# 第一章 软件工程学

## 1．1软件工程

在计算机系统发展的早期时代（60年代中期以前），通用硬件相当普遍，软件却是为每个具体应用而专门编写的。这时的软件通常是规模较小的程序，编写者和使用者往往是同一个（或同一组）人。这种个体化的软件环境，使得软件设计通常是在人们头脑中进行的一个隐含的过程，除了程序清单之外，没有其他文档资料保存下来。

从60年代中期到70年代中期是计算机系统发展的第二代时期，这个时期的一个重要特征是出现了“软件作坊”，广泛使用产品软件。但是，“软件作坊”基本上仍然沿用早期形成的个体化软件开发方法。随着计算机应用的日益普及.软件数量急剧膨胀。在程序运行时发现的错误必须设法改正，用户有了新的需求时必须相应地修改程序；硬件或操作系统更新时，通常需要修改程序以适应新的环境。上述种种软件维护工作，以令人吃惊的比例耗费资源。更严重的是，许多程序的个体化特性使得它们最终成为不可维护的。“软件危机”就这样开始出现了门968年北大西洋公约组织的计算机科学家在联邦德国召开国际会议，讨论软件危机问题，在这次会议上正式提出并使用了“软件工程”这个名词，一门新兴的工程学科就此诞生。

### 1．1．1 软件危机介绍

1 软件危机：是指在计算机软件的开发和维护过程中所遇到的一系列严重问题。这些问题绝不仅仅是“不能正常运行的”软件才具有的，实际上几乎所有软件都不同程度地存在这些问题。概括地说，软件危机包含下述两方面的问题：如何开发软件，怎样满足对软件的日益增长的需求；如何维护数量不断膨胀的已有软件。具体地说，软件危机主要有下述一些表现：

(1) 对软件开发成本和进度的估计常常很不准确。实际成本比估计成本有可能高出一个数量级，实际进度比预期进度拖延几个月甚至几年的现象并不罕见。

(2) 用户对“已完成的”软件系统不满意的现象经常发生。软件开发人员常常在对用户要求只有模糊的了解，甚至对所要解决的问题还没有确切认识的情况下，就仓促上阵匆忙着手编写程序。

(3) 软件产品的质量往往靠不住。软件可靠性和质量保

证的确切的定量概念刚刚出现不久，软件质量保证技术（审查、复审和测试 ）还没有坚持不懈地应用到软件开发的全过程中，这些都导致软件产品发生质量问题。

（4）软件常常是不可维护的。很多程序中的错误是非常难改正的，实际上不可能使这些程序适应新的硬件环境，也不能根据用户的需要在原有程序中增加一些新的功能。“可重用的软件”还是一个没有完全做到的、正在努力追求的目标，人们仍然在重复开发类似的或基本类似的软件。

（5）软件通常没有适当的文档资料。计算机软件不仅仅是程序，还应该有一整套文档资料。这些文档资料应该是在软件开发过程中产生出来的，而且应该是“最新式的”（即和程序代码完全一致的 ）。

（6）软件成本在计算机系统总成本中所占的比例逐年上升。软件成本随着通货膨胀以及软件规模和数量的不断扩大而持续上升。美国在 1985年软件成本大约已占计算机系统总成本的 90％。

（7）软件开发生产率提高的速度，远远跟不上计算机应用迅速普及深入的趋势。软件产品“供不应求”的现象使人类不能充分利用现代计算机硬件提供的巨大潜力。

### 1．1．2　产生软件危机的原因

在软件开发和维护的过程中存在这么多严重问题，一方面与软件本身的特点有关，另一方面也和软件开发与维护的方法不正确有关。

软件不同于硬件，它是计算机系统中的逻辑部件而不是物理部件。

软件缺乏可见性，在写出程序代码并在计算机上试运行之前，软件开发过程的进展情况较难衡量，软件开发的质量也较难评价。

软件在运行过程中不会因为使用时间过长而被“用坏”如果运行中发现错误，很可能是遇到了一个在开发时期引入的在测试阶段没能检测出来的故障。因此，软件维护通常意味着改正或修改原来的设计，这就在客观上使得软件较难维护。

