软件开发基本原则（一）—— 策略和因素

1 概 述

时间 -- 成本 -- 质量（或特性）是评价软件项目成败的三个关键指标，这三个指标之间相互影响和制约，形成了所谓的“项目管理三角形”。要提高质量或增加特性意味着成本和时间的增加，或两者都增加；要在时间不变的前提下缩减开发成本或成本不变的前提下缩减时间则意味着质量的下降或特性的削减。

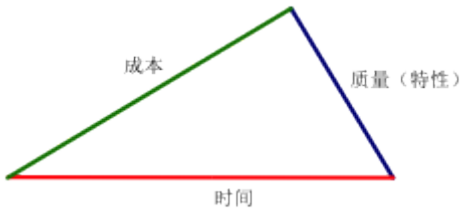

图 1-1 项目管理三角形

上述分析其实只是理论上的“理想平衡”状态。现实工作中往往出现的情形是：要么时间超过计划，要么成本超过预算，要么质量达不到要求，要么三个指标都达不到预期。

典型例子：

由于客户的压力需要尽量缩减开发时间，由于企业间的竞争和盈利压力需要尽量节约成本，因此需要一个人做两个人的工作，一个月做两个月的工作，同时压缩需求分析、设计、测试、评审和项目会议等活动。可想而知，即使软件的构建阶段能够按时完成，但做出的软件质量是难以保证的。更糟糕的还在后面：由于质量的低劣，构建阶段结束后对系统进行集成测试时，很多问题就会暴露出来：对某些需求的理解有误差，导致这部分功能要重新分析、设计、编码和测试；架构设计缺乏整体思维导致系统不同模块各自为政，产生大量重复的难以维护的代码；编码太仓促导致一大堆的Bug；沟通不畅顺导致模块接口不兼容……从而项目被带入了修改无限循环地带，即使勉强上线发布，修改还是一直持续，直至最后，没有人再敢接近这套代码，对这个项目谈虎色变。

软件开发项目有其自身规律和原则，只有遵守其原则并付诸相应的实践才可能使项目健康稳定地前进。本文讲述的是软件开发的基本原则，它是通用的，几乎适用于所有的软件开发项目。不同项目可以根据自身特点在原则的指导下定义相应的项目开发实践。

2 策略和因素

2.1 总体策略

要避免混乱低效的开发，就要求每个人能够放弃他们自己的一些坏习惯，通过采取以下四种策略实现快速开发：

1、 避免典型错误

2、 打好开发基础

3、 管理风险，避免灾难发生

4、 采用面向进度的实践

图 2.1-1 快速开发的四跟支柱

典型错误：是指一些经常被许多人使用的无效的开发实践，如：不现实的预期，缺乏计划，功能蔓延和银弹综合症等。将在第3章详细讲解。

开发基础：是指项目开发过程中管理、技术、质量保证等方面行为和活动，如：计划编制，需求管理和技术回顾等。将在第4章详细讲解

风险管理：是指对有可能影响项目的风险进行评估和控制。将在第5章讨论进度计划相关的风险。

面向进度的实践有以下三类

- 面向速度的实践：可以提升开发速度，帮助你更快的交付软件
- 面向进度风险的实践：可以降低计划风险，帮助你的项目平稳推进
- 面向可视化的实践：可以提高进程的可视化程度，帮助你掌握项目动态

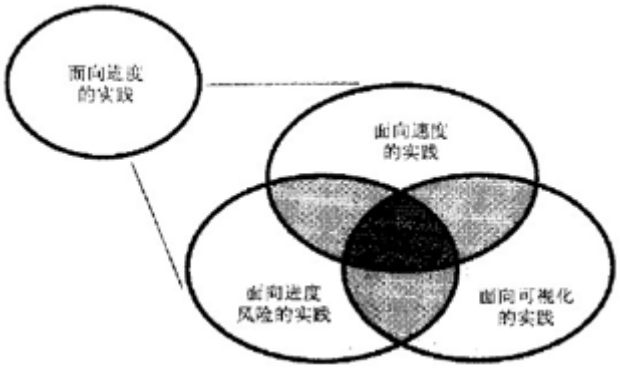

图 2.1-2 面向进度的实践

图2.1-1所示的前三根柱子为可能的最佳进度提供了最重要的支撑，虽然可能不是最理想的，但却是最需要的。也就是说，即使不借助于面向进度的实践方法，也可能实现较优化的项目进度；但是，如果仅仅依赖面向进度的实践却不可以支撑可能的最佳进度计划。

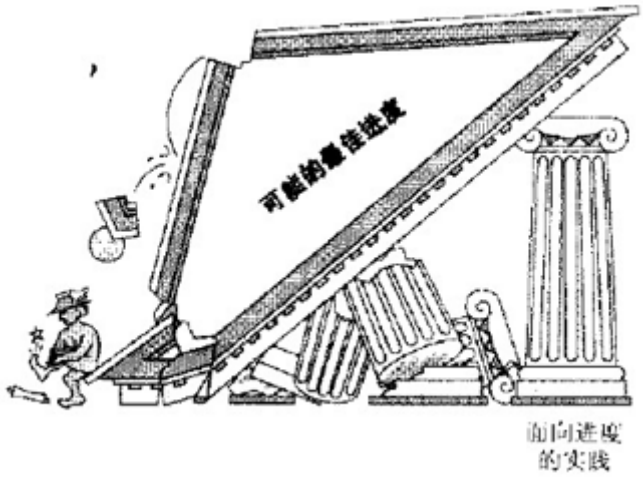

图 2.1-3 仅仅依赖面向进度的实践不足以支撑最佳进度计划

2.2 软件开发的四维

每个软件项目都有四个重要的维：

- 人员：完成任务要么快，要么慢
- 过程：优化人员的工作效率，或者浪费人员的时间
- 产品：以自我完善的形式定义，或者阻碍人员达到最好效果的形式定义
- 技术：促进或者阻碍开发的实现

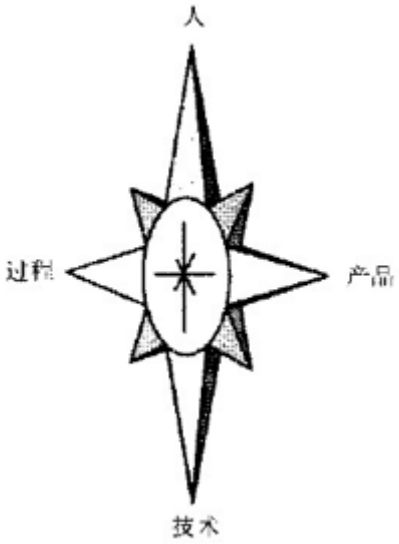

图 2.2-1 开发速度的四维

2.2.1 人员

- 具有不同深度与广度经验的人的生产效率差异超过10：1。
- 具有同等经验的人的生产效率差异在10：1范围内。
- 具有不同经验的小组的生产效率差异在5：1范围内。
- 具有相似经验的小组的生产效率差异在2.5：1范围内。

研究数据：

人件极大地影响着生产效率，任何关注提高生产效率的组织首先必须有一套良好的人员激励、团队合作、员工选择及培训机制

发挥人员最大潜能，缩短项目周期的方法：

1、 项目成员的选择

五个原则：

- 用更少更好的人
- 使任务与人员的技能和动机相匹配
- 帮助人员自我实现，而不是强制地把他推到他最有经验或最需要他的岗位上
- 人员选择应强调人员之间的互补与协调性
- 尽快排除或替换不称职的人员

2、 团队组织结构

人员的组织方式对人员的工作效率有很大影响，调整项目团队以使之与项目规模、产品特点以及进度目标相匹配。特定的软件项目也可以从适宜的专门组织中受益。

3、 人员激励

人员激励能激发人的动力，从而付出额外的努力工作；它适用于不同组织、不同项目和不同人员。人员激励是达成快速开发的最具潜力方法

2.2.2 过程

研究数据：

Hughes Aircraft、Lockheed、Motorola、NASA、Raytheon 和 Xerox 等组织通过对开发过程的改进将产品上市时间缩短了一半，降低成本、减少错误为原来的1/3~1/10。

