变为负数，导致顾客反要给银行支付巨额的利息。为了解决“千年虫”问题，有关方面花费了大量的人力、物力和财力。

2003 年 8 月 14 日下午 4 时 10 分，美国及加拿大部分地区发生历史上最大的停电事故，15 日晚逐步恢复，经济损失达 250 亿到 300 亿美元。原因在于俄亥俄州的第一能源公司下属的电力监测与控制管理系统软件 XA/21 出现错误，系统中重要的预警部分出现严重故障，负责预警服务的主服务器与备份服务器连接失控，错误没有得到及时通报和处理，最终多个重要设备出现故障，导致大规模停电。

2007 年 8 月 14 日 14 时，美国洛杉矶国际机场电脑发生故障，60 个航班的两万多名旅客无法入关，直至次日凌晨 3 时 50 分，所有滞留旅客才全部入关。原因在于包含旅客姓名和犯罪记录的部分数据系统瘫痪。

据推测，（美国）由于软件缺陷而引起的损失额每年高达 595 亿美元。这一数字相当于美国国内生产总值的 0.6%。

……

## 1.2.2　软件缺陷概述

软件缺陷是计算机软件或程序中存在的某种破坏正常运行能力的问题、错误，或者隐藏的功能缺陷。

软件缺陷包括缺陷标识、缺陷类型、缺陷严重程度、缺陷产生的可能性、缺陷优先级、缺陷状态、缺陷来源、缺陷根源等属性。

① 缺陷标识是标记某个缺陷的唯一的标识，可以使用数字序号表示。

② 缺陷类型是根据缺陷的自然属性划分缺陷种类，如表 1.1 所示。

表 1.1　　软件缺陷类型列表

| 缺陷类型 | 描述 |
| --- | --- |
| 功能 | 影响各种系统功能、逻辑的缺陷 |
| 用户界面 | 影响用户界面、人机交互特性，包括屏幕格式、用户输入灵活性、结果输出格式等方面的缺陷 |
| 文档 | 影响发布和维护，包括注释、用户手册、设计文档 |
| 软件包 | 由于软件配置库、变更管理或版本控制引起的错误 |
| 性能 | 不满足系统可测量的属性值，如执行时间、事务处理速率等 |
| 系统/模块接口 | 与其他组件、模块或设备驱动程序、调用参数、控制块或参数列表等不匹配、冲突 |

③ 缺陷严重程度是指因缺陷引起的故障对软件产品的影响程度。“严重等级”描述在测试条件下，一个错误在系统中的绝对影响，如表 1.2 所示。

表 1.2 软件缺陷严重等级列表

| 缺陷严重等级 | 描述 |
| --- | --- |
| 致命（Fatal） | 系统任何一个主要功能完全丧失，用户数据受到破坏，系统崩溃、悬挂、死机或者危及人身安全 |
| 严重（Critical） | 系统的主要功能部分丧失、数据不能保存，系统的次要功能完全丧失，系统所提供的功能或服务受到明显的影响 |
| 一般（Major） | 系统的次要功能没有完全实现，但不影响用户的正常使用。例如：提示信息不太准确，或用户界面差、操作时间长等一些问题 |
| 较小（Minor） | 使操作者不方便或遇到麻烦，但它不影响功能的操作和执行，如个别的不影响产品理解的错别字、文字排列不对齐等一些小问题 |

④ 缺陷产生的可能性指缺陷在产品中发生的可能性，通常可以用频率来表示，如表 1.3 所示。

表 1.3 缺陷产生的可能性列表

| 缺陷产生的可能性 | 描述 |
| --- | --- |
| 总是（Always） | 总是产生这个软件缺陷，其产生的几率是 100% |
| 通常（Often） | 按照测试用例，通常情况下会产生这个软件缺陷，其产生的几率为 80%～90% |
| 有时（Occasionally） | 按照测试用例，有的时候会产生这个软件缺陷，其产生的几率为 30%～50% |
| 很少（Rarely） | 按照测试用例，很少产生这个软件缺陷，其产生的频率为 1%～5% |

⑤ 缺陷优先级是指缺陷必须被修复的紧急程度。“优先级”的衡量弥补了在严重性中没有考虑的重要程度因素，如表 1.4 所示。

表 1.4 软件缺陷优先级列表

| 缺陷优先级 | 描述 |
| --- | --- |
| 立即解决（P1 级） | 缺陷导致系统几乎不能使用或测试不能继续，需立即修复 |
| 高优先级（P2 级） | 缺陷严重，影响测试，需要优先考虑 |
| 正常排队（P3 级） | 缺陷需要正常排队等待修复 |
| 低优先级（P4 级） | 缺陷可以在开发人员有时间的时候被纠正 |