软件不同于一般程序，它的一个显著待点是规模庞大，而且程序复杂性将随着程序规模的增加而成指数上升。

目前相当多的软件专业人员对软件开发和维护还有不少糊涂观念，在实践过程中或多或少地采用了错误的方法和技术，这可能是使软件问题发展成软件危机的主要原因。

对用户要求没有完整准确的认识就匆忙着手编写程序是

许多软件开发工程失败的主要原因之一。

一个软件从定义、开发、使用和维护，直到最终被废弃要经历一个漫长的时期，这就如同一个人要经过胎儿、儿童、青年、中年、老年，直到最终死亡的漫长时期一样。通常把软件经历的这个漫长的时期称为生命周期。

问题定义，也就是确定要求解决的问题是什么；

可行性研究，决定该问题是否存在一个可行的解决办法；

需求分析，也就是深入具体地了解用户的要求，在所要开发的系统（不妨称之为目标系统）必须做什么这个问题上和用户取得完全一致的看法；

设计（通常又分为总体设计和详细设计两个阶段）；

编写程序阶段；

测试（需要的工作量通常占软件开发全部工作量的 40％一 50％），最终交付使用。

编写程序只是软件开发过程中的一个阶段，而且在典型的软件开发工程中，编写程序所需的工作量只占软件开发全部工作量的 10％一 20％。

另一方面还必须认识到程序只是完整的软件产品的一个组成部分，在上述软件生命周期的每个阶段都要得出最终产品的一个或几个组成部分（这些组成部分通常以文档资料的形式存在）。这也就是说一个软件产品必须由一个完整的配置组成，软件配置主要包括程序、文档和数据等部分。应该清除只重视程序而忽视软件配置其余成分的糊涂观念。

严重的问题是，在软件开发的不同阶段进行修改需要付出的代价是很不相同的，在早期引入变动，涉及的面较少，因而代价也比较低；而在开发的中期软件配置的许多成分已经完成，引入一个变动要对所有已完成配置成分都做相应的修改，不仅工作量大，而且逻辑上是更复杂，因此付出的代价剧增；在软件“已经完成”时再引入变动，当然需要付出更高得多的代价。

通过上面的论述不难认识到，轻视维护是一个最大的错误。许多软件产品的使用寿命长达 10 年甚至 20 年，在这样漫长的时期中不仅必须改正使用过程中发现的每一个潜伏的错误，而且当环境变化时（例如硬件或系统软件更新换代）还必须相应地修改软件以适应新的环境，特别是必须经常改进或扩充原来的软件以满足用户不断变化的需要。统计数据表明，实际上用于软件维护的费用占软件总费用的 55％一 70％。软件工程学的一个重要目标就是提高软件的可维护性，

减少软件维护的代价。

### 1．1．3　消除软件危机的途径

为了消除软件危机，首先应该对计算机软件有一个正确的认识。应该彻底消除在计算机系统早期发展阶段形成的“软件就是程序”的错误概念。一个软件必须由一个完整的配置组成，事实上，软件是程序、数据及相关文挡的完整集合。其中，程序是能够完成预定功能和性能的可执行的指令序列；数据是使程序能够适当地处理信息的数据结构。虽然表面上看来在这个定义中列出了软件的五个配置成分，但是，方法和规则通常是在文挡中说明并在程序中实现的。

软件开发不是某种个体劳动的神秘技巧，而应该是一种组织良好、管理严密、各类人员协同配合、共同完成的工程项目。必须充分吸取和借鉴人类长期以来从事各种工程项目所积累的行之有效的原理、概念、技术和方法，特别要吸取几十年来人类从事计算机硬件研究和开发的经验教训。

应该推广使用在实践中总结出来的开发软件的成功的技术和方法，并且研究探索好更有效的技术和方法，尽快消除在计算机系统早期发展阶段形成的一些错误概念和做法。

应该开发和使用更好的软件工具。正如机械工具可以“放大”人类的体力一样，软件工具可以“放大”人类的智力。

## 1．2 软件工程

软件工程：是指导计算机软件开发和维护的一门工程学科。采用工程的概念、原理、技术和方法来开发与维护软件，把经过时间考验而证明正确的管理技术和当前能够得到的最好的技术方法结合起来，已经济地开发出高质量的软件并有效的维护它，这就是软件工程。