过程是指软件开发生命周期中定义的一系列工作流程和活动的集合。可以概括为以下三类：

- 基本过程：包括获取过程、供应过程、开发过程、运作过程、维护过程和管理过程
- 支持过程：包括文档过程、配置管理过程、质量保证过程、验证过程、确认过程、联合评审过程、审计过程以及问题解决过程
- 组织过程：包括基础设施过程、改进过程以及培训过程

忽略过程容易造成工作效率低下，工作目的交叉重复，产品质量难以保证等问题；另一方面，如果过程过于严格、过于官僚同样会挫伤人员的积极性，或者由于执行过程的成本过高而影响实际的工作效率。

组织可以对现有的过程进行裁剪和调整，制定出适合特定项目的过程；或者可以为项目从头开始定义过程。无论是裁剪过程或是定义过程，应该把关注点放在以下几个方面：

1、 避免返工

软件项目节省时间一个最直接的方式就是确定过程，避免重复工作。如果在项目最后阶段改变需求，就可能不得不重新设计、编码和测试；如果直到系统测试阶段才发现设计有问题，就可能不得不扔掉已经细化的设计和编码。

2、 质量保证

质量保证有两个目的

- 确保交付的产品能够达到可接受的质量水平
- 在各阶段以最少的时间和成本代价查出错误

应尽早在错误发生的时候就查出来，错误在产品中停留的时间越长，清楚错误所花费的时间和成本就越多。质量保证是任何开发过程中必不可少的部分。

3、 开发基础

一系列的软件工程实践活动形成了开发基础，如：分析、设计、构建、集成和测试等。在过程中对开发基础加以关注，并定义良好的工作规范和任务集合能防止项目失控。

4、 风险管理

与进度相关的风险管理是开发过程必要的组成部分。风险管理虽然不能直接提高开发速度，但它是避免项目灾难的有效实践。

5、 资源目标

资源包括人力资源、环境资源和软硬件资源等。优化资源的调配有助于提高生产率。

6、 生命周期计划

生命周期计划是基本的管理计划，有助于确定软件项目要进行的活动集合和资源分配。每种周期模型都有其适用范围和缺点，为项目选择适当的生命周期模型能有效提高工作效率或降低项目风险。

