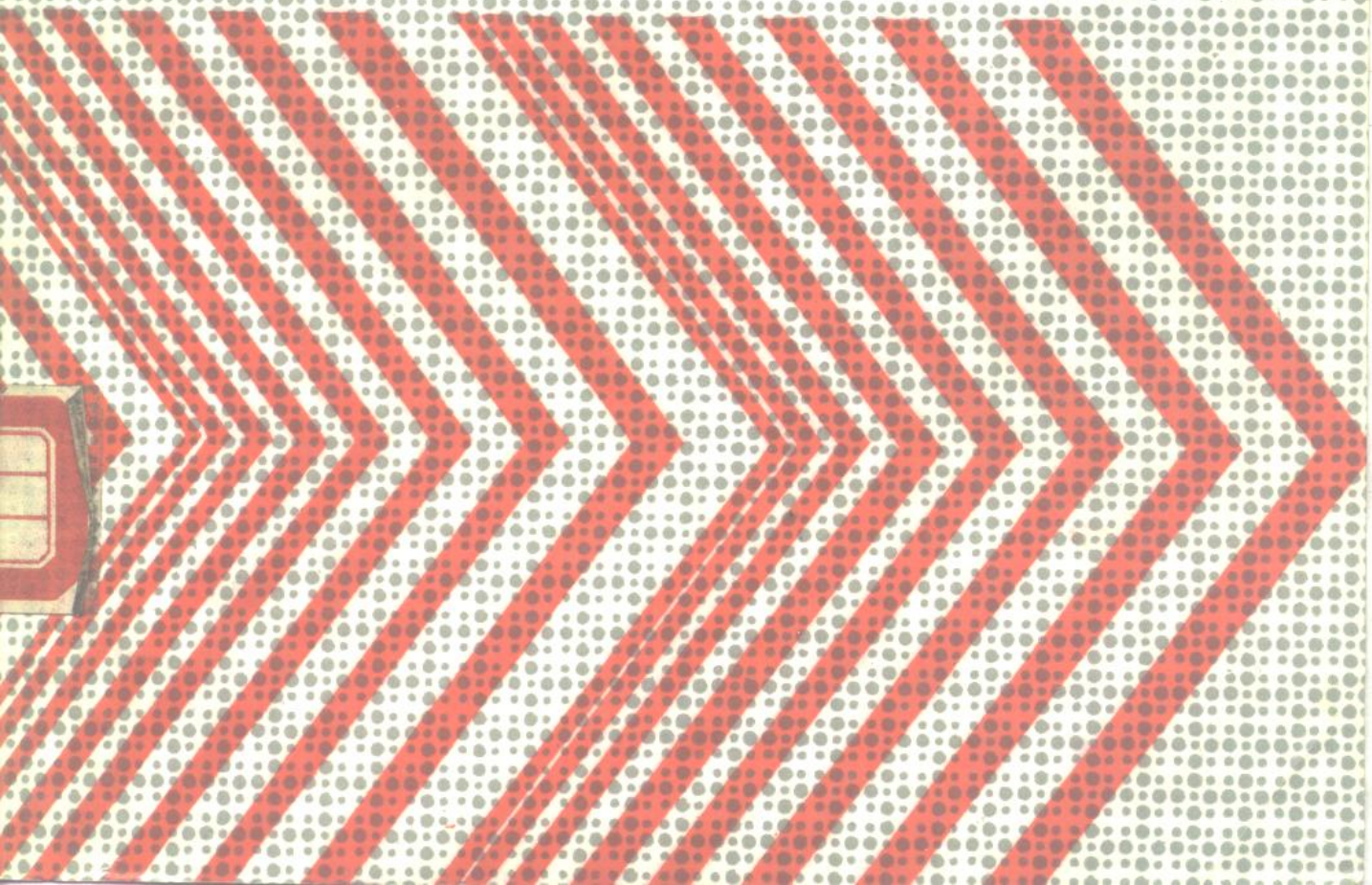
软件开发指南

何培民

清华大学出版社



软件开发指南

何培民 编著

清华大学出版社

内 容 简 介

本书系统地介绍了计算机应用系统的软件开发步骤、方法以及所涉及的软件工具，包括软件的可行性研究，需求分析，概要设计，详细设计，软件实现，软件测试，软件开发管理等。

本书通俗易懂，简明扼要，是作者多年从事软件开发的经验总结，是所有软件开发人员的必备参考书。

软 件 开 发 指 南

何培民 编著

☆

清华大学出版社出版

北京 清华园

北京昌平环球科技印刷厂印刷

新华书店总店科技发行所发行

☆

开本：787×1092 1/16 印张：10 字数：250 千字

1991年3月第1版 1991年3月第1次印刷

印数：00001～10000

ISBN 7-302-00804-3/TP·289

定价：4.60 元

前　言

随着计算机应用的发展，计算机软件开发的工作量越来越大。但是，无论是在办公自动化、教育、通讯、交通、工程设计等领域，还是在这些领域中的某一应用方面，由于缺乏科学的系统管理，缺少各开发步骤的检验标准，形成了不同设计者的不同设计风格，致使设计出的软件产品难以移植、推广和维护。

本书力求以通俗易懂的方式介绍应用系统的软件开发步骤、方法以及所涉及的软件工具。撰写本书的目的在于使计算机软件设计人员能够按照一定的规矩、方法和必要的程序进行软件开发；也使从事计算机软件或项目的管理人员能够对软件开发的全过程有所了解。不同的读者可以跳过书中的某些章节而直接阅读感兴趣的内容。