一般来讲，缺陷严重等级和缺陷优先级相关性很强，但是，具有低优先级和高严重性的错误是可能的，反之亦然。例如，产品徽标一旦丢失了，这种缺陷虽然是用户界面的产品缺陷，但是它影响了产品的形象，因此它是优先级很高的软件缺陷。

⑥ 缺陷状态是指缺陷通过一个跟踪修复过程的进展情况，也就是在软件生命周期中状态的基本定义，如表 1.5 所示。

表 1.5 软件缺陷状态列表

| 缺陷状态 | 描述 |
| --- | --- |
| 激活或打开（Active or Open） | 问题还没有解决，存在源代码中，确认“提交的缺陷”，等待处理，如新报的缺陷 |
| 已修正或修复（Fixed or Resolved） | 已被开发人员检查、修复过的缺陷，通过单元测试，认为已解决但还没有被测试人员验证 |
| 关闭或非激活（Close or Inactive） | 测试人员验证后，确认缺陷不存在之后的状态 |

续表

| 缺陷状态 | 描述 |
| --- | --- |
| 重新打开 | 测试人员验证后还依然存在的缺陷，等待开发人员进一步修复 |
| 推迟 | 这个软件缺陷可以在下一个版本中解决 |
| 保留 | 由于技术原因或第三方软件的缺陷，开发人员不能修复的缺陷 |
| 不能重现 | 开发不能复现这个软件缺陷，需要测试人员检查缺陷复现的步骤 |
| 需要更多信息 | 开发能复现这个软件缺陷，但开发人员需要一些信息，例如缺陷的日志文件、图片等 |

⑦ 缺陷来源是指缺陷所在的地方，如文档、代码等，如表 1.6 所示。

表 1.6　　软件缺陷来源列表

| 缺陷来源 | 描述 |
| --- | --- |
| 需求说明书 | 需求说明书的错误或不清楚引起的问题 |
| 设计文档 | 设计文档描述不准确、和需求说明书不一致的问题 |
| 系统集成接口 | 系统各模块参数不匹配、开发组之间缺乏协调引起的缺陷 |
| 数据流（库） | 由于数据字典、数据库中的错误引起的缺陷 |
| 程序代码 | 纯粹因编码中的问题所引起的缺陷 |

⑧ 缺陷根源是指造成上述错误的根本因素，用以寻求软件开发流程的改进、管理水平的提高，如表 1.7 所示。

表 1.7　　软件缺陷根源列表

| 缺陷根源 | 描述 |
| --- | --- |
| 测试策略 | 错误的测试范围，误解测试目标，超越测试能力等 |
| 过程、工具和方法 | 无效的需求收集过程，过时的风险管理过程，不适用的项目管理方法，没有估算规程，无效的变更控制过程等 |
| 团队/人 | 项目团队职责交叉，缺乏培训。没有经验的项目团队，缺乏士气和动机不纯等 |
| 缺乏组织和通信 | 缺乏用户参与，职责不明确，管理失败等 |
| 硬件 | 硬件配置不对、缺乏，处理器缺陷导致算术精度丢失，内存溢出等 |
| 软件 | 软件设置不对、缺乏，操作系统错误导致无法释放资源，工具软件的错误，编译器的错误，千年虫问题等 |
| 工作环境 | 组织机构调整，预算改变，工作环境恶劣（如噪声过大） |

# 1.3　软件工程概述

## 1.3.1　软件工程三要素

软件工程是一门研究用工程化方法构建和维护有效的、高质量的软件的学科，它包括 3 个要素：方法、工具和过程。

软件工程方法为软件开发提供了“如何做”的技术。它包括多方面的任务，如项目计划与估算、软件系统需求分析、数据结构、系统总体结构的设计、算法过程的设计、编码、测试以及维护等。

软件工具为软件工程方法提供了自动的或半自动的软件支撑环境。目前，计算机辅助软件工

程（CASE）将各种软件工具、开发机器和一个存放开发过程信息的工程数据库组合起来，形成一个软件工程环境。

软件工程的过程则是将软件工程的方法和工具综合起来以达到合理、及时地进行计算机软件开发的目的。过程定义了方法使用的顺序，要求交付的文档资料，为保证质量和协调变化所需要的管理以及软件开发各个阶段完成的里程碑。

### 1.3.2 软件开发过程模型

软件开发模型是软件开发全过程、软件开发活动以及它们之间关系的结构框架，主要为软件项目的管理提供里程碑和进度表，并给软件开发提供原则和方法。

下面介绍 RUP 过程和敏捷过程。

1. RUP

RUP（Rational Unified Process，Rational 统一过程）是 Rational 公司（现归属 IBM 公司）推出的一种软件过程产品，以统一建模语言（UML）描述软件开发过程。