### 1．2．1　软件工程的介绍

1968年在第一届 NATO会议上给软件工程这个术语定义为“软件工程就是为了经济地获得可靠的而且能在实际机器上有效地运行的软件，而建立和使用完善的工程原理”。

1993年 IEEE更全面更具体的定义：“软件工程是：①把系统的、规范的、可度量的途径应用于软件开发、运行和维护过程，也就是把工程应用于软件；②研究①中提到的途径。”

软件工程具有下述本质特性：

1 软件工程关注于大型程序的构造

把一个人在较短时间内写出的程序称为小型程序，把多人合作用时半年以上才写出的程序称为大型程序——程序系统

2 软件工程的中心课题是控制复杂性

软件的复杂性主要不是由于问题的内在复杂性造成的，而是由必须处理的大量细节造成的

3 软件经常变化

4 开发软件的效率非常重要

软件供不应求日益严重

5 和谐地合作是开发软件的关键

纪律是成功地完成软件开发项目的关键

6 软件必须有效地支持它的用户

仅仅用正确的方法构造系统还不够，还必须构造出正确的系统，提供产品、用户手册、培训资料

7 在软件工程领域中是由具有一种文化背景的人替具有另一种文化背景的人创造产品

软件工程师不仅缺乏应用领域的实际知识，他们还缺乏领域的文化知识

## 1．2．2软件工程的基本原理

著名的软件工程专家 B．w．Boehm综合这些学者们的意见并总结了 TRW 公司多年开发软件的经验，于 1983年在一篇论文中提出了软件工程的七条基本原理。他认为这七条原理是确保软件产品质量和开发效率的原理的最小集合。这七条原理是互相独立的，其中任意六条原理的组合都不能代替另一条原理。然而这七条原理又是相当完备的，人们虽然不能用数学方法严格证明它们是一个完备的集合，但是，可以证明在此之前已经提出的 100 多条软件工程原理都可以由这七条原理的任意组合蕴含或派生。

下面简要介绍软件工程的七条基本原理：

1 用分阶段的生命周期计划严格管理

2 坚持进行阶段评审

有两个理由：第一，大部分错误是在编码之前造成的，例如，根据 Boehm等人的统计，设计错误占软件错误的 63％，编码错误仅占 37％；第二，错误发现与改正得越晚，所需付出的代价也越高 （参见图 1．1和图1．2）。

3 实行严格的产品控制

当改变需求时，为了保持软件各个配置成分的一致性，必须实行严格的产品控制，其中主要是实行基准配置管理。所谓基准配置又称为基线配置，它们是经过阶段评审后的软

件配置成分 （各个阶段产生的文档或程序代码 ）。基准配置管理也称为变动控制：一切有关修改软件的建议，待别是涉及到对基准配置的修改建议，都必须按照严格的规程进行评审，获得批准以后才能实施修改。绝对不能谁想修改软件（包括尚在开发过程中的软件），就随意进行修改。

4 采用现代程序设计技术

人们一直把主要精力用于研究各种新的程序设计技术。60 年代末提出的结构程序设计技术，已经成为绝大多数人公认的先进的程序设计技术。以后又进一步发展出各种结构分析 (SA)与结构设计 (SD)技术。实践表明，采用先进的技术既可提高软件开发的效率，又可提高软件维护的效率。

5 结果应能清楚地审查

软件产品不同于一般的物理产品，它是看不见摸不着的逻辑产品。软件开发人员（或开发小组 ）的工作进展情况可见性差，难以准确度量，从而使得软件产品的开发过程比一般产品的开发过程更难于评价和管理。为了提高软件开发过程的可见性，更好地进行管理，应该根据软件开发项目的总目标及完成期限，规定开发组织的责任和产品标准，从而使得所得到的结果能够清楚地审查。

6 开发小组的成员应该少而精

这条基本原理的含义是，软件开发小组的组成人员的素质应该好，而人数则不宜过多。开发小组人员的素质和数量是影响软件产品质量和开发效率的重要因素。素质高的人员的发效率比索质低的人员的开发效率可能高几倍至几十倍，而且素质高的人员所开发的软件中的错误明显少于素质低的人员所开发的软件中的错误。此外，随着开发小组人员数目的增加，因为交流情况讨论问题而造成的通信开销也急剧增加。当开发小组人员数为 N时，可能的通信路径有 N(N 一 1）／ 2条，可见随着人数 N的增大，通信开销将急剧增加。因此，组成少而精的开发小组是软件工程的一条基本原理。