本书的初稿实际上是一本汇集计算机软件工作者的经验和大量短小精悍的实例的教材。经济管理学院石成孝花费了大量心血，对全文进行了修改；北方交通大学陈景艳审阅了全文，提出了很多修改意见。在此表示十分的敬意和感谢。

由于水平有限，书中错误和不妥之处在所难免，敬请读者不吝批评指正。

作　者

1988.10

目　录

第一章 绪 论

人类正在由工业化社会进入信息化社会，以计算机产业和计算机应用服务业为支柱的信息工业是这个社会的主要基础之一。目前世界各主要工业国家的信息工业的增长速度都已大大超过整个国民经济的增长速度。计算机在国民经济的各个领域，诸如国防、科学、教育乃至社会生活中发挥着越来越大的作用。

计算机软件是计算机应用的灵魂。在计算机日益获得广泛应用的过程中，其软件在计算机系统中的比重不断增加。表现为软件与硬件的费用在计算机应用系统中的成本比例不断增加。

随着计算机应用范围的不断扩大，软件系统的规模和复杂程度也不断增加。美国第四代宇宙飞船对软件的要求几乎成指数倍增长。70年代末期的“穿梭号”飞船的软件规模已达到4000万行目标码。在这种情况下，对软件的需求与满足程度之间的矛盾越来越尖锐化。即使在软件产业最发达的美国，软件需求量每年大约递增12%，而软件人员的劳动生产率每年只能递增4%，也就是说，满足软件需求的积压程度，将以每年8%的速度递增。这就是人们常说的当今世界普遍存在的“软件危机”。

“软件危机”的主要表现为以下几点：

1. 计算机应用系统越来越庞大。软件复杂程度越来越高。以程序员各自为政的小生产者手工业式和小集团作坊式的生产已无法驾驭大型软件系统的开发。

2. 由于软件发展的历史短，人们来不及积累大量的历史数据和模型作为系统开发的指南。因而导致进度和费用估计粗略，缺乏切实的生产率指标，进度一拖再拖，费用一增再增。整个软件缺乏有效的管理措施。