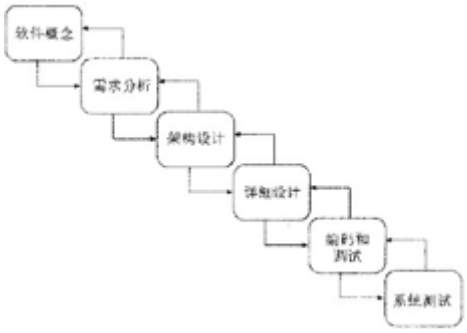

图 2.2.2-1 纯瀑布模型

图 2.2.2-2 瀑布模型的另一种形式——鲑鱼生命期模型

图 2.2.2-3 编码修正模型（一种不规范的模型）

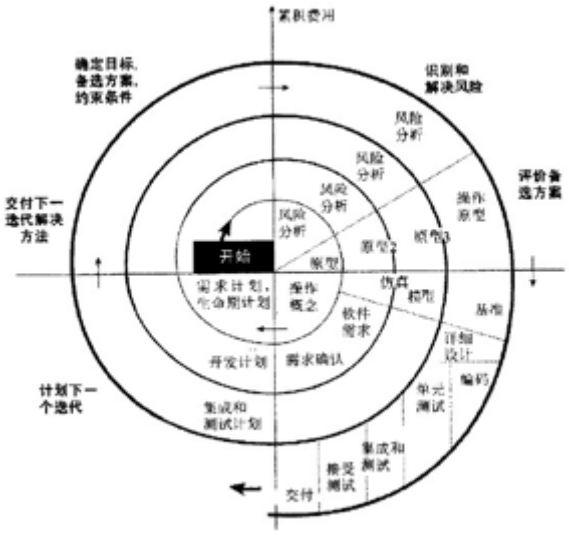

图 2.2.2-4 螺旋模型

图 2.2.2-5 生鱼片模型

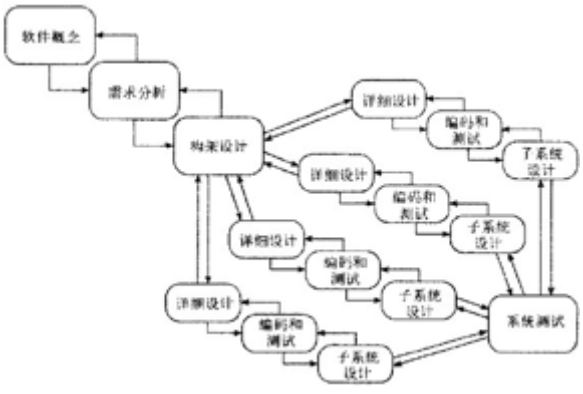

图 2.2.2-6 包含子项目的瀑布模型

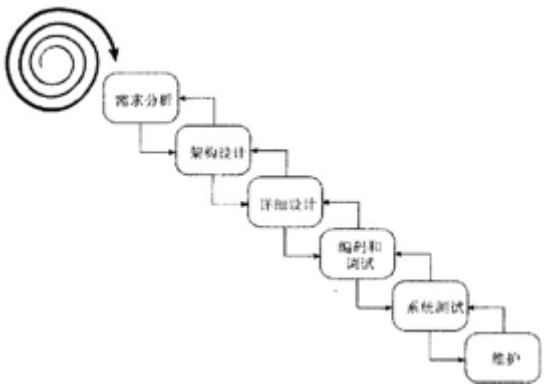

图 2.2.2-7 能够降低风险的瀑布模型（对需求分析和架构设计阶段采用螺旋模型）

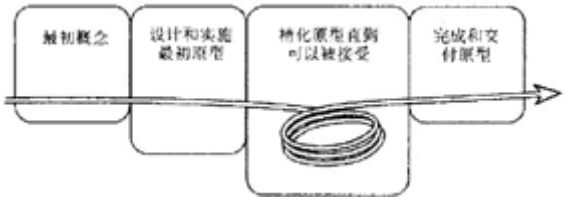

图 2.2.2-8 渐进原型模型

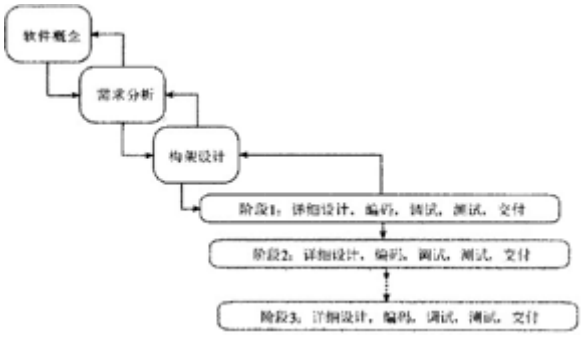

图 2.2.2-9 阶段交付模型

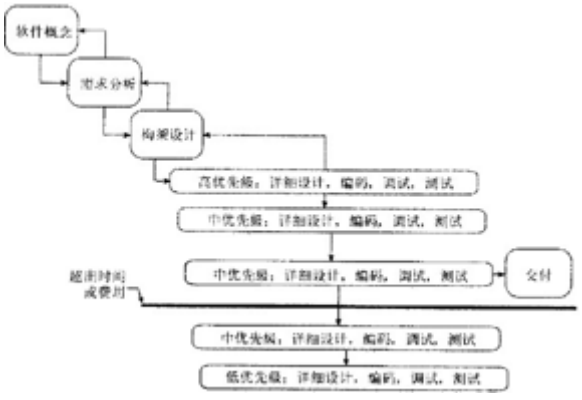

图 2.2.2-10 面向进度模型

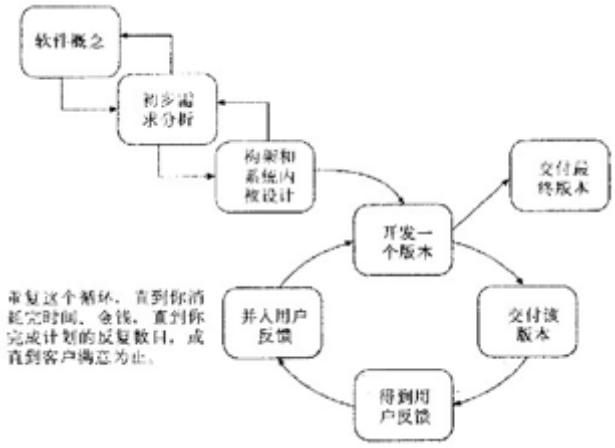

图 2.2.2-11 渐进交付模型

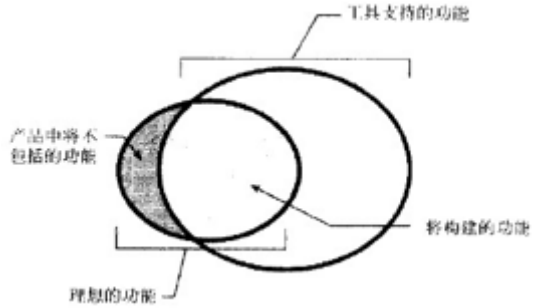

图 2.2.2-12 面向开发工具的设计模型

生命期模型的能力	纯瀑布	编码修正	螺旋	经过修改的瀑布模型	渐进原型
没有充分理解需求	差	差	好	介于好到一般之间	好
没有充分理解架构	差	差	好	介于好到一般之间	介于差到一般之间
开发高可靠性的系统	好	差	好	好	一般
开发带有极大的成长性的系统	好	介于差到一般之间	好	好	好
管理风险	差	差	好	一般	一般
可以强制执行预先定义的进度	一般	差	一般	一般	差
低管理费用	差	好	一般	好	一般
允许中途变更	差	介于好到差之间	一般	一般	好
提供给用户可视的进展情况	差	一般	好	一般	好
提供给管理者可视的进展情况	一般	差	好	介于一般到好之间	一般
需要极少的管理和开发经验	一般	好	差	介于差到一般之间	差

生命期模型的能力	阶段交付	渐进交付	面向进度的设计	面向工具的设计	商品软件
没有充分理解需求	差	介于一般到好之间	介于差到一般之间	一般	好
没有充分理解架构	差	差	差	介于差到好之间	介于差到好之间
开发高可靠性的系统	好	介于一般到好之间	一般	介于差到好之间	介于差到好之间
开发带有极大的成长性的系统	好	好	介于一到好之间	差	N/A
管理风险	一般	一般	介于一般到好之间	介于差到一般之间	N/A
可以强制执行预先定义的进度	一般	一般	好	好	好
低管理费用	一般	一般	一般	介于一般到好之间	好
允许中途变更	差	介于一般到好之间	介于差到一般之间	好	差
提供给用户可视的进展情况	一般	好	一般	好	N/A
提供给管理者可视的进展情况	好	好	好	好	N/A
需要极少的管理和开发经验	一般	一般	差	一般	一般

7、 面向客户开发

谁是客户？

对客户的理解取决于场合，可能是项目委托人，最终用户，市场人员或者老板。

现代软件开发非常关注客户的需求与期望，开发出合符产品规格的软件只是完成了一半工作，另一半是帮助客户配置出产品能够实现的功能，而实现这些功能所花费的时间通常远远多于确定纸面上的产品规格所需要的时间。

将自己站在客户的角度考虑问题是避免大量返工的最好方法。同时应该建立有效的客户沟通渠道，合理控制客户的期望值。

2.2.3 产品

在软件开发的四维中，最切实的维是产品维。对产品规模和产品特性的关注，意味着巨大的缩短计划进度的机会。削减了产品功能通常就可以缩短产品开发周期

1、 产品规模

产品规模是对开发进度影响最大的一个因素。构建软件所需的工作量的增长比产品规模的增长要快得多，并且增长是不成比例的，所以产品规模的缩小将大大提高开发速度。将中等规模的软件削减一半通常可以使工作负荷削减2/3。

2、 产品特性

产品的一些非功能性需求或额外关注点会影响设计的复杂度和构建的工作量，如对性能、稳定性、可维护性和可扩展性等要求很高的产品比没有这些特性要求的产品需要更长的开发周期。

2.2.4 技术

从使用低效的工具转为使用高效的工具是提高开发速度的快捷方法。选择有效的工具并管理好由此带来的风险也是提高开发速度的方法。

软件开发基本原则（二）—— 典型错误

大多数典型错误其表面都具有诱惑性，给人们一种诱人的前景，但通常却不能产生期望的结果。

“想挽救进度已经落后的项目吗？---- 给项目补充更多人员！”

下面分别按照人员、过程、产品和技术四个维度列出36个典型错误。

人 员

典型错误1：挫伤积极性

对人员不够关心和重视；过度的进度压力；缺乏激励；过分夸张的激励等。

典型错误2：人员素质低

人员能力欠佳，工作效率低，甚至做多错多。

典型错误3：对有问题的员工失控

不对有问题的人员采取措施是项目组成员对领导最常见的抱怨。

典型错误4：英雄主义

强调个人英雄主义会导致发生额外的风险，也会削弱在软件开发过程中多个角色的合作。

典型错误5：项目后期加入人员

盲目地在项目后期加入人手等于火上浇油。

典型错误6：办公室环境拥挤嘈杂

拥有安静、隐蔽办公环境的人员比工作在嘈杂、拥挤环境中的人员往往会有更好的工作业绩表现。

典型错误7：开发人员与客户之间发生摩擦

主要原因是缺乏沟通。这种摩擦耗费时间，它会转移客户和开发人员双方对项目工作的注意力。

典型错误8：不现实的预期

过高的期望值和主观的不切实际的设想。是导致开发人员和客户或项目经理之间的摩擦常见原因之一。

典型错误9：缺乏有效的项目支持

软件开发项目的许都方面都需要高层的支持，包括实际的计划、变更控制以及新型开发方法的采用等。缺乏有效的高层支持事实上注定了项目的失败。

典型错误10：缺乏各种角色的齐心协力

软件开发中所有主要人员必须齐心协力专注于项目，包括高层支持者、项目领导、项目成员、市场人员、最终用户、客户和任何项目介入者。

典型错误11：缺乏用户介入

没有用户早期介入的项目充满需求误解的风险，易受项目后期功能蔓延的威胁。

典型错误12：政治高于物质

“政治家”型项目强调“管理至上”，主要精力集中在他们与经理的关系上。将政治凌驾于结果之上对软件项目会造成极大伤害。

典型错误13：充满想象

闭上眼睛毫无理由地希望某事将像想象那样运作。很多软件开发问题都是由于充满想象造成的。

想象示例：

项目组不知道他们能不能按时完成项目，但他们认为如果每个人能更努力工作，并且不出现问题，他们应该能完成项目。

我们无需向客户演示最新的修改，我们确信这个效果是客户想要的。

项目组错过了一个里程碑好几天了，他们说会更努力工作赶上下一个里程碑，我想他们能够及时赶上的。

过 程

典型错误14：过于乐观的计划

定制过于乐观的项目计划相当于自己为项目失败画出了底线，导致缩短分析、设计等关键性前期开发活动；同时也向开发人员施加了额外压力，会长期对开发人员的自信心和生产率造成巨大伤害。