7 承认不断改进软件工程实践的必要性

遵循上述六条基本原理，就能够按照当代软件工程基本原理实现软件的工程化生产，但是，仅有上述六条原理并不能保证软件开发与维护的过程能赶上时代前进的步伐，能跟上技术的不断进步。因此， Boehm提出应把承认不断改进软件工程实践的必要性作为软件工程的第七条基本原理。按照这条原理，不仅要积极主动地采纳新的软件技术，而且要注意不断总结经验。

## 1．2．3软件工程方法学

通常把在软件生命周期全过程中使用的一整套技术方法的集合称为方法学，也称为“范性”。在软件工程领域中，这两个术语的含义基本相同。

软件工程方法学包括3个要素：方法、工具和过程。

方法：完成软件开发的各项任务的技术方法

工具：运用方法提供的自动的或半自动的软件工程支撑环境

过程：为了获得高质量的软件所要完成的一系列任务框架，规定了完成各项任务的工作步骤

目前使用得最广泛的软件工程方法学分别是传统方法学和面向对象方法学

1 传统方法学

传统方法学又称生命周期方法学或结构化范型。软件工程采用的生命周期方法学就是从时间角度对软件开发和维护的复杂问题进行分解，把软件生命的很长周期依次划分为若干个阶段，每个阶段有相对独立的任务，然后逐步完成每个阶段的任务。对于任何两个相邻的阶段而言，前一阶段的结束标准就是后一阶段的开始标准。在每一个阶段结束之前都必须进行正式严格的技术审查和管理复审，从技术和管理两方面对这个阶段的开发成果进行检查，通过之后这个阶段才算结束；如果检查迈不过，则必须进行必要的返工，并且返工后还要再经过审查。审查的一条主要标准就是每个阶段部应该交出“最新式的”（即和所开发的软件完全一致的）高质量的文档资料，从而保证在软件开发工程结束时有一个完整准确的软件配置交付使用。

2 面向对象方法学

是一种以数据为主线，把数据和对数据的操作紧密地结合起来的方法。面向对象方法学具有下述四个要点：

（1）把对象作为融合了数据及在数据上的操作行为的统一的软件构件。

（2）把所有对象都划分成类。

（3）按照父类与子类的关系，把若干个相关类组成一个层次结构的系统。

（4）对象彼此间仅能通过发送消息互相联系。

面向对象方法学的出发点和基本原则，是尽量模拟人类习惯的思维方式，使开发软件的方法与过程尽可能接近人类认识世界解决问题的方法与过程，从而使描述问题的问题空间（也称为问题域）与实现解法的解空间（也称为求解域）在结构上尽可能一致。

1.3 软件生命周期

一般说来，软件生命周期由软件定义、软件开发和软件维护三个时期组成，每个时期又进一步划分成若干个阶段。

软件定义时期通常进一步划分成三个阶段，即问题定义、可行性研究和需求分析。

开发时期具体设计和实现在前一个时期定义的软件，它通常由下述四个阶段组成：总体设计，详细设计，编码和单元测试，综合测试。其中前两个阶段又称为系统设计，后两个阶段又称为系统实现。

维护时期的主要任务是使软件持久地满足用户的需要。具体地说，当软件在使用过程中发现错误时应该加以改正；当环境改变时应该修改软件以适应新的环境；当用户有新要求时应该及时改进软件以满足用户的新需要。

1 问题定义

问题定义阶段必须回答的关键问题是：“要解决的问题是什么 ?”

2 可行性研究

对于上一个阶段所确定的问题有行得通的解决办法吗 ?

