第 1 章 软件工程概论

当前，人类社会的发展进入了电子化时代，而电子化时代的一个重要标志则是计算机已经应用到人类生活的各个方面，吃、穿、住、行无一能够离开计算机的帮助，而计算机软件则是使计算机应用更加普及的一个助推剂，天上的飞机应用计算机软件进行控制与操作，水下的潜艇需要计算机软件进行方位的辨识和测量；工人生产、农民耕种、学生学习、军人练兵、商人经商，各行各业的人们都在借助于计算机软件进行学习和工作。开发高质量、高性能的计算机软件成为许多人、许多企业乃至许多国家的迫切需求。

软件工程就是为了研究软件本身而兴起的一门重要学科，它不仅要研究软件的开发方法、开发的周期特征，还要研究软件过程的质量控制、风险控制以及软件整体过程的管理与控制等。本章将从软件工程的起源、软件工程的研究内容以及软件工程的框架等方面对软件工程进行综合介绍。

1.1 软 件 工 程

当软件已经应用于社会生活的各个方面，当软件已经成为一种经济和社会发展的巨大驱动力时，没有人怀疑软件对世界的影响力，也没有人怀疑软件正在使人们的生活变得舒适和美好。从性能、功能、服务以及影响等各个方面看，今天的软件和过去相比已经发生了巨大的改变。软件本身不仅是一个产品，能够带来需要的各种信息，如商业信息、外部世界的信息，同时也提供了获取各种信息的手段；相对于早期程序员的单兵作战，如今的软件开发已经均由软件开发团队来完成，每个人在软件项目中承担的只是复杂系统的一部分；软件开发分工更加精细，成员合作更加密切。但是，面对一些规模庞大、复杂度较高的软件系统，早期软件开发过程中存在的问题依然会出现：

(1)为什么仍然没有更好的办法获取用户的需求?

(2)为什么还是不能在规定的时间和预算下完成项目开发?

(3)为什么不能在将软件产品交给用户之前发现软件中存在的所有错误?

(4)为什么仍然不能控制好软件的项目进展?

这些问题一直是人们在软件开发过程中重点关注的问题,也构成了软件工程需要研究的过程和方法。

1.1.1 软件及其特征

计算机软件是指与计算机系统操作相关的各类程序、规程、规则以及任何与之有关的数据和文档资料。软件在计算机系统中和硬件相互依存,共同完成系统功能,具体说来,软件就是:

(1)在运行中能够提供所希望的功能和性能的所有指令的集合(即程序)。

(2)使程序能够适当地操作信息的数据结构。

(3)描述程序研制过程和操作的文档。

显然,软件不仅包括常规意义上的计算机程序,也包括程序产生过程中需要的各类相关文档。从软件的定义不难看出,软件是一组可以运行的程序和文档的集合,它提供了我们需要的产品信息,这些信息不仅可以帮助处理个人和公司信息,也能帮助企业利用这些信息走向世界,因为今天软件的影响已经非同昔比,互联网时代的软件不仅已经使世界变得如此之小,而且也带给了世界一个又一个神话。

1)软件特征

软件作为一种产品,是逻辑的而非有形的物理部件。因此,它不同于一般意义上的产品,而是具有许多非常独特的产品特征:

(1)软件是被设计的,而非制造的,具有抽象性。软件是人们通过智力活动,将知识和技术转化为信息的一种产品,它是被研制和开发出来的,可以通过高质量的设计提高软件产品的质量,和硬件不同的是,硬件一般是有形的,可见的,如果因在制造过程中出现问题而影响产品质量,解决起来将非常困难;而软件是记录在介质上的产品,无法看到其具体形态,只能通过运行、测试来了解其性能,而且如果出现问题,改正起来也比较容易。

(2)软件没有“磨损性”。一般来说,软件不会像硬件那样,在长时期的使用过程中会产生机械磨损、硬件老化等问题,而且硬件随着时间的推移,故障率是逐步提升的,硬件故障随时间变化曲线如图 1-1 所示;软件则没有上述问题,软件的故障往往是在使用之初的一段时间较高,一方面可能是由于未发现的故障所致,另外一方面是用户使用后提出新的需求需要完善,当使用一段时间后,软件的故障率越来越低,甚至趋于零,软件故障随时间变化曲线如图 1-2 所示。

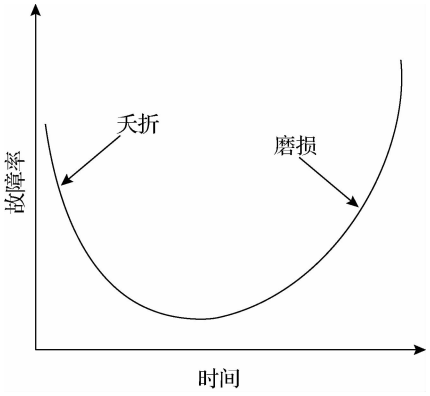


图 1-1　硬件故障随时间变化曲线

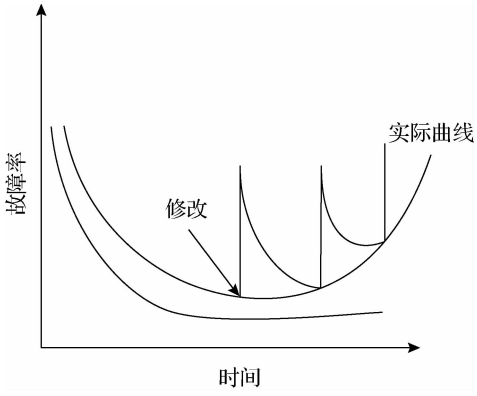


图 1-2　软件故障随时间变化曲线

(3)重用技术使软件开发效率大大提高。对于硬件来说,硬件工程师设计一个产品时,一旦定义好接口标准和协议,便可以从大量的标准部件中进行查阅和选择。但是对于软件来说,目前大量的软件依然处于定制阶段,因此,软件的开发效率依然很低。近年来,软件工程师已经开始尝试软件复用技术,即在软件开发过程中,将一些具有相对独立功能的模块进行独立设计,以便在具有同样功能的系统中对模块进行重用,显然,这种重用不仅可以大大提高软件开发效率,减轻开发人员的负担,而且还可以降低软件成本,提高软件开发质量。

目前,软件重用不仅包括代码重用,也包括设计结果重用,如数据流图的重用以及测试案例和分析结果重用等。

2)软件分类

日常生活中接触到的软件多种多样,特别是信息化的快速发展,软件已经是无处不在。但是,身处不同行业以及不同年龄段的人对软件的关注度也不尽相同,所以,很难给出一个严格、合理的软件分类,从应用领域划分,软件可以粗略分为以下两种:

(1)系统软件。它是一组与计算机硬件紧密相关的软件,可以使计算机的各个相关部件与相关软件及数据能够协调工作。系统软件不仅自身可以和硬件频繁交互、资源共享,而且可以为其他软件程序服务,可以提供复杂的数据结构、多个外部接口以及多用户支持。

(2)应用软件。应用软件一般指为某一特定领域开发的、服务于某一特定行业的软件。日常生活中接触最多的软件即为这类软件,例如,商业、服务业类的数据库应用软件;航空航天、汽车导航等用的智能控制软件;用于室内外装潢设计、CAD 制图等的计算机辅助设计软件,办公自动化系统,中文信息处理系统等。应用软件多是定制产品,其应用一般建立在系统软件之上。

1.1.2　软件工程的起源

计算机硬件技术的不断进步,使硬件成本大大降低,同时其质量和性能也在稳步提高。与此相反,随着软件规模的扩大和复杂性的增强,软件成本却不断攀升,软件的开发效率也不能适应软件技术的发展,一些成熟软件的维护也无法进行,造成了大量人力、物力的浪费,这些情况导致软件行业停滞不前,软件危机产生。

所谓软件危机，实际上是指软件开发和维护过程中遇到的一系列严重问题。软件开发过程中的问题是指如何用科学的软件开发方法和理论指导日益增长的复杂软件的开发，在合理的经费预算下用合适的软件开发手段完成合同规定的软件功能、性能的内容；维护过程中的问题主要是指如何维护已经开发完成的大量软件，并使维护变得轻松和容易。

软件危机的具体表现如下：

(1)历史方面。虽然有许多成功的软件案例，但是由于人们相信一些软件神话，例如，虽然人们相信软件的需求在不断变化，但还是认为这些变化不足为奇，因为灵活的软件设计是足以应付的。所以，软件延期交付、超出预算以及提交时依然存在错误等情况时有发生，这些在过去软件项目完成过程中存在的问题至今仍然屡见不鲜。2004 年一家软件研究机构完成的一项对9 236个软件项目的研究统计结果发现，仅有 29%的项目取得了成功，18%的项目在完成之前被取消或者没有完成，其余 53%的项目属于超预算完成、延期交付或者部分功能没有完成。这些情形可以概括为如图 1-3 所示。

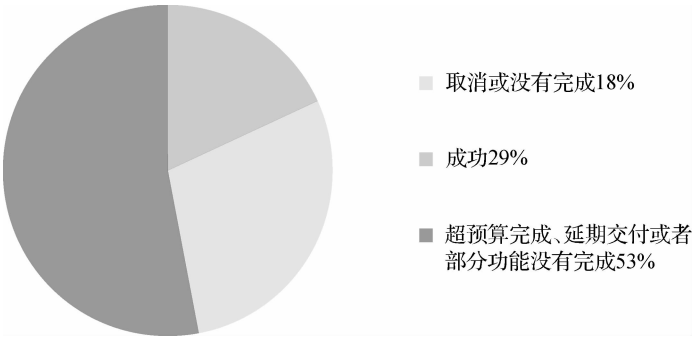


图 1-3　2004 年完成的软件统计结果

显然，由于大量软件不能按期交付，不仅对软件企业的生存造成了巨大影响，同时存在错误的软件交付到用户手中也会给用户造成巨大的损失，严重的会给人们的生活带来巨大的影响和灾难。例如，1991 年的海湾战争中，一枚飞毛腿导弹穿越了爱国者导弹防御系统，击中了沙特阿拉伯达兰市附近的一个兵营，造成了 28 位美国人死亡，98 人受伤。这正是由于爱国者导弹的软件中存在一个累计计时算法错误所致。

目前，这种软件危机依然存在，而且随着软件规模和复杂度的提高以及软件种类的日益增多，软件危机还会持续且更加难以预测。

(2)经济方面。软件新技术的层出不穷与开发手段的不断提高，给软件开发人员带来了极大便利。这就使人们更加愿意采用新的软件技术进行软件开发，以提高软件的开发速度，增加软件企业的效益。但是从问题的两面性考虑，这种做法在软件设计领域有时并非可取，主要原因是：

①一项新的编程技术的引进虽然能够加快软件开发速度，提高软件编程效率，但是不得不考虑引进新编程技术需要付出的代价。一般地，引进新的编程技术，不仅要对软件编程人员进行相关的技术培训，而且培训后的编程人员还要有一段时间的熟悉和消化过程，真正做到能够熟练编程更是需要一个相对长的时间，因此，在编程语言的选择方面，要权衡这些因素和利益得失，否则，就有可能陷入混乱。

②用新的编程语言开发的软件虽然节省了大量的编程时间，但是其维护并不轻松，相对

于使用熟悉的编程语言，如果软件开发人员需要负责长期的软件维护，成本的核算仍然是一个不确定的变量。

(3)维护方面。软件维护之所以变得越来越困难，主要取决于人们对软件维护的认识。

首先，一些人认为，写出了软件程序并能够将其正常运行，工作也就结束了。显然，这种观点忽视了软件的生命周期过程，一个软件交付给用户后，真正的软件开发工作才完成了全部工作量的 20%～30%，较长时期的软件开发工作都是在软件交付给用户后才开始的。

其次，软件能否很好地得到维护，一要有相应的经费支持，二要有完善的文档资料可供参阅。在这方面，同样存在误解，一些人认为，一个成功的项目唯一应该提交的是能够运行的程序，殊不知，软件开发人员的流动使得软件维护变得并不轻松，如果没有相应的软件配置指导、程序设计说明书以及各类辅助文档资料，进行软件维护可以说是天方夜谭。

在占据软件生命周期最长的软件维护阶段，费用预算同样也被许多开发和管理人员所忽视。从图 1-4(a)中可以看出，大概有三分之二的费用用于软件交付后的维护。图 1-4(b)显示交付后软件维护占的费用更大，许多软件组织将软件预算的 70%～80%甚至更多用于软件交付后的维护。

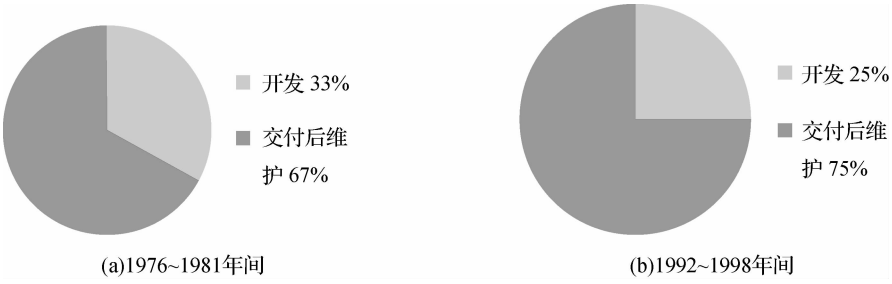


图 1-4　软件开发和交付后维护所占的近似成本

从软件危机的表现方面看，其历史已经很长，且蔓延趋势明显；经济效益是软件开发人员和企业的命脉，天下没有免费的午餐，没有效益就没有人愿意开发软件；软件维护是软件生命周期中最长的阶段，如果维护阶段没有保障，那么没有用户愿意开发并使用软件，因此，为了促进软件产业的健康发展，必须有一套科学、合理的软件管理和技术手段来保障软件项目的顺利实施，保障软件客户的最大权益，使软件更好地服务于社会。于是，1968 年在前联邦德国召开的国际会议上，人们提出了软件工程这一概念，意在运用工程学的基本原理和方法来组织和管理软件的开发和生产。

1.1.3　软件工程的概念

自 1968 年提出软件工程这一术语以来，软件工程的概念和内涵得到了软件界专家的不断丰富和完善，经过长期的探索和研究，形成了软件工程的核心内容，其中一些经典性的结论如下：

(1)P. Wegner 和 B. Boehm 认为，软件工程可以定义为将科学知识运用在设计和构造计算机程序，以及开发、运行和维护这些程序所要求的有关文档。

(2)Fritz Bauer 在北大西洋公约组织会议上给出的定义为:软件工程是为了经济地获得可靠的和能在实际机器上高效运行的软件而确立和使用的健全的工程原理和方法。

(3)IEEE 给出的软件工程的定义是:软件工程是开发、运行、维护和修复软件的系统方法。

(4)IEEE 给出的一个更加综合的软件工程的定义是:将系统化的、规范的、可度量的方法应用于软件的开发、运行和维护的过程,即将工程化应用于软件中。

1.1.4 软件工程内容

尽管不同的软件专家对软件工程的定义有所不同,但是大家都认可的理念是:软件工程是应用工程化的方法对软件开发进行管理。软件工程的最终目的是采用工程化的概念、原理、技术和方法来进行软件的开发和维护,用正确和科学的管理方法及开发技术对软件工程进行完美性实施。从理论上来看,软件工程研究的内容有程序正确性证明理论、软件可靠性理论、软件成本估算模型、软件开发模型以及模块划分原理等;从技术角度来看,软件工程研究软件开发方法学、软件工具和软件开发环境等,具体说明如下:

(1)软件需求。软件需求是为了解决实际问题而施加在软件产品上的要求和约束,它是软件开发的基础和条件。软件危机的重要原因之一就是软件需求工作完成得不彻底,同时,软件需求在软件生命周期过程中也在不断变化,因此,软件工程要研究软件需求的分析技术和方法,从而使软件需求在软件开发过程中可控。

(2)软件设计。软件设计是系统分析完成后,软件开发之前的一个软件过程,在该过程中为了实现软件的各种功能和性能,软件工程师需要对软件方案给出一个模型及整体架构,同时给出相应的模块和算法模型的详细描述。软件工程在这个过程中需要研究软件的模块划分原则以及软件设计的测试方法等内容。

(3)软件构建。软件构建是指创建软件的过程和方法,软件工程需要研究产生软件的方法学,包括软件编码、测试、集成以及调试工具开发技术等。

(4)软件测试。通俗地讲,软件测试过程就是验证软件是否满足需求分析过程中定义的功能和性能,它是产生高质量软件的保证。软件工程研究的是软件的测试理论和方法,包括算法测试、性能测试以及开源代码和可重用模块的测试等。

(5)软件维护。软件维护是软件交付给用户后开始的软件过程。因为产品交给用户后,不可避免地会暴露出软件存在的质量问题,同时,软件在用户的使用环境中也可能会发生变化,这些问题都是在软件维护阶段需要解决的,软件工程研究的是软件维护的步骤和方法,以及如何使软件维护过程更加严谨和经济,如何和前面的软件过程进行更好的融合。

(6)软件配置管理。一些现代软件都具有非常高的复杂度和庞大的规模,因此软件开发过程的人、财、物以及软件的变更管理变得越来越困难。软件工程要研究合适的软件配置管理方法,用该方法对软件过程的始终进行全方位的管理、监控和跟踪,从而使软件过程中的任何变化都不会影响软件质量,且不会引起混乱或者带来风险。

(7)软件工程管理。软件工程管理是软件过程的管理活动,软件工程研究该管理方法的

目的是使管理活动更加具体化、系统化和流程化，能够量化一些软件工程的活动内容。

(8)软件工程过程。软件工程过程研究软件生命周期中的相关技术和管理方法，和软件工程管理的研究内容不同，该部分研究的是过程的科学与合理，例如，如何使软件生命周期的需求获取、设计开发、测试以及维护过程协调一致，如何在保证软件质量的前提下加快软件的开发过程等。

(9)软件工程工具和方法。软件工程工具研究的目的是为软件开发提供高效的计算机辅助设计工具，同时减少软件开发人员的工作强度。软件工程方法研究的目的是使软件工程活动系统化并最终取得软件系统的成功，方法可以是一种符号和词汇表，也可以是一种特定的开发和管理方法。

(10)软件质量。软件质量虽然有多种不同的解释和定义，但从根本上讲，客户对软件的满意程度反映出软件质量的高低，软件开发的最终目的是开发出用户高满意度的软件，软件工程在这个方面研究的是软件质量保证的体系理论和方法，以及软件质量的度量理论和方法等内容。

1.2　软件工程框架

软件工程的框架可概括为软件工程的目标、过程以及实现原则。其中，目标定义了软件工程进行的最终目的，即建立高可靠性的软件；为了保证软件工程的顺利实施，软件过程必须在严格的控制和管理下进行；而软件工程的实现原则要求软件工程不仅要科学实施、科学设计和科学施工，同样也必须遵守软件性质中所定义的过程准则。

1.2.1　软件工程的目标和原则

软件工程的目标非常明确，就是在给定成本、进度的前提下，利用正确的理论和科学的方法研制、开发和生产出具有有效性、可靠性、可理解性、可维护性、可重用性、可适应性、可移植性、可追踪性和可互操作性的满足用户需求的高质量软件产品。而为了实现这一目标，必须采取相应的开发、管理、过程控制等措施，具体如下：

(1)利用工程化的思想进行软件的开发和维护。要想实现软件工程的目标，必须有一套成熟的软件开发过程模型，从工程的组织、管理和实施等多个方面进行精心组织、合理调配、协同工作。

不同规模、不同类型的软件要采用不同的组织方式进行开发，借鉴已有的多种成功经验，并在需求分析、软件设计、系统实现以及集成测试等多个过程中推行，探索新的技术和有效的成功方法，合理组织开发团队，提高程序员团队的凝聚力，激发程序员的创新和思考能力，使软件过程的分工更加细致与合理，在过程的每一个阶段都有质量控制。

(2)充分利用计算机辅助开发和设计工具。随着软件开发技术的不断更新，软件开发的

技术手段已经得到了极大增强，计算机辅助设计工具在软件开发过程中的作用已经越来越明显，充分利用这些辅助设计工具不仅可以大大减轻开发人员的工作强度，而且可以使软件开发更加容易，软件过程的管理更加方便。辅助开发工具的使用可以使软件的开发周期大大缩短，同时生产的软件质量也更加可靠。因此，充分利用计算机辅助开发和设计工具是现代软件工程过程的重要手段。

(3)利用现代化的手段进行软件过程的管理。软件危机产生的更深层次的原因是软件过程的管理非常困难，软件生命周期的各个过程之间相互联系又相互制约，对各个生命周期过程需要的人员层次的认识也不尽相同，因此，软件工程过程中的管理不同于一般工程项目的管理，必须通过一定的项目管理工具、配置管理工具和管理计划来保证软件项目的顺利实施，以获取高质量的软件。

软件工程的目标不仅要通过科学的软件项目实施来取得，也要借鉴过去大量的软件开发和管理经验。从软件工程概念的提出到现在，专家学者们也提出了许多软件工程实践准则，其中，软件工程领域的著名学者 Barry W. Boehm 综合诸多学者的意见提出了六条软件工程基本准则，这些准则之间相互独立，从完备角度看，它们的组合是缺一不可的最小集合，人们提出的其他准则都可以由这几个准则的组合而得到，可以将它们概括如下：

(1)用分阶段的生命周期计划严格管理。软件从定义、分析、设计、实施到最终被废弃需要经过一个漫长的过程，这个过程被定义为软件的生命周期。由于在软件不同的生命周期过程中需要的人员种类不同，完成的项目内容和需要的时间也不同，因此，需要将这些过程分离开来，在不同阶段制定不同的实施和管理计划，并要求参与到各个阶段的成员严格按照过程计划实施，最后提交应该完成的阶段性内容，这样，将软件的管理划分到不同的生命周期阶段有助于把项目中可能发生的问题分解到不同的阶段，可以通过不同阶段的审核尽早地发现和消除风险。

(2)坚持进行阶段评审。软件开发由人来完成，因此，软件中产生错误在所难免，但是不能到软件完成后再谈软件质量保证，必须在软件过程的不同阶段进行错误识别和改正，主要理由如下：

①软件中存在的错误大部分是在软件进行编码之前存在的。据统计，设计错误占软件错误的 63%，而编码错误仅占 37%，例如，在对喷气推动实验室为美国国家航空航天局无人太空计划开发的软件进行的 203 次检查结果统计中发现，平均每页规格说明书中有 1.9 个错误，平均每页设计说明书中有 0.9 个错误，而平均每页代码中只有 0.3 个错误。

②错误发现的越晚，付出的代价越高。例如，图 1-5 描述了 1974～1980 年间部分软件项目在不同阶段纠正一个错误所花费的平均成本。从图中可以看出，在分析阶段纠正相同的错误仅需 3 美元，而在交付软件后的维护阶段纠正相同的错误则需要花费 200 美元，这主要是由于交付给用户后纠正错误所需要的工作量巨大所致。因为软件交付后的错误纠正的工作内容不仅是代码修改，还涉及编译、链接以及所有相关文档的更新。