典型错误15：缺乏足够的风险管理

如果你不主动管理风险，风险随时会来找你，打乱你的开发计划。

典型错误16：承包人导致的失败

如果不对承包商加以认真管理，交付可能延期，并且质量难以保证。

典型错误17：缺乏计划

没有计划的项目就像飘荡在海洋中的小船，没人知道会飘到哪里。

典型错误18：在压力下放弃计划

很多项目组定制了计划，但遇到了麻烦时就放弃计划。项目失败的原因不是在于放弃计划本身，而是不能及时修订计划制定替代计划，并一头栽进编码和问题处理中。

典型错误19：在模糊的项目前期浪费时间

由于花在审批、预算等前期工作的时间过长，或需求无限循环等原因，导致压缩开发计划。项目前期节省几周或几个月时间比将开发计划压缩同样时间来得更容易、更廉价，风险也更少。

典型错误20：前期活动不符合要求

研究数据：

前期被跳过的活动或工作通常在后期会以10倍到100倍的代价来完成。如果一项工作在项目初期需要5小时完成，那么在项目后期你至少需要50小时才能完成它。 （Fagan 1976，Boehm and Papaccio 1988）

典型错误21：设计低劣

前期活动不符合要求的一个特殊情况就是设计低劣。高压环境导致设计缺乏周密思考往往导致设计低劣。

典型错误22：缺少质量保证措施

研究数据：

项目前期砍掉1天的质量保证活动，到项目后期就需要3到10天的处理代价。（Jones 1994）

典型错误23：缺少管理控制

缺少管理控制点就难以对项目的阶段和状态进行跟踪，因此不能知道项目是否按正常轨道前进。

典型错误24：太早或过于频繁的集成

在构建未完全锁定时，进行过早的集成或额外的集成不利于产品，它仅仅是在浪费时间，延长进度。

典型错误25：项目估算时遗漏必要的任务

训、公司和部门会议，技术评审会议等活动在项目估算时通常被遗漏。

典型错误26：追赶计划

当进度落后时不重新检查任务和调整计划，而是简单地决定把进度赶上来。

另一种情况是，当产品出现变更却没有做相应的计划调整

典型错误27：鲁莽编码

没有足够的需求基础和清晰的架构设计而进行“边编码边修改”造成太多重复工作和返工，这样的做法使项目大多以失败告终

产 品

典型错误28：需求的镀金

项目的产品要求要求比实际需求多得多的产品特性或复杂功能，却又不给进度计划分配足够的时间。

典型错误29：功能蔓延

在整个开发过程中，项目平均会有25%的需求变更，对软件计划至少有25%的影响。如果任由客户不断提出新需求，项目就会一直都做不完

典型错误30：开发人员的镀金

开发人员着迷于新技术，有时渴望在自己的产品中使用这些技术，而不管那些技术是否适合或是否会对系统整体造成破坏。

典型错误31：又推又拉的交易

管理者批准进度落后的项目顺延，但同时又给这个项目加入新任务。

典型错误32：研究导向的开发

软件开发进度是完全有理由可以预测的，而软件研究进度甚至理论上都是不可预知的，不能采用像软件研究一样的工作方式引导项目开发。

技 术

典型错误33：银弹综合症

过于相信某些技术宣传（某种开发过程、某种程序设计方法、某种开发语言），缺少在特定环境下使用这些工具的必要信息。当团队寄望利用他们来解决进度问题时，不可避免会失败的。

典型错误34：过高估计了新技术或方法带来的节省量

无论采用多少新工具或方法，以及这些工具或方法有多好，他们很少能够大幅度提高生产率。软件开发由多个任务组成，特定的工具或方法只会可能提高特定任务的生产效率。同时，它们所带来的效率常常被学习它们所花费的时间抵消了。

典型错误35：项目中间切换工具

在项目中间更换工具时，伴随使用新工具而带来的人员学习和掌握的过程、重复的工作、不可避免的错误等会彻底抵消它所带来的益处。

典型错误36：缺乏自动的源代码控制手段

缺乏自动的源代码控制容易造成版本冲突、历时版本丢失、更新丢失等一系列问题，并浪费大量的时间处理这些问题。

软件开发基本原则（三）—— 基本原则

“回顾一下被选为‘最佳项目’的十个软件项目，如果说有所发现的话，那就是——最佳的项目一定是建立在最佳的软件开发基础之上的。我们都知道软件开发基础对于优秀软件的作用，但差别在于大多数软件的基础薄弱，这样不可避免地使自己陷入麻烦之中”

（Bill Hetzel 1993）

本章的范畴只限定在确定软件开发的基本原则，解析他们是如何影响开发计划的，同时提供参考信息。

本章书把软件开发基本原则实践分为三类：管理实践，技术实践和质量保证实践。

管理的基本原则

管理原则由以下几部分组成：

判定产品规模（包括功能、复杂度和其它产品特性）

根据产品规模分配资源

制定资源计划

监控、引导资源以保持项目方向不会偏离

1. 项目估算和进度安排

一个运行良好的项目一般通过三个基本步骤来定制软件开发进度表。

首先估算项目规模大小

然后估算完成这样规模的项目需要付出的代价

最后基于这种估算定制项目进度计划

如果估算不准确就会降低开发效率，所以说估算和项目进度计划是软件开发的基础。精确的估算时进行有效规划的必要前提，而有效的规划又是有效开发的必要条件。

2. 计划编制

计划一个软件项目应该包括以下活动：

- 项目估算和时间进度
- 确定项目需要多少人参与、需要什么样的技能、合适加入以及具体人选
- 确定项目组的运作方式
- 确定项目采用的生命周期模型
- 管理风险
- 确定项目策略（例如：如何控制产品的特色，是否需要购买部分产品组建）

3. 跟踪

跟踪是一个基本的软件管理行为。如果不跟踪一个项目就不能管理它，就不会知道计划是否被贯彻执行了，也不会知道下一步该做什么，同时也无法监控项目风险。有效的跟踪能使项目组在还有时间做点什么来改正错误的时候，尽早发现进度表上的问题。

制定了一个项目计划就要跟踪检查它是否在按计划进行，包括对它的进度、费用和质量等目标的检查。典型的管理级跟踪控制包括：任务列表、进展状况会议、进展报告、里程碑审查、预算报告以及走查管理等。典型的技术级跟踪包括：技术审查、技术审计和标志着里程碑是否完结的质量关口等。

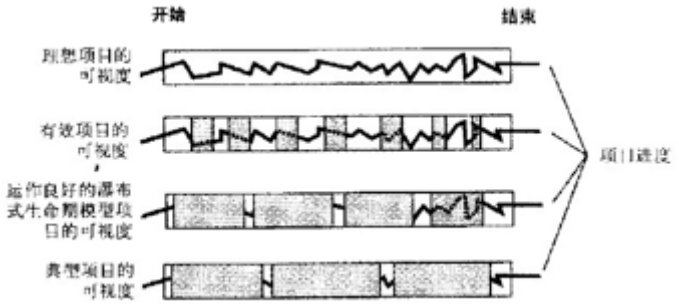

图 4.1.3-1 不同类型项目的可视度

4. 量度

老板问你：“我们能够在9各月内开发出这个产品吗？”——你怎么回答？！

为了使开发更有效，你需要具备软件量度方面的基本知识。你需要了解收集数据的尺度基准，包括应该要收集什么数据，如何获得这些数据。你还需要具备用来分析状态，质量和生产率的详细基准方面的知识。任何公司想要进行快速的开发就要收集这些基本的尺度，这样才可以知道他们的开发速度是否正在改善或后退。

技术基本原则

1984年有关“现代程序设计实践方法——技术的基本原则”的一份研究，详细论述了不使用这些基本原则就不可能具有高的生产率的内容。图4.2-1展示了研究的结果。

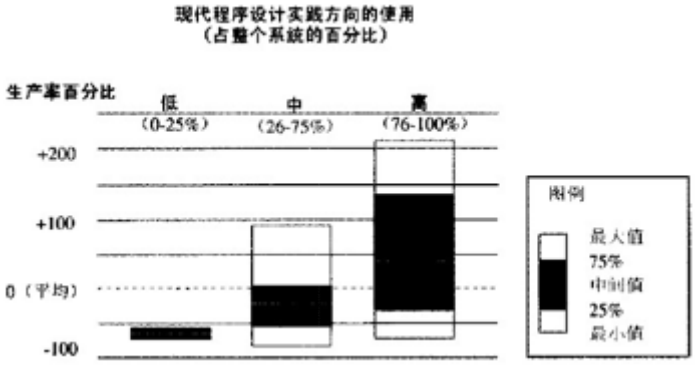

图 4.2-1 生产率与“现代程序设计实践方法”的关系

（不广泛地使用“现代程序设计实践方法”就无法具有高的生产率）

很显然，不采用现代程序设计实践方法的项目不可能具有高的生产率。但技术基本原则的应用，就其本身而言，不足以创造高的生产率。一些项目使用了大量现代程序设计实践方法，但是仍旧和那些完全没有使用该方法的项目具有一样的生产率。因此，注意技术的基本原则是很必要的，但却不足以达到快速开发的目的（例如犯了某些典型错误）。