3 需求分析

这个阶段的任务仍然不是具体地解决问题，而是准确地确定“为了解决这个问题目标系统必须做什么”，主要是确定目标系统必须具备哪些功能。这个阶段的一项重要任务是用正式文档准确地记录对目标系统的需求，产生规格说明书。

4 总体设计

这个阶段必须回答的关键问题是：“概括地说，应该怎样实现目标系统”。应设计出低成本、中等成本、高成本 3 种方案，推荐最佳方案。

5 详细设计

这个阶段的关键问题是“应该怎样具体实现这个目标系统。”设计每个模块，确定实现模块功能所需要的算法和数据结构。

6 编码和单元测试

这个阶段的关键任务是写出正确的容易理解、容易维护的程序模块

7 综合测试

这个阶段的关键任务是通过各种类型的测试使软件达到预定的要求。

集成测试：根据设计的软件结构，把经过单元检验的模块按某中选定的策略结合起来，在装配过程中对程序进行必要的测试。

验收测试：按照规格说明书的规定，由用户对目标系统

进行验收。

8 软件维护，通常有四类维护活动：

改正性维护：诊断和改正在使用过程中发现的软件错误

适应性维护：修改软件以适应环境的变化

完善性维护：根据用户的要求改进或扩充软件使它更完善

预防性维护：修改软件为将来的维护活动预先做准备

1.4 软件过程

## 1．4．1瀑布模型

1 特点

(1) 阶段间具有顺序性和依赖性

这个特点有两重含义；第一，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；第二，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获得正确的结果。

(2) 推迟实现的观点

实践表明，对于规模较大的软件项目来说，往往编码开始得越早最终完成开发工作所需要的时间反而越长。

瀑布模型在编码之前设置了系统分析与系统设计的各个阶段，分析与设计阶段的基本任务规定，在这两个阶段主要考虑目标系统的逻辑模型，不涉及软件的物理实现。

清楚地区分逻辑设计与物理设计，尽可能推迟程序的物理实现，是按照瀑布模型开发软件的一条重要的指导思想。

(3) 质量保证的观点

软件工程的基本目标是优质、高产。在瀑布模型的每个阶段都应坚持两个重要做法：

第一，每个阶段都必须完成规定的文档，没有交出合格的文档就是没有完成该阶段的任务。完整、准确的合格文档不仅是软件开发时期各类人员之间相互通信的媒介，也是运行时期对软件进行维护的重要依据。

第二，每个阶段结束前都要对所完成的文档进行评审，以便尽早发现问题，改正错误。

事实上，越是早期阶段犯下的错误，暴露出来的时间就越晚，排除故障改正错误所需付出的代价也越高。因此，及时审查，是保证软件质量，降低软件成本的重要措施。

瀑布模型的成功在很大程度上是由于它基本上是一种文档驱动的模型。两个图 1。 3 1。 4

## 1．4．2快速原型模型

1 快速建立起来的可以在计算机上运行的程序，能完成的功能往往是最终产品的一个子集。

2 快速原型模型的特点是不带反馈环，基本上按线性顺序开发。

原型系统已经通过与用户交互而得到验证，据此产生的规格说明文档正确地描述了用户需求，，不会因为规格说明文档的错误而进行较大的返工。

开发人员通过建立原型系统已经学到了许多东西，在设计编码阶段发生错误的可能性比较小，自然减少了反馈。

## 1．4．3增量模型

又称渐增模型。使用增量模型开发软件时，把软件产品作为一系列的增量构件来设计、编码和测试。每个构件由多个相互作用的模块构成，能完成特定的功能。第一个增量构件往往实现软件的基本需求，提供最核心的功能。