（1）RUP 各个阶段

RUP 分为 4 个顺序的阶段，分别是初始阶段、细化阶段、构建阶段和交付阶段。每个阶段结束于一个主要的里程碑；每个阶段本质上是两个里程碑之间的时间跨度。在每个阶段的结尾执行一次评估以确定这个阶段的目标是否已经满足，如果评估结果满意，允许项目进入下一个阶段。其中每个阶段又可以进一步分解迭代。一个迭代是一个完整的开发循环，产生一个可执行的产品版本，作为最终产品的一个子集。产品增量式地发展，从一个迭代过程到另一个迭代过程，直到成为最终的系统。

如图 1.2 所示，RUP 的过程可用二维坐标来描述。横轴通过时间组织，是过程展开的生命周期特征，体现开发过程的动态结构，用来描述它的术语主要包括周期、阶段、迭代和里程碑；纵轴以内容来组织为自然的逻辑活动，体现开发过程的静态结构，用来描述它的术语主要包括活动、产物、工作者和工作流。

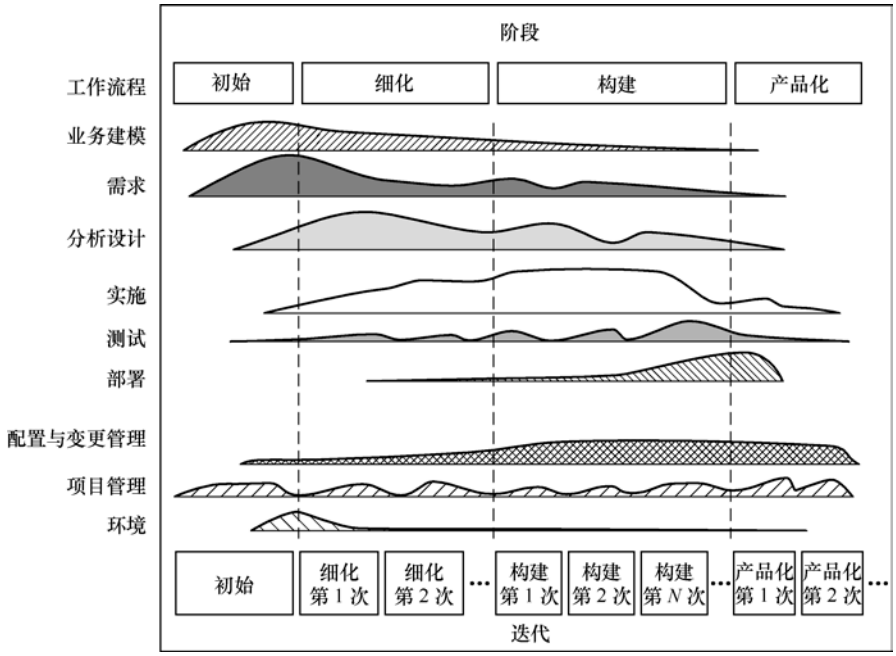


图 1.2 RUP 的过程图

下面依次介绍 RUP 软件生命周期的四个阶段。

① 初始阶段

初始阶段的目标是为系统建立商业案例并确定项目的边界。为了达到该目的必须识别所有与系统交互的外部实体，并在较高层次上定义交互的特性。这包括识别出所有用例并描述几个重要的用例。本阶段具有非常重要的意义，在这个阶段中所关注的是整个项目进行中的业务和需求方面的主要风险。

在初始阶段应该取得如下成果。

- 蓝图文档，即关于项目的核心需求、关键特性、主约束的总体蓝图。
- 初始的用例模型（约占总体的 10%～20%）。
- 初始的项目术语表。
- 初始的商业案例，包括商业环境、验收标准等。
- 初始的风险评估。
- 项目计划。
- 一个或多个原型。

② 细化阶段

细化阶段的目标是分析问题领域，建立坚实的体系结构基础，制订项目计划，消除项目中具有最高风险的因素。为了达到该目的，必须在理解整个系统的基础上，对体系结构作出决策，包括其范围、主要功能和性能等非功能需求。同时为项目建立支持环境，包括创建开发案例，创建模板、工作准则并准备工具。

细化阶段的成果如下。

- 用例模型，所有的用例和参与者都已被识别出，并完成大部分的用例描述。
- 补充非功能性要求以及与特定用例没有关联的需求。
- 软件体系结构的描述。
- 可执行的软件原型。
- 修订过的风险清单和商业案例。
- 整个项目的开发计划，该开发计划应体现迭代过程和每次迭代的评价标准。
- 更新的开发案例。
- 初步的用户手册。