1. 需求管理

研究数据：

典型项目平均会经历25%的需求变化，从而至少产生25%的额外费用和时间。

一项针对8000多个项目的调查显示，导致项目推迟发布、超出预算、功能比预期减少的最重要的三个原因——缺乏用户的介入、不完善的需求分析和用户不断改变需求，都和需求管理有关。（Standish Group1994）

一项软件工程研究所的调查也有相同的结论：超过半数的项目都遭遇过不充分的需求管理的麻烦。(Kitson and Masters 1993）

需求管理就是收集需求，把需求记录成文档、电子邮件、用户界面串连脚本、可实现的原型等形式，然后依此来跟踪设计和编码，并随时管理、修改需求，以适应项目后续的过程。

成功的需求管理取决于了解足够的不同的实践经验，以便能够为特定项目选择可借鉴的一种。

需求管理的基础：

需求分析方法：包括结构分析、数据结构分析和面向对象分析

系统建模实践：如类图表、数据流图表、实体关系图表、数据字典符号和状态跃变图表

沟通实践：如联合应用开发、用户界面原型和常规会谈实践等

需求管理和其它生命周期类型的关系：如渐进原型、阶段交付、螺旋模型、瀑布模型和编码修正

需求管理在两个方面对开发速度发挥着巨大的调节作用：

首先，正规的需求管理中，需求收集往往比其他软件开发活动完成得要从容些。如果能加快需求步伐而不伤害质量，就可以缩短总的开发时间。

第二，正确地把需求摆在首位，往往要比被动地这样做所花的时间少得多。一些需求管理实践基本原则能够减少需求变化的数量，其他开发实践的基本原则能够减少因需求改变而产生的费用。

想象一下，如果把需求变化从25%减少到10%，同时把每个需求变化导致的费用减少5%-10%，那么综合的效果会怎样呢？

2. 设计

研究数据：

一个设计上的错误如果到系统测试时才被发现，那么花费的修补时间要比它在设计阶段时被发现所花费的时间多10倍。(Dunn 1984）

设计是系统构建、项目进度计划、项目跟踪和项目控制的基础。

体系结构和设计的基本原则：

主要设计风格：如面向对象设计、结构化设计和数据结构设计

基础设计概念：如信息隐藏、模块化、抽象、封装、聚合、耦合、层次、继承、多态、基本算法和基本数据结构

对典型挑战性事件的标准设计：包括异常处理、国际化、本地化、便携性、字串存储、输入输出、内存管理、数据存储、浮点算法、数据库设计、性能和复用

对特殊领域应用程序设计的独有思考：例如财务应用、科学应用、嵌入式系统、实时系统、安全性要求高的软件等

架构安排：如子系统组织、分层结构、子系统通信方式和典型的系统架构

设计工具的使用

3. 构建

当构建开始时，项目成功与否大多就已经注定了。需求管理和设计对开发进度计划的调节作用比构建的调节作用大得多，这意味着小的波动可以导致进度的重大变化。

尽管构建是一个低层次的活动，但是它确实可以提供许多机会进一步改进时间效率低的任务或优化一些任务。例如，花时间对那些无需镀金的功能进行镀金；调试那些无用的多余代码，或者对那些并不知道是否需要优化的片段尽心优化。

构建的基本原则：

- 编码实践：如变量和函数命名、版面布局和文档
- 数据相关概念：如作用范围、持续和捆绑时间
- 特定数据类型的使用方针：如通用基础数据类型、枚举、常量、数组和指针
- 控制相关的概念：如组织整齐的代码、条件的使用、循环的控制、复杂度的控制、特殊控制结构的使用（goto、return、递归）
- 断言和其它以代码为核心的错误检测方法
- 对例程、模块、类和文件代码打包的规则
- 单元测试和调试实践
- 集成策略：如增量式集成、大爆炸式集成和渐进开发
- 代码优化策略和实践
- 与所使用的特定编程语言相关的其他事情
- 使用构建工具：如编译环境、群组工作支持、源代码控制、代码库和代码生成器

4.软件配置管理

软件配置管理(SCM)是管理项目成果的一种实践方法，能使项目在全程中保持一致的状态。SCM包括评估变更、跟踪变更、处理多版本，以及在不同时间保存项目成果的备份等实践。

质量保证基本原则

很多公司现阶段开发软件都有一定的不当之处，使得他们的开发时间比需要的长。在调查了4000个软件项目后，Capers Jones 递交报告说，糟糕的质量是进度被拖延的最普遍的原因之一。他还说，中途被取消的项目中，大约有一半是由于其糟糕的质量。（Jones 1994）

一项软件工程研究所的调查显示，大约有60%的公司遭受着不适当的质量保证体系的困扰。（Kitson and Masters 1993）。

在过大的时间压力下发布的产品，其错误率是正常情况下的4倍。有进度问题的项目经常是在进行艰苦的工作而不是轻松活跃的工作，关注质量被认为是有些奢侈。但其结果却是项目进展缓慢，并陷入更深的进度问题中。（Jones 1994）

重做有缺陷的需求、设计和编码通常花费整个软件开发成本的40%—50%。（Jones 1968b，Boehm 1987a）

最糟糕的情况下，在运行中的软件项目只修改一次软件需求问题的花费通常是在需求分析阶段所花时间的50到200倍。（Boehm and Papacio 1988）

大约60%的错误通常在设计阶段就存在了。（Gilb 1988）

如果可以尽早地预防并修正漏洞，可以节省大量时间，在进度的安排上占了先机。

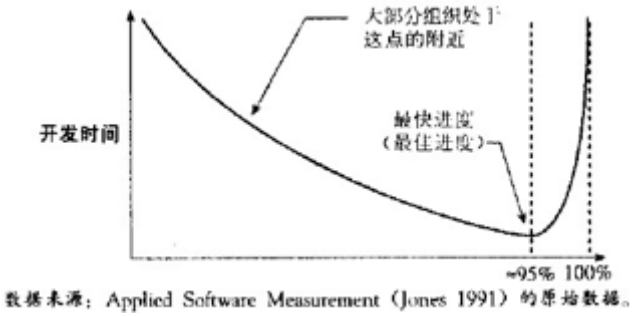

图 4.3-1 错误率和开发时间的关系

（大多数情况下，具有低错误率的项目同时实现了最短日程的目标）

1. 易错模块

易错模块是那些容易存在或多或少漏洞的模块。

研究数据：

IBM 的 IMS 项目中，57%的漏洞存在于7%的模块中。（Jones 1991）

程序中20%的模块包含了80%的错误。（Boehm 1987b）

高错误率的模块开发起来要比其它模块更加昂贵和耗时，如果普通模块开发每个功能点要花费$500—$1000，那么易错模块每个功能点就要花费$2000—$4000。（Jones 1994）

易错模块往往比系统中的其它模块更复杂，缺乏结构化，或者不同寻常的庞大，并且往往在背负压力下开发，往往没有被完全测试过。对软件开发特别重要的一个方面就是对易错模块的质量保证。