3. 产品质量低劣，出错率高，可靠性差，无法满足用户的真正要求，需求矛盾日益尖锐。

4. 应用系统的需求量日益增长，整个社会对软件人员的需求急剧上升，造成软件人员奇缺，供求关系失调。软件生产满足不了社会需求的发展。

5. 已开发的软件维护成为最头痛的问题。软件维护费用占了整个软件生存周期总费用的70%～80%，其工作量占整个工作的80%～90%。而且，随着时间的推移，系统的可靠性下降，再使用修修补补的办法来改进现有系统，难以满足用户的愿望。

为了克服“软件危机”，世界各国都在努力研究新的软件开发方法；制定软件工业化生产规范；研制与开发新的软件工具，并进一步发展成软件工程环境。这些内容构成了80年代最重要的工程科学——软件工程，它是软件产业的技术基础和前提。

自从1968年第一次提出“软件工程”概念以来，这一门科学在理论和实践方面不断发展，从而使整个软件的开发、管理和维护过程发生了深刻变化，提高了软件开发效率和软件质量。

我国计算机应用经过一段起步过程以后，目前已进入了一个蓬勃发展的新时期，正在经历着一个从单项应用到系统应用，从本地批处理应用到大范围网络联机应用的转变过程。据

统计，1987年全国已安装大、中、小型计算机8000多台，微型机20多万台，许多大型信息管理系统及软件计划正在酝酿和实施之中。在这种形势下，总结和吸收国外软件开发经验，在系统初创阶段就将软件开发纳入软件工程的轨道，可以使我国的计算机应用水平提高到一个新的水平，从而加速四个现代化建设的步伐。

本书重点阐述软件开发的整个过程中应遵循的过程和步骤，不仅告诉你要做什么，更重要的是告诉你怎样做。

1.1 软件工程原理

软件工程是在计算机应用日益广泛、对于软件产品的功能、质量、成本及生产率不断提出更高要求的形势下，逐步形成的一门新学科，是一种非常新颖而又不够成熟的工程分支。

软件从定义、开发、维护直到报废、结束其使命，要经历软件计划、需求分析、概要设计、详细设计、实现、组装测试、确认测试、运行和维护等八个阶段。这些构成了软件生存周期。软件工程就是应用于计算机软件的定义、开发和维护的一整套工序、方法、工具、文档和实践标准。软件工程，除了具有一般工程的共性外，还有它的特殊性，即软件开发是非实物性的，是人的逻辑思维过程，具有不可见性、抽象性和知识密集性。因此，软件生产规范、软件开发工具和环境、软件工程管理等方面，既要以工程科学为基础，又要适应软件开发的特殊性。概括起来，软件工程的基本原理是：

1. 严格按计划进程管理；
2. 每一活动步骤要编制出完整精确的文档；
3. 坚持进行阶段评审；
4. 实现严格的产品基线控制；
5. 采用能够加速软件开发的各种方法和工具。

软件产品的定义、开发和维护活动的全过程，必须遵守上述原理，才能保证产品质量、缩短开发时间、降低开发和维护费用，并使整个活动科学化、条理化，更有成效。

1.1.1 计划管理

一个软件项目，除了硬性规定的交付日期之外，缺乏科学而周密的计划，是软件开发工作中的普遍现象。调查研究结果表明，不成功的软件项目中，有50%以上是由于计划不周造成的。因此，建立周密的计划，并在生产活动中严格按计划进行管理，是软件项目取得成功的先决条件。

软件工程计划按软件开发活动步骤应该包括：

1. 项目实施总计划；
2. 软件配置管理计划；
3. 软件质量保证计划；
4. 测试计划；
5. 安全保密计划；
6. 系统安装计划；
7. 运行和维护管理计划。

这些计划要面向开发过程的各个段阶。参与各阶段工作的技术人员必须严格按计划行事。各层次的管理人员要按计划监督和管理项目开展工作，决不受任何内部、外部条件干扰随意修改计划。必要的计划修改，必须经过严格的审批手续才能生效，否则同样会造成软件开发工作的混乱，甚至失败。