③ 构建阶段

在构建阶段，组件和应用程序的其余功能被开发并集成为产品，所有的功能都被彻底地测试。从某种意义上说，构建阶段是一个制造过程，其重点为管理资源及控制运作，从而降低成本、加速进度和优化质量。

构造阶段的成果是可以交付给最终用户的产品，它至少包括以下内容。

- 集成于适当操作系统平台上的软件产品。
- 用户手册。
- 当前版本的描述。

④ 交付阶段

交付阶段的重点是确保软件对最终用户是可用的。交付阶段可以跨越几次迭代，包括为发布做准备的产品测试，基于用户反馈的少量的调整。在这一阶段，用户反馈应主要集中在产品调整、设置、安装和可用性问题上。

（2）RUP 核心工作流

RUP 共有 9 个核心工作流，分为 6 个核心过程工作流和 3 个核心支持工作流。6 个核心过程工作流可能使人想起传统瀑布模型中的几个阶段，但应注意迭代过程中的阶段是完全不同的，这些工作流在整个生命周期中一次又一次被执行。

① 商业建模

商业建模工作流描述了如何为新的目标组织开发一个构想，并基于这个构想在商业用例模型和商业对象模型中定义组织的过程、角色和责任。

② 需求

需求工作流的目标是描述系统应该做什么，并和开发人员和用户达成共识。为了达到该目标，要对需要的功能和约束进行提取、组织、文档化，理解系统所解决问题的定义和范围。

③ 分析和设计

分析和设计工作流是将需求转化成系统的设计，为系统开发一个健壮的结构使其与实现环境相匹配，设计活动以体系结构设计为中心，其结果是一个设计模型和一个可选的分析模型。

④ 实现

实现工作流的目的包括以层次化的子系统形式定义代码的组织结构，以组件的形式（源文件、二进制文件、可执行文件）实现类和对象，实现可执行的系统。

⑤ 测试

测试应确保软件中所有组件的正确集成，检验所有的需求是否已被正确实现，识别并确认缺陷在软件部署之前已被提出并处理。

⑥ 部署

部署工作流的目的是成功地生成版本并将软件分发给最终用户。部署工作流描述了那些与确保软件产品对最终用户具有可用性相关的活动，包括软件打包、生成软件本身以外的产品、安装软件以及为用户提供帮助。

⑦ 配置和变更管理

配置和变更管理工作流描绘了如何在多个成员组成的项目中控制大量的产物。配置和变更管理工作流提供了准则来管理演化系统中的多个变体，跟踪软件创建过程中的版本。工作流描述了如何管理并行开发、分布式开发，如何自动化创建工程，同时也对产品修改原因、时间、人员保持审计记录。

⑧ 项目管理

软件项目管理平衡各种可能产生冲突的目标，管理风险，克服各种约束并成功交付使用户满意的产品。其目标包括：为项目的管理提供框架，为计划、人员配备、执行和监控项目提供实用的准则，为管理风险提供框架等。

⑨ 环境

环境工作流的目的是向软件开发组织提供软件开发环境，包括过程和工具。环境工作流集中于配置项目过程中所需要的活动，同样也支持开发项目规范的活动，提供了逐步的指导手册并介绍了如何在组织中实现过程。

（3）以用例驱动为核心

开发软件系统的目的是要为该软件系统的用户服务。因此，要创建一个成功的软件系统，必须明白该软件的用户需要什么。“用户”并不仅仅局限于人，还包括其他软件系统。一个用例就是系统向用户提供一个有价值的结果的某项功能，所有用例结合起来就构成了“用例模型”，该模型

描述系统的全部功能。用例模型是从用户的利益角度出发进行考虑，设计人员创建一系列用例模型，开发人员审查每个后续模型，以确保它们符合用例模型。

**2. 敏捷过程**

传统计划驱动的开发方法往往将大量的开发时间用于开发文档的撰写和维护，而真正花在代码上的时间较少，并且由于依赖过程控制，而不是程序员的自我管理，导致开发过程管理复杂且低效，极大地阻碍了软件开发效率。因此，产生了一种轻量级的开发方法。

（1）简介

2001 年是全球软件行业具有历史意义的一年，“敏捷联盟”成立并发表了对传统软件开发过程具有颠覆意义的《敏捷宣言》，即“通过开发软件和帮助别人开发软件，我们找到了一些更好的开发软件的方式。通过这一工作，我们得出了这些价值：