2. 测试

最寻常的质量保证实践就是毋庸置疑地进行测试，两种基本的测试方法

- 单元测试：程序员检查他自己的代码是否工作正常
- 系统测试：独立测试员检查整个系统是否如期望的那样正常运行

研究数据：

测试的有效性差异是巨大的。单元测试可以找到程序中10%—50%的漏洞；系统测试可以发现20%—60%的程序漏洞。加在一起，累积的漏洞检测率经常少于60%。（Jones 1986a）

剩下的错误要么通过其它的查错技巧（如技术回顾）发现，要么就是在产品发布后被最终用户发现。

平衡测试和快速开发的最佳办法是在坏消息出现之前做好计划——设置对坏消息的测试，尽早地发现问题。

3. 技术回顾

技术回顾包括在需求、设计、编码和测试等事件中用于查错的所有类型的回顾。回顾在形式上和效果上是多样的，它在开发速度上比在测试上扮演更重要的角色。下面讲述最常见的几种回顾。

1）走查

走查是指任何两个以上的开发人员以增进软件质量为目的所召开的回顾技术工作会议。走查可能是最平常的非正式回顾，走查可以在写设计说明书时，设计和编码完成之前就发现漏洞。

研究数据：

走查可以发现30%—70%的程序漏洞。（Myers 1979，Boehm 1987b，Yourdon 1989b）

2）代码阅读

代码阅读时比走查更正式些的回顾方式，但仅适用于代码。代码阅读时，写这段代码的程序员把代码清单交给两个或更多的审阅者审阅，审阅者阅读代码，并把发现的错误报告给编写者。

研究数据：

NASA 的软件工程实验室的一项研究发现：代码阅读能发现的漏洞是测试时能发现的漏洞的两倍。（Card 1987）

3) 检查

检查是一种正式的技术回顾，它被认为是在整个项目中最具效率的查错方式。使用检查的方法，开发人员需要接受检查的特殊训练，并且在检查中扮演重要的角色。在检查会议之前“仲裁人”发布产品要被检验评估的消息和检查列表，“审阅人”在会议前检查程序，在检查会议上“作者”通常要解释要检验的东西，“审阅人”鉴别错误，“书记员”记录错误。在会后“仲裁人”写一份报告说明每个漏洞和处理办法。

在项目中可以使用检查对需求分析、用户界面原型、设计、编码及其他认为的过程查错。

研究数据：

检查可以查出程序中60%—90%的漏洞，这点比走查或测试要好。因为可以在开发的早期应用，因此，检查方法被证明可以节约10%—30%的开发时间。（Gilb and Graham 1993）

一项对大型程序的调查结果显示，在检查上每花1小时，就可以避免在维护上33个小时的花费。检查比测试有效20倍以上。（Russel 1991）

软件开发基本原则（四）—— 风险管理

1988年，Peat Marwick 针对600家成功公司的调查结果显示，35%的公司有过软件项目失控的经历。（Rothfeder 1988）

1982年，Allstate 公司宣布其公司运营全部要实行自动化。他们启动了一个将耗时5年投资800万美元的大型项目，而在花费了6年和1500万美元后，Allstate 公司重新调整了目标和最终期限，重新调整后的预算大约1亿美元。

1988年，Westpac Banking 公司决定重新设计他们的信息系统。他们做了5年、8500万美元的计划。3年后，在花费了1.5亿美元却依然收效甚微时，Westpac Banking 公司为了减少损失，取消了这个项目，并为此裁员500人。（Glass 1992）

从项目管理的角度来看，有五大硬性知识领域：范围管理、进度管理、成本管理、质量管理和风险管理。风险会出现在前面四个领域的各个过程中，只有有效地消除可能发生的危险因素，才能确保项目顺利推进。项目的风险贯穿于整个项目过程，因此整个项目的生命周期都应该坚持有效的风险管理。

根据风险的内容，可以把风险归为以下几类：

- 产品规模风险：与软件的总体规模相关的风险
- 商业影响风险：商业风险影响到软件开发的生存能力
- 客户特性风险：与客户的素质以及开发者和客户沟通能力相关的风险
- 过程定义风险：与软件过程定义相关的风险

- 开发环境风险：与开发工具的可用性及质量相关的风险
- 技术风险：技术风险是指在设计、实现、接口、验证、维护、规约的二义性、技术的不确定性、陈旧的技术等方面存在的风险
- 人员数目及经验带来的风险：与参与工作的软件工程师的总体技术水平及项目经验相关的风险

软件项目的风险主要体现在四个方面：需求、技术、成本、进度。风险管理是一个相当重要的话题，但涉及的问题太多，很难在本章中全部囊括，本章主要讲述进度相关的风险管理。

风险管理要素

软件风险管理就是在风险成为影响软件项目成功的威胁之前，识别、处理并消除风险。可以在几个层次上定位风险管理：

1、危机管理 — 救火模式，即在风险已经造成麻烦后才着手处理它们

2、失败处理 — 觉察到了风险并迅速做出反应，但只是在风险发生之后

3、风险环节 — 事先制定好风险发生后的补救措施，但不做任何防范措施

4、着力预防 — 将风险识别与风险防范作为软件项目的一部分加以规划和执行

5、消灭根源 — 识别和消除可能发生的风险的根源

本章描述如何定位第4、5个层面上的进度风险管理。

总体来讲，风险管理由风险评估和风险控制组成：

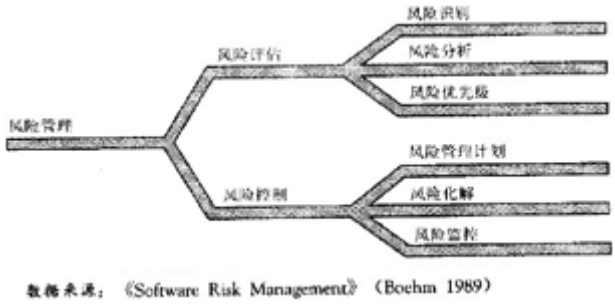

图 5.1-1 风险管理由风险评估和风险控制组成

1. 风险评估

- 风险识别：建立一个潜在破坏项目进度的风险列表
- 风险分析：评估每一个风险出现可能性及其影响，判定风险的级别
- 风险优先级：按风险影响大小排出一个风险优先级列表，这个列表将作为风险控制的基础

2. 风险控制

- 风险管理计划：定制一个应对每个重要风险的方案，同时应确保每一个单独的风险管理计划相互之间以及与项目计划之间保持一致
- 风险化解：执行每一个重要风险所对应管理计划
- 风险监控：对解决风险的过程进行监控。还包括：识别新的风险，并将其加入到正在进行的风险管理进程；在风险列表中移除已经化解的风险等工作

风险识别

1. 最常见的进度计划风险

- 功能蔓延
- 需求镀金或开发人员镀金
- 质量不稳定
- 计划过于乐观
- 设计欠佳
- 银弹综合症
- 研究导向的开发
- 人员薄弱
- 承包人导致的失败
- 开发人员与客户之间发生摩擦

2. 进度计划风险列表

下面列出了详尽的可能对软件进度有负面影响的潜在风险。除了这里所列出的风险，大多数项目都有其特定的风险，如：

“Joe 要退出项目组，除非可以允许他带自己的小狗来上班，而管理层还没有决定是否同意他这样做”—— 这样的风险就要靠自己识别了！

潜在的进度计划风险： （资料来源：Gilb 1988、Boehm 1989 Pressman 1993、Thomsett 1993、Jones 1994）

类 型	风 险
1、计划编制	1) 计划、资源和产品定义全凭客户或上层领导口头指令，而且不完全一致 2) 计划是优化的，是“最佳状态”（但不现实，只能算是“期望状态”） 3) 计划忽略了必要的任务 4) 计划基于使用特定的小组成员，而那个小组成员其实指望不上 5) 在限定时间内无法建成已定规模大小的产品 6) 产品规模比估计的要打（代码行数、功能点或与前一产品规模的百分比） 7) 工作量大于估计数（代码行数、功能点、模块等） 8) 进度已经拖延的项目在重新评估时过于优化或忽略项目历史 9) 过度的进度压力造成生产力下降 10) 目标日期提前，但没有相应地调整产品范围或可用资源 11) 一个任务的延时导致相关任务的连锁反应 12) 涉足不熟识的产品领域，花费在设计和实现上的时间比预期要多