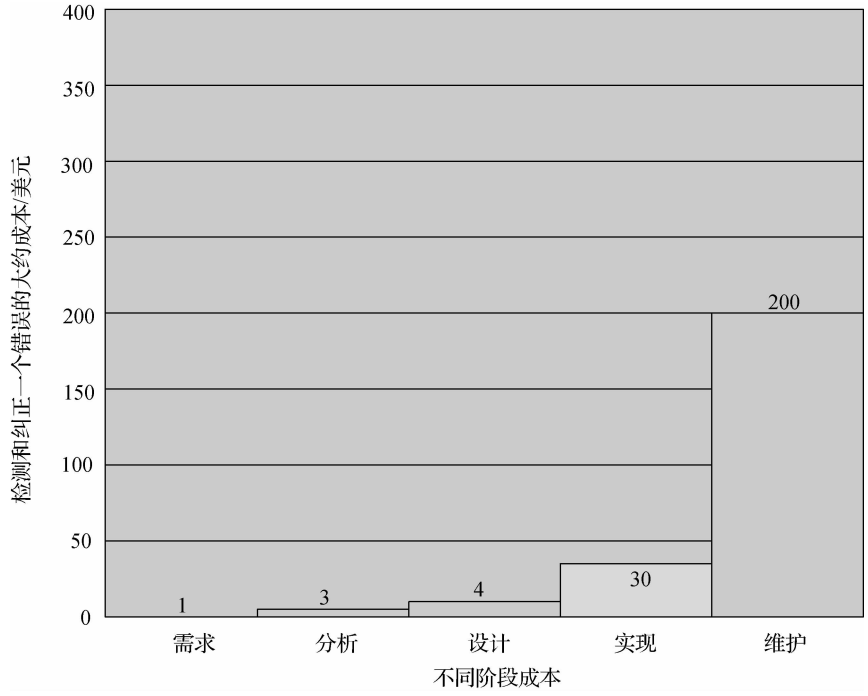


图 1-5　软件过程的不同阶段纠正相同错误所需成本

(3)实行严格的产品控制。实行严格的产品控制有两层含义：

首先，软件产品在开发过程中，一般都希望在需求阶段定义的功能和性能等不要发生改变，但是，由于需求获取手段的限制以及用户环境的改变，需求改变不可避免地要发生，对于这种由于需求改变而导致软件配置改变的情形，必须采取非常严格的管理措施，对需求改变的评审、需求变更的实施以及需求变更后产品的测试等均要实行严格的配置管理，务必保持配置的一致性，不能出现没有经过批准和评审的随意变更。

其次，应该考虑产品开发过程中的一致性检查，目前，大量的软件产品是借助于开发团队来完成的，这样，产品开发过程中就可能出现团队协作的非同步情形，因此，产品开发过程中必须定义严格的代码接口协议并对完成的模块经常进行一致性检验，由于这种开发过程涉及的因素比较多，如程序员本身的人为因素、团队组织形式以及经济和法律等因素，因此，这种产品的严格控制要充分考虑项目本身的特点。

(4)采用现代程序设计技术。软件工程注重在项目规定的日期内完成高质量、可运行的软件产品，因此，程序设计技术是软件工程研究的主要内容之一，从结构化程序设计技术到目前流行的面向对象程序设计技术，软件设计技术的进步使软件开发变得更加快捷，也使软件维护变得更加容易。但是，对于一些特定的软件项目，如追求高可靠性的软件项目，在选择程序设计技术方面还须谨慎考虑。

(5)结果清晰并应该能够被审查。和一般的物理产品不同，软件产品是一种逻辑产品，它是以代码的形式存放的，如果没有实际的运行效果，很难检查软件的开发进度，这也是软件工程难于管理的原因之一。因此，为了对软件的进度进行合理的控制和管理，必须根据软件的功能和性能等总体目标，清楚标识阶段性成果的检查标志，使完成的内容清晰可见，并

且能够经受得住质量检验人员的审查。

(6)开发小组的人员应该少而精。软件开发是一项高强度的智力活动,因此,要求软件开发人员不仅要爱好软件开发工作,有良好的道德自律,更需要有良好的从业素质。具有丰富软件实践经验的高素质软件技术人员不仅可以提高软件的生产效率,而且还可以产生高质量的软件。之所以要求开发小组的人员少而精,同样也是因为,较多的软件开发人员会增加更多的相互交流,从而耽误更多的开发时间,因为一个具有 N 个成员的开发小组,可能的通信路径有$\frac{N(N-1)}{2}$条。显而易见,人员增加,相互交流的时间也会增加。

一般情况下,遵循上述六条基本原则可以生产出高质量的软件,但是软件技术的发展和新问题的出现也会促使我们不断改进软件过程的实践活动,吸收先进的软件开发技术和软件工程的新思想、新方法,以灵活适应现代软件的特点。例如,可以不断收集软件开发过程中导致软件不能按时交付的因素,不断吸取软件团队开发过程中的经验教训等,这样,软件工程的基本准则将在实践中不断得到创新。

1.2.2 软件工程过程

软件工程的目的是生产高质量的软件,而进行生产则需要遵循一定的步骤,这个步骤的完整线路图就是软件工程过程,Howard Baetjer 给出的软件工程过程的描述为:“因为软件像所有资本一样是具体化的知识,也因为从宏观看来知识最初是分散的、不明的、隐藏的且不完整的,软件开发是一个社会的学习过程。该过程是一个对话,其中必须变成软件的知识被集中到一起并具体化到软件中,该过程提供了用户和设计者间、用户和演化的工具间以及设计者和演化的工具(技术)间的交互。它是一个迭代的过程,在其中演化的工具本身作为通信的媒介,随着每新一轮的对话,从参与的人员引导出更有用的知识。”

确切地说,软件工程过程就是一个迭代学习的过程,而且过程的输出结果,即软件,是在这个过程进行中收集、精练和组织的知识的具体化。在迭代学习过程中,如果不考虑项目的复杂性以及规模特征,软件工程过程一般可以分为如下三个阶段:

第一个阶段可以称为软件工程定义阶段,在该阶段的主要任务是明确“做什么”。即在软件定义过程中要明确软件要处理哪些信息,需要产生什么样的输出效果,系统要完成哪些功能,又有哪些性能需求和约束条件,系统要建立什么样的界面,实际客户的最关键需求是什么等。虽然这些内容在定义阶段需要完成,但是对于一个具体的软件工程项目,可能需要根据软件本身进行处理,但是从软件工程的生命周期过程考虑,这个阶段主要包括可行性分析、需求分析和软件项目计划等内容。

第二个阶段集中体现在“如何做”。这个阶段要解决第一个阶段定义的问题,即如何进行输入信息的处理以产生需要的输出信息,用哪些手段和方法实现项目的功能和性能,对输出结果如何测试其正确性,项目的具体细节,如编码标准、测试手段与方法如何实现等,从软件生命周期的过程考虑,该阶段包括软件的概要设计和详细设计阶段、系统编码与集成阶段、软件测试阶段。

第三个阶段集中体现在如何对“变化”进行处理。所谓变化一般是指软件外部环境的变化、客户当前需求的变化以及软件使用过程中出现的一些错误等,任何一个软件项目都可能

有这些具体的变化，如果说在第一个阶段已经做好了应对这些变化的详细安排，那么这个阶段主要是如何解决这些变化，软件生命周期的系统维护阶段的工作则是第三阶段工作的具体体现。

本 章 小 结

首先，本章描述了软件的具体定义，给出了软件不同于硬件的具体特征。软件的这些特征表明软件是一种逻辑产品，不同于其他有形的物理部件，所以软件具有一些如非磨损等独特特性，其工程实现过程也有别于硬件的生产过程。

其次，本章分析了软件危机产生的根源，从历史方面、经济方面和软件维护方面给出了软件危机的不同种类，这些不同种类的软件危机已经严重影响了软件产业的发展，因此，为了促进软件产业的健康发展，需要有一套科学与合理的软件管理和技术手段来保障软件项目的顺利实施，使软件更好地服务于社会，于是，人们提出了软件工程的概念。本章给出了不同时期人们对软件工程的理解，同时对软件工程的研究内容进行了详细论述。

最后，本章结合软件工程的研究内容，引出了软件工程实施的最终目标，即生产高质量、符合用户需求的软件产品，并从软件工程实现的基本准则方面进行了解释，虽然人们提出了软件工程的多种准则，但是本章给出的六条准则及其组合基本涵盖了软件工程专家提出的软件工程准则，当然，软件工程的实践是不断发展的，软件工程的理论和方法也在不断创新中，所以在软件实践活动中坚持不断改进软件过程、吸收先进的软件开发技术和软件工程的新思想，可以使软件开发过程变得更加科学与合理。

习　题　1

1）选择题

（1）在生活和学习中使用过各种各样的软件，从软件本身考虑，软件定义包括的内容有（　　）。

A. 可以运行的程序　　B. 开发工具

C. 原始代码　　D. 详细设计说明书

（2）关于软件故障，下面论述正确的是（　　）。

A. 软件交付用户后的一段时间内，故障明显，以后故障率逐渐下降

B. 软件交付用户后的一段时间内，故障明显，一段时间后，故障率会趋于零，之后故障率可能会有大有小

C. 软件交付用户后，故障率应该为零，因为交付的软件没有错误

D. 软件交付用户后，故障会不断出现

（3）假定一个软件系统的开发费用估计为 50 000 元人民币，从软件工程的角度考虑，软

件交付后的维护费用大约应该为(　　)。

A. 5 000 元　　B. 10 000 元

C. 不需要　　D. 35 000 元

(4)一个软件产品在交付 7 个月后,发现了一个问题,纠正该错误花费了 6 000 元人民币,错误的起因是因为一条模糊语句,那么如果在分析阶段能够纠正该错误,需要的费用是(　　)。

A. 100 元左右　　B. 500 元左右

C. 1 000 元左右　　D. 2 000 元左右

(5)开发团队在软件项目进行过程中,为了能够相互协调软件的开发和进度等,需要进行相互沟通,具有 3 个人的开发小组和 5 个人的开发小组,其相互沟通的渠道分别为(　　)。

A. 3 和 5　　B. 1 和 6

C. 3 和 8　　D. 3 和 10

2)问答题

(1)为什么会产生软件危机?

(2)什么是软件工程? 软件工程的基本原则是什么?

(3)软件工程过程中,为什么要坚持阶段性评审?

(4)假定公司经理将一份源程序清单交付给你,希望你找出其中存在的 bug,你将如何对经理解释?

第 2 章 软件过程模型

软件过程是指为了获得高质量的软件而进行的一系列相关活动。这些活动的执行可以是顺序的、重复的、并行的、嵌套的或者是有条件引发的，不论软件过程的形式和种类如何，它们的最终目标都是在软件目标的指导下，借助于科学的软件过程开发出质量优良、性能稳定、充分满足用户需求的高水平软件，因此，软件过程的研究非常必要，它不仅是科学的，而且必须是合理的。

软件过程通常由多个阶段组成，它不仅需要定义软件过程需要的资源（包括人力资源、环境资源、财力资源等），而且需要给出将需求转换为软件产品的方式和方法，每个阶段之间的次序、重叠、迭代等关系往往与软件项目的范围、规模、复杂性等密切相关。软件过程的每个阶段都有特定的工作内容，包括数据的输入与输出等，并会产生特定的软件成果，包括模型、代码以及文档资料等。因此 ISO 9000 将过程定义为“把输入转化为输出的一组彼此相关的资源和活动”。

本章将给出软件生命周期的基本过程，并给出经典的软件过程模型的特征描述、优点以及相互比较等内容。

2.1 软件生命周期的基本过程

每个产品都有一个完整的生产过程，从产品的概念形成、外形和功能设计、制造到产品成形，再到产品最终失去存在的价值，这个过程就构成了一个产品的生命周期。对于软件产品也是如此，软件产品的生命周期包括问题定义、可行性研究、需求分析、概要设计、详细设计、编码与测试、综合测试、运行与维护等过程，在这一系列过程中，软件从概念定义到开发完成，从用户欣赏使用到不断地完善更新，直到最后退出其应用的历史舞台，软件过程的不同阶段作用不同，但其目的都只有一个，即使软件具备更高的质量和更好的性能，从而最大限度地满足用户需求。

1)问题定义

问题定义阶段的主要目标是明确系统的任务是什么,要解决什么问题,这是开始一个软件项目最重要的事情,如果问题定义模糊不清,那么下面的相关过程就无从谈起,再做什么工作都是徒劳。明确问题不仅需要软件开发者具有良好的软件开发经验,也需要有良好的用户沟通和协调能力,当然,问题定义更需要用户的积极配合和认可。

2)可行性研究

可行性研究即在明确解决问题的基础上,对解决该问题需要的人力资源、财力资源以及各种技术资源进行分析,提出解决该问题的各种可行性,并进行经济效益、社会效益等的可行性分析。在此基础上,提出切实可行的解决方法,并对该方法的技术因素、社会因素、人为因素等进行综合分析,从而给出系统实现的可行性研究报告,可行性研究必须给出充分的论据证明项目可行与否。

3)需求分析

需求分析是软件开发的重要过程,这个阶段必须明确软件要完成哪些功能,软件需要具备哪些性能。需求分析离不开用户的帮助,只有和用户多交流才有可能使需求更加明晰,当然,获取需求分析的方法很多,能否最大限度地获得用户的需求分析,不仅要靠软件开发者的努力,也取决于用户对软件的清楚描述及其相关的配合工作。这个阶段的一个重要输出是项目规格说明书,它详细说明了用户的需求,系统需要完成的功能、满足的性能、需要的环境和各种约束条件等内容。项目规格说明书需要经过必要的评审后才可以进入软件的下一个过程。

4)概要设计

概要设计又称总体设计,是对系统的一个初步设计,设计内容包括系统的实现方案以及体系结构,如果设计过程中具有不同的设计方案,也可以分别列出其优缺点,以供用户选择。最后,对选定的方案应该按照合理的次序和逻辑关系给出系统的模块划分,划分原则是应使模块之间的耦合度最小。

5)详细设计

详细设计阶段的主要任务是将问题实现具体化,即用什么样的手段或者方法实现系统的功能模块,具体来说,就是对系统的各个模块进行具体设计,设计内容包括算法的数学模型、数据结构、测试代码以及实现步骤等。详细设计也称模块设计,不仅包括程序的结构,也包括必要的详细规格说明。详细设计的要求是代码编写人员参照详细设计应该能够写出程序代码。

6)编码与测试

编码与测试阶段的工作是在详细设计的基础上,进行代码编写,并完成相关代码的测试工作,也就是将详细设计转化为符合规范和要求的程序代码。编码阶段需要注意的是应该严格遵守设计阶段的编码规范,应用可重用的软件组件前要清楚组件的来龙去脉以及使用规则。测试阶段应该注意不仅要进行代码质量的测试,也要进行软件功能的测试。经过该阶段,软件的代码编写基本完成,各个模块功能基本实现。

7)综合测试

综合测试也称集成测试，也就是将经过测试的软件模块进行总体集成，之后进行系统测试，根据技术规格书以及测试案例不仅要进行软件功能的整体测试，而且要进行软件的整体性能测试，综合测试要注重检查程序之间的接口以及一些数据的信息。参加测试的人员既有软件开发人员，也应该包含用户，在完成综合测试的基础上，写出测试报告，从而对软件的质量给出一个综合的评价，以确定软件是否具备移交给用户的条件。

8)运行与维护

软件开发完成交付给用户即是运行与维护工作的开始，软件的特征就是不断地进行软件维护以满足用户日益变化的需求。软件维护并不是一件非常简单的工作，它需要软件人员具备非常丰富的软件开发技术经验，具备非常强的现场交流和现场处置能力。软件维护的步骤是非常严格的，不是用户提出什么新的要求，开发人员就必须立即进行改正，而是要经过维护分析阶段、维护计划提出、维护计划审批、软件维护、维护测试与验收等一系列的步骤，同时任何一次软件维护必须具备完整的维护文档，这样，会给软件维护人员的工作带来巨大的便利。

在实际的软件开发过程中，软件开发的过程取决于软件的复杂度、规模、种类、开发环境以及所应有的技术等多种因素，对于小规模的软件，循序渐进地遵循软件的这些开发过程可能会事与愿违，因为这样的过程显得过于复杂；而对于大规模的复杂软件，这些过程可能还不够，还需要对这些软件过程进行组合优化，以寻找出更好的开发过程，因此，一个科学的软件过程应该是适合于所承担的项目特点的任务集合，这些任务集合包括软件过程、里程碑事件以及软件产品等。

为了研究适合于各类软件的开发过程，人们提出了各种各样的软件过程模型，即软件开发的过程以及任务框架。过程模型可以直观显示软件开发的全过程，明确规定软件过程要完成的一些主要活动和开发策略。软件过程模型不仅使开发过程有章可循，同时也使软件过程更加规范和有效，软件目标更加明确。

目前，人们提出的软件过程模型虽然适应的软件类型不同，但它们都是大量软件工程实践的结晶，因此，对于现代的软件开发具有一定的指导意义。

2.2　瀑 布 模 型

瀑布模型是温斯顿·罗伊斯于 1970 年提出的软件过程模型，直到 20 世纪 80 年代早期，它一直是唯一被广泛采用的软件开发模型。瀑布模型的经典之处在于，首次把工程概念引入到软件开发工作中，是一个文档驱动的模型。

瀑布模型的核心思想是按工序将问题化简，将功能的实现与设计分开，便于分工协作，采用结构化的分析与设计方法，将逻辑实现与物理实现分开。根据软件过程的开发步骤和多年的研究经验，瀑布模型将软件过程开发和维护的复杂过程进行了时间层次上的划分，将软件生命周期划分为计划制定、需求分析、软件设计、程序编写、软件测试和运行维护等六个

基本活动，并且规定了它们自上而下、相互衔接的固定次序，顺序如同瀑布流水，环环相扣，逐级连接。瀑布模型如图 2-1 所示。

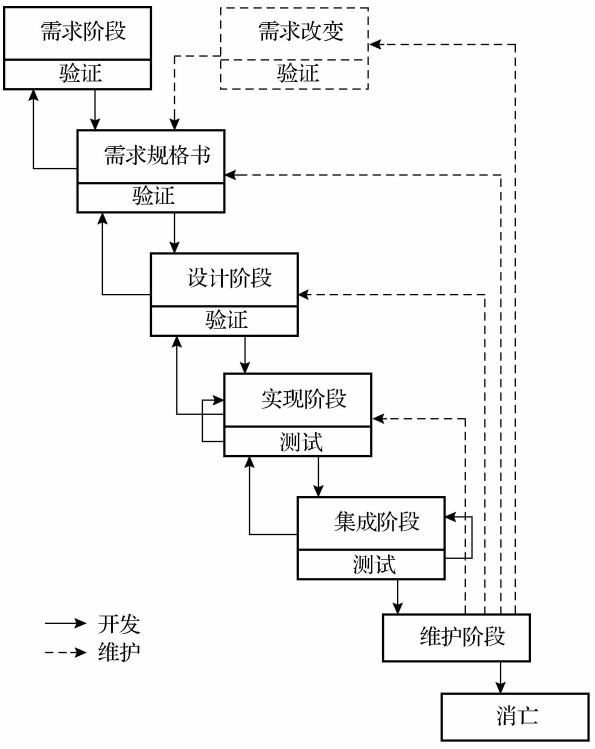


图 2-1　瀑布模型

在瀑布模型中，软件开发过程的每个里程碑都以事先需求分析定义的大量文档为标识，该文档将成为下一个过程继续的基础，即模型中的上一项活动的工作成果，将作为输出传给下一项活动，如果该项活动工作评审得到认可，那么下一项活动可以进行，否则返回前一项活动甚至更前项的活动。显然，该过程表现在两个方面：一是必须等前一阶段的工作完成之后，才能开始后一阶段的工作；二是前一阶段的输出文档就是后一阶段的输入文档。因此，只有前一阶段的输出文档正确，后一阶段的工作才能开始并可能获得正确的结果。

2.2.1　瀑布模型分析

由于瀑布模型具有严谨和顺序特征，因此瀑布模型注重软件质量的高度可靠，并不一味追求软件开发的速度，具体说来，瀑布模型的特点如下：

(1)阶段间的顺序性和相互依赖性。所谓顺序性是指瀑布模型具有严格的阶段顺序特性，前一过程在经过严格的评审后，后一过程才能继续，也就是说，前一阶段的输出文档，就是后一阶段的输入文档。依赖性是指后一阶段的启动严格依赖于前一阶段有正确的输出，否则后一阶段将无法启动，这种严格的顺序性要求每一阶段必须严谨，论证和设计必须周到、细致，否则该软件过程将无法完成，即使完成，如果进行中发现中间阶段存在问题，那么修改和完善也将付出极大的工作量。

(2)质量保证贯穿瀑布模型全过程。瀑布模型在不考虑时间代价的基础上,强调软件开发的质量,并将过程贯穿于软件开发过程的每一个阶段:

①每一阶段都要完成规定的相关文档,如果没有完成文档,那么该阶段的任务就没有完成,相应的后续过程将无法继续。从软件过程的角度考虑,显然编写代码并不是基于要完成的任务,重要的是系统需求分析和系统设计,否则,急急忙忙地编写代码,最后可能会事与愿违。

②每一阶段都要对文档进行评审,尽管评审的方式可能不同,但是评审仍然会起到发现问题、消除隐患的作用。发现隐患越早,付出修改的代价越小,如果直到系统的测试阶段才发现问题,那么对系统进行维护的代价将是巨大的。

2.2.2　瀑布模型的适用范围

瀑布模型是经过了大量的软件实践而提出的,也有大量成功案例,严格的过程管理使模型具备如下优点:

(1)对于功能和性能明确、完整,需求无重大变化的系统,瀑布模型是一个良好的软件开发过程的选择。由于瀑布模型的基础在于需求分析,因此,只要前期将系统目标、软件功能与性能进行完整和精确的定义,那么在模型提供的模板指导下,进行分析、设计、编码、测试等,将会使软件系统得到一个完美的结果。

一般地,国防科研院所的软件系统、大型工业如钢铁企业等使用的软件系统,均可采用瀑布模型来指导软件开发过程。

(2)瀑布模型通过设置过程里程碑,来明确每个阶段的任务与目标,这样,有利于软件组织为每个阶段分别制订开发计划、进行成本预算、组织相关资源。

由于瀑布模型具有非常明确的阶段划分,因此易于使软件组织在每个阶段组织精干力量进行任务攻关,当通过阶段评审后,再集中力量进行下一阶段的任务,这样,将能够在一定程度上保证软件的按时交付,并能够保证软件达到最高的标准。

但是,瀑布模型过分强调文档的作用,并要求每个阶段都要仔细验证,同时,从模型结构可以看出,这种模型的线性过程太理想化,随着软件系统复杂度和规模的不断扩大,该模型的缺点将非常突出,具体如下:

(1)模型不适应需求经常发生变更的软件环境。瀑布模型的成功建立在需求清晰、完备的基础上,但是由于软件应用行业广泛,软件规模和复杂度日益增加,因此需求分析也变得越来越困难,需求也同样成为一个软件系统需要控制的目标,这样,瀑布模型就难以适应这类软件的开发过程,因为需求的不断变更可能会引起软件阶段的混乱,同时也会造成大量的人力、物力和财力的损耗。所以,有人形象地把采用瀑布模型进行商业软件开发称为“在沙滩上盖楼房”。