- 个体和交互胜过过程和工具
- 可以工作的软件胜过面面俱到的文档
- 客户合作胜过合同谈判
- 响应变化胜过遵循计划

也就是说，尽管右边的项也有价值，但我们认为左边的项更有价值。”

从那以后，在敏捷过程价值观的指导下诞生了很多具体的开发过程，包括 XP（极限编程）、FDD（功能驱动开发）、SCRUM 开发过程等，其中 XP 是影响最为广泛的一种开发过程。

（2）敏捷开发的特征

敏捷方法具有如下两个主要特征。

第一个特征是开发采用适应性方法，经过多次迭代开发过程逐步逼近实际需求。这种开发方法以小型发布、不断集成测试为核心。每一次的小型发布都经过严格测试后集成到最终产品中，保证每一次小型发布都是经过测试的高质量的代码。开发过程以代码为核心，而不是以文档为核心。传统的以文档为主要输出的设计过程（需求分析、高层架构设计、概要设计、详细设计等）被大大压缩。设计以简单为原则，小组通过密切而有效率的交流达到对设计的统一理解和认识。进行代码的编写、测试、发布、重构，然后进入第二次迭代。

敏捷开发的第二个特征是以人为本。传统计划驱动方法企图以文档、过程为核心，从而抹杀人的重要性。在敏捷方法里，程序员在软件开发中不再是单纯被管理的对象，而是开发的主体。所有的主要设计策略的制定、开发方法的选择、需求的确定都由程序员决定，以往开发过程中的对开发过程的严格控制、检测，软件的各种测量等都大大简化。

（3）敏捷开发的价值与局限

敏捷方法抛弃了烦琐的文档管理，依靠程序员主动、开放、频繁、面对面的高效交流来达成对需求、目标、设计实现的理解；敏捷方法抛弃了机械、严格的过程控制，依赖程序员和开发团队的高标准自我要求：严格的自律，团队合作精神，个人高度自觉的主动性，责任感。敏捷方法的高效和高质量实际上是以程序员的高素质和开发团队的高度合作的开发文化为基础的。敏捷方法的实施前提是必须找到愿意并有能力实施敏捷方法的团队。XP 的创始人 Beck 也曾建议过有些情况是不适合采用 XP 方法的，比如不能接受 XP 文化、团队规模过大、重构开销过大等。

## 1.3.3　软件过程能力评估及 CMM/CMMI

1987 年 9 月，卡内基—梅隆大学的软件工程研究所（SEI）发布了一份能力成熟度框架以及

一套成熟度问卷。4 年后，SEI 推出了 CMM 1.0 版。CMM 1.0 版在成熟度框架的基础上建立了可用的模型，该模型能够有效地帮助软件公司建立和实施过程改进计划。近几年，SEI 又推出了 CMM 2.0 等版本，同时进入了 ISO 体系，称为 ISO/IEC15504（软件过程评估）。

1. CMM

CMM 作为软件过程改良和评估模型，也称为 SEI SW-CMM。该模型于 1991 年发布，其核心是把软件开发视为一个过程，进行过程的监控和研究。CMM 提供了一个软件过程改进的框架，这个框架与软件生命周期和所采用的开发方法无关。

CMM 为软件企业的过程能力提供了一个阶梯式的进化框架，该框架共有 5 级，分别是初始级、可重复级、已定义级、已管理级和优化级。除第一级外，每一级都设定了一组目标，如果达到了这组目标，则表明达到了这个成熟级别，可以向下一个级别迈进，如图 1.3 所示。

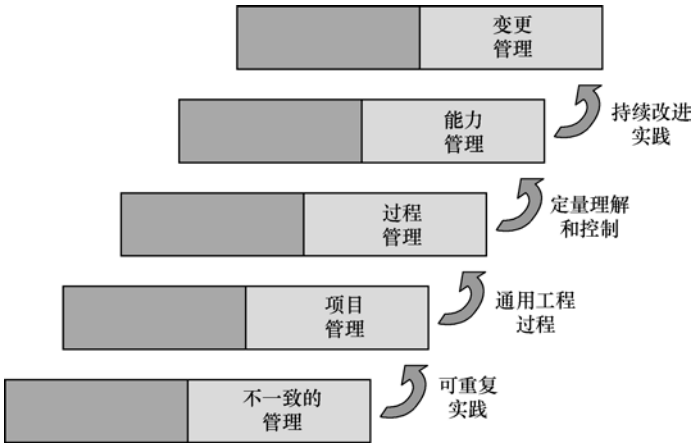


图 1.3　CMM 的 5 层模型

（1）初始级

在这个阶段，软件开发过程表现得非常随意，偶尔会出现混乱的现象，只有很少的工作过程是经过严格定义的，开发成功往往依靠的是某个人的智慧和努力。此时的软件机构基本没有健全的软件工程管理制度，其软件过程完全取决于项目组的人员配备，具有不可预测性。人员变了过程也随之改变。往往由于缺乏健全的管理和周密的计划，延期交付和费用超支的情况经常发生。