2、组织和管理	1）项目缺乏一个用凝聚力的最高领导人 2）由于前期乏力，项目长时间被搁置 3）解雇和削减开支导致项目小组能力下降 4）仅由管理层或市场人员进行技术决策，导致计划进度延长 5）低效的项目组织结构降低生产率 6）管理层审查或决策的周期比预期的时间长 7）预算削减打乱项目计划 8）管理层作出了打击项目组积极性的决定 9）非技术的第三方的工作比预期延长（预算批准、设备采购、法律审查等） 10）计划性太差，无法达到期望的开发速度 11）项目计划由于压力而放弃，导致开发混乱低效 12）管理层强调英雄主义而忽略客观确切的状态报告，错过发现和改正问题的机会
3、开发环境	1）设施没有及时到位 2）设施到位但不配套 3）设施拥挤、杂乱或破损 4）开发工具没能及时到位 5）开发工具不如期望的那样有效 6）开发工具的选择不是基于技术需求，不能提供计划要求的性能 7）开发人员需要长时间创建工作环境或切换新开发工具 8）新开发工具的学习周期比预期的长，内容繁多
4、最终用户	1）最终用户坚持新的需求 2）最终用户对于最后交付的产品不满意，要求重新设计或重做 3）最终用户不买进项目产品，无法提供后续支持 4）最终用户的意见未被采纳，造成产品无法满足最终用户的期望而必须重做
5、客户	1）客户坚持新的需求 2）客户对规则、原型和规格的审核和决策周期比预期的长 3）客户没有或不能参与规划和原型审查工作，导致需求不稳定和耗时的变更 4）客户沟通或答复的时间比预期长 5）客户坚持技术决策而导致进度计划延长 6）客户对开发进度管理过细，导致实际进展缓慢 7）客户提供的组件无法与开发的产品匹配，导致额外的设计和集成工作 8）客户提供的组件质量欠佳，导致额外的设计、测试、集成和客户关系管理工作 9）客户要求的支持工具和环境不兼容、性能差或功能不完善，导致生产率降低 10）客户不接受交付的软件，尽管它满足合同的条款要求 11）客户期望的开发速度无法达到
6、承包商	1）承包商没有按承诺交付组件 2）承包商递交的组件质量低下无法满足要求，必须再花时间改进 3）承包商没有买进项目开发需要的工具，进而无法提供需要的性能水平
7、需求	1）需求已经成为项目的基准，但变化仍在继续 2）需求定义欠佳，但进一步的定义会扩展项目的范畴 3）添加额外的需求 4）需求定义含糊的部分需要的处理时间比预期多
8、产品	1）错误发生率高的模块需要比预期更多的设计、实现和测试工作 2）校正质量低下的产品需要比预期更多的设计、实现和测试工作

	3）在一个或多个新兴领域推过产品技术使得进度延长或不可预期
	4）由于软件的功能错误使得需要重新设计和实现
	5）开发额外不需要的功能（镀金）延长了计划进度
	6）要满足产品规模的要求，需要比预期长的时间，包括重新计划、设计和实现
	7）严格要求与原有系统兼容，需要进行比预期更多的计划、设计、实现和测试
	8）要与不受项目组控制的其他系统集成，导致无法预料的设计、实现和测试工作
	9）要求在不同操作系统或软硬件环境下运行将花费比预期更长的时间
	10）在不熟识或未经检验的软件环境中运行，产生未预料到的问题
	11）在不熟识或未经检验的硬件环境中运行，产生未预料到的问题
	12）开发一些全新的功能模块比预期花费更长时间
	13）依赖未成熟的技术，使得进度延长或不可预期
9、外部环境	1）产品依赖政府的政策或制度，而政策或制度不可预期
	2）产品依赖草拟中的技术标准，而最后的标准不可预期
10、人员	1）招聘人员所花时间比预期长
	2）培训、工作许可证或其他项目的收尾等作为先决条件的任务不能按时完成
	3）开发人员和管理层之间关系不佳导致决策缓慢影响进度
	4）项目组成员没有全身心投入项目，进而无法达到要求的产品质量水平
	5）缺乏激励措施，士气低下，降低生产力
	6）缺乏必要的规范，增加了工作失误几率和重复工作
	7）某些人需要更多时间适应新的软件工具和环境
	8）某些人需要更多时间适应新的硬件工具和环境
	9）项目结束前，成员调离团队或离职
	10）项目后期加入新开发人员，额外的培训和沟通降低现有成员的生产率
	11）项目成员不能有效地一起工作
	12）项目组成员间有冲突，导致沟通不畅、设计欠佳、接口错误和额外的重复工作
	13）有问题的成员没有调离项目组，损害了其他成员的积极性
	14）项目的最佳人选没有加入项目组
	15）项目的最佳人选已加入项目组，但因政治或其它原因未能合理使用
	16）没有找到项目急需的，具有特殊技能的人
	17）关键人物只能兼职参与
	18）项目人员不足
	19）任务的分配与人员技能不匹配
	20）人员工作的进度比预期的慢
	21）项目管理人员怠工导致计划的进度失效
	22）技术人员怠工导致工作遗漏和质量低下
11、设计和实现	1）设计过于简单，无法确定主要事件，并导致重新设计和实现
	2）设计过于复杂，导致一些不必要的工作，影响实现效率
	3）设计质量低下，导致重复设计和实现
	4）使用不熟识的方法和技术，导致额外的培训时间
	5）使用低级语言开发产品，导致生产率比预期低
	6）一些必要的功能无法使用现有的代码或库实现，必须采用新库或重新实现
	7）代码和库质量低下导致需要额外的测试、错误修正或重做
	8）过高估计了工具对计划进度的节省量
	9）独立开发的模块无法有效集成，需要重新设计或重做

1）大量的书面工作导致进度比预期慢
2）进度跟踪不准确，导致无法预知项目是否已落后于计划进度
3）前期的质量保证行为不真实，导致后期重复工作
4）质量跟踪不准确，导致无法得知影响进度的质量问题
12、过程 5）不够正规（缺乏对软件开发标准和策略的遵循），导致沟通不足和质量问题
6）过于正规（教条地遵循软件开发标准和策略），导致过多耗时于无用的工作
7）向管理层撰写进度报告占用开发人员的时间比预期多
8）风险管理不够重视，导致没有发现重大的项目风险
9）软件项目风险管理花费的时间比预期多

风险分析

1. 风险暴露量

一种很有用的风险分析方法就是风险暴露量。风险暴露量就是风险发生的概率乘以损失的程度。举例来说：如果你认为“完成需求分析比原计划延长4周的概率是30%”，那么风险暴露量就是4周*30%=1.2周。

风 险	发生概率	损失程度（周）	风险暴露量（周）
计划过于乐观	50%	5	2.5
由于要完全支持自动从主机更新数据而造成的额外需求	5%	20	1.0
由于市场变化而增加额外的功能	35%	8	2.8
图形格式子系统接口不稳定	25%	4	1.0
设计欠佳——需要重新设计	15%	15	2.25
项目审批超过预计时间	25%	4	1.0
设施未能及时到位	10%	2	0.2
为管理层撰写进程报告占用开发人员的时间比预期的多	10%	1	0.1
承包商的图形格式子系统推迟交付	10~20%	4	0.4~0.8
新的编程工具没有节省预期的时间	30%	5	1.5

表 5.3.1-1 风险暴露量

1）评估损失程度

损失程度常常比发生概率更容易估算，在表5.3.1-1中，完全可能很精确地估计出由于增加“完全支持自动从主机更新数据”而增加的研发时间是20个月。如果有时损失程度不容易直接估算出来，还可以把损失分解为更小的部分分别进行估算，之后将各个小的独立评估结果累加得出合计估算值。