分解时唯一必须准守的约束条件是，当把新构件集成到现有软件中去时，所形成的产品必须是可测试的。

优点：

1 能在较短的时间内向用户提交可完成部分功能的产品

2 逐步增加产品功能可以使用户有较充裕的时间学习和适应新产品，减少一个全新的软件可能给客户带来的冲击

困难：

1 在把每个新的增量构件集成到现有的软件体系结构中时，必须不破坏原来已经开发的产品

2 软件体系结构必须是开放的

3 本身具有矛盾性，一方面要求开发人员把软件看作一个整体，另一方面要求开发人员把软件看作构件序列，且构件间彼此独立

## 1．4．4螺旋模型

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。

1 确定阶段目标，为完成阶段目标选择方案，设定这些方案的约束条件

2 用建造原型的方法来排除上述方案中潜在的风险

3 用瀑布模型开发

4 评价该阶段工作，计划下阶段工作

优点

1 软件重用

2 测试量减少

3 维护是模型的另一个周期

4 螺旋模型主要是用于内部开发的大规模软件项目，它是风险驱动的。

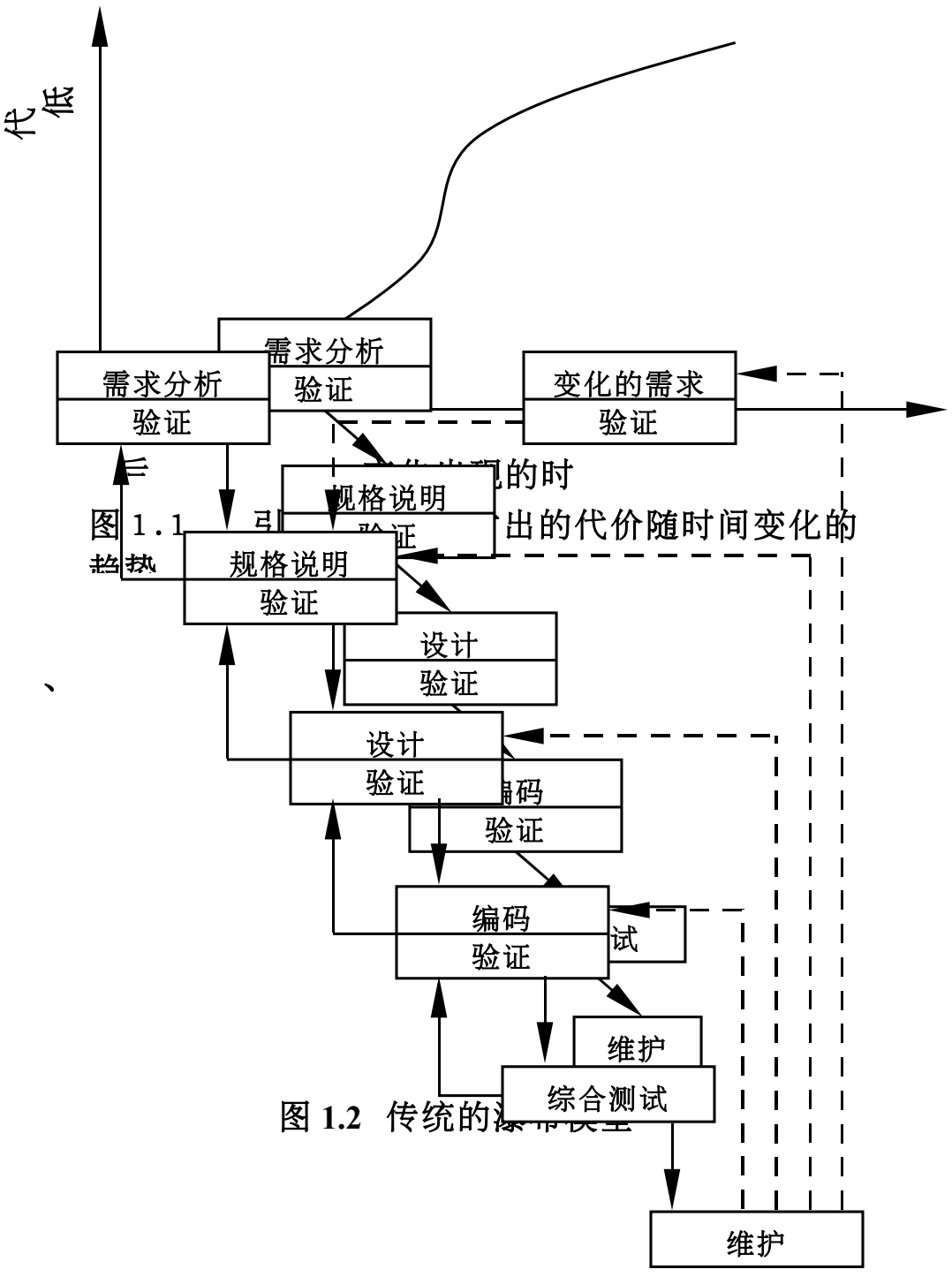


图 1.1 引[illegible]出的代价随时间变化的趋势