总之，处于 1 级成熟度的软件机构，其过程能力是不可预测的，其软件过程是不稳定的，产品质量只能根据相关人员的个人工作能力而不是软件机构的过程能力来预测。

（2）可重复级

这一阶段已经建立了基本的项目管理过程。按部就班地设计功能、跟踪费用，根据项目进度表进行开发。对于相似的项目，可以重用以前已经开发成功的部分。处于 2 级成熟度的软件机构，针对所承担的软件项目已建立了基本的软件管理控制制度。通过对以前项目的观察和分析，可以提出针对现行项目的约束条件。项目负责人跟踪软件产品开发的成本、进度以及产品的功能和质量，并识别出为满足约束条件所应解决的问题。此时，软件的需求是条理化的，而且其完整性是受控制的。软件机构已经制定了项目标准，并且能确保严格执行这些标准。项目组与客户及承包商已经建立起一个稳定的、可管理的工作环境。

处于 2 级成熟度的软件机构的过程能力可以概况为软件项目的策划和跟踪是稳定的，已经为

一个有纪律的管理过程提供了可重复以前成功实践的项目环境。软件项目工程活动处于项目管理体系的有效控制之下，执行着基于以前项目的准则且合乎现实的计划。

（3）已定义级

在这一阶段，软件开发的工程活动和管理活动都是文档化、标准化的，是被集成为一个有组织的标准开发过程。所有项目的开发和维护都在这个标准基础上进行定制。处于 3 级成熟度的软件机构，有一个固定的过程小组从事软件过程工程活动。过程小组可以利用过程模型进行过程例化活动，从而获得一个针对某个特定的软件项目的过程实例，并投入过程运作而开展有效的软件项目工程实践。同时，过程小组还可以推进软件机构的过程改进活动。在该软件机构内实施培训计划，能够保证全体项目负责人和项目开发人员具有完全承担任务所要求的知识和技能。

处于 3 级成熟度的软件机构的过程能力可以概况为，无论是管理活动还是工程活动都是稳定的。软件开发的成本、进度以及产品的功能和质量都受到控制，而且软件产品的质量具有可追溯性。

（4）已管理级

这一阶段已经对软件开发过程和产品质量的测试细节有了很好的归纳，产品和开发过程都可以定量地分解和控制。

处于 4 级成熟度的软件机构的过程能力可以概况为软件过程是可度量的，软件过程在可度量的范围内运行。这一级的过程能力允许软件机构在定量的范围内预测过程和产品质量趋势，在发生偏离时可以及时采取措施予以纠正，并且可以预期软件产品是高质量的。

（5）优化级

这一阶段通过建立开发过程的定量反馈机制，不断产生新的思想，采用新的技术来优化开发过程。处于 5 级成熟度的软件机构，可以通过对过程实例性能的分析和确定产生某一缺陷的原因，来防止再次出现这种类型的缺陷，对任何一个过程实例的分析所获得的经验教训都可以成为该软件机构优化其过程模型的有效依据，从而使其他项目的过程实例得到优化。这样的软件机构可以通过从过程实施中获得定量的反馈信息，在采用新思想和新技术的同时测试它们，从而不断地改进和优化软件过程。

处于 5 级成熟度的软件机构的过程能力可以概况为软件过程是可优化的。这一级的软件机构能够持续不断地改进其过程能力，既对现行的过程实例不断地改进和优化，又借助于所采用的新技术和新方法来实现未来的过程改进。

总而言之，根据软件生产的历史与现状，CMM 框架可用 5 个不断进化的层次来表达：其中初始层是混沌的过程；可重复层是经过训练的软件过程；已定义层是标准一致的软件过程；可管理层是可预测的软件过程；优化层是能持续改善的软件过程。任何企业所实施的软件过程都可能在某一方面比较成熟，在另一方面不够成熟，但总体上必然属于这 5 个层次中的某一个层次。在某个层次内部，也有成熟程度的区别。在一个较低层次的上沿，很可能与一个较高层次的下沿非常接近，此时由这个较低层次向该较高层次进化也就比较容易。反之，在一个较低层次的下沿向较高层次进化就比较困难。在 CMM 框架的不同层次中，需要解决带有不同层次特征的软件过程问题。因此，一个软件开发企业首先需要了解自己处于哪一个层次，然后才能够对症下药地针对该层次的特殊要求解决相关问题。任何软件开发企业在致力于软件过程改善时，只能由所处的层次向紧邻的上一层次进化，即软件过程的进化是渐进的，而不是跳跃的。