1.1.2 文档

软件开发一般需要人力和自动化资源的重大投资。为了合理地使用这些投资，保证软件开发成功，并得到有效的运行和维护，必须对软件开发进行严格管理。进行管理的基础是对被管理对象的了解。然而、软件开发是脑力劳动，具有不可见性，他人是不大容易了解的。为了解实现对软件开发过程的管理，在开发工作的每一阶段，都需按照规定的格式编写出完整准确的文档资料。文档的作用是：

1. 作为开发人员在一定阶段内承担任务的工作结果和结束标志；

2. 向管理人员提供软件开发工作的进展情况，把软件开发过程中的一些“不可见”的事物转换成“可见”的文字资料，以便管理人员在各个阶段检查开发计划的实施情况，使之能够对工作结果进行清晰的审计；

3. 记录开发过程中的技术信息，以便协调工作，并作为下一阶段工作的基础；

4. 提供有关软件维护、培训、流通和运行信息。有助于管理人员、开发人员、操作人员和用户之间工作的了解；

5. 向潜在用户报导软件的功能和能力，使之能判断该软件能否适合使用者的需要。

软件文档有两类：一类是开发过程中填写的各种图表，可称之为工作文档；另一类是所编制的技术资料或技术管理资料，可称之为产品文档。

1.1.3 评审

评审是软件工程的主要原理之一。

软件质量的保证工作不能推迟到实现阶段（即编码阶段）结束之后进行，这是因为：

1. 程序中的大部分错误是在编码之前造成的。据统计，设计阶段产生的错误大约占63%，而编码错误仅占37%。

2. 错误的检测与改正时间越晚，所付出的代价也就越高。通过对一些大型软件项目分析表明：如果在设计阶段发现一个错误，改正所需的费用假设为1；那么同样的错误若在测试开始前发现，其费用则为6.5；而在测试期间发现，其费用则为15；而在交付使用之后发现，其费用可高达67。

此外，错误还会被“放大”，即早期存在的一个错误，如果没有及时纠正，会造成更大的错误。这种情况可用图 1-1 缺陷放大模型来说明。

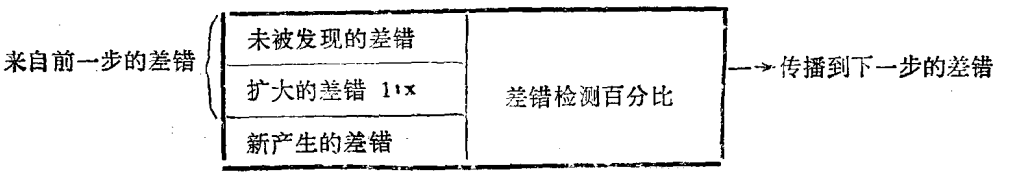


图 1-1 缺陷放大模型

图1-2说明无评审和有评审两种不同的结果。无评审时软件开发结束遗留24个差错，而有评审仅残留6个差错，二者差异很大。当然评审也要付出代价，但要小得多，一般只是无评审时所付代价的1/3。大量实践表明，代价宁肯花在评审上，也不要带到维护阶段，否则代价就更大。

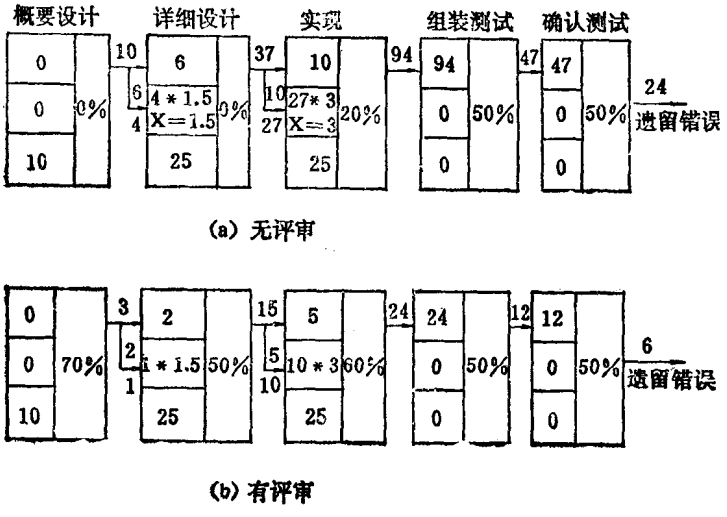


图 1-2 缺陷扩大

评审要严肃认真。根据质量保证计划，对每个阶段的工作结果和文档，由专门的评审小组进行评审。如果评审中发现错误或异常问题，应指示有关人员对文档进行修改，修改后再次评审，直到评审小组通过为止。任何一步的工作未经评审通过，不得开展下一步工作。