2）评估发生概率

估算发生概率比估算损失程度更具主观性。有许多实践方法可以提高主观评估的精确度，例如：

由最熟识系统的人评估每个风险的发生概率，然后保留一份风险评估审核文件

每个人对风险进行独立评估，然后讨论评估的合理性，直到达成共识

使用“形容词标准”，如非常可能、很可能、可能、或许、不大可能和不可能等，然后把口头评估转换成量化的评估

2. 项目的延期和缓冲

由于我们只谈论进度风险，所以可以累加所有的风险暴露量来得到项目总风险暴露量。这个项目的总风险暴露量为12.8~13.2周，这意味着，如果不做任何风险管理的话计划可能要延期12.8~13.2周。如果例子中的项目历时25周，那么超出预计值12.8~13.2周就很显然要进行风险管理了。

风险优先级

项目通常花费80%的金钱解决20%的问题，所以风险管理的重点是关注那最重要的20%的部分。（Boehm 1989）

如果只关注进度计划风险而不是关注所有的风险，确定风险优先级的工作就变得比较容易了。在风险评估表中按风险暴露量从大到小排序看看会得到什么结果：

序 号	风 险	发生概率	损失程度（周）	风险暴露量（周）
1	由于市场变化而增加额外的功能	35%	8	2.8
2	计划过于乐观	50%	5	2.5
3	设计欠佳——需要重新设计	15%	15	2.25
4	新的编程工具没有节省预期的时间	30%	5	1.5
5	图形格式子系统接口不稳定	25%	4	1.0
6	由于要完全支持自动从主机更新数据而造成的额外需求	5%	20	1.0
7	项目审批超过预计时间	25%	4	1.0
8	承包商的图形格式子系统推迟交付	10~20%	4	0.4~0.8
9	设施未能及时到位	10%	2	0.2
10	为管理层撰写进程报告占用开发人员的时间比预期的多	10%	1	0.1

表 5.4-1 排序后的风险暴露量

排序后的风险评估表实际上就形成了一个粗略的风险优先级列表。如果能成功地处理风险列表中的前5个风险，就有希望将超出预期计划的时间减少10.05周，如果能成功地处理后5个风险，则只能将超出预期计划的时间减少2.7~3.1周。一般情况下，你的时间最好花在风险列表中靠前的风险上。

表5.4-1的排序只是粗略的，你可能想将损失更大的风险排在优先级更高的位置，确保他们不会发生（如：上表第6个风险）。

你也可以把一些关联的风险排在比各自优先级更高的位置，如表5.4-1中第5和第8个风险。让承包商开发图形格式子系统接口，其组合风险要远大于它们各自的风险。

这里确定的风险优先级是比较粗糙的，因为用于确定风险的数据都是估计的，因此优先级本身

也就是个相对主观的值。所以，对风险的客观公正态度，是风险管理必要的组成部分。

风险管理计划

编制风险管理计划的重点是制定一个计划，以处理风险分析中确定的高优先级风险。风险管理计划可以简单地理解为一段一段的风险管理描述，如：每个风险的起因，表现形式，可能发生的时间、地点，为什么发生以及怎样发生等。风险管理计划同时也包含：监控风险，处理新风险，关闭已化解的风险等计划内容。

风险化解

特定风险的化解在许多情况下都决定于特定风险本身，下面列出一些化解风险的一般方法：

1、 避免风险

不要做冒险的活动。例如，你的项目进度比较紧，有一个工具提供商声称它的工具能提高你的开发速度，但是你的团队成员从来没使用过这个工具，这时你就不应该冒险采用这个工具。首先该工具不一定像它声称的那样好，其次对工具的学习和熟识需要一个过程，如果冒险用它很可能会得不偿失。

2、 转移风险

有时，项目某部分的风险对项目另一部分来说却不是风险，因此可以将它转移到另一部分。例如，可以负责大部分的系统设计工作，但不要承担不熟识的部分的设计工作，让比较有把握的人负责设计这部分。

3、 购买风险相关信息

如果不能确切知道风险的严重性，可以做一些调研。也可以请外面的咨询顾问帮你进行评估。

4、 消除产生风险的根源

例如，系统中某部分的设计可能面临严重的问题，可将其作为一个研究项目彻底重新设计。

5、 接受风险

如果风险的后果较小，而处理它的成本太高，那么可以接受这个风险，不做任何处理。

6、 发布风险

让上级领导、市场人员、客户知道有关的风险以及可能的后果。这样，即使风险真的发生了，他们也不会感到太惊讶。

7、 控制风险

定制计划应对风险无法化解的情形。例如分配额外的资源来测试设计中令人担心的那部分系统，

并留出额外时间处理测试发现的问题。

8、 记住风险

总结项目相关的风险及其处理办法、处理效果等，为未来的项目建立一组风险管理计划。

风 险	控制方法
1、功能蔓延	a) 使用基于用户的实践 b) 使用增量开发实践 c) 控制功能集 d) 采取针对变更的设计
2、需求镀金或开发人员镀金	a) 修正需求 b) 时间锁定开发 c) 控制功能集 d) 使用舍弃原型实践 e) 基于进度表的开发
3、质量低劣	a) 给 QA 留出时间，注重质量保证基础
4、计划过于乐观	a) 采用多估算实践 b) 多个估算员和自动估算工具有原则地进行谈判 c) 基于进度表的开发 d) 使用增量开发实践
5、设计低劣	a) 要有清晰的设计活动和足够的设计时间 b) 进行设计检查
6、银弹综合症	a) 要有一定的生成率要求 b) 建立软件量度计划 c) 建立软件工具库
7、研究导向的开发	a) 不要试图进行研究的同时强调加快开发速度 b) 使用基于风险的生命周期模型警惕地进行风险管理
8、人员薄弱	a) 招募顶尖人才 b) 项目开始前招聘或预定关键成员 c) 培训 d) 团队建设
9、承包商失败	a) 检查参考资料 b) 外包前分析承包商能力 c) 积极管理承包商
10、开发人员与客户发生摩擦	a) 采用面向进度的实践

表 5.6-1 常见进度计划风险的控制方法

风险监控

由于风险在项目推进过程中会增强或减弱，所以需要对风险进行监控，检查每个风险的化解程度，关闭完全化解的风险，添加新产生的风险。

1、 前十风险列表

最有效的风险监控手段之一就是建立前十风险列表，该表包含每个风险当前的级别、以前的级别、已经上表的次数和上次审核后风险化解的步骤等。

前十风险列表最有意义的方面是促使你定期查看和思考这些风险，并对重要的变化给予警告。

前十风险列表的示例：

本周排序	上周排序	已上列表周数	风险	风险化解进展
1	1	5	功能蔓延	采取分阶段交付的方式；需要对市场人员和最终用户解释
2	5	5	设计低劣——要求重新设计	按已定规范设计，并请专家按规范审核
3	2	4	测试领导还未到岗	优秀后选人员已经分派了工作，等待主管负责人分配人员
4	7	5	图形格式子系统接口不稳定	图形格式接口设计计划前移；设计还未完成
5	8	5	承包商开发的图形格式子系统延迟交付	约见有经验的合同联络人，要求承包商指派正式的联络人
6	4	2	开发工具延迟交付	7个工具中已交付5个，采购小组已将余下的工具列为高优先级
7	—	1	项目经理审核周期变长	按规范评估
8	—	1	客户审核周期变长	按规范评估
9	3	5	计划过于乐观	按计划准时完成第一阶段里程碑
10	9	5	增加完全支持自动从主机更新数据的功能	研究手动更新的可行性，参考功能蔓延风险
—	6	5	设计负责人的时间花在以前的项目上	以前的项目组已经转移到其他办公室

表 5.7-1 前十风险列表

2、 中间检查

虽然前十风险列表可能是最有效的风险监控手段，但每个项目也应该包括中间过程检查。在完成每个主要里程碑后进行一次小规模的检查是非常有益的实践。

3、 风险官员

有时任命风险官员很有用，风险官员的职责就是对项目风险提出警告，防止项目经理和开发人员忽视计划中的风险管理。