2. CMMI

CMMI 是 SEI 于 2000 年发布的 CMM 的新版本。CMMI（Capability Maturity Model Integration，

能力成熟度模型集成）将各种能力成熟度模型，包括 Software CMM、Systems Eng-CMM、People CMM 等，整合到同一架构中去，由此建立包括软件工程、系统工程和软件采购等在内的模型集成，以解决除软件开发以外的软件系统工程和软件采购工作中的需求。

软件企业采用多种模型改进过程能力时，往往会存在以下一些问题。

① 软件企业不能集中不同过程改进的能力以取得更大成绩。

② 软件企业往往要进行一些重复的培训、评估和改进活动，从而增加了成本。

③ 由于不同模型有些说法不一致，或活动不协调，甚至相抵触，导致难以理解。

正因为如此，美国国防部把各种能力成熟度模型集成为 CMMI，其基本思想如下。

① 解决软件项目过程改进难度增大的问题。

② 实现软件工程的并行与多学科组合。

③ 实现过程改进的最佳效益。

CMMI 具有如下两个功能：第一，软件采购方法的改革；第二，从集成产品与过程的角度出发，包含健全的系统开发原则的过程改进。

CMMI 纠正了 CMM 存在的一些缺点，消除了不同模型之间的不一致和重复，降低了基于模型改善的成本，指导组织改善软件过程，提高产品和服务的开发、获取和维护能力，因此更加适用于企业的过程改进实施。