(2)瀑布模型不适应时间紧、任务急的软件过程。瀑布模型由于有严格的顺序过程,可能会要求在某一过程的进行过程中,等待其他成员完成其所依赖的任务才能进行下去,这样,有可能等待的时间比开发的时间要长,这被称为“堵塞状态”,这种情形显然无法适应时间紧、任务急的软件过程。而追求效益最大化是软件企业的终极目标,所以,一些软件企业不会选择花费时间相对较长的瀑布模型,而一些与时间无关、只求质量的软件则适用于瀑布模型。

2.3 快速原型模型

快速原型模型是一个增量型软件过程模型，它强调极短的开发周期。可以充分利用软件组件开发使快速原型模型的实现成为可能。快速原型模型如图 2-2 所示。

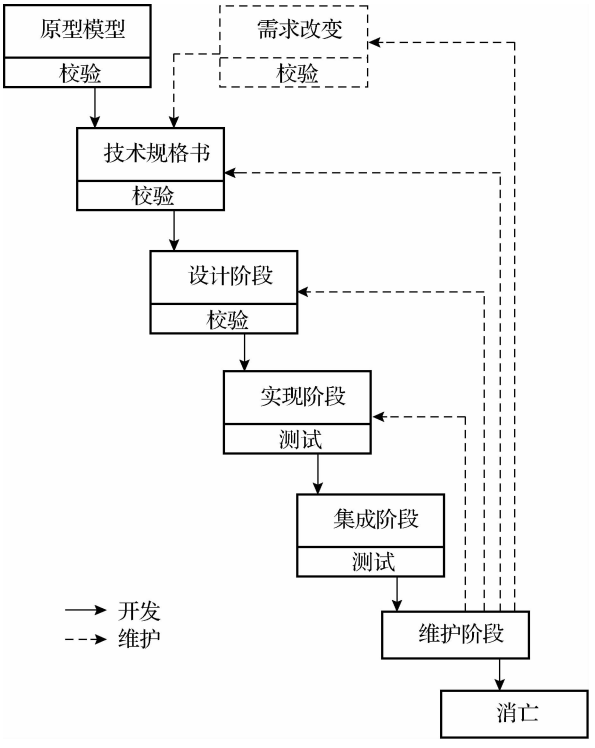


图 2-2 快速原型模型

快速原型模型的目的是快速建造一个可以运行的软件原型，以便和用户进一步讨论、分析，从而理解用户需求，与用户达成共识，最终在确定客户需求的基础上开发出客户满意的软件产品。快速原型模型允许在需求分析阶段对软件的需求进行初步而非完全的分析和定义，之后根据对需求的初步理解快速设计出软件系统的原型，该原型将向用户展示待开发软件的全部或部分功能和性能；用户对该原型进行测试评定，给出具体改进意见以丰富和细化软件需求；开发人员据此对软件进行修改完善，直至用户满意认可之后，再进行软件的完整实现及测试和维护。

借助于快速原型模型，开发人员可以首先应用相对较少的成本以及较短的周期设计一个简单的、但可以运行的系统原型向用户演示或让用户试用，以便及早澄清并检验一些主要设计策略，并在此基础上再开发实际的软件系统。快速原型模型的类型主要分为以下三种：

(1)探索型原型。将快速原型用于软件开发的需求分析阶段，目的是借助于原型的展示清晰化用户的需求，确定用户期望的软件功能和性能特征，探索各种方案的可行性。该方法

主要针对开发目标模糊，用户与开发商对项目都缺乏经验的情况，通过快速原型可以明确用户的需求。

(2)实验型原型。这种原型主要用于系统设计阶段，用该原型以确定实现方案是否合适和适用。对于一个大型的复杂系统，若软件开发人员对设计方案没有十分的把握，可以通过这种原型来证实设计方案的正确性。

(3)演化型原型。当把快速原型提交给用户后，该原型系统并不被抛弃，而是在和用户沟通后，将原型不断扩充演化并最终变为适用的软件系统。演化型原型要求在设计原型之初就要考虑周到，并按照软件工程的思想进行设计。

要想实现快速原型，可以采用软件重用技术，利用现成的大量可重用组件，争取时间，尽快向用户提供原型模型；也可以应用一些类似的、具有一定相关性的软件，利用它们的一些相似界面和功能，向用户提供部分或全部功能，以达到快速开发的目的。

显然，快速原型模型可以处理模糊需求，这样就可以克服瀑布模型的缺点，减少由于软件需求不明确带来的开发风险，其优点还在于：

(1)借助于快速原型，开发人员可以和用户进行更有效的沟通，从而使用户更加清晰自己的真正需求。

(2)快速原型的软件开发过程是线性的，不存在瀑布模型中的反馈循环，因此快速原型模型的“快速”本质对一些新产品在争取客户方面可以推荐利用。

快速原型模型的缺点是：所选用的开发技术和工具不一定符合主流的发展；快速建立起来的系统结构加上连续的修改可能会导致产品质量低下。

2.4　增 量 模 型

增量模型也称渐进模型，即软件采用渐进开发的形式进行，将软件分割成一些独立的增量部分，每个增量构件分别进行设计、实现、集成和测试，这样，当多个增量构件最终完成时，系统开发工作完成。增量模型如图 2-3 所示。

在增量模型中，每一个增量构件是由多种相互作用的模块所形成的提供特定功能的代码片段构成的，这个增量构件在系统中具有某种独立的功能属性，借助于该构件用户可以完成系统的某种功能需求，实际上，在增量模型中各个阶段交付的构件并不是一个可运行的完整产品，而是客户需求中一个可运行的产品子集，这样，当整个产品被分解成若干构件，开发人员逐个构件地交付产品时，开发人员可以较好地应对用户的需求变化，客户也可以不断地看到所开发的软件，从而降低软件开发风险。

显然，增量模型和瀑布模型在实现过程上是完全不同的，瀑布模型是一次提交完整的可以运行的产品，而增量模型则是逐次提交可以运行的功能，直至最终提交完成所有的功能组件，这种实现方式的优点如下：

(1)增量模型的第一个增量构件往往是软件的核心功能，它实现了软件的基本需求，而一些需求可能不够明晰的功能特征则留待下一个构件进行完成，这样，在第一个增量构件完成后，开发人员可以通过和用户的交流进一步增加对下一个构件的需求的理解，从而可以对

症下药地开发，避免盲目性，而一次性完成全部功能的设计实现则没有这样的机会和可能，在软件需求日渐复杂的今天，增量模型的这种特性对用户和开发人员来说都是非常有益的。

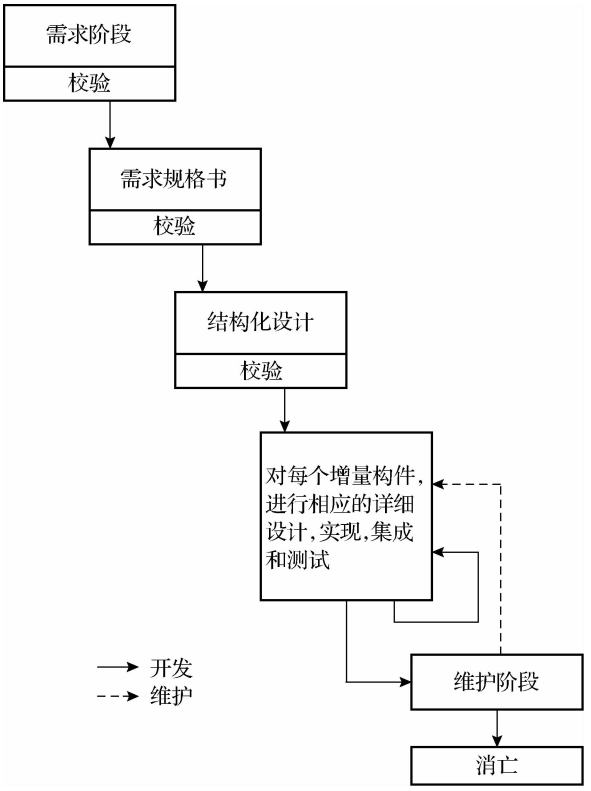


图 2-3　增量模型

(2)对开发企业来说，增量模型可以使企业更合理、更灵活地进行人员分配，特别是对于财力紧张的企业来说，也是一个不错的选择。如果项目是以模块来付款的话，当完成一个包含一个或多个模块的增量构件时，就会有一部分资金回笼，这样，企业可以不用投入大量资金，而只需循序渐进地完成项目即可。即使一些需求在开发过程中需要更改，只要这个增量构件足够小，其对整个项目完成的影响对企业来说也是可以承受的。

同样，增量模型也存在一定的实现困难，具体如下：

(1)由于各个增量构件是逐渐融合到已有的软件体系结构中的，加入构件必须不破坏已构造好的系统构件部分，因此，需要系统具备开放式的体系结构。

(2)在开发过程中，需求的变化是不可避免的。增量模型的灵活性可以使其适应这种变化的能力大大优于瀑布模型和快速原型模型，但也很容易退化为边做边改模型，从而使软件过程的控制失去整体性。

(3)增量模型存在潜在风险性。在增量模型中，系统的需求分析、技术规格书以及结构化设计顺序完成后，才开始各个增量模块的详细设计、编码和测试，这时，各个模块可能出现并行完成的情形，也就是说，第一个模块详细设计完成后，代码编写和测试随即开始，而设计团队可能立即开始第二个模块的详细设计，该设计完成后，代码编写和测试也随即开始，这样，同时开始设计、编码和测试模块的工作就有可能产生一种风险，即第一个模块的详细设

计出现问题，可能会产生大规模的软件混乱，因为后面的设计是和前面的设计相关联的，并且具有设计理念的一致性，所以，如果不对增量过程进行严格的管理，过程也存在较大的潜在风险，如图 2-4 所示。

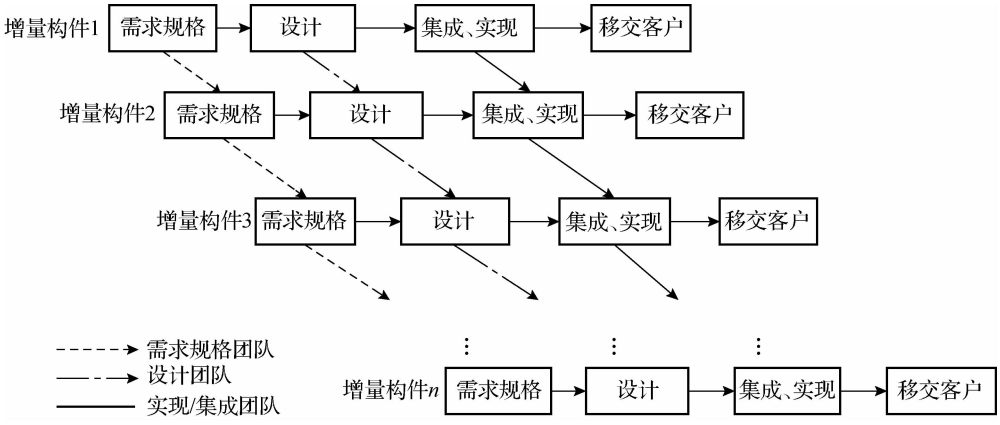


图 2-4　存在风险的增量模型

2.5　螺 旋 模 型

软件开发过程的一个重要步骤是风险评估，不仅在需求分析阶段应该进行风险分析，在软件过程的其他阶段也应该进行风险评估，只有这样，才能使软件开发过程中的风险最小化，并最大限度地提高软件产品质量。从快速原型的实现过程看，快速原型模型可以较大程度地降低需求分析过程中的风险，但是它不可能降低软件过程的其他风险。

1988 年，巴利·玻姆(Barry Boehm)提出了软件系统开发的螺旋模型，该模型将瀑布模型和快速原型模型结合起来，强调了其他模型所忽视的风险分析过程，它的简单描述即为在瀑布模型的每个阶段之前，先进行风险分析，如图 2-5 所示。

螺旋模型的基本思想是使用原型及其他方法来尽量降低风险。例如，通过原型开发，可以更好地理解用户的实际需求，同时通过对原型进行必要的性能测试，可以知道原型需要的约束条件能否满足，其要求的软件性能能否实现。

但是仍然有一些风险无法通过原型测评来降低，而且原型模型不适合于大型的复杂软件系统。螺旋模型的详细模型图如图 2-6 所示。其中，旋转角表示过程的螺旋形进展，螺旋的每一圈代表一个阶段，在阶段的起始期，确定阶段的目标，选择是否达到目标。经过这个阶段，得到达成目标的策略。接着从风险分析角度对策略进行分析，如果风险不能降低或者排除，那么可能立即终止项目，可能的情况下，即使项目继续进行，那么规模上也会大大减小，如果已经成功排除所有风险，那么启动下一个步骤，最后对该阶段的结果进行评估，并计划下一个阶段。

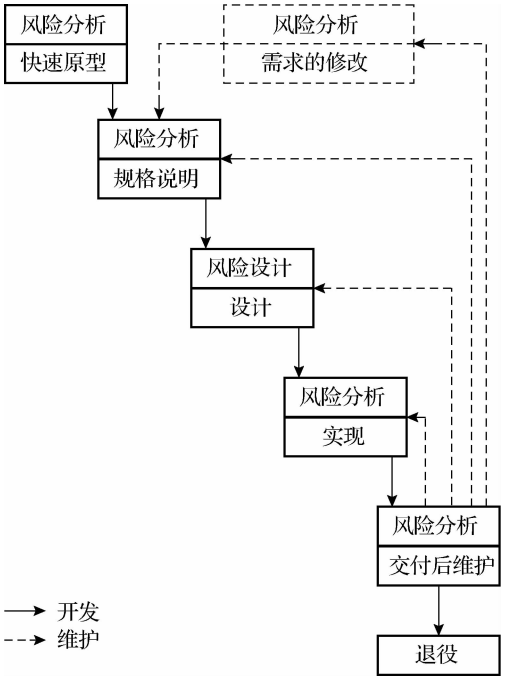


图 2-5　螺旋模型

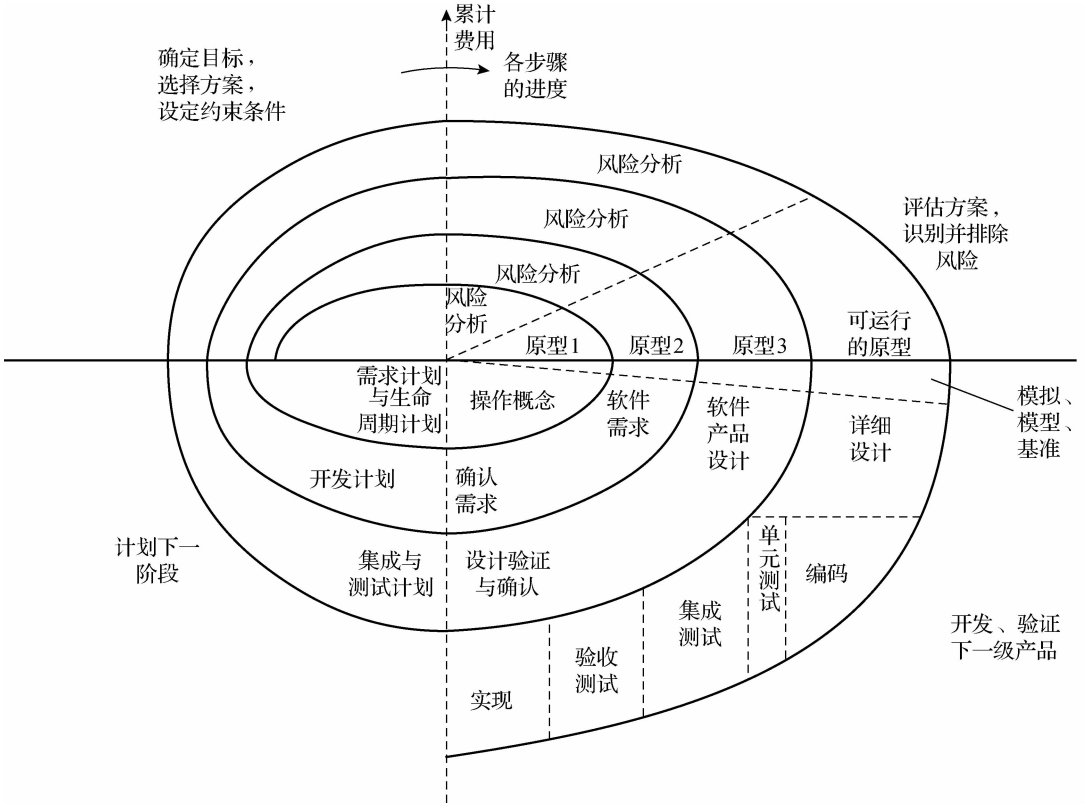


图 2-6　完全螺旋模型

显然，螺旋模型是一个风险驱动的软件开发模型，软件的每个增量或模块都用一个原型实现来最大限度地降低风险，风险排除后，即可进行一系列的软件增量活动，即进入软件过程的瀑布模型，最后进行评估并进入下一个软件增量。这样，软件过程沿着螺旋线每转一圈，即表示产生出一个更加完善的软件版本，在风险可控的情形下，过程沿着螺旋线向外逐渐延伸，最终形成一个用户满意的产品。

和其他模型相比，螺旋模型的主要优势在于它是风险驱动的，能够使开发者和客户较好地对待和理解每一个演化级别上的风险，特别适合于大规模的复杂软件系统的开发；当然，风险驱动也要求软件开发人员具有丰富的风险评估经验，否则当风险真正出现时，开发人员可能还认为一切正常，这将是最大的风险。

同时，螺旋模型也存在自身的问题，首先，过多的迭代次数会增加软件开发成本，延迟提交时间；其次，开发人员无法使用户确信演化方法的可控性，而且由于螺旋模型假定软件开发的各个阶段是相互独立的，而这对于大型的复杂软件系统几乎是不可能的事情，这些都是螺旋模型需要进一步改进的地方。

2.6　同步稳定模型

尽管前面所讲的各种软件过程模型各有其优点，并已经有许多成功案例，但是作为世界著名的软件公司，微软公司却独辟蹊径，开创了自己独特的软件过程模型，即同步稳定模型。

在同步稳定模型中的需求分析阶段，通过和众多潜在客户的交流最大限度地获取用户需求，而后将需求按照重要性分为不同的模块，这样将每个模块分为几个小的开发团队共同且并行完成。这里的同步即每天工作结束时，每个团队将当天开发的组件全部集成到一起，对集成后的产品进行测试，解决任何检测到的错误和问题，从而使当天的工作得到稳定和巩固，模块完成后，就是一个合格的产品模块。

对各个模块实行同样的同步开发策略，并定期执行部分已经开发成功的产品，可以使开发人员对系统更加了解，从而也可能对一些需求变动进行修改。

显然，这种开发模式要求很高，它不仅需要组织纪律性极高的研究开发团队，而且需要高效的软件工程管理团队，所以，它不一定适合每个软件开发组织。

至此，本章已经给出了几种不同的软件过程模型，并分析了它们各自的优缺点，在现实的软件项目中，由于项目的规模、复杂度等均有不同，因此，选择哪种软件过程模型要看具体情况而定，因为软件开发组织既要考虑软件的质量，也不能忽视公司的经济效益；既要考虑软件过程的完善和组织计划的严密，又要考虑软件本身特定的客户和环境，因此，在软件开发过程中，要根据软件自身的诸多特征选择适合的软件过程模型，瀑布模型、快速原型模型、增量模型、螺旋模型以及同步稳定模型的对比如表 2-1 所示。

表 2-1　几种软件过程模型的比较

生命周期模型	优　　点	缺　　点
瀑布模型	结构简单清晰、文档驱动，方法严谨	需求不确定时，交付的产品可能不适用
快速原型模型	可以快速实现一个可以运行的系统初步模型，确保最终产品满足用户需求	具有增量模型的缺点，同时最初的原型可能会被完全抛弃
增量模型	每一个增量均是一个可以运行的产品，投入人力、物力等资源较少	各个增量之间耦合度太大将导致过程难以控制、系统难以实现
螺旋模型	风险驱动，有益于软件产品质量提高，易于软件重用	仅适用于大规模的软件开发，且要求开发人员有风险评估经验
同步稳定模型	保证时间和进度的同步性，易于调动开发人员的创造性	要求开发团队的管理相当优秀，有待于在其他软件公司进行检验

本章小结

软件过程是软件生命周期中若干活动的集合，使用这些活动的目的是开发出高质量的软件产品，针对软件的规模、复杂性和应用场景的不同，如何进行软件活动的组织是软件过程模型的研究内容。本章论述了几种经典的软件过程模型，如瀑布模型、快速原型模型、增量模型、螺旋模型以及同步稳定模型，分析了这些模型的特性，并对这些模型进行了比较。

瀑布模型是一种规范化的文档驱动的过程模型，具有非常严格的过程评审过程，它适合于需求明确，对完成时间不太计较且具有高质量要求的软件，适合于国防类的软件开发。

快速原型模型的目的是通过快速建立软件原型，以获取用户真实的需求，从而实现软件的快速开发。尽管软件的早期版本可能遭到废弃，但是利用目前存在的大量软件组件，这种开发过程确实可行，对于一些需求都不太清楚或确定的新软件，它确实是一种快速占领市场的优选过程模型。

增量模型的特点是通过逐步开发软件的模块，逐次完成软件的开发，对于资金不足、人手短缺的开发企业，这是一种快速回笼资金、用较少人手换取较大效益的开发方法。

螺旋模型是一种风险驱动的软件过程模型，它适合于大型复杂软件系统的开发，当开发人员具有丰富的软件风险评估经验时，这种过程模型极易取得成功。

随着软件工程和软件过程研究的不断深入，目前人们也提出了多种新颖的软件过程模型，如敏捷软件开发模型，该模型注重团队内部的沟通与合作，更加注重和用户的沟通与协同，因此，这种模型也能够更加积极应对软件开发过程中用户需求的变化，并对这些变化作出相应调整和技术变化；另外一种更加灵活的开发模型是基于构件的开发模型，该模型支持复用构件库中的一个或多个软件构件，通过组合手段高效率、高质量地构造软件系统，虽然该模型的应用效率更多地依赖于构件库的健壮性，但是，这种开发模型可以更好地提高开发效率，它允许多个项目同时开发，能够大大降低开发成本，提高软件项目的可维护性，并可实

现分步提交软件产品。

本质上讲，没有任何一种软件过程模型可以适用于所有的软件项目，针对特定的软件，必须根据其特征进行软件过程模型的选择或者使用模型的组合来进行软件开发，软件过程必须科学、合理，才能真正开发出适合用户需求的、高质量的软件产品。

总之，通过本章的学习，可以对软件过程有一个清楚的印象，对经典的软件过程模型有一个较清楚的理解，能够在软件开发的过程中，根据软件规模和特点进行基本过程模型的选择和使用，通过不断对模型的理解和消化，进而能够对模型进行一些组合与优化，从而能够对软件过程进行更好的选择和控制，以开发出高质量与高性能的软件。

习　题　2

1)选择题

(1)软件生命周期的(　　)阶段是软件进行功能测试、性能测试以及是否满足用户需求的基础。

A. 可行性研究　　B. 需求分析　　C. 概要设计　　D. 详细设计

(2)在一个控制系统软件中，控制算法的数学模型至关重要，如果没有该模型，软件将无法完成，那么该算法的模型应该在软件过程的(　　)阶段给出。

A. 可行性研究　　B. 需求分析　　C. 概要设计　　D. 详细设计

(3)对于大规模的复杂软件系统，为了产生较高质量的软件，它最适合应用的软件过程模型是(　　)。

A. 瀑布模型　　B. 增量模型　　C. 快速原型模型　　D. 螺旋模型

(4)软件过程的增量模型要求，在把每个增量模块集成到现有的软件体系结构中时，必须不能破坏已经开发出的软件产品，这要求增量模型应该满足(　　)。

A. 软件体系结构的开放性　　B. 增量模块必须保持相互独立

C. 增量模型的模块不能并行开发　　D. 增量模型必须有相应的测试规定

(5)快速原型模型的本质是快速开发出原型系统，加快软件的开发进度，原型的目的是用于获取用户需求，因此，需求确定后，原型模型如果可以继续使用，则要求(　　)。

A. 原型模型注重考虑用户功能性需求

B. 原型模型注重考虑用户性能需求

C. 原型模型按照软件工程的思想进行设计

D. 原型模型可以充分利用组件开发

2)问答题

(1)瀑布模型作为经典的软件过程模型，在什么样的软件组织和软件特征下可以采用该过程模型进行软件开发?