1.1.4 基线控制

软件开发各阶段所产生的文档、报告、清单和数据，构成软件配置。全部软件配置是一个软件产品的真正代表，必须使其保持精确。此外，软件开发中的某一阶段或在维护阶段的需求变更，均要引起软件配置的改变，这种改变必须严格加以控制和管理，保持修改信息，并把精确、清晰的信息传递到下一个软件开发步骤。软件配置管理就是对软件配置的标记、控制、评审、记载等一整套的活动。

在软件开发过程中，对某一交付文档实行软件配置控制的时刻定义为基线。如果在文档建立期间，过早地施加严格管制，就可能降低生产率。只有在特定的软件工程步骤完成以后，并经过正式的评审和批准，才能实施软件配置控制。这就是所谓基线控制。软件配置基线分计划基线、需求基线、设计基线、实现基线，测试基线等。凡某一基线以后的工作要变动前一基线的文档，就需对文档的全部修改进行评审。

1.1.5 软件开发方法和工具

软件工程原理鼓励研制和采用各种先进的软件开发方法和工具，以便不断提高软件生产率。从软件工程概念提出的初期，人们着重研究的是各种新的程序设计技术，其中自顶向下程序设计、结构程序设计、主程序员负责制等是70年代初期产生发展起来的。这种程序

设计方法和技术可使软件开发效率改善因子达1.51，使维护效率改善因子达2.07。

继程序设计技术、方法的改进，70年代中期相继发展了许多适用于软件开发各阶段的方法。其中面向数据流的方法（Yourdon方法）和面向数据结构的方法（Jackson方法）是较为流行和实用的方法。

但是这些软件方法和技术是建立在以图表为工具的手工作业基础之上的。软件开发人员只为别人开发自动化的工具，而在自己的生产活动中依靠的仍是人的密集型劳动。显然这无法根本扭转软件生产率低的落后局面。

为了改变这种局面，一个直接而有效的途径是从软件人员角度出发，在软件工程实践方面提供一套软件开发和维护的支持，这就是目前在国际上引起广泛重视的软件工具，进而发展成软件工程环境。

所谓软件工具一般是指为了支持软件人员开发和维护软件而使用的软件。例如项目估算工具、需求分析工具、设计工具、编码工具、测试工具和维护工具等。

但是就一般的软件工具而言，它们对大型软件开发和维护项目的支持能力是有限的，即使可以使用众多的软件工具或工具系统，但由于这些工具之间往往相互隔离、独立存在，无法支持一个统一的软件开发和维护过程。因此，目前人们又在工具系统的整体化及集成化方面开展一系列研究工作，使之形成完整的软件工程环境。其目的是使软件工具支持整个软件生存周期，不仅能支持软件开发和维护中的个别阶段，而且能支持从软件计划、需求分析、设计、编码测试到运行维护等所有阶段。并且做到不仅支持各阶段中的技术工作，还要支持管理和服务工作，以保持项目开发的高度可见性、可控性和可追踪性。

1.1.6 对待软件工程的错误概念

为了按时获取高质量的软件产品，软件开发必须认真贯彻执行基于上述原理制定的软件工程方法。但是，软件工程实践所提出的技术和方法，在某些部门并未得到认真的贯彻。究其原因，除这门新兴的学科尚有许多不足，需进一步发展完善以外，主要在于习惯势力和头脑中的错误观念在起作用。这方面的问题，归结为：