## 1.4 软件工程与软件测试

软件开发与软件测试具有密不可分的关系，从以下几个方面进行介绍。

### 1. 测试与开发各阶段的关系

图 1.4 所示为软件测试与软件开发各阶段的关系，软件测试在开发阶段具有如下作用。

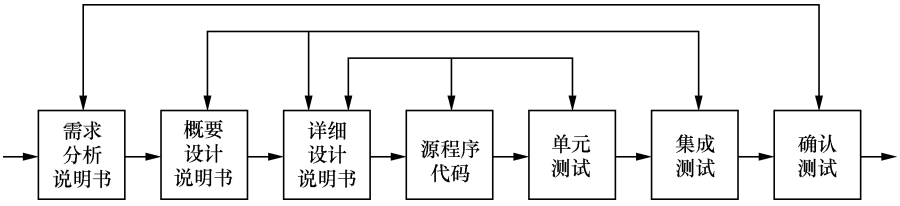


图 1.4 软件测试与软件开发过程的关系

① 项目规划阶段：负责从单元测试到系统测试的整个测试阶段的监控。

② 需求分析阶段：确定测试需求分析，制订系统测试计划，评审后成为管理项目。

③ 详细设计和概要设计阶段：确保集成测试计划和单元测试计划完成。

④ 编码阶段：由开发人员进行自己负责部分的测试代码。在项目较大时，由专人进行编码阶段的测试任务。

⑤ 测试阶段（单元、集成、系统测试）：依据测试代码进行测试，并提交相应的测试状态报告和测试结束报告。

### 2. 测试与开发的并行性

图 1.5 所示为软件测试与软件开发之间的并行关系。

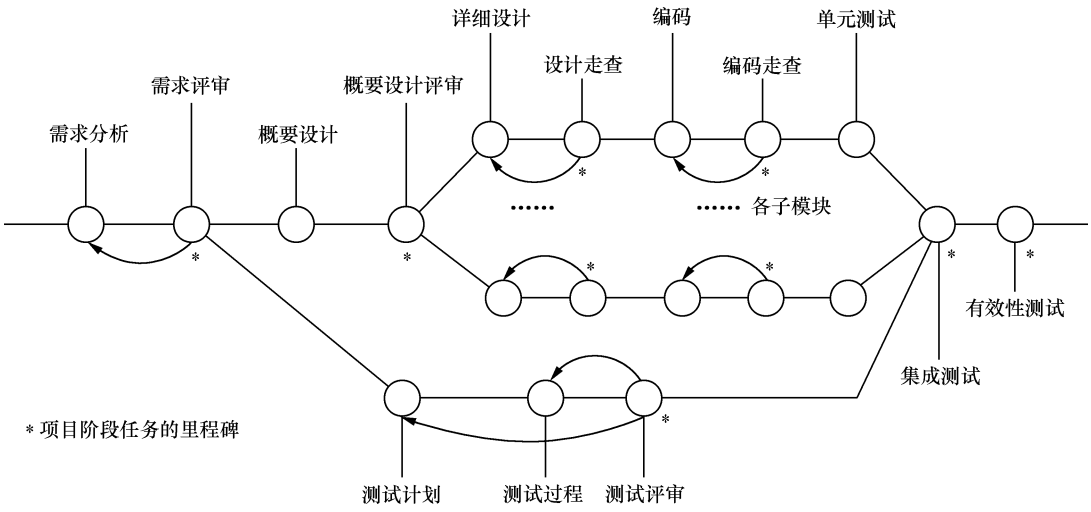


图 1.5　软件测试与软件开发的并行性

### 3. 完整的软件开发流程

完整的软件开发流程如图 1.6 所示。从图 1.6 中可以看出测试过程和开发过程贯穿软件过程的整个生命周期，相辅相成、相互依赖，具体概括为如下 3 个关键点。

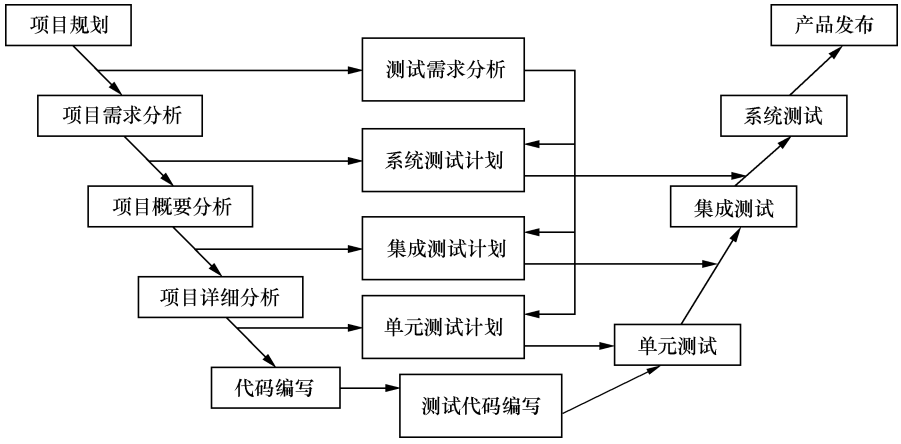


图 1.6　完整的开发流程

① 测试过程和开发过程是同时开始、同时结束、两者保持同步的关系。

② 测试过程是对开发过程中的阶段性成果和最终产品进行验证的过程，所以两者相互依赖。前期测试过程更多地依赖于开发过程；而后期开发过程更多地依赖于测试过程。

③ 测试工作的重点和开发工作的重点可能是不一样的，两者有各自的特点。

## 思考与练习

### 一、填空题

1. 计算机系统发展的早期所形成的一系列错误概念和做法，已经严重地阻碍了计算机软件的

开发，甚至有的_____根本无法维护，只能提前报废，造成大量人力、物力的浪费，从而引发了软件危机。为了研究解决的方法，计算机科学技术领域中的一门新兴的学科逐步形成了，这就是_____。

2. 软件工程方法学包括的三要素：_____、_____、_____。

## 二、选择题

1. 软件本身的特点和目前的软件开发模式使隐藏在软件内部的质量缺陷不可能完全避免，在下列关于导致软件质量缺陷的原因的描述中，不正确的是（　　）。

A. 软件需求模糊以及需求的变更，从根本上影响着软件产品的质量

B. 目前广泛采用的手工开发方式难以避免出现差错

C. 程序员编码水平低下是导致软件缺陷的最主要原因

D. 软件测试技术具有缺陷

2. 下列哪种方法会减少成本（　　）。

A. 让客户去找缺陷　　B. 发现缺陷而不是预防它们

C. 预防缺陷而不是发现它们　　D. 忽视小的缺陷

3. 实施缺陷跟踪的目的是（　　）。

A. 软件质量无法控制　　B. 问题无法量化

C. 重复问题接连产生　　D. 解决问题的知识无法保留

E. 确保缺陷得到解决　　F. 使问题形成完整的闭环处理

4. 下列关于极限测试，说法不正确的是（　　）。

A. 相对传统的软件开发方法，极限编程可以随时应对新增或改变的需求

B. 极限编程的单元测试是由编码人员完成的测试

C. 极限编程要求在编码之前先设计测试

D. 验收测试由用户来完成，编码人员不必在现场

## 三、简答题

1. 什么是软件？软件经过了哪几个发展阶段？
2. 软件缺陷是什么？
3. RUP 是什么？具有什么特征？
4. 敏捷开发有什么特征？
5. 软件开发过程模型是什么？
6. CMMI 与 CMM 的关系是什么？