(2)利用软件过程的快速原型模型可以很好地得到用户的实际需求，而这也是瀑布模型的缺点之一，试将快速原型和瀑布模型进行软件过程模型的组合，并给出模型的分析。

(3)在软件过程的生命周期中,哪个过程最长?给出你对这个过程的理解。

(4)对于一个小型软件公司,如果人手紧张,资金不足,哪种软件过程模型更适合于软件开发过程?试举例说明。

(5)螺旋模型为什么只适用于大型的复杂软件系统?软件开发人员如果不具备风险评估经验,最终可能会出现什么情况?

第 3 章 软件计划与管理

和任何工程项目的实施过程相同，软件项目开始前同样要进行项目的计划设计以及管理计划，不仅要进行资源需求计划制定、技术计划制定、质量计划制定，同样也要进行相应的各类风险估计与跟踪，包括人力资源风险、技术风险以及不可抗拒的自然和社会风险等。项目计划是否细致完善、项目过程能否完备跟踪、项目风险是否可测可控、项目质量是否可靠可行，这些都直接决定着软件项目的成败，所以在软件规模日益庞大，软件复杂性越来越高的今天，软件项目的计划与管理，是项目顺利实施的关键因素之一，它不仅是软件过程的开始，同时也会贯穿软件过程的始终。

3.1 计划的内容和目标

所谓软件计划，实际上就是在项目开始前的一系列估算工作，管理者和软件项目实施人员必须估算需要完成的工作，估算需要的人力资源、环境资源以及相应的资金和设备，同时要估计项目从开始到完成需要的时间、估计项目各个节点需要完成的时间、估计项目过程中可能出现的各类风险及其相应的应对策略。显然，由于这种估算是在预测未来，所以允许具有某种程度的不确定性。

3.1.1 项目计划的内容

软件项目计划包含的内容、软件过程以及软件的实施之间密切相关，这些内容计划不完备，则可能导致软件项目实施过程中出现问题，严重的将影响软件项目的完成与质量。计划内容详细表述如下：

1)软件范围

项目计划的首要活动就是要确定软件范围。范围阐述该项目进行的原因或意义，形成

项目的基本框架，使项目所有者或项目管理者能够系统地、逻辑地分析项目关键问题及项目形成中的相互作用要素，使项目相关责任人在项目开始实施前或项目相关文档编写前，能够就项目的基本内容和结构达成一致；项目范围应该能够列出项目成果清单，以便项目完成时作为对项目评估的依据。范围说明可以作为项目整个生命周期监控和考核项目实施情况的基础，还可以作为项目其他相关计划的基础。

2)项目进度计划

进度计划描述的是项目中各项工作的开展顺序、开始时间、完成时间及相互依赖衔接关系的计划。通过进度计划的编制，项目实施形成了一个有机的整体。进度计划是进度控制和管理的依据，可以分为项目进度控制计划和项目状态报告计划。在进度控制计划中，要确定应该监督哪些工作、何时进行监督、监督负责人是谁，用什么样的方法收集和处理项目进度信息，怎样按时检查工作进展和采取什么调整措施，并把这些控制工作所需的时间、人员、技术和物资资源等列入项目总计划中。

3)项目质量计划

软件质量保证计划针对具体待定的项目，给出质量保证人员的检查计划，软件过程的相关资源、规范、程序以及标准的使用计划。项目质量计划应当包括保证与控制项目质量有关的所有活动。

4)项目资源计划

确定项目范围计划和进度计划后，资源计划就是决定在项目的每项工作中用什么样的资源（人、软件、硬件、特定设备、相关信息以及资金等），在各个阶段使用多少资源，这既包括自身的内部资源，也包括可以使用的外部资源。

5)项目沟通计划

软件沟通计划就是制定项目实施过程中相关人之间信息交流的内容、人员范围、沟通方式、沟通时间或频率等，这些沟通要求的约定可以通过项目沟通计划来约束，也可以通过项目论坛的形式来体现。

6)风险对策计划

软件实施过程中难免存在各种各样难以预测的风险，因此，项目风险对策计划必须在项目开始前进行对策研究，该计划包括人员风险对策、资金风险对策、技术风险对策以及不可抗拒风险对策等。

7)项目采购计划

采购计划即软件实施过程中可能需要采购产品（包含软件、硬件、接口以及网络设备等）的时间、产品到达的过程控制等计划。

8)变更控制、配置管理计划

由于软件项目的特定性质，也由于计划无法保证过程完全和计划一致，因此软件过程的变更和管理显得非常重要，变更控制、配置管理计划就是要提出变更的程序以及控制手段，该计划能够给项目相关人员提供变更的手段和相应的变更依据。

3.1.2　项目计划目标

软件项目计划的目标是为管理者提供一个管理框架，使其能够对资源、成本以及软件过程的进度有一个合理的估算，这些估算可以包含“最好的情况”，也可以包含“最坏的情形”，这样可以使估算能够在一个合理的范围内。具体来说，项目计划应该能够达到如下基本明确的目标：

(1)项目模块之间的相互依存和相互独立性划分清晰。对于任何一个软件项目，当把任务进行划分时，必然会存在一些任务项目关联，而一些任务则可能相对独立。一些相互关联的任务可能需要顺序进行，而一些相互独立的任务则可以同时进行，因此，项目计划必须将任务分解明晰，这样可以有效提高时间利用率。

(2)不同类型人员的工作内容和工作时间划分明确。一个软件项目需要一定的人员在一定的时间内完成。当一个人分别参与多个项目时，这种计划时间约束就显得非常有必要，特别是在任务顺序进行时，必须计划好不同类型人员的时间任务。

(3)软件项目相关人员的责、权、利、罚划分仔细。项目能否顺利完成，很大程度上取决于项目参与人员的责任心，因此，项目计划必须明确相关人员的责任，即项目中谁负责哪些事情，出现问题找谁解决和处理；相关人员的权利，即在项目中具有哪些可以全权处理的事情，享受到的利益，项目人员保质保量完成相关任务应该享受到的利益；相应的惩罚，即项目相关人员在项目过程中因渎职、怠工或其他原因导致任务不能按期完成或者给公司造成损失而应该受到的惩罚。

(4)软件项目过程中的任务里程碑等事件划分分明。软件里程碑就是项目中的关键节点，项目计划必须能够将里程碑时间划分清楚，里程碑事件能否在规定的时间内完成有可能关系着整个软件工程的成败，因此，事件的时间和节点必须清晰，而且必须给予充分的人员和财务的保障。

3.1.3　项目计划实现方法

软件项目计划可以有多种表示方法，既可以用简单易懂的文字形式呈现，也可以用一目了然的图表形式展现。一些典型的方法如下：

1)甘特图表示法

甘特图是在 20 世纪初由亨利·甘特开发的，它基本上是一种线条图，横轴表示时间，纵轴表示要安排的活动，线条表示期间计划的和实际的活动完成情况。甘特图可以直观地表明任务计划在什么时候进行，以及实际进展与计划要求的对比。图 3-1 是用微软公司的 Office project 生成的 Gantt 图。

甘特图是对简单项目进行计划与排序的一种常用工具。用于解决负荷和排序问题时较为直观，它能使管理者先为项目各项活动做好进度安排，然后再随着时间的推移，对比计划进度与实际进度，进行监控工作。调整注意力到最需要加快速度的地方，从而使整个项目按期完成。

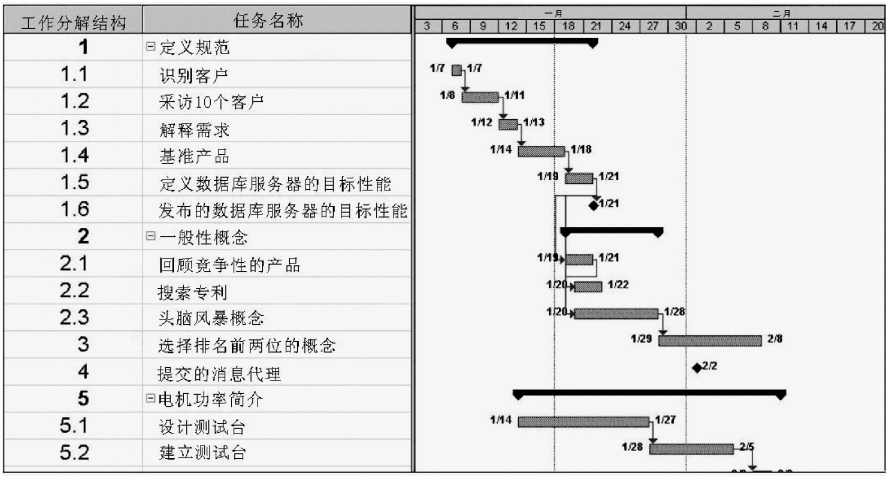


图 3-1　用微软公司的 Office project 生成的 Gantt 图

首先，应用甘特图要求把项目分解成许多离散的却非常具体的活动。“离散”意味着每项活动都有明确的开头和结尾。项目分解活动的完成，同时也意味着这些软件过程活动顺序的确定。虽然不同的项目管理人员可能会给出不同的顺序，但是有经验的项目管理人员会全盘考虑项目中的各个综合因素以及项目中多种因素对项目完成的利害关系，并最终确定采用哪个顺序。

其次，应用甘特图要求对每项活动作出时间估算。如果项目的工期已经明确，意味着每项活动要花费多少时间必须有切实可行的估算根据，虽然这不一定能够确切实现，但对管理项目进行估算有实质性的帮助。

甘特图表示法具有简单、直观、容易掌握和应用的特点，但是作为一种计划的表示方法，它也并非完美，其存在的问题如下：

(1)任务可能和人不相关，这就需要一套完善的解决方法。

(2)当依赖关系不明确时，需要设计另外的解决方法。

(3)图中无法显示一个人的负荷情况。

(4)软件过程中的关键路径无法明确表示。

(5)无法记录原计划和实际情况的不同点。

上述问题对于一个具体的软件工程项目来说重要性可能不尽相同，但是有些确实是非常重要的，例如，软件过程中的关键路径在软件实施过程中就是非常重要的内容。

2)波特图表示法

波特图(project evaluation and review technique，PERT)表示法是利用网络分析制订计划以及对计划予以评价的技术，又称为计划评审技术。该方法能协调整个计划的各道工序，合理安排人力、物力、时间、资金等资源，以加速项目计划的完成。在现代计划的编制和分析手段方面 PERT 表示法是现代项目管理的重要手段和方法，已被广泛使用。

图 3-2 所示是一个波特图，其中节点代表事件(这里用数字 1～8 表示)，当任务完成时发生；连接事件的线代表以日程时间(以天计算)标记的任务(活动)；粗线显示的是关键路径。

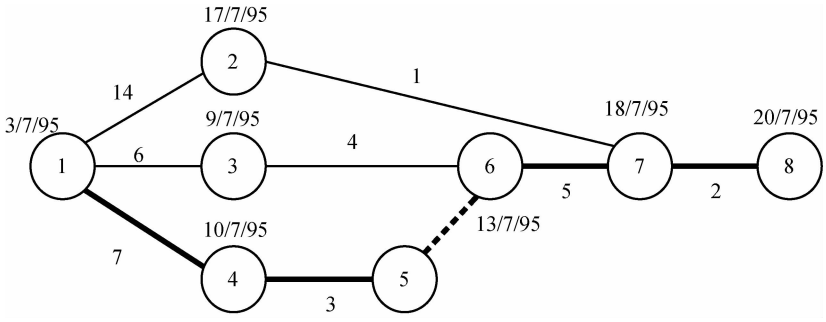


图 3-2　波特图

在波特图中，一个项目网络是由一组称之为节点的圆圈和线组成的，线用于把节点连接起来，以给出活动顺序的形象表达。其中，节点代表项目活动，而线表示活动顺序，从而在任务之间得出了明确的依赖关系，此外，任务的顺序通过从左到右的连接流给出，但从时间上来说，水平轴不一定是线性的。

一个完备的波特图有三个要素：事件、活动和关键路线。事件表示主要活动结束的那一点；活动表示从一个事件到另外一个事件的过程；而关键路线是波特图网络中花费时间最长的事件和活动的序列。图 3-2 所示的波特图使用了一系列连接在一起的节点，从而使任务更加明晰。

波特图可以比甘特图更紧凑，但是需要以一个线性的时间尺度为代价才能做到。还可以将任务所需的时间资源逐个列举出来，但是当需要快速查看哪些方面使用了大部分时间资源时，波特图会变得很困难。

图 3-3 是一个简单的软件工程波特图，图中的每个圆圈表示开发工程中的一项具体任务，圈内的数字表示完成该项具体任务所需的时间（单位为星期），圆圈之间的箭头表示各项任务完成的先后顺序和相互依赖关系。例如，制订测试计划要用 4 周，进行软件测试要用 12 周，而只有在测试计划制订好后才能开始软件平台测试和数据测试，并且只有当数据测试和软件平台测试都完成后，才能进行产品测试等。

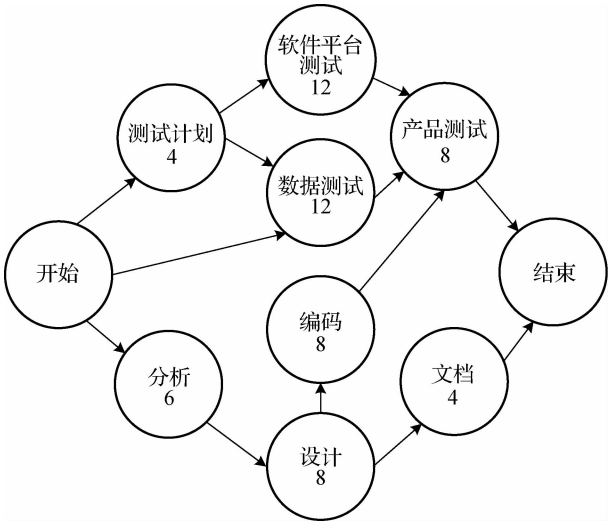


图 3-3　一个软件项目的波特图

3)关键路线法

关键路线法(critical path method,CPM)是应用网络模型描述工程项目所有活动的内容和顺序关系,并据此选择最优计划方案的一种计划管理方法。它是美国杜邦公司在1957年研究成功的。利用关键路线法能在网络模型上直观地分析大型工程项目所需时间和费用的关系,从而找到缩短工程日期和节约费用的关键所在。

CPM图和波特图类似,但包括一个对于"关键路径"的明确指示,该任务序列定义了项目中的最低时间量。换句话说,这些任务的延误会导致整个项目的延误。一个或多个这样的任务序列是一直存在的;CPM图使这些路径(通常只有一个)变得明确。否则,CPM图就具有和波特图同样的长处和劣势,因此,两者往往集中在一块形成一种技术。

对于复杂的、时间紧迫的项目,为了保持项目的进度,CPM图可能提供了必需任务关键路径的一个清楚提示。图3-4是一个典型的CPM图。

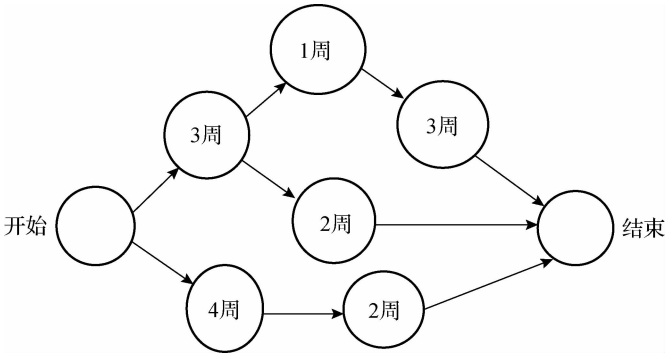


图3-4 CPM图

3.2 成本估算与跟踪

软件成本是项目进行之前必须要考虑的因素,客观上,软件成本可以分为内部成本(internal cost)和外部成本(external cost)两种。内部成本主要包括开发团队、项目管理人员以及项目相关支持人员的薪金,开发需要的软、硬件成本以及一般的管理费用,如房租、水电等。外部成本是指软件产品报给客户的价格,显然,外部成本应该是内部成本与企业利润的总和。

软件成本贯穿于软件开发过程的始终,包含软件计划、需求分析、设计、编码、测试、集成到维护全过程的人力、物力消耗。由于软件项目开发过程中有太多的不确定因素,所以项目初期就要进行成本估算,以期提供一个可以接受风险的估算成本。一位总经理曾经被提问:在选择一个项目管理者时,什么特质是最重要的?他的回答是"具有在错误真正发生之前就能够预先知道的能力"。实际上,这也意味着,对于一个管理者应该"在未来还是一团迷雾的时候,具有实施估算的勇气和能力"。

在估计软件成本时，由于未来诸多不确定因素都可能对项目产生不可估计的影响，因此估算本身具有与生俱来的风险，而这种风险也导致了项目的一些不确定性。一个基本的软件成本估计框架如图 3-5 所示。

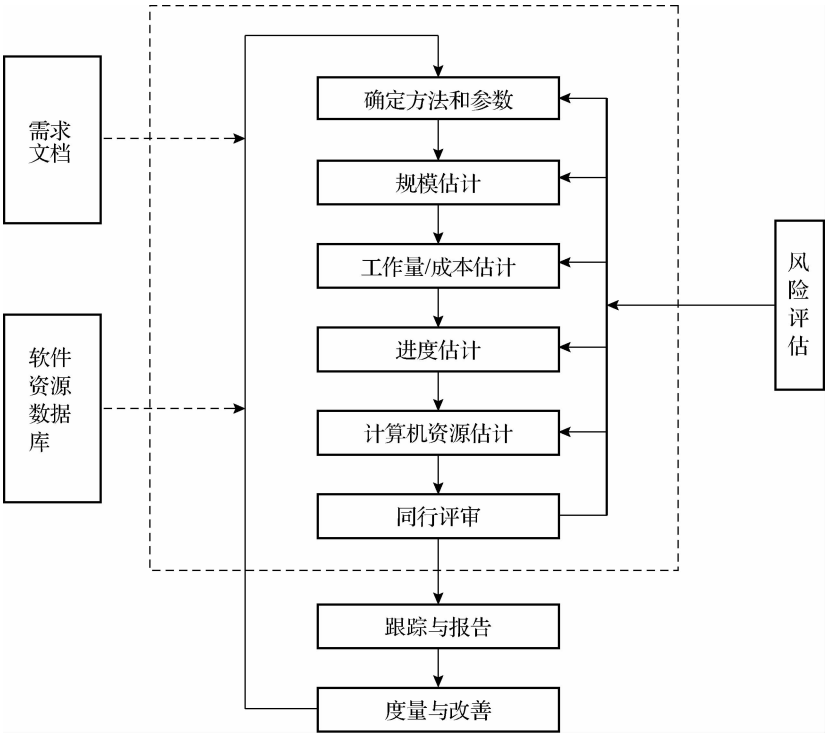


图 3-5　成本估算框架

3.2.1　项目分解成本估算方法

如果以项目本身作为一个整体来考虑成本，成本估算有时显得过于复杂，因为项目本身的复杂性、不确定性以及项目结构的不确定性程度等都将对估算本身产生重要影响，因此将项目进行分解，将问题细化，可以得到较好的估算方法。

但是无论如何，要想精确估计软件成本几乎是一件不可能完成的事情。其中最重要的因素是人的因素，不难理解，在一个软件项目中有经验的程序员和一个编程新手的产出效率会有相当大的差别，这就会导致项目成本和完成周期出现很大不同；同样，项目进行过程中关键技术人员的离职也会对软件成本的估计造成巨大的偏差。

那么，衡量人员产出效率的软件成本的研究中，软件规模如何度量呢？这是软件度量备受关注的研究内容之一，目前软件规模度量方式有代码行度量和功能点度量两种基本方法。

1）代码行度量

代码行（lines of code，LOC）和千行交付的源代码指令（thousands of delivered source instructions，KDSI）是软件度量最简单的方法之一，该方法的基本思想是用软件程序的源代

码行数(注释行除外)作为程序复杂性的度量。显然,随着软件系统规模的日益庞大和软件编程技术的快速发展,代码行度量也存在着如下一些弊端:

(1)代码行度量无法适应快速发展的程序设计语言。代码行度量初期,程序设计语言比较单一,因此用该方法度量是可行的,但是随着编程语言的快速发展,不仅出现了面向对象的程序设计语言,而且又有了用户界面良好、非过程化程度高、面向问题的第四代编程语言,而这些语言既包括程序代码,又可能包含声明语句、注释语句以及宏指令,这就对代码度量非常不利,因为不同语言编制的程序可能代码量相差很大。例如,用汇编语言编制一个功能可能需要 50 条指令,而用 C 语言可能只需要 8 条指令,这种差别并非用代码行就能够度量的。

(2)没有可以接受的统一的代码行标准。对代码行没有公认的可接受的标准定义。不同的公司、软件行业机构对编程标准有不同的要求,并且这些要求并不公开。例如,有的程序代码中有空代码行、注释代码行、数据声明行以及复用的代码行等,同样程序中也存在删除和修改的代码行,这样若使用代码行计数,则显得非常复杂,同样,面向对象的程序设计中类的继承也存在同样的问题,所以即使对同一个软件产品的代码行计算,不同的计算方式也可能带来数倍的差异。所以利用代码行度量法在不同机构之间进行生产率的比较是不可行的。

(3)某些代码行的信息难以确定。代码行适用于软件开发阶段,但是在项目早期,由于需求不稳定,设计不成熟,实现技术和手段都不确定,所以准确地估算代码量非常困难。

对于软件的代码行度量,虽然美国软件专家 Capers Jones 称“使用代码行数进行涉及多种语言和生命周期活动的生产率研究,应该被认为是一种职业的不良习惯”,但是对于使用单一化编程语言的软件来说,它仍不失为一种有效的选择。

2)功能点度量

功能点度量由 Albrecth 和 Gaffney 于 1979 年提出,用于对软件规模进行评定、评估和度量,它是一种面向功能的软件度量方法。

传统的功能点是从软件的需求分析入手,将软件功能分为五类,即外部输入 EI(external input)、外部查询 EQ(external query)、外部接口文件 EIF(external interfaces or files)、内部逻辑文件 ILF(internal logic file)和外部输出 EO(external output),并分别给予这些功能不同的复杂度,最终计算出软件的功能点数目。五种功能的定义如下:

(1)外部输入。计数每个用户的输入,对终端用户的输入进行处理。

(2)外部查询。计数终端用户的查询请求,软件以联机输出的形式产生实时响应。

(3)外部接口文件。计数为了和外部系统进行数据交换而使用的接口文件。

(4)内部逻辑文件。计数逻辑主文件,包括顺序文件、数据库文件以及临时文件等。

(5)外部输出。计数每个用户输出,一般指报表、屏幕、出错等信息,用于向用户提供应用的信息。

一般地,上述功能的复杂度分为三种,即低、中、高,功能点度量的复杂度如表 3-1 所示。

表 3-1　功能点度量的复杂度

功能要素	复杂度		
	低	中	高
EI	3	4	6
EQ	3	4	6
EIF	5	7	10
ILF	7	10	15
EO	4	5	7

完成了各个功能要素的计数后，将它们分别乘以各自的复杂度因子，即可得到整个项目无调整的功能点(unadjusted function point，UFP)，即总计数值。最后采用下式计算出功能点(FP)：

$$FP=UFP\times TCF \tag{3.1}$$

式中，$TCF=0.65+0.01\times\sum F_i$，$TCF$ 是技术复杂度因子，它是对 14 个技术复杂度因子效果的测量值，F_i($i=1\sim14$)是基于对下列问题的回答而得到的复杂度调整值：

(1)系统需要可靠的备份和恢复吗?