图 1.2 传统的瀑布模型

图 1.3 实际的瀑布模型

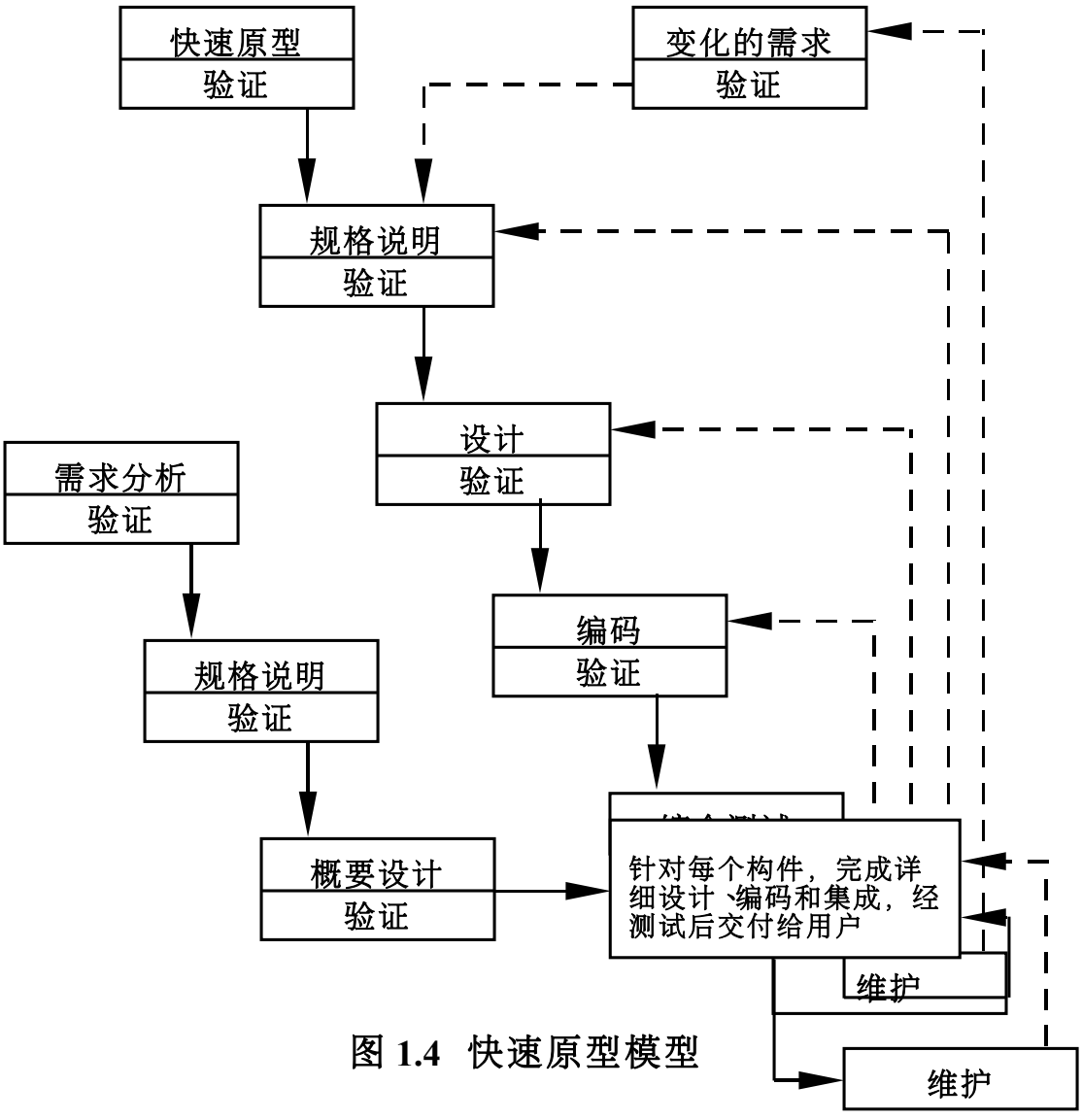


图 1.4 快速原型模型

图 1.5 增量模型

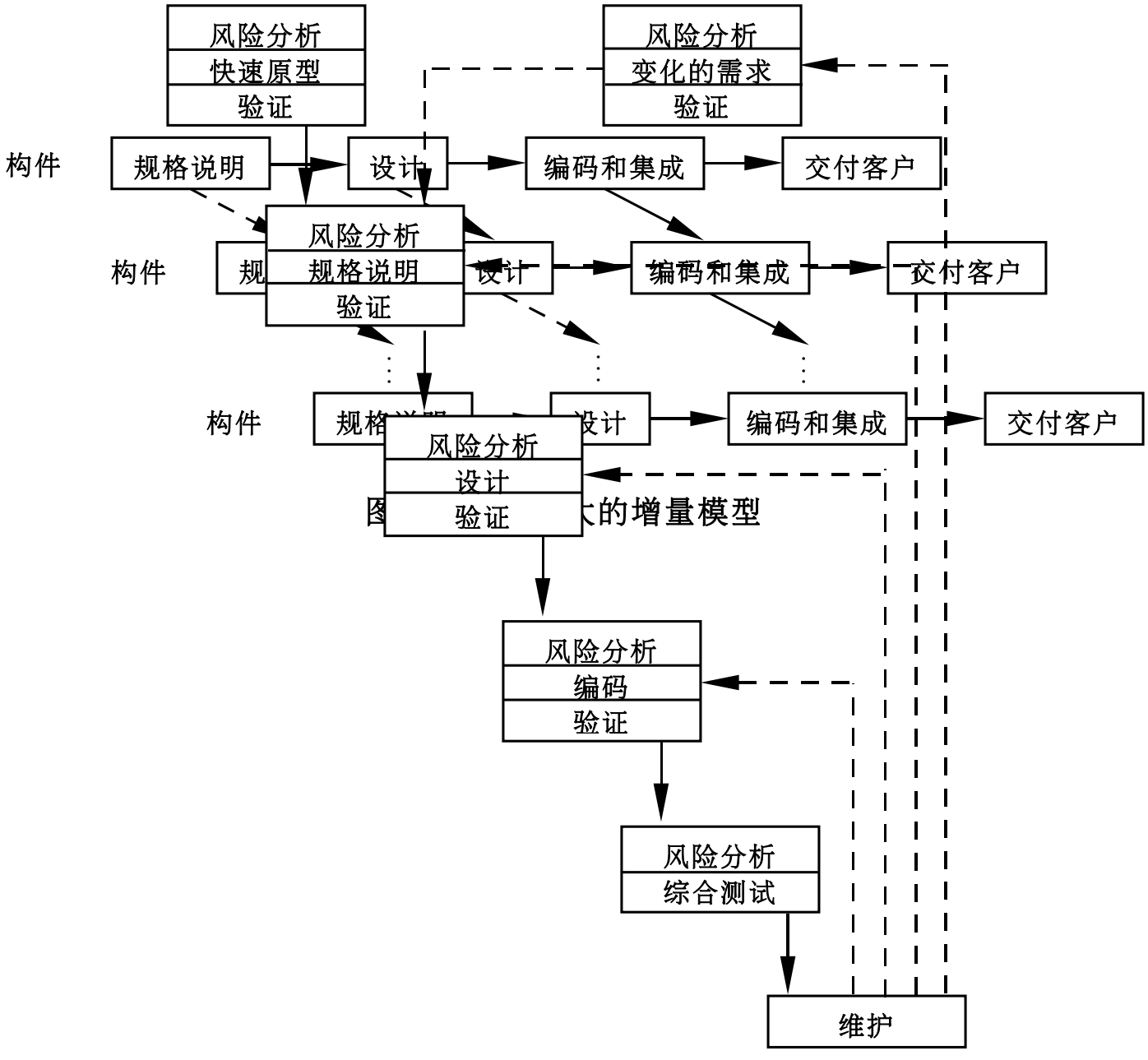
风险分析
快速原型
验证
风险分析
变化的需求
验证
构件
规格说明
设计
编码和集成
交付客户
风险分析
规格说明
验证
构件
规
设计
编码和集成
交付客户
构件
设计
编码和集成
交付客户
风险分析
设计
验证
风险分析
编码
验证
风险分析
综合测试
维护

图 1.7 简化的螺旋模型