(2)需要数据通信吗?

(3)有分布处理功能吗?

(4)性能很关键吗?

(5)系统是否在一个现存的、重负的操作环境中运行?

(6)系统需要联机数据登录吗?

(7)联机数据登录是否需要在多屏幕或多操作之间切换以完成输入?

(8)需要联机更新主文件吗?

(9)输入、输出文件或者查询很复杂吗?

(10)内部处理复杂吗?

(11)代码需要被设计成可复用的吗?

(12)设计中需要包括转换及安装吗?

(13)系统的设计支持不同组织的多次安装吗?

(14)应用的设计方便用户修改和使用吗?

这些调整值范围在 0～5 之间。其中，0 表示不存在或者是没有影响，5 表示自始至终都有重大影响。当然，对于各个功能要素复杂度的确定，仍然难以克服主观倾向。

3.2.2　经验估算模型

软件成本的另外一个估计方式是通过经验算法模型来实现的，这种模型的基本指导思想是找到软件工作量的各种成本影响因子，并判定它对工作量所产生影响的程度是可加的、乘数的还是指数的，以期得到最佳的模型算法表达形式。当某个因子只影响系统的局部时，一般说它是可加性的。例如，给系统增加源指令、功能点实体、模块、接口等，大多只会对系统产生局部的可加性的影响。当某个因子对整个系统具有全局性的影响时，则说它是乘数的或指数性的，例如，增加服务需求的等级或者不兼容的客户等。

作为一种典型的成本估算模型，构造性成本模型(constructive cost model，COCOMO)于1981年由Boehm提出，它包含三个基本模型，即将产品作为一个整体估计的宏观模型(macro-estimation model)、中等模型(intermediate model)和从细节处理产品的微观模型(micro-estimation model)，这些模型将软件开发所需要的工作量表示为如下软件规模和软件开发工作量因子的函数：

$$Effort = a \times (size)^b \times \prod_{i=1}^{n} EM_i \tag{3.2}$$

式中，$Effort$ 是工作量，单位为人/月；a 和 b 是两个常数，取值与软件规模、系统复杂度有关；$size$ 是软件源代码行数，以千行为单位；EM_i 是工作量因子，这些因子根据产品的复杂度划分为15种，取值如表3-2所示。

表3-2　COCOMO模型的复杂度因子

项目特征	复杂度级别	复杂度					
		非常低	低	额定	高	非常高	特别高
产品属性	要求的软件可靠性	0.75	0.88	1.00	1.15	1.40	
	数据库规模		0.94	1.00	1.08	1.16	
	产品复杂度	0.70	0.85	1.00	1.15	1.30	1.65
计算机属性	执行时间限制			1.00	1.11	1.30	1.66
	主存限制			1.00	1.06	1.21	1.56
	虚拟机的变更性		0.87	1.00	1.15	1.30	
	计算机周转时间		0.87	1.00	1.07	1.15	
人员属性	分析员能力		1.46	1.19	1.00	0.86	0.71
	应用经验		1.29	1.13	1.00	0.91	0.82
	程序员能力		1.42	1.17	1.00	0.86	0.70
	虚拟机知识		1.21	1.10	1.00	0.90	
	编程语言经验		1.14	1.07	1.00	0.95	
项目属性	现代编程实践的使用	1.24	1.10	1.00	0.91	0.82	
	软件工具的使用	1.24	1.10	1.00	0.91	0.83	
	要求的开发进度表	1.23	1.08	1.00	1.04	1.10	

而 a 和 b 的取值则依赖于软件的组织模式，每种模式对应的值见表3-3。

表3-3　不同开发模式的参数表

开发模式	工作量 $Effort = a \times (size)^b \times \prod_{i=1}^{n} EM_i$
组织型	a=3.2，　b=1.05
半独立型	a=3.0，　b=1.12
嵌入型	a=2.8，　b=1.20

关于如何确定软件项目的工作量因子，Boehm 给出了一些建议性的准则，以提供一个可以确定有关参数的指导，具体如下：

(1)如果软件模块的控制操作主要由一系列结构化编程的构造(如 if-then-else，do-while，case 等)组成，那么复杂度可以定为“非常低”。

(2)如果模块间的操作是嵌套操作，那么级别可以定为“低”。

(3)如果软件具备模块间的控制和决策表的加入，可以将级别定为“额定”。

(4)如果模块间的操作是高度嵌套的，带有组合断言，包括队列和堆栈，那么级别可以定为“高”。

(5)如果软件出现可重入和递归编程以及固定优先集中断处理，那么级别可以定为“非常高”。

(6)如果软件使用可动态改变优先级的多个资源的调度和微代码级的控制，那么级别可以定为“特别高”。

由于 COCOMO 在提出初期只有瀑布模型存在，而随着软件技术与规模的不断发展，其计算精度也逐渐减小，于是 Boehm 等人于 2000 年提出了 COCOMO Ⅱ，该模型支持各种软件的生命循环模型，COCOMO Ⅱ和早期 COCOMO 的主要区别如下：

(1)COCOMO 只包含一个基于代码行数(KDSI)的总模型；而 COCOMO Ⅱ为软件过程的不同阶段提供了三个不同的模型系列，即应用于早期工作流阶段的应用组合模型(application composition model)、应用于体系结构设计阶段的早期设计模型(early design model)和应用于软件开发阶段的后架构模型(post-architecture model)。

(2)成本因子设计不同。在 COCOMO 中，常数 b 取决于产品制造的模型，只取三个不同的数值；但是在 COCOMO Ⅱ中，模型常数 b 由 5 个属性决定，即先例性 PREC(precedentedness)、开发灵活性 FLEX(development flexibility)、早期体系结构/风险化解 RESL(architecture/risk resolution)、团队凝聚力 TEAM(team cohesion)、过程成熟度 PMAT(process maturity)。每个属性都划分为从很低到特高 6 个级别，最后由下式确定 b 的值：

$$b = 1.01 \times 0.01 \times \sum_{i=1}^{5} W_i \prod_{i=1}^{17} f_i \tag{3.3}$$

显然 b 的取值范围为 1.01～1.26，该模式划分更加精细和灵活。

(3)成本驱动因子的不同。在 COCOMO 中，成本因子只有 15 种，而在 COCOMO Ⅱ中，成本因子为 17 种，而且其中的 7 种相对于 COCOMO 是全新因子。

3.2.3　成本跟踪与控制

成本计划后进行跟踪与控制是非常必要的，因为只有通过将项目实际开发工作量和估计值进行比较，才能不断发现问题，进而解决问题。如果实际发生情况和预测估计情况发生较大偏差，必须及时调整来减少这种偏差，弱化由于这种情况造成的不良影响，以保证软件项目目标的顺利完成。

成本跟踪同样贯穿于软件项目的所有过程，它不仅跟踪成本的去向，同样也要控制软件进度是否和软件成本成比例，如果成本符合项目初期的预算，就不必进行调整与控制；反之，如果成本超出预期，就必须根据当初定义的目标进行控制以解决问题。否则到了时间节点，

产品虽然完成了但质量和成本均达不到要求，项目仍然是失败的。

有了监控但项目仍然有可能延期，或者说仍然有可能达不到当初定义的质量和成本要求，问题的根源可能在于只跟踪并没有控制，即只是发现问题而没有寻找问题的根源和解决方法，只应急处理问题而没有提前观察各种征兆。实际上，这些问题是监控中最常见的现象，必须避免。跟踪控制应该解决如下问题：

(1)任务本身的成本估算问题。成本出现偏差首先要考虑工作量的估算是否合理，是否考虑了工作中存在的技术难点，是否考虑了项目成员自身的技能、工作效率和沟通能力，是否考虑了其他应该考虑的风险。虽然任务计划下达给项目成员时可能已经获取了承诺，但很多时候承诺本身是无用的，是否可以承诺或者是否能完成承诺与项目成员的个人经验和技能有很大的关系。

当偏差出在估算上，而且后续项目都是采用相同估算模式的情况时，必须调整项目计划。对于短期型的项目，如果这时才发现是项目成员技能的问题，那么想通过培训来提高技能以取得立竿见影的效果往往是不现实的。

如果项目任务中存在技术攻关或技术难点需要解决，工作量往往是很难估计的，而且一旦无法攻破技术难题，将对项目进度产生致命影响。所以，项目估算时，必须进行风险分析和应对，提前对技术问题进行预研和攻关，开发原型，积累相关的知识和经验。

成本估算出现问题的根源可能在于对历史项目或版本的数据采集和分析不够，准确的估算依赖于专家的经验，但专家的经验同样依赖于历史项目和历史数据。估算问题的另外一个根源也可能在于对项目成员技能和生产率水平没有较清楚的认识，一个软件类任务的完成往往由于个人技能而产生巨大的生产率差异和进度差异。

(2)任务本身的粒度问题。对于一个软件项目，出现 1～2 天的偏差很容易得到纠正。而如果出现 1～2 周的偏差就很难再进行纠正。任务本身的粒度直接影响着成本估算的偏差大小。若任务本身的粒度较大，则不适宜进行估算跟踪，任务本身是否会出现偏差已经无法取决于跟踪者，而是取决于执行者对于大粒度的任务是否有很好的任务细分和自我控制能力。

对于一个具体的任务，如果进度偏差最多允许一周，那么需要把任务粒度细化到周，按周来进行跟踪；如果最多允许偏差 1～2 天，那么需要把任务粒度细化到天，按天来进行跟踪。细粒度的跟踪目的就是要消除不确定性因素和风险，尽早发现任务中的问题，这样才可能有时间来解决问题和纠正偏差。

如果项目任务的粒度较大，而任务内部本身也要求进行跟踪，那么此时跟踪一般只能由项目成员自己进行。对于粒度大的任务，如果任务没有细分，成员反馈的任何完成 40％或者 70％的任务百分比都是不可靠或主观的数据，因为项目成员的自我监控能力对进度是否存在偏差有重要的影响，在这种情况下，任务是否能够按期完成对于项目经理来说是不可控的，因此项目经理必须对成员有充分的了解和信任。

(3)项目经理对业务和技术领域不熟悉。对于项目经理来说，对业务和技术领域的熟悉是必须具备的工作能力。有了这些能力才可能和项目组一起得出比较好的项目分解和任务工作量的估算；有了这些能力才可能实现细粒度任务的划分，并定义清晰的出入口准则。

在项目的跟踪过程中体现较多的往往是项目管理能力，但在项目控制过程中体现更多的则是业务和技术能力。控制的目的是真正发现问题的根源，解决问题并纠正偏差。

如果项目管理人员只是一味地对项目过程进行跟踪，发现问题但不能及时找出问题的根源，那么项目有可能深陷泥潭。如果项目管理者不仅进行跟踪，而且出现问题时，在第一时间介入任务或问题本身，挖掘问题根源，调整任务粒度，改进监控方式，就可以使项目及时得到纠正，问题及时得到解决，从而使项目走向正轨。但是，对项目的跟踪与控制技能需要管理者不断地积累知识和经验，不断地提高自己的业务和技术能力，这样才可能成为一个合格的项目管理者。

3.3 人 员 管 理

一个软件项目除了必须具备高素质的软件开发人员外，同样也必须具备优秀的项目管理人员，如何对项目参与人员进行合理有效的管理，最大限度地规避人员风险是软件项目管理的一个重要内容。

由于软件项目的规模越来越大，因此项目组人员间的分工合作也越来越细致，即使是一个模块，也有可能需要一个项目团队来完成，但是由于项目组人员间的能力不同、分工不同，因此，必须有合理的组织方式才能使团队高效运行。例如，如果编码工作量需要 1 名程序员 12 个月才能完成，那么理论上讲，4 名程序员 3 个月也可完成该工作量。但是事实上并非如此，由于程序员之间进行编码实现需要进行沟通和交流，而这需要占用大量的时间；同样当设计工作已经完成，但还没有将任务完全分发给整个开发团队时，人员合作的效率也会非常低，这与程序员本身的能力无关。

因此，一个优秀的团队组织与管理形式非常重要，它不仅可以高效地管理一个软件项目团队，而且能够充分发挥软件开发团队的最大潜能。当然，针对不同的项目背景、项目类型分别应用不同的团队组织形式也是非常必要的。也就是说，任何软件项目的管理者不仅要清楚人是软件项目中的决定性因素，而且必须具备针对不同软件项目组织不同团队的技能，否则，在软件项目管理上就不容易取得成功。

3.3.1 项目参与者

对于一个软件项目，参与的人员一般可以分为以下 5 种：

(1)高级管理者。负责对项目的全局和关键节点进行掌控，对项目的主要业务问题进行技术和管理决策，是项目成败的关键指挥者。

(2)项目管理者。执行高级管理者的决策，对软件项目制定相应的计划、组织和控制；对项目组相关人员进行管理和激励。

(3)开发人员。在管理者的计划和控制下，应用专门的技术进行产品的开发和设计。

(4)客户。负责说明待开发软件需求的人及其他风险的承担者。

(5)最终用户。软件提交给客户后，最终使用软件产品并和软件直接进行交互的人。需要说明的是，客户可能就是最终用户，但最终用户可能不是客户。

3.3.2 人员管理模型

对于参与软件项目的成员，不论是项目管理者还是软件技术高手，都应该明白一点，即任何人单凭一己之力已经很难应对日渐复杂的软件项目，加入或者建立一个合适的软件开发团队不仅可以更好地展现软件高手的技术技能，而且可以使管理者能够高效管理一个团队，生产出高质量的软件产品。

对于管理者，更应该清楚的是，给一个已经落后于计划进度的团队增加成员，不仅不会增加项目的进度，而且可能会使进度更加落后，这个著名的 Brook 法则已经成为项目管理者的共识。目前，就编程团队的组织形式而言，经典的解决方式有两种，即民主团队组织形式和主程序员团队组织形式，除此之外，还有其他几种组织形式，分别介绍如下：

1)民主团队组织形式

一般地，程序员都喜爱自己的程序，将自己的程序作品看做生命中的杰作，甚至生命的延续，所以，尽管他们知道程序中存在这样或者那样的问题，但也不乐意其他人说程序的缺点，或者批评程序。

针对上述情形，Weinberg 于 1971 年提出了民主团队的组织形式。该组织形式的基本原则是无我编程(egoless programming)，Weinberg 希望在民主团队组织形式里，程序中出现 bug 或者错误是非常正常的事情，并非不可原谅，也希望大家一起合作找出程序中的问题所在，大家应该认为程序属于这个团队，而不是属于个人。民主团队组织形式的特点如下：

(1)团队内部人员的地位平等，没有绝对的技术权威，项目的技术决策由大家共同策划完成，队内的通信是平行的。

(2)由于团队对程序中出现的错误有清醒和正确的认识，因此队内成员乐意请大家协助发现并解决问题，这种积极认真的态度可以产生高质量的软件产品。

(3)由于有轻松的工作氛围，有群策群力的工作态度，因此这种团队组织形式有利于进行科技攻关，解决技术难题。

(4)如果有技术能力出众的成员，这样的团队组织往往有较高的工作效率，也能够生产质量一流的软件产品；反之，如果大家的技术都平庸，没有一个绝对的权威做指导，也可能造成软件项目的夭折。

因此，承担科技攻关项目的国家大型科研院所以及国防科研部门等，由于大多没有特定的时间和空间限制，因此多数采用这种团队组织形式进行科研和生产。

2)主程序员团队组织形式

1972 年，Mills 提出了主程序员团队的概念，其团队结构如图 3-6 所示。

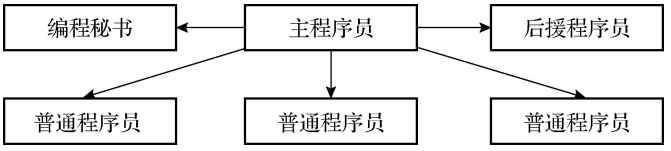


图 3-6 主程序员团队结构

显然，这种团队组织形式虽然包括六名程序员，但是他们的二人沟通渠道却只有 5 条，而在民主团队里，这样的沟通渠道需要 15 条。主程序员团队组成人员有主程序员、后援程序员、编程秘书以及 1～3 名普通程序员，职责分别介绍如下：

(1)主程序员。负责团队管理、技术设计以及技术指导。普通程序员之间不进行沟通交流，主程序员负责给他们提供详细设计和编码指导，同时负责检查和评审每个人的工作，因此它不仅应该是优秀的程序员，也应该是具有丰富实践经验的管理者。

(2)后援程序员。几乎具备与主程序员同样的技术与管理技能，因此在对项目的熟悉和把握程度上，也几乎和主程序员等同，其主要负责测试用例计划以及其他独立于设计过程的任务。当主程序员由于健康或者其他原因不能出现在岗位上时，后援程序员是主程序员位置的最佳人选。

(3)编程秘书。这个岗位也是该组织的一个重要工作岗位，编程秘书负责维护项目产品库以及项目文档，并负责源代码的编译、链接、加载、执行以及运行测试用例。因此，这里的秘书是技术娴熟的程序员，是团队中的重要成员，而非普通意义上的秘书。

(4)普通程序员。负责代码输入、编辑、编译以及测试等，不需要做其他工作。

显然，主程序员团队组织的运行更多地依赖于一个具有良好管理经验与超高编程技能的主程序员，也正是基于这个原因，《纽约时报》项目在基于主程序员团队组织的运行下取得了巨大成功。但是，在现代社会中，这种组织形式也暴露出了明显的弊端，主要体现在如下几点：

(1)主程序员难寻。在现代社会，找到一个技术出众的人才也许并不难，但是找到一个技术精湛且善于管理的人才却并不容易，因为技术和管理属于两个不同的专业范畴，极少有人做到在两个专业领域都应对自如，游刃有余。

(2)后援程序员难当。姑且不论后援程序员的能力问题，单论后援程序员的待遇问题，就很少有人愿意担当这样一个薪水低、责任大的角色。

(3)编程秘书苦不堪言。喜欢做程序员的人主要是喜欢自己创造性的工作，当将一个需求变成作品时，那种欣慰是无法形容的，同时程序员最不喜欢做的就是事务性的工作，如编写大量的项目相关文档等，但这正是编程秘书的工作，因此少有人乐意从事这类工作。

3)现代团队组织形式

前已述及，民主团队的最大优势在于团队每个成员均有积极向上的心态，他们勇于发现自己或者别人代码中出现的问题，这非常有助于形成高效和高质量的代码，然而，这种优势在主程序员团队是无法实现的，因为主程序员负责代码质量，并依此对程序员进行评价与考核，这样，程序员就会产生心理芥蒂，从而不愿意发现自己或者别人代码中的错误。

鉴于民主团队组织形式和主程序员团队组织形式各自具有的特点，将二者进行融合，则是现代软件开发团队应该采取的主要方式。

因此，为了使程序员更加积极主动地发现代码中存在的问题，团队的管理者应该分为技术主管和行政主管，两者各自主管不同的工作，具体说来就是，技术主管只负责技术范围内的事情，如代码审查；而行政主管则负责程序员的绩效考核，不参与代码审核与技术事务的处理。团队结构如图 3-7 所示。

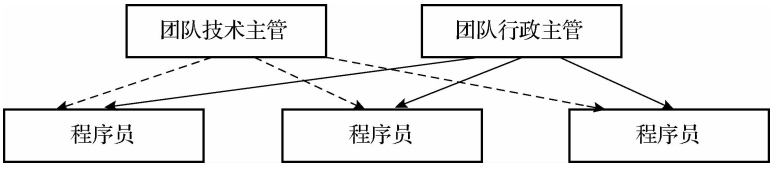


图 3-7　现代程序员团队结构

当然，软件开发小组的团队人数比较少，当遇到的软件项目规模比较大时，可以将这种组织形式进行扩展，基于团队技术主管的组织结构扩展图如图 3-8 所示。

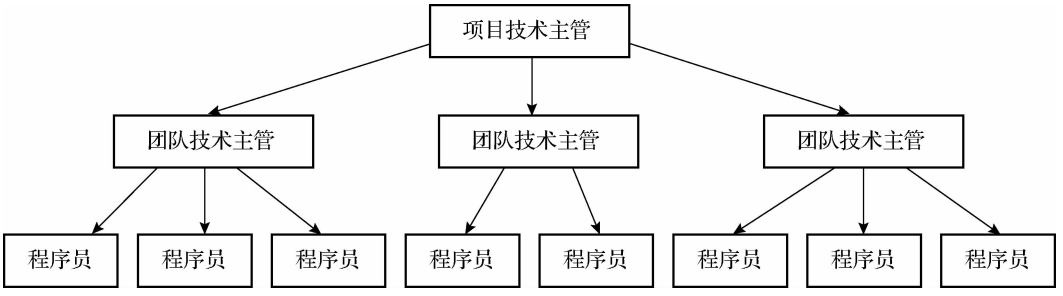


图 3-8　大型项目的技术管理组织结构

同样，非技术方面的基于项目行政主管的组织结构扩展也可类似得出。如果遇到更大规模的软件，同样可以扩充组织结构的层次。

这种融合了民主团队组织形式和主程序员团队组织形式优点的现代软件开发团队组织形式可以更进一步扩展其优势，使管理层内部和技术层内部的交流与沟通成为可能，这种能够集思广益的分散式决策过程，有益于发挥各级人员，特别是程序员的主动性、积极性和创造性，该组织形式特别适用于科技攻关类项目，其组织结构图如图 3-9 所示。

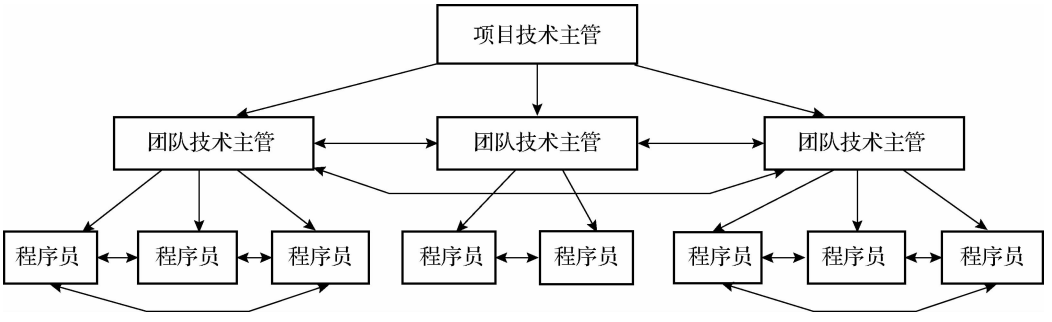


图 3-9　包含分散决策的团队组织形式

4)同步-稳定团队组织形式

该组织形式为微软公司采用的一种组织管理形式，该组织人员由 1 个项目经理、3～8 名软件开发人员和相对应的测试人员组成，他们可能负责软件项目的几个模块。

这种形式的团队鼓励单个成员的创造性，他们可以在一天内随心所欲，但是必须遵守团队的时间和项目计划的约定，即必须在规定的时间内完成一天内实现的组件的测试和调试，这样，可以保证项目能够同步进行，多人可以同步合作完成一个复杂的系统任务。

5)开源编程团队组织形式

开源项目目前已经越来越受到程序员的喜爱，根本原因是程序员对产品设计完成时的成就感，同时也因参与开源项目而获得了新的知识和技能，因此，如果想主持一个开源项目，必须具备如下条件：

(1)项目主持人具有出色的策划和组织能力，能够给项目描绘出一个灿烂前景，这样，才能吸引更多的开源编程爱好者参与其中。

(2)开源项目具有技术上的相对先进性。对于程序员来说，不仅积累经验重要，掌握新的知识和技能更为重要。因为程序员需要的不仅是成就感，也需要通过技能更新从而在未来的组织中获得更大的晋升。

因此，开源项目团队的组织人以及小组核心成员的策划技能和人格魅力非常重要。

3.4　资 源 管 理

软件项目中的资源包括人力资源、技术资源和设备资源，资源模型如图 3-10 所示。这些资源均包括四个主要特征，即资源描述、可用性陈述、需要该资源的时间和该资源被使用的持续时间，在项目开始初期，必须对这 4 类资源特征进行计划和估计。

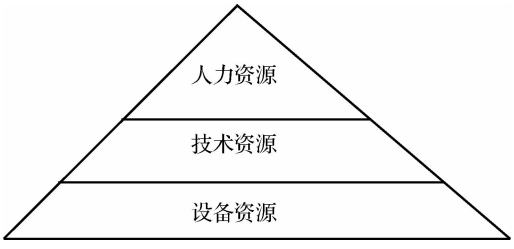


图 3-10　资源模型

3.4.1　资源组成

1)人力资源

人力资源是软件项目中最核心和最重要的资源，对软件项目的实施和成败起着决定性的作用。软件过程的依赖资源首先是开发和实施过程的人，人再进一步利用和消耗其他资源。软件演化过程中的人力资源是指参与软件实施过程的所有人员，根据其所承担的任务和职责可以分为不同的角色。如项目需求分析员、软件架构师、程序员等。对人力资源进行管理的目的是通过组织、协调、控制和监督等手段，依据各人员的能力和特点，分配其最适合的工作，以达到人尽其才。

2)技术资源

技术资源处于金字塔的中间，是软件演化过程中的技术保障。它包括可复用的构件、各种技术文档、软件过程涉及的各种技术因素，也包括一些虚拟的服务，如数据库服务、数据传

输等，其中最重要的是可复用的软件资源。

所谓软件的可重用性是指软件生命周期过程的可重用性，包括项目的组织、软件需求、设计、文档、实现、测试方法和测试用例等都是可以被重复利用或借鉴的有效资源。基于重用的软件开发方法希望通过重复使用有效的、被验证过的软件资源来提高软件质量和效率，其中构件是重用的基本单位。软件计划过程中主要考虑以下四种类型的构件：

(1)成品构件。已经存在的软件，能够从第三方买到或者已经存在于过去开发的软件中。

(2)具有完全经验的构件。以前完成的一些规约、设计、代码或者数据，和现在开发的项目非常类似，且相关技术人员对这些构件的适用范围非常熟悉，应用产生的风险较小。

(3)具有部分经验的构件。虽然是以前完成的成熟构件，但是和现在项目的关联度不大，而且相关人员对这些构件的应用范畴经验不足，所以要应用这些构件，需要进行适当修改，故风险较大。

(4)新构件。项目组为当前项目需要开发的新的软件构件。

显然，要想对这些可重用的软件资源进行有效管理，软件构件库的分类方法和指导思想必须以软件重用为目的，使构件使用者能够做到以下几点：

(1)基于重用属性的快速检索。构件检索是软件重用过程中重要的环节。为了从众多的构件中检索到最需用的构件，需要有效的检索方法和工具。目前，构件的分类和检索方法有很多种，从构件表示出发可分为人工智能方法、超文本方法和信息科学方法；而根据复杂度和检索效果的不同则可以分为基于文本、基于词法描述及基于规约的编码和检索三类。

(2)能够正确理解与评价构件。虽然人们在构件的开发过程遵循了公共的软件工程规范，并在构件库的说明文档中全面、准确地说明了构件的功能、行为以及相关的论域等知识，但如果软件人员希望复用那些原先并非为复用而设计的构件，那么它们在项目的开发过程中未必适用。此时软件人员必须借助计算机软件辅助工具对待选构件进行分析，进而帮助对构件的理解，从而达到能够应用的目的。

(3)可以借助说明来修改构件。当用户无法直接使用构件库中的原始构件来完成项目的功能时，有时只需要对构件进行简单修改即可适应新的需求，因此，构件库必须提供相应的构件文档并能够抽象出更高层次的应用信息。

(4)能够进行基本的构件合成。使用者应该能够将可复用构件库中的构件相互连接，或将它们与当前软件项目中的软件元素相连接以构成最终的目标系统。构件合成技术大致可分为基于功能的、基于数据的和面向对象的合成技术。

3)设备资源

设备资源处于金字塔的底部，是最基础的软件项目资源，它是软件项目的基石。在设备资源的基础之上其他资源才能发挥作用。它由物理资源，如计算机资源、网络资源、存储设备等，以及外部环境资源组成。

3.4.2 资源特征

软件项目管理计划的核心功能之一是识别资源需求，匹配和分配资源，调度和监视资源，从而尽可能高效地利用资源，因此，必须清晰地标注资源特征属性。资源特征属性主要

如下：

(1)一般属性。包括资源的标识、名称、类型(是属于人力资源、技术资源还是设备资源)、位置(技术资源和设备资源的地理位置，各人员所属部门等)、描述等，其中资源的描述对于设备资源是指描述其型号、价格、性能参数、保修期、制造厂商、购入日期、获得资源的来源地等，对于人力资源则描述其工作经验和能力、性格特点等。

(2)使用特征。包括资源的成本、工作时间以及可用性；资源的共享性，即资源是否可被多个过程活动所共享；资源的独占性，即资源是否被独自使用；资源的可移动性以及资源的优先级和动态分配策略。

(3)能力属性。即资源所具有的功能，描述资源完成活动的基本能力，是一个资源能力的集合。

3.4.3　资源管理

软件项目进行过程中资源管理的目标是高效、合理地利用资源，这种利用具有并发性、共享性和随机性等特点，同时也具有特殊性，资源管理主要表现在如下几个方面：

(1)人力资源是软件演化过程中资源管理的重点。软件项目的关键是掌握和运用知识的人，由于人员才能的差异以及完成活动的人员的多种组合方式等因素会对活动的完成周期产生重大影响，从而增加了求解的复杂性。

(2)资源的多样性和分布自治性。软件开发过程中所涉及的资源既包括传统意义上有形的物理资源，又包括无具体形态的人力资源、可重用构件库、数据库服务等技术资源。另外由于软件开发同样包括在原来软件系统的基础上进行完善和改进，而不同的资源可能分属于不同的机构或部门，这些机构对资源有不同地使用、调度策略，因此，如何统一管理这些资源，使资源得到及时与合理的使用，是资源管理需要解决的一个主要问题。

(3)软件各个过程子目标的不一致性。软件进程中涉及众多的子过程，各子过程对资源的使用和分配有不一致甚至矛盾的目标、策略和需求模式，这就要求资源管理支持它们之间的协商以及不同目标之间的资源需求均衡。

(4)动态性与自适应性。随着软件过程的不断深入，用户需求和资源状态始终处于动态变化中，与此相对应，资源提供者与资源消费者的身份、需求等也可能发生变化，因为资源信息具有不确定性，而资源的配置和能力也在动态变化，所以，资源管理需要具有一定的自适应性能力以及有效且高效地利用资源和处理失败的容错能力。

资源管理对软件过程中所涉及的所有资源进行管理，其功能包括资源采集与整理、编制资源需求计划、分配和回收资源。图 3-11 是资源管理的体系结构图，详细介绍如下：

(1)资源采集与整理系统。收集各种资源实体，按照资源的不同类型，将其输入一个可供各活动访问的共享资源数据库。

(2)资源需求编制系统。编制资源计划的过程就是确定为完成各项活动需要什么资源和这些资源的数量的过程。资源计划一般关注资源内容(需要什么)和资源在时间上的分配(什么时候需要)。因此，资源计划是资源和时间的一系列组合表。

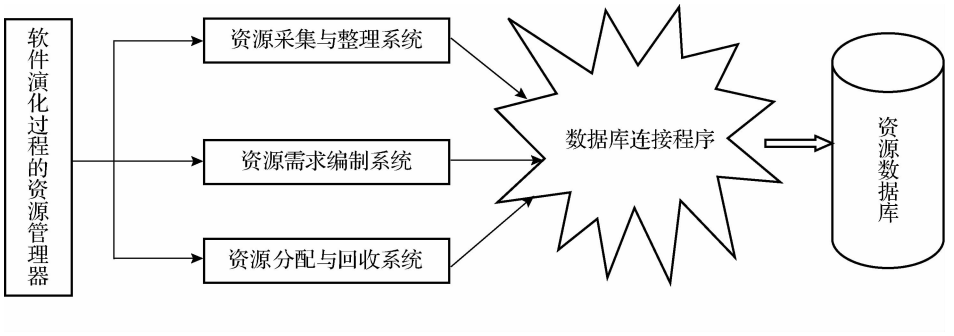


图 3-11　资源管理的体系结构图

编制资源需求计划可以采用专家判断法，即邀请多位专家，利用专家的经验估算资源的需求；也可以采用相似比较估算法，即通过和以往相似的项目进行比较，以对相同或相近的活动所需要的资源数据进行估算，对不同的部分再用其他方法估算等。

(3)资源分配和回收系统。资源分配就是按照一定的策略，为申请使用资源的过程或活动分配资源并记录资源的使用情况。例如，资源的使用者、提出申请时间、满足申请时间、使用资源时间、结束使用时间等。资源的回收是指资源在结束使用之后，系统回收该资源并将其资源可用性属性设置为可用，以等待下一次资源申请到来后，依据分配策略再分配给资源申请者。

3.5　过 程 管 理

一个软件过程可以表示为一个公共过程框架(common process framework，CFP)，该框架由若干包括任务集合的框架活动组成，这些活动可以适用于任何软件工程项目，而不用考虑软件的规模和复杂性，图 3-12 所示为软件公共过程管理框架。

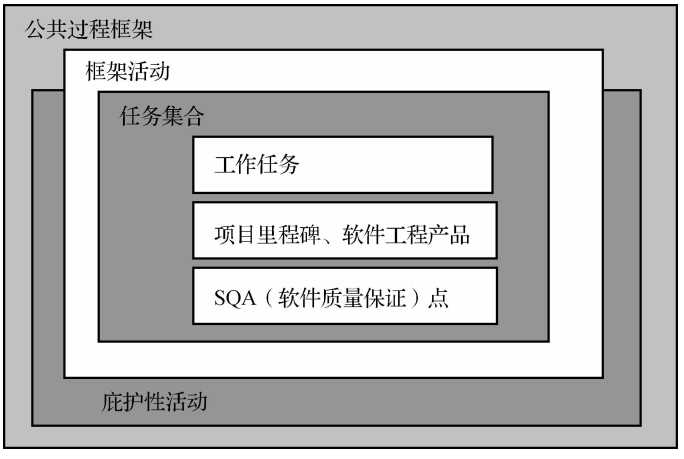


图 3-12　软件公共过程管理框架

公共过程框架的任务集合包括工作任务、项目里程碑、软件工程产品和软件质量保证点，这些可以使软件框架的活动被修改以适应不同种类和需求的软件项目。另外，软件的庇护性活动独立于软件框架之外，它适合并贯穿于软件过程的始终。

3.5.1　过程分解

对于一个具体的软件过程，过程模型的选择取决于软件和团队本身的多种因素，一旦选定，不论是线性模型还是其他类型的软件过程模型，公共过程框架都会适用于该模型。

但是，实际软件工程过程中的任务却是变化的，如需求分析活动的内容、项目质量的保证措施等都可能在进行过程中和计划内容有些差异。将这些问题的过程展开，就形成了过程的分解，分解过程中应该遵守典型的 W5HH 原则，即软件过程的分解必须强调项目目标、里程碑和进度、责任、管理技术和方法以及需要的资源。

3.5.2　能力成熟度模型的集成

1)CMM 和 CMMI 简介

1987 年，美国卡内基・梅隆大学软件研究所(SEI)受美国国防部的委托，针对软件企业的软件过程能力的度量提出了软件过程成熟度模型(capability maturity model for software, CMM)，它是对于软件组织在定义、实施、度量、控制和改善其软件过程的实践中各个发展阶段的描述。CMM 的核心是把软件开发视为一个过程，并根据这一原则对软件开发和维护进行过程监控与研究，以使其更加科学化、标准化。CMM 自 1987 年开始实施认证，现已成为软件业最权威的评估认证体系。

CMM 是一种用于评价软件公司能力并帮助其改善软件质量的方法，它侧重于软件过程的管理及工程实施能力的提高与评估。CMM 分为五个等级：一级为初始级，二级为可重复级，三级为已定义级，四级为已管理级，五级为优化级。共计 18 个过程域。

该模型认为，只要集中精力建立有效的软件工程过程的基础结构，不断进行管理的实践和过程的改进，就可以克服软件生产中的困难，但软件过程的改善不可能在一夜之间完成。因此，CMM 为软件企业的过程能力提供了一个阶梯式的改进框架，它基于过去所有软件工程过程改进的成果，吸取了以往软件工程的经验教训，提供了一个基于过程改进的框架；它指明了一个软件组织在软件开发方面需要管理哪些主要工作、这些工作之间的关系以及以怎样的先后次序一步一步地做好这些工作而使软件组织走向成熟，这些在其五个等级中得到了明显的体现：

(1)初始级。在此阶段，软件企业没有健全的软件管理方法和制度，项目的成功与否取决于项目人员的技术水平和自我约束能力。

(2)可重复级。建立了基本的项目管理过程来跟踪成本、进度和功能监控，制定了必要的过程规则，能重复使用过去在设计、技术和管理方面取得的类似应用项目的成功经验。

(3)已定义级。软件过程已文档化、标准化,所有项目都综合成整个软件开发组织的标准软件过程来开发和维护,同时采用评审的办法来保证软件的质量。

(4)已管理级。制定了软件过程和产品质量详细的度量标准,对软件过程和工作产品都有定量的理解、跟踪和控制。

(5)优化级。通过过程的量化反馈,不断利用新方法、新技术促使软件过程持续改进,使生产率和质量得到稳步提升。

2000 年 8 月,为了利于软件企业进行整个组织的全面过程改进,SEI 决定整合不同模型中的最佳实践,建立集成化能力成熟度统一模型(capability maturity model integration,CMMI),CMMI 是以 3 个基本成熟度模型为基础综合生成的,分别是面向软件开发的 SW-CMM(software-CMM)、面向系统工程的 SE-CMM(system engineering-CMM)以及面向并行工程的 IPD-CMM(integrated product development-CMM),它融合了 6Sigma、TQM、ISO 9000 等标准体系的核心思想,其主体内容包括:

(1)软件工程能力成熟度模型(SW-CMM)。软件工程的对象是软件系统的开发活动,要求实现软件开发、运行、维护活动的系统化、制度化和量化。

(2)系统工程能力成熟度模型(SE-CMM)。系统工程的对象是完整系统的开发活动,可能包括也可能不包括软件。系统工程的核心是将客户的需求、期望和约束条件转化为产品解决方案,并对解决方案的实现提供全程的支持。

(3)集成产品开发能力成熟度模型(IPD-CMM)。集成产品开发是指在产品生命周期中,通过所有相关人员的通力合作,采用系统化的进程来更好地满足客户的需求、期望和要求。如果项目或企业选择 IPD 进程,需要选用模型中与 IPD 相关的实践。

(4)外购能力成熟度模型(SS-CMM)。外购协作过程适用于那些供应商的行为对项目的成功与否起关键作用的项目。主要内容包括:识别并评价产品的潜在来源、确定需要采购的产品的目标供应商、监控并分析供应商的实施过程、评价供应商提供的工作产品以及对供应协议和供应关系进行适当的调整。

2)CMMI 和 CMM 比较

CMMI 作为 CMM 的改进模型,列举了许多实践中的例子进行说明,同时,和 CMM 相比,CMMI 降低了对度量的要求和实施难度,更具有全局性和可操作性。在模型设计方面,CMM 模型只有一种表示法,即阶段式表示法,将软件组织的成熟度划分为五个等级,而 CMMI 模型采用了两种表示法:阶段式表示法和连续式表示法,保留了 CMM 中的阶段式表示法,便于企业组织间能力成熟度的比较,又增加了连续式表示法,进一步促使企业组织更加切合实际地进行内部软件过程的改进,在关键过程域方面,CMM 有 18 个关键过程域(KPA),用于促进软件过程的改进,在 CMMI 中去掉了“关键”,称之为过程域(PA),CMMI 有 22 个过程域(PA)。两者过程域的对比如表 3-4 所示。

表 3-4　CMM 和 CMMI 的关键过程域对比

成熟度等级	CMM 的关键过程域(KPA)	CMMI 的过程域(PA)
第二级:已管理级 CMM:可重复级	需求管理	需求管理
	软件项目计划	项目计划
	软件项目跟踪与监督	项目监控
	软件子合同管理	供应商协议管理
	软件质量保证	产品过程质量保证
	软件配置管理	配置管理
		度量分析
第三级:已定义级	组织过程焦点	组织过程焦点
	组织过程定义	组织过程定义
	培训大纲	组织培训
	组间协调	集成化项目管理
	集成软件管理	风险管理
	集成软件管理	风险管理
	软件产品工程	需求开发 技术解决方案 产品集成 确认 验证
	同行评审	
		决策分析与解决方案
第四级:定量管理级 CMM:已管理级	定量过程管理	定量项目管理
	软件质量管理	组织过程绩效
第五级:优化级	缺陷预防	原因分析与解决方案
	技术变更管理 过程变更管理	组织革新与部署

从表 3-4 中可以看出,CMM 2 级的度量分析分散在每个关键过程域中,而 CMMI 增加了度量分析过程域;CMM 3 级的软件产品工程关键过程域,在 CMMI 中被分解为需求开发、技术解决方案、产品集成、确认和验证 5 个过程域;CMM 3 级的同行评审关键过程域融合到 CMMI 的验证过程域;CMM 3 级的集成软件管理关键过程域所阐述的风险管理,在 CMMI 中形成了一个独立的风险管理过程域。同时,CMM 3 级的组间协调关键过程域转化为 CMMI 的集成化项目管理过程域;CMMI 3 级增加了决策分析与解决方案过程域;CMM 4 级的定量过程管理和软件质量管理关键过程域转变为 CMMI 的定量项目管理和组织过程绩效过程域;CMM 5 级的缺陷预防关键过程域转变为 CMMI 的原因分析与解决方案过程域,技术变更管理和过程变更管理关键过程域合并为 CMMI 的组织革新与部署过程域。

3.5.3 过程管理

软件过程是指软件进行中的需求分析、软件设计、软件实现、软件测试、软件维护等一系列过程，而软件过程的管理则由这些不同阶段的状态、方法、技术和开发、维护软件的人员以及相关计划、文档、模型、编码、测试、手册等组成。软件过程管理的目的是最大限度地提高软件生产率和软件产品的质量，而要达到这样的目的并非一朝一夕能够完成，必须不断地改进软件过程和软件实施规范，最终实现软件过程的高效和可控。

(1)需求管理。在CMM中，要求项目技术人员与需求提出者一起得出对需求的真正理解，并能够得到项目参与者与项目提出者对需求的确认，同时，在项目进行过程中管理需求变更，维护/保持需求、保证项目计划和产品开发之间的双向可追踪性，必须识别出项目计划、开发产品和需求之间的矛盾(或不一致性)。

(2)项目计划。制定软件项目需要的资源计划，包括人力、物力和财力，质量保证，质量评审与质量跟踪计划，项目的节点完成计划，并能够进行计划的跟踪、变更与控制，同时必须制定相应的风险，特别是不可预测的风险应对计划。

(3)项目跟踪与监控。软件项目跟踪和监控的目的是建立软件项目实际进展状态的可视性，使管理者能在软件项目性能偏离开发计划时采取有效措施。管理者监控软件活动主要通过在所选定的时间点或里程碑处，将实际的软件规模、工作量、成本和时间表与计划相比较来确定进展情况。其主要目标是对照软件计划，跟踪实际结果和性能；当实际结果和性能明显偏离软件计划时，采取纠正措施并加以管理直到结束；对软件约定的更改得到受影响的组和个人的认可。项目跟踪和监控中需要明确软件项目跟踪与监控的执行约定和执行的活动；跟踪和监控目标的度量和分析方法以及相关的验证和实施措施。

(4)子合同管理。子合同管理是一个基本的关键过程域。其目的就是选择合格的软件承包商，并可进行有效的管理。软件子承包商的选择应由项目责任者(业主或主承包商)负责，子承包商的选择是基于能力的，对分包人的选择可能处于战略经营同盟或技术的考虑。项目的责任者与子承包商对所承包的项目责任要有一致的认同，并保持不断交流。项目的责任者负责根据合同的责任跟踪子承包商的实际工作结果。

(5)质量保证。软件质量保证是CMM的一个关键过程域，在软件开发过程中起着非常重要的作用。其主要工作包括：评审软件工程活动、审计软件产品、将结果通知项目组成员及相关经理。要想得到高质量的软件，必须考虑以下条件：

①软件项目开发过程遵循明确定义好的既定规则，由此所获得的利益远大于为它所付出的代价。先有稳定、明确的用户需求再进行开发，虽然进度可能有所延迟，但与开发后发现不是用户所需要的产品相比，代价要小得多。

②确保人员的独立性。为了较好地开展软件质量保证工作，软件质量保证人员应该是独立的，与项目经理没有任何行政隶属关系，对其考核与评价也不应由项目经理做出，同时也不能承担本项目中除软件质量保证外的其他任何工作。

③保证质量标准的客观性。软件质量保证的目的是给管理者提供可视性。如果存在主观因素，管理者看到的就不是软件开发过程的真实情况，这一点对软件质量保证工作至关重要。

(6)配置管理。软件配置管理是CMM/CMMI中的关键过程域,也是贯穿于CMM软件改进过程的关键过程域,软件从需求分析开始到最后提交产品要经历多个阶段,每个阶段的工作产品又会产生不同的版本,如何在整个生存期内建立和维护产品的完整性是软件配置管理的目的。软件配置管理主要包括如下内容:

①版本控制。版本控制是全面实行软件配置管理的基础,用于对系统不同版本进行标识和跟踪,版本标识的目的是便于对软件过程中的不同版本加以区分、检索和跟踪,以表明各个版本之间的关系。

②变更控制。软件生存期内全部的软件配置是软件产品的核心,必须使其保持精确,软件工程过程中某一阶段的变更均能引起软件配置的变更,这种变更必须严格加以控制和管理,并保证将修改信息精确、清晰地传递到软件工程过程的下一阶段。

3.6 质 量 管 理

软件质量,即软件产品与明确或者隐含定义的需求相一致的程度。具体地说,软件质量是软件符合明确叙述的功能和性能需求、文档中明确描述的开发标准以及所有专业开发的软件都应具有的隐含特征的程度。软件质量直接影响着软件的使用与维护,因此,无论是软件开发人员还是用户,都对软件质量非常关心,因为质量好的软件不仅可以为开发人员减轻大量的售后维护工作,也省去用户许多不必要的烦恼;而质量差的软件不仅给软件开发者带来大量的维护工作,更可能给用户带来灾难性的后果,因此软件质量至关重要。

影响软件质量的因素有很多,通常包含人的因素、项目研发的各个过程的复杂性、测试技术规范的局限性、质量管理的困难性、软件人员的传统习惯、开发规范、开发工具的支持不够等。

显然,软件产品质量贯穿软件过程的始终,软件产品的质量高低,取决于软件过程的质量控制。

3.6.1 质量度量模型

在软件开发过程中,对即将交付给客户的产品进行科学与合理的评估是非常有必要的,而软件质量度量则提供了一个定量的方法来评价产品的质量,在软件产品完成之前进行评估,可以大大减少质量评估中的主观性。

为了对软件质量进行合理的度量,需要给出软件质量的基本特征,并对这些特征进行量化,从而达到质量度量的目的,目前,基于软件质量特征的质量度量模型主要有以下几种:

1)McCall 质量模型

该模型由 McCall 于 1979 年提出,模型将软件质量特征分为三类:第一类是表现软件运行特征的要素,其中有正确性、可靠性、有效性、完整性和可用性;第二类为软件能够被修改使用的特征,包含可维护性、灵活性和可测试性;第三类为软件适应新的系统环境的能力特征,包括可移植性、可重用性和互联性。

根据上述软件特征，McCall 将软件质量分解到能够度量的层次，提出了 FCM 三层模型，即质量要素(factor)、衡量标准(criteria)和量度标准(metrics)。

(1)质量要素。质量要素是描述和评价软件质量的一组属性，包括软件的功能性、可靠性、易用性、效率性、可维护性、可移植性等质量特性以及将质量特性细化产生的副特性。

(2)衡量标准。衡量标准综合反映某一软件质量要素的特征，衡量的主要内容包括精确性、稳健性、安全性、通信有效性、处理有效性、设备有效性、可操作性、培训性、完备性、一致性、可追踪性、可见性、硬件系统无关性、软件系统无关性、可扩充性、公用性、模块性、清晰性、自描述性、简单性、结构性、文件完备性等。

(3)量度标准。量度标准由各个使用单位自行定义，对软件过程的各个阶段，如软件的需求分析、概要设计、详细设计、编码、测试、确认、维护与使用等，进行质量度量。

表 3-5 给出了 FCM 三层模型的详细描述。

表 3-5　FCM 模型描述

软件质量特征	描　　述
正确性	在预定环境下，软件满足设计规格说明以及用户预期目标的程度，要求软件本身没有错误
可靠性	软件按照设计要求，在规定时间和条件下不出故障，持续运行的程度
有效性	为了完成预定功能，软件系统所需要的计算机资源的多少
完整性	为某一目的而保护数据，避免它受到偶然的或者有意的破坏、改动或者遗失的能力
可使用性	对于一个软件系统，用户学习使用软件及为程序准备输入和解释输出所需工作量的大小
可维护性	为满足用户新的要求，或当环境发生了变化或运行中发现了新的错误时，对一个已经投入运行的软件进行相应诊断和修改所需工作量的大小
灵活性	修改或改进一个已经投入运行的软件所需工作量的大小
可测试性	测试软件以确保其能够执行预定功能所需工作量的大小
可移植性	将一个软件系统从一个计算机系统或环境移植到另外一个计算机系统或环境中运行时所需工作量的大小
可重用性	一个软件(或软件的部件)能再次用于其他应用(该应用的功能与此软件或软件部件所完成的功能有关)的程度
互联性	又称相互操作性，连接一个软件和其他系统所需工作量的大小。如果这个软件要联网与其他系统通信或要把其他系统纳入自己系统的控制之下，必须有系统间的接口使之可以连接

2)Boehm 模型

Boehm 模型是由 Boehm 等人在 1978 年提出来的质量模型，该模型从软件总体功效考虑，虽然在表达质量特征的层次性上与 McCall 模型非常类似，但是，它是基于更为广泛的一系列质量特征的，其中包含了硬件性能的特征，这在 McCall 模型中是没有的。其特征类型如图 3-13所示。

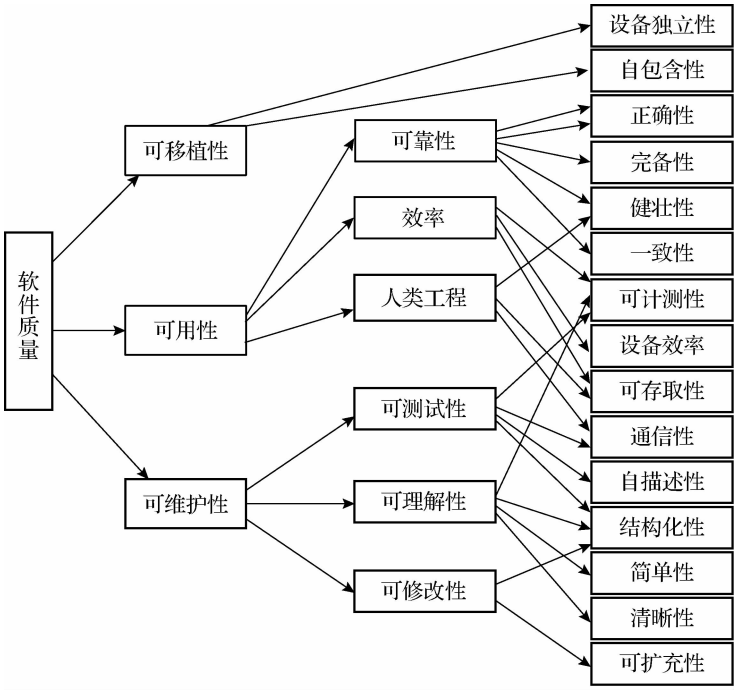


图 3-13　Boehm 质量度量模型

3）ISO 9126

除了著名的 ISO 9000，ISO 也发布了 ISO 9126 质量特征，该特征是基于软件使用标准以及质量特征制定的，其模型如图 3-14 所示。

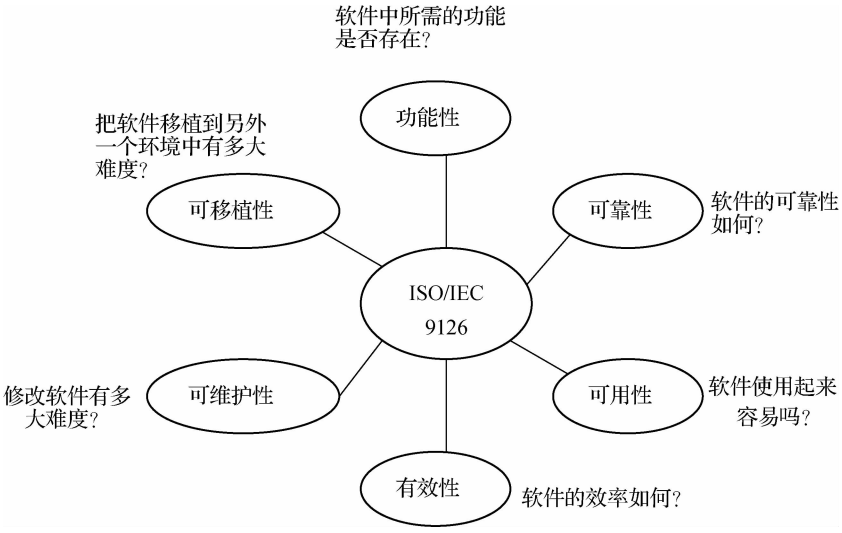


图 3-14　ISO 9126 质量模型

ISO 9126 质量模型的制定是基于 McCall 模型和 Boehm 模型的。它与这些模型具有基本相同的结构方式，如图 3-15 所示。除此之外，ISO 9126 还包括一个功能参数，通过这些参

数可以识别软件产品的内部和外部质量特性。

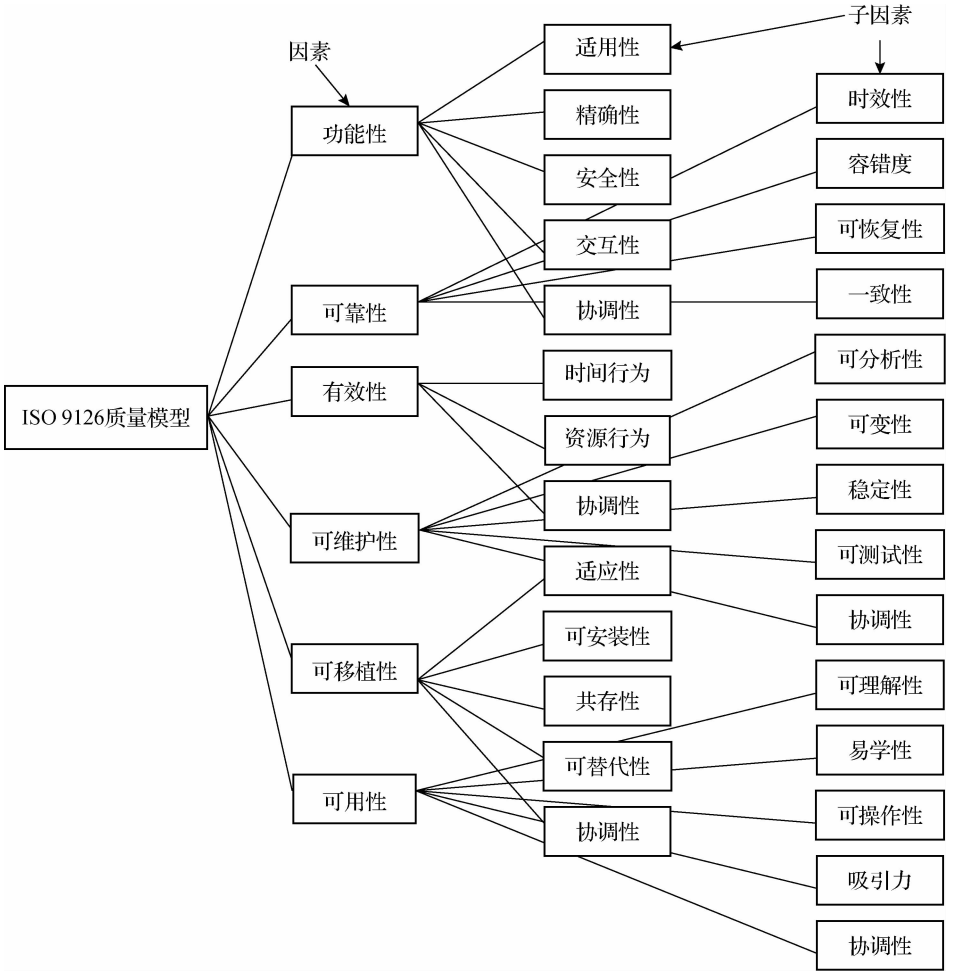


图 3-15　ISO 9126 质量特征

各项质量因素以及相应的子因素的定义如下：

(1)功能性。一个涉及一系列功能及其相应属性存在性的属性集合。该功能能够满足明确或者隐含的需求。

①适用性。涉及一系列功能及其相应任务的存在性和适用性的软件属性。

②精确性。一系列揭示对于权限、协商结果或者影响的规定的软件属性。

③安全性。一系列涉及无论是意外还是蓄意安排的情况下，能够防止对程序和数据进行未经授权访问的软件属性。

④交互性。一系列涉及与指定系统交互能力的软件属性。

⑤协调性。能够使软件在法律或类似规定下，遵循相关的应用程序标准、公约或规定的软件属性。

(2)可靠性。一系列在特定的期限和条件下，有关软件维持其性能水平能力的软件属性。

①时效性。有关由于软件的缺陷导致的故障发生频率的软件属性。

②容错度。在软件故障或者指定接口受到侵害时，能够维持指定的性能水平的能力的软件属性。

③可恢复性。在失败的情况下，重建其性能水平和恢复受其影响的数据所具备的时间和精力上的能力的软件属性。

④一致性。软件性能是否满足系统的标准。

(3)有效性。在特定条件下，有关软件性能水平和资源的使用量之间关系的一系列软件属性。

①时间行为。在功能运行时，有关响应和处理时间以及吞吐量的软件属性。

②资源行为。有关使用的资源量以及功能运行时的持续时间的软件属性。

③协调性。软件产品在效率方面符合某些标准或者习惯的能力。

(4)可维护性。涉及作出指定修改所需努力的一系列软件属性。

①可分析性。为诊断缺陷、失败原因或为确定需要修理的故障部件时，涉及的所需付出努力的软件属性。

②可变性。涉及修改、故障排除或环境变化时所需努力的一系列软件属性。

③稳定性。涉及修改时发生不可预知后果的风险的一系列软件属性。

④可测试性。有关验证修改后的软件所需要的努力的软件属性。

⑤协调性。见上文。

(5)可移植性。有关把软件从一个环境转移到另一个环境所需能力的一系列软件属性。

①适应性。在不使用为使软件达到此目的而提供的其他行为或手段时，有关为适应不同的指定环境时的时机的软件属性。

②可安装性。有关在指定的环境下，安装软件所需努力的软件属性。

③共存性。有关使软件遵循标准或涉及可移植性的公约的一系列软件属性。

④可替代性。在其他指定的软件环境下，涉及使用该软件替换其他软件的时机和所需努力的一系列软件属性。

(6)可用性。涉及使用时需要的努力，以及这种用法对应的个别评估的一系列软件属性。

①可理解性。涉及用户为认识逻辑概念及其应用所付出的努力的软件属性。

②易学性。有关用户为学习其应用(如操作控制、输入、输出等)所付出的努力的软件属性。

③可操作性。涉及用户为操作和操作控制所付出努力的软件属性。

④吸引力。软件产品吸引用户的能力。

⑤协调性。能够使软件在法律或类似规定下，遵循相关的应用程序标准、公约或规定的软件属性。

3.6.2　软件质量保证

为了保证软件质量，必须具备一系列的软件质量保证(software quality assurance，SQA)人员和措施，参与软件质量保证的人员可以简单分为如下两类：

(1)软件设计工程师。他们通过采取可靠的技术方法和措施，进行正规的技术评审，执

行严格细致的软件测试计划来保证产出高质量的软件。

(2)专职 SQA 小组。这些专业质量保证小组的人员,通过执行质量保证计划中的计划、监督、记录、分析和报告等来辅助软件工程师提高软件质量。

需要说明的是,SQA 小组人员和软件开发人员应该保持组织上的相对独立性,即应该属于不同的管理者,任何一方的管理人员都无权调动另外一方的人员,对于软件组织来说,似乎增加了人员成本,但是,这些成本和高质量的软件来比显然微不足道,因为软件质量是软件的生命,如果软件带着错误或者漏洞被移交给客户,那么它不仅会给公司带来致命的信誉危机,同时由软件质量问题而导致的软件维护工作量也无疑将大大增加。

为了使软件质量保证措施的顺利实施以及保证软件以最优的质量提交到用户手中,软件质量保证人员必须参加一系列质量保证活动,其中最重要的活动之一就是正规技术评审。

正规技术评审的要点如下:

(1)评审原则。评审应该面对产品,不针对个人,因此,应该避免较大的争论发生,将问题做好详细记录,在汇集大家的意见后集中上报讨论。

(2)评审人员。参与评审的人员不能太多,但是要具有代表性,既有软件设计人员,又有资深专家,还有质量控制人员。

(3)评审结论。参加评审的人员在产品评审结束后,必须给出一个清晰的结论。

(4)评审报告。评审完成后,必须给出可以存档的评审报告,报告应该明确评审内容、评审时间、参加人员和评审结论。

3.7 可靠性管理

软件可靠性(software reliability)是指软件系统在规定的时间内及规定的环境条件下,完成规定功能的能力。软件可靠性是软件系统的固有特性之一,它表明了一个软件系统按照用户的要求和设计的目标,执行其功能的正确程度。软件可靠性与软件缺陷有关,也与系统输入和系统使用有关。可靠性常用的一些度量术语如下:

(1)缺陷。缺陷是一个存在于软件需求、设计或者源代码中,并且运行时可能产生意想不到的或者不完整的行为的漏洞。缺陷是静态的,不执行源代码就可以检测并清除。出于可靠性的目的,不会触发软件故障的缺陷不会被跟踪和衡量。质量缺陷可能影响软件质量的其他部分,比如软件的维护缺陷和测试用例或者文档中的缺陷。

(2)错误。错误是执行相关源代码时触发一个软件缺陷导致的结果。错误不是用户可见的。例如,内存泄露或者一个需要上层栈重新传输的数据包损坏。错误可能是导致软件故障的过渡状态。简单的缺陷没有中间错误状态。

(3)故障。故障是指一个客户(或者业务系统)观察或检测到了一个不可接受的操作,该操作与已定义的软件功能之间存在较大差别。故障是可见的,是运行时的错误表现。

从理论上说,可靠的软件系统应该是正确、完整、一致和健壮的。但是实际上任何软件都不可能达到百分之百的正确,而且也无法精确度量。一般情况下,只能通过对软件系统进行测试来度量其可靠性。软件可靠性包含了以下三个要素:

(1)规定的时间。软件可靠性只是体现在其运行阶段,所以将“运行时间”作为“规定的时间”的度量。“运行时间”包括软件系统运行后工作与挂起(开启但空闲)的累计时间。由于软件运行的环境与程序路径选取的随机性,软件的失效为随机事件,所以运行时间属于随机变量。

(2)规定的环境条件。环境条件指软件的运行环境。它涉及软件系统运行时所需的各种支持要素,如支持硬件、操作系统、其他支持软件、输入数据格式和范围以及操作规程等。不同环境条件下软件的可靠性是不同的。具体地说,规定的环境条件主要是描述软件系统运行时计算机的配置情况以及对输入数据的要求,并假定其他一切因素都是理想的。有了明确规定的环境条件,还可以有效判断软件失效的责任是在用户方还是在研制方。

(3)规定的功能。软件可靠性还与规定的任务和功能有关。由于要完成的任务不同,软件的运行剖面会有所区别,则调用的子模块就不同(即程序路径选择不同),其可靠性也就可能不同。所以要准确度量软件系统的可靠性必须首先明确它的任务和功能。

3.7.1　可靠性度量

软件可靠性度量是指依据软件测试及其生命周期过程活动中所获得的可靠性数据,对软件可靠性设计过程及其软件产品、资源、环境等的定量测量,是一个认知和理解的过程。它贯穿于软件生命周期全过程。软件可靠性度量为软件的可靠性测试、分析、评估、验证等提供依据和指南,为软件可靠性质量管理和工程管理提供决策支持信息。

ISO/IEC 9126 系列标准和 ISO/IEC 14598 系列标准是最权威、最具影响力的软件质量度量标准,也是迄今为止软件可靠性度量的重要依据。GJB 5236 以此为基础所给出的软件质量模型由 6 个特性及其 32 个子特性构成,它将可靠性作为最重要的质量特性之一。CMMI 为了支持系统地度量软件开发进度、产品及过程,采用了规范的度量行为描述,定义了测量分析过程,规定了两个特定目标和支持这两个特定目标的 8 个特定惯例。这些工作奠定了软件可靠性度量的基础。软件可靠性度量的内涵如图 3-16 所示。

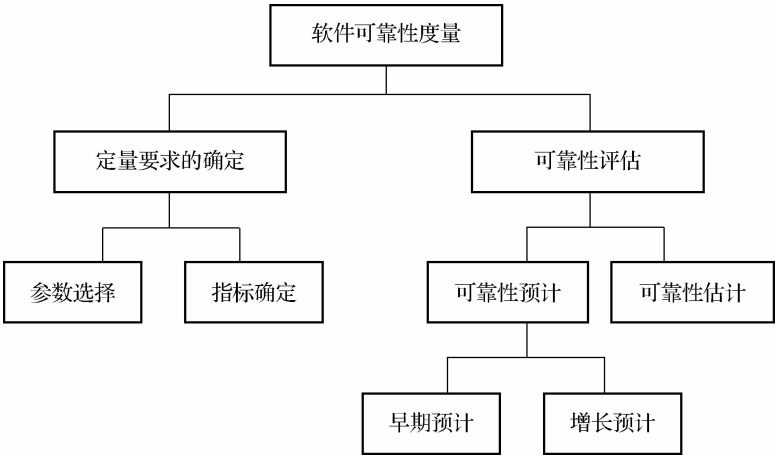


图 3-16　软件可靠性度量的内涵

软件的可靠性主要与系统的设计与实现有关，而与硬件的物理磨损没有关系。可靠性的度量方式是“平均故障间隔时间(MTBF)”，其表达式为：

$$MTBF=MTTF+MTTR$$

其中，$MTTF$ 和 $MTTR$ 分别表示“平均故障时间”和“平均修复时间”。

由于故障是一种不可以接受的结果或者行为，因此必须要清楚软件需要间隔多长时间才出现故障，即需要知道 MTBF，同样也必须要知道当软件出现了故障，修复该故障需要多长时间，即 MTTR。

可靠性度量的另外一种方式是软件可用性度量。软件可用性是指在某个时间点上程序能够按照需求执行的概率，其定义为：

$$可用性=\frac{MTTF}{MTTF+MTTR}\times 100\%$$

显然，有两种方式可以增大软件的可用性，即提高软件的可靠性(增大 MTTF)或者减少停机时间(减少 MTTR)。减小 MTTR 的方式可以有如下几种：

(1)实施硬件冗余来掩饰最有可能发生的故障。

(2)提高故障检测的速度。

(3)通过有效实施软件重启加快软件和系统恢复速度。

可靠性度量的另外一种度量方式是采用缺陷移除效率(DRE)，如果将项目作为一个整体来考虑，DRE 定义如下：

$$DRE=\frac{E}{E+D}$$

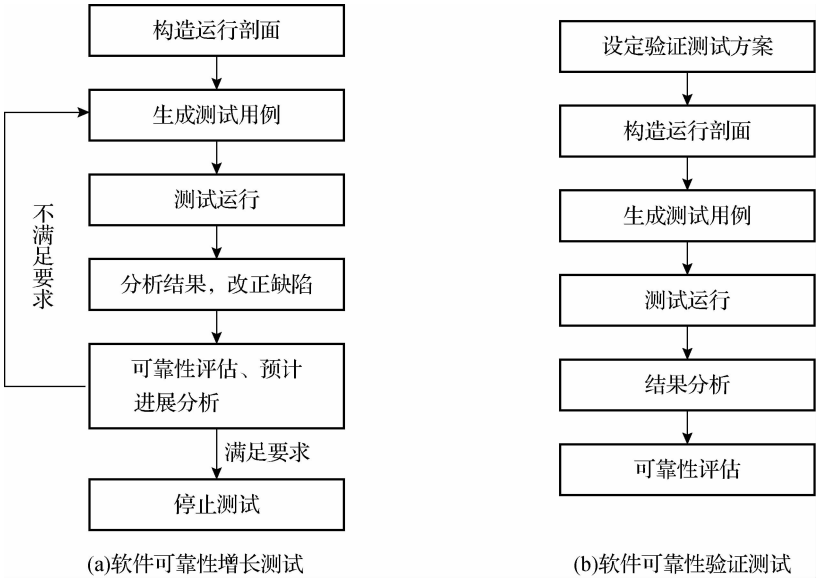
其中，E 是软件交给最终用户之前发现的错误数，D 是软件交付之后发现的缺陷数。

显然，使 DRE 接近于 1 的方式有两种：一种是使 D 为 0，即软件交付之后发现的缺陷数为 0，但是，这基本上是很难做到的；另外一种方式是使 E 充分大，D 比较小，即软件交给用户之前发现大量的错误，软件交付之后发现较小数量的缺陷。因此，用 DRE 作为软件可靠性的度量指标的目的在于鼓励软件企业在将软件交给用户之前发现尽可能多的错误。

3.7.2 可靠性测试

软件可靠性测试是在预期的使用环境中，为软件可靠性评估、鉴定等而对软件实施的一种测试。软件可靠性测试是面向故障的测试，每一次测试均代表用户将要完成的一组操作，使测试成为最终软件产品运行的预演。软件可靠性测试的主要技术包括运行剖面构造技术、可靠性测试数据生成技术、可靠性测试环境构造技术等。

软件可靠性测试通常包括可靠性增长测试和可靠性验证测试两类，如图 3-17 所示。可靠性增长测试和可靠性验证测试是从不同的角度理解、分析和处理故障数据的。可靠性增长测试以迭代的方式进行，根据测试过程中所检测出和跟踪到的故障，使用基于软件可靠性增长模型和统计推理的可靠性评估方法进行故障强度的估计以及测试进展跟踪。可靠性验证测试是软件产品发放或交付前为确定其在风险限度之内的可接收程度而组织实施的最终测试，它是最终检验而不是调试。



(a)软件可靠性增长测试　　(b)软件可靠性验证测试

图 3-17　可靠性测试分类

3.7.3　可靠性设计

因为软件可靠性管理始终贯穿于软件生命周期过程，所以想要取得好的软件可靠性，必须从需求阶段就着手准备，并在不同阶段给出相应的设计方案。

1)需求阶段可靠性设计

需求阶段的软件可靠性要求是：确定重要的软件功能，包括必要的、关键的和非必要的；明确界定可以接受的软件故障率；指出所有影响软件可用性的行为，同时必须为软件升级、重新启动和系统重新开始定义可以接受的间隔时间。

2)设计阶段可靠性设计

软件设计应该使用多层次的方法，包括以下三个要素：

(1)系统的体系结构。明确所有需要软件的系统级功能，并通过进行系统级故障模式分析确定软件在监测和处理硬件故障模式中的作用。

(2)高层次设计(high-level design，HLD)。根据功能的重要性和对故障的脆弱性确定所有的模块。必要功能是最频繁执行的；重要功能不经常被执行，但是完成了系统的关键操作(如重启、关机、备份等)；脆弱点是指可能标记缺陷集合的点(如同步点、硬件和模块接口、初始化和重启代码等)。

(3)低层次设计(low-level design，LLD)。定义模块的可用行为(如重新开始、重试、重新启动、冗余等)。明确功能脆弱的部分，功能应该是面向容错技术的，其专注于简单的实现和恢复行动。

3.8 风险管理

所谓风险，即将来可能产生严重后果的因素。相对于软件来说，软件风险即将来可能导致软件彻底失败的因素。由于现代软件项目周期长、规模大、涉及范围广、风险因素数量多且种类繁杂，因此在其生命周期内面临的风险多种多样，而且风险因素之间的内在关系错综复杂，内外风险因素交叉影响，这些都可能直接导致软件开发人员最终面临失败结局。

根据 Standish Group 对美国超过 8 000 个各种民用、军用软件项目的调查，1996 年、2000 年和 2004 年的软件项目成功率分别为 27%、28%和 29%，其他则部分或者完全失败。我国的相关研究表明，软件项目成功率不足 30%。由此可见，软件项目开发有很高的风险。风险是随机的，不是百分之百发生的，软件风险有如下两个重要特征：

(1)不确定性。风险可能发生也可能不发生，没有百分之百确定发生的风险。

(2)损失。如果风险变成了现实，就会产生恶性后果或者损失。

因此，在软件过程的不同阶段进行风险的识别和评估就显得非常重要。

3.8.1 风险分类

根据风险的不确定性程度和因风险而导致的损失程度划分，软件风险可以分为如下三种：

(1)项目风险。该风险是指项目的预算、进度、各种资源、需求、项目规模、复杂性以及系统结构的不确定性等方面出现问题，从而可能导致项目本身不能按期完成并可能超出预算。这类风险的发生对项目本身产生的影响是致命的。

(2)技术风险。此类风险主要涉及技术本身产生的直接后果，其表现形式为：技术不成熟导致产品质量出现问题；技术存在缺陷导致软件设计存在问题；技术陈旧导致接口、测试和维护设计困难；技术太先进导致和其他系统融合存在困难等。技术风险的直接后果是设计出的软件系统存在一定的质量问题。

(3)商业风险。商业风险一般包括软件产品设计特别超前，目前无人问津；产品进行过程中遇到企业进行战略调整、高层人员调整、资金短路等导致软件过程无法依照既定方针进行。

显然，不论是哪种风险，对于软件企业和软件开发团队来说，都无法预知或者避免，因为有些事情，特别是人的事情无法预测，所以从可预测的角度分析，可以将软件风险划分为：

(1)已知的风险。通过各种渠道和技术分析得到的确定的风险，这类风险由于能够实现估计，所以不会对项目本身构成威胁。

(2)可预测的风险。能够从过去的项目经验、项目本身的一些人员以及涉及的一些技术因素中推测到的风险，这种风险由于可以预先采取一些措施，因此对项目的影响也较小。

(3)不可预测的风险。就像抛掷一枚硬币而不知道正面出现还是反面出现一样，这类风险一旦出现，很可能对项目产生致命影响。

3.8.2　风险识别

风险识别就是要识别出项目风险之所在和引起风险的主要因素，并对其后果作出定性估计。它的理论实质也就是有关知识、推断和搜索的理论。它需要回答以下问题：项目中有哪些潜在的风险因素？这些风险因素会引起什么风险？这些风险所引起后果的严重程度如何？风险识别一般运用分解原则将复杂的事物分解成比较简单的、容易被认识的事物。将大系统分解成小系统，这也符合人们分析问题、认识事物的规律。在实践中用于识别项目风险的方法有很多，目前常用的有头脑风暴法、德尔菲法、情景分析法等。

(1)头脑风暴法(brain storming)也称集体思考法，是以专家的创造性思维来索取未来信息的一种直观预测和识别方法。此法由美国人奥斯本于 1939 年首创，从 20 世纪 50 年代起就得到了广泛应用。头脑风暴法一般在一个专家小组内进行，以宏观智能结构为基础，通过专家会议，发挥专家的创造性思维来获取未来信息。这就要求主持专家会议的人在会议开始时的发言中能激起专家们的思维“灵感”，促使专家们感到急需回答会议提出的问题，通过专家之间的信息交流和相互启发，诱发专家们产生“思维共振”，以达到互相补充并产生“组合效应”，从而获取更多的未来信息，使预测和识别的结果更准确。我国于 20 世纪 70 年代末开始引入头脑风暴法，并受到了广泛的重视和采用。

(2)德尔菲法(Delphi method)又称专家调查法，它是 20 世纪 50 年代初美国兰德公司研究美国受前苏联核袭击风险时提出的，并在世界上快速地盛行起来。它是依靠专家的直观能力对风险进行识别的方法，现在此法的应用已遍及经济、社会、工程技术等各领域。用德尔菲法进行项目风险识别的过程为：由项目风险小组选定项目相关领域的专家，并与这些适当数量的专家建立直接的函询联系，通过函询收集专家意见，然后加以综合整理，再匿名反馈给各位专家，再次征询意见。这样反复经过四至五轮，逐步使专家的意见趋向一致，作为最后识别的根据。我国在 20 世纪 70 年代引入此法，已在许多项目管理活动中进行了应用，并取得了比较满意的结果。

德尔菲法的最大特点是专家与专家之间不见面，背靠背地进行，不发生任何关系，这就减少了权威、资历、口才、人数和心理等因素的影响，对那些缺乏客观资料和历史数据的情况比较适用，尤其适用于项目启动阶段的风险预测。

(3)情景分析法(scenarios analysis)是由美国 SIIELL 公司的科研人员 Pierr Wark 于 1972 年提出的。它根据发展趋势的多样性，通过对系统内外相关问题的系统分析，设计出多种可能的未来前景，然后用类似于撰写电影剧本的手法，对系统发展态势作出自始至终的情景和画面的描述。当一个项目持续的时间较长时，往往要考虑各种技术、经济和社会因素的影响，此时可用情景分析法来预测和识别其关键风险因素及其影响程度。情景分析法对以下情况特别有用：提醒决策者注意某种措施或政策可能引起的风险或危机性的后果；建议需要进行监视的风险范围；研究某些关键性因素对未来过程的影响；提醒人们注意某种技术的发展会给人们带来哪些风险。情景分析法是一种适用于对可变因素较多的项目进行风险预测和识别的系统技术，它在假定关键影响因素有可能发生的基础上，构造出多重情景，提

出多种未来的可能结果，以便采取适当措施防患于未然。情景分析法从20世纪70年代中期以来在国际上得到了广泛应用，并产生了目标展开法、空隙添补法、未来分析法等具体应用方法。一些大型跨国公司在对一些大项目进行风险预测和识别时都陆续采用了情景分析法。但是因其操作过程比较复杂，目前此法在我国的具体应用还不多见。

3.8.3 风险评估

风险量化就是对风险发生的可能性及其后果作出定量的估计，也就是对风险作出定量的测度。它要回答的是风险有多大的问题。它是在对项目进行风险识别和初步分类之后所要做的工作，其对象是项目的各单个风险，而非项目整体风险。通过风险估计，有关人员可以加深对项目自身和环境的理解，力争使项目所有的不确定性和风险都经过充分、系统而有条理的考虑，从而寻找实现项目目标的可行而又较优的方案。目前，对项目进行风险量化的方法很多，如主观评分法、层次分析法（AHP）、概率分析法（随机型风险估计法）、蒙特卡洛模拟法、故障树分析法（fault tree analysis，FTA）、影响图分析法（influence diagram）等。

故障树分析法是美国贝尔电话实验室的维森于1962年首先提出的，我国于1976年开始介绍和研究这种方法，并随之应用于许多项目中，并取得了许多成果。故障树是由一些节点及它们之间的连线所组成的，每个节点表示某一具体事件，而连线则表示事件之间的关系。FTA是一种演绎的逻辑分析方法，遵循从结果找原因的原则，分析项目风险及其产生原因之间的因果关系，即在前期预测和识别各种潜在风险因素的基础上，运用逻辑推理的方法，沿着风险产生的路径，求出风险发生的概率，并能提供控制各种风险因素的方案。

3.8.4 风险应对

在项目开始初期，做好各类风险的应对计划是控制风险最有效的手段。为了制定风险应对计划，需要熟悉优先风险列表，合理计划和分配资源。风险计划的关键是考虑当前决策对未来的影响，因此，通常采取以下风险应对策略：

（1）风险避免。通过改变项目计划或者一些现存策略以消除项目风险或保护项目目标不受风险影响。

（2）风险接受。有选择地接受可以承担的风险后果，或者项目组被动接受。

（3）风险退避。由于无法避免存在的风险，因此通过改变技术条件、人员资源等方法以消除对项目存在的特定威胁。如增加额外补贴，减少项目范围等。

（4）风险储备。在项目进度计划中加入风险计划并在项目风险影响的预期中作出预算，即对项目意外风险预留应急费用和进度计划。用于项目较新或灰度较大时，可以用来防止项目进度超时或费用超支的风险应对策略。

（5）风险转移。风险转移就是设法将风险转移给能够处理风险的第三方，转移风险只是将管理风险的责任转移给另一方，它不能消除风险。转移风险几乎总是会伴有向接受风险的一方支付风险成本的情况发生。

（6）风险缓解。指项目组在风险发生之前可以采取的步骤措施，使得某一负面风险事件

的概率或其后果降低到可以接受的程度。显然，早期采取措施降低风险发生的概率或风险对项目的影响，比在风险发生后再补救更为有效，对照风险可能的概率及其后果，缓解的成本应是合理的。

(7)风险研究。就已经存在或者发生的风险，进行大量的研究、分析和测试，以期获得对风险的认识能力和管理能力。

3.8.5　风险缓解、监控和管理

风险缓解、监控和管理的主要目的是评估一个被预测的风险是否真正发生了，同时保证为风险而定义的缓解步骤被正确地实施了。

风险缓解要求定义文档的标准并建立相应的机制以确保文档能够被及时建立，当项目进行时，风险监控也随即开始，而如果风险缓解工作失败或者监控的风险真正发生了，即风险变成了现实，那么应该采取相应的管理措施，以降低风险的危害，风险监控和管理主要包含以下几个方面：

(1)在整个项目过程中，时刻监督风险的发展与变化情况，确定伴随某些风险的消失而来的新的风险并制定相应的处理措施。

(2)保证风险应对计划的执行并评估风险应对计划执行效果。评估的方法可以是项目周期(阶段性)回顾、绩效评估等。

(3)对突发的风险或“接受”的风险采取适当的权变措施。通过风险监控过程，持续更新风险应对计划，对特定风险事项进行文档化跟踪，并通过重复上述各步骤保证项目风险始终处于受控状态。

3.8.6　风险管理模型

软件风险管理研究始于 20 世纪 70 年代，Boehm 于 1989 年出版的专著《软件风险管理》奠定了该领域的理论基础。在随后近 30 年中，又陆续出现了几种风险管理方法，并出现了很多成果。典型的风险管理体系有：

(1)Boehm 体系。Boehm 提出的风险管理模型把风险管理活动分为风险评估和风险控制两个阶段；风险评估阶段主要包括风险识别、风险分析、风险排序 3 项子步骤；风险控制阶段包括风险管理计划、风险解决和风险监控 3 项子步骤。Boehm 思想的核心是 10 大风险因素清单，其中包括人员短缺、不合理的进度安排和预算、不断的需求变动等。针对每个风险因素，Boehm 给出了一系列的风险管理策略，例如，对待不合理的进度安排和预算，可以采用增量式开发方法或进行软件重用等。

(2)Charette 体系。1989 年 Charette 设计了称为风险分析和管理的体系，其包含的两大阶段分别为分析阶段和管理阶段，每个阶段都包含三个过程，风险分析阶段分为风险辨识、风险估计、风险评价；风险管理阶段分为风险计划、风险控制、风险监督。每个阶段内的过程活动并不能完全分离，有相互重叠甚至交错反复的现象。显然，Charette 体系从结构上来说和 Boehm 体系非常类似，两者最大的不同就是 Charette 体系认为风险管理是一个不断

反复循环的过程。

(3)SEI 体系。SEI 作为世界上著名的旨在改善软件工程管理实践的组织,在软件项目风险管理方面做了大量的工作,并于 1999 年前后分别以技术报告和手册等形式公布了基于分类的风险辨识(TBQ)、连续风险管理(CRM)、软件风险评估(SRE)、软件风险管理—能力成熟度模型(RM-CMM)和团队风险管理(TRM)。SEI 风险体系最明显的特点就是可操作性,注重了与软件开发过程的紧密结合。体系中的理性思考都以指导实践步骤为主要目的,基本上摒弃了复杂的数学运算,更加符合风险管理和项目成本之间的博弈结果。

图 3-18～图 3-20 所示分别为上述三个风险管理模型的体系结构图。

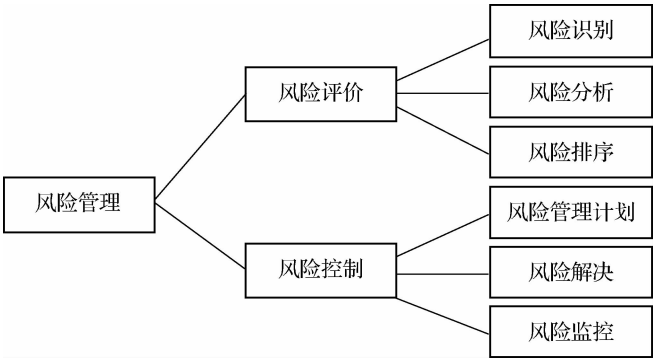


图 3-18　Boehm 风险管理体系图

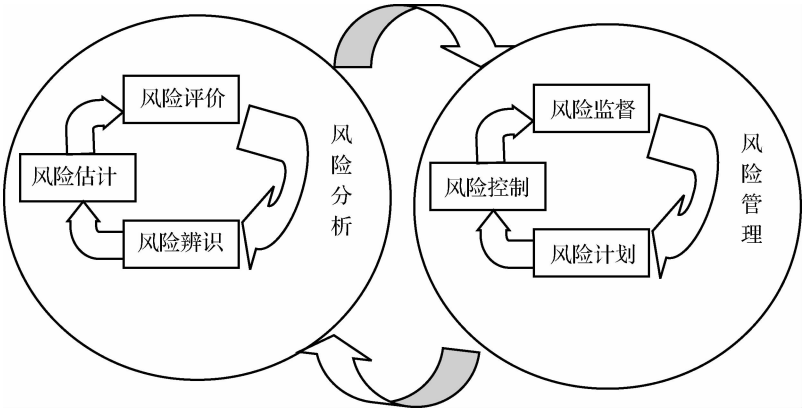


图 3-19　Charette 风险体系结构图

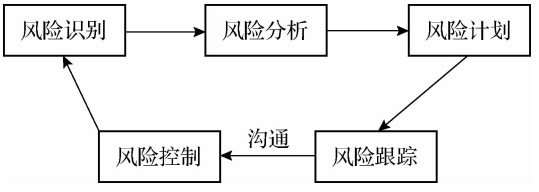


图 3-20　SEI 风险体系结构图

本 章 小 结

软件计划和任何项目计划一样，其重要性不言而喻，不仅要估算项目成本，更要估算需要的人员、技术和设备等资源，还要在软件过程中有效地管理这些资源，因此，本章给出了软件计划的内容和目标以及项目成本估算的方法，尽管这些方法，如 LOC 等，还存在着一些问题，但是在特定情形下，它们还是非常有效的。在项目管理方面，本章给出了几种程序员团队的组织管理形式，并给出了一些不同项目的团队组织形式建议；在软件过程方面，依据能力成熟度模型集成（CMMI），可以进行软件过程的度量和管理，同时根据 CMMI 和 ISO 9001，可以进行软件的质量管理和度量，并给出了几种典型的质量度量模型；在软件可靠性方面，本章给出了软件可靠性的几种度量方式，并给出了提高可靠性的方法；最后给出了软件风险的分类和识别方法，并描述了几种典型的风险管理模型。

总之，本章介绍了软件计划和管理的核心内容，通过本章的学习，可以对软件的计划内容有一个明晰的印象，能够掌握软件成本估算的一些基本算法，能够清楚描述软件过程中的质量管理、可靠性管理和风险管理，并对软件风险管理的重要性有一个清醒的认识。

习　题　3

1）选择题

（1）应用于软件工程的 ISO 质量保证标准是（　　）。

A. ISO 9000　　B. ISO 9001

C. ISO 9002　　D. ISO 9003

（2）软件可靠性问题主要与（　　）有关。

A. 需求分析阶段信息采集错误　　B. 设计和实现阶段出现错误

C. 人员问题　　D. 操作问题

（3）质量控制的一个关键问题是所有工程产品（　　）。

A. 必须在预算内准时提交

B. 有完全的文档

C. 有用于输出的可以度量的技术指标

D. 提交给用户之前必须完全通过测试

（4）软件风险的两个主要特点是（　　）。

A. 风险管理系统　　B. 已知的和未知的风险

C. 不确定性和损失　　D. 人员和预算

(5)有效的风险管理计划需要说明(　　)。

A. 风险避免　　B. 风险监控

C. 意外风险　　D. 上述所有项

(6)风险的三个领域是(　　)。

A. 商业风险、个人风险、预算风险

B. 项目风险、技术风险、商业风险

C. 计划风险、技术风险、个人风险

D. 管理风险、技术风险、设计风险

(7)(　　)可以确定一个软件项目需要的人员数量。

A. 软件开发工作量确定后　　B. 项目预算大小

C. 项目系统的复杂性评估后　　D. 上述所有内容

(8)软件项目计划的目的是(　　)。

A. 使客户确信项目是可行的

B. 应用历史项目的数据

C. 让项目经理做出合理的成本和计划估计

D. 确定可能的利润

(9)基于代码行度量的软件估算技术需要基于(　　)问题分解。

A. 信息范围值　　B. 项目计划

C. 软件功能　　D. 过程活动

(10)基于功能点度量的估计技术需要用(　　)进行问题分解。

A. 信息域的值　　B. 项目计划

C. 软件工程　　D. 过程活动

2)问答题

(1)民主团队和开源编程团队组织间有什么区别？哪个团队组织更适合于进行科技攻关类的项目研究？给出你的理解。

(2)为什么说人力资源是软件项目中最重要的资源？如何管理人力资源？

(3)给出两种以上软件可靠性的度量方法及其描述。

(4)软件项目中常用的风险识别方法有哪些？试举例说明。

(5)软件质量的核心要求是什么？为什么 SQA 小组人员和软件开发人员应该保持相对的独立性？

3)计算题

(1)假定目标产品有 7 个简单的输入、2 个一般的输入和 10 个复杂的输入。有 56 个一般的输出、8 个简单的查询、12 个一般的主文件和 17 个复杂的接口，计算未调整的功能点的数量。

(2)假定由你负责开发一个 67KDSI 的嵌入式产品，除了数据库的规模级别非常高，以

及软件工具的使用非常低之外，其他都是额定的，请问使用中级 COCOMO 模型，以人月计算的估计工作量是多少？

(3)假定由你负责开发一个 49KDSI 的有组织模式的产品，并且每个方面都是额定的：

①假设成本是每人月为 9 900 美元，该项目的估算成本将是多少？

②如果开始时你的团队集体辞职了，但是你找到了一个经验更为丰富的团队来完成，但是成本也增加到每人月 12 900 美元，那么人员变化后成本情况如何变化？