1. 有一个对总目标的一般描述，就可以开始编码，细节可以在以后补充。

实际上，软件定义不准确是软件开发失败的主要原因。前期工作不能忽视，一个关于功能、性能、接口、设计约束和有效性准则的详细描述是不可缺少的。这些只能在用户和软件开发设计人员之间进行全面充分的讨论之后才能确定。

2. 写出了程序并且能运行，就算完成任务了。

软件开发不等价于编程序。最初是计划，接着是需求分析、设计、实现、测试，而最后还有软件维护。

3. 软件需求不断变化，软件很灵活，所以改变也容易。

确实，对软件的要求经常有变化，但是变化所付出的代价在不同阶段上的差别是很大的（见图1-3）。

在软件定义期间，改变需求容易适应。用户在评审需求说明时提出修改建议，这时改变容易适应，对代价影响不大。在设计期间，所做的改变，其代价影响迅速增长。因为这时软件结构已经建立，资源已经安排，改变需要引起修改设计、增加资源，会引起动乱。在实现编码和测试期间，改变目标、功能、性能、接口或任何其它软件特性会严重影响代价。在一

个项目的末期要求改变，会比早期要求同样的改变，所付代价要高出一个数量级。

4. 评审是多余的，太费时间。

评审是进行软件管理控制和技术控制的重要措施，尤其是在软件计划阶段，除评审外没有其它手段能鉴别项目的可行性。在软件生存周期中，各个阶段均应坚持严肃认真的评审工作，否则无法管理控制软件质量。

5. 进度拖后了，只要加人就可赶上去。

软件开发与机械制造、建筑工程不同，不是单纯的机械作业或手工作业过程，而是知识密集的逻辑过程。在一个拖了进度的软件项目中增加人员往往会使它更加拖延。这是因为，当新人增加时，由于弄清情况需要学习、需要原有工作人员进行频繁的通讯交流，因而减少了用于生产性开发工作的时间。

6. 软件产品只要设计出能运行的程序就行了，不需要那么多文档资料。

一个能运行的软件仅仅是配置的一部分，文档资料构成了软件开发的基础，每个阶段必须产生相应准确的文档，不能后补，更不能草率从事。更重要的是，这些软件配制，是软件维护的指南。

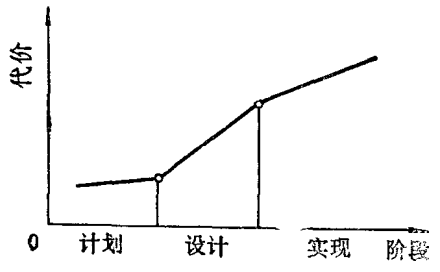


图1-3 改变软件所需代价

7. 软件能运行，维护是极少的，一旦需要改变，也很容易对付过去。

人们往往对软件维护的现实性估计过低，这会给管理人员和技术人员造成过重的精神压力。如果一个系统在预算中给维护工作投入10%～15%的人力和投资，但实际上可能要达到55%～70%。

有关这类错误认识还有许多。造成这些错误认识的原因也是多方面的。有些是计算机发展带来的，有些是由于人们的习惯势力的影响。但更主要的是缺乏对软件工程的教育和管理人员缺乏这方面的知识造成的。

1.2 软件生存周期

软件生存周期建立了软件工程事件的顺序表。要想使软件开发获得成功，必须遵守软件工程原理，准确地把握软件生存周期各个阶段的任务、实施步骤，完成标志及交付的文档。

一般说来，一个软件从酝酿到结束其使命，要经历定义、开发和维护三大阶段，按其作业内容又可将它们划分八个阶段（见图1-4）。

应该注意的是，在软件生存周期的每个阶段上，都要对各主要任务进行评审。

1.2.1 定义阶段

如图1-4所示，软件的目标和计划是在定义阶段确定的，软件产品是在开发阶段制作的，而它的真正作用是在运行维护阶段发挥的。如果没有对用户要求和现实环境进行准确、综合分析，就难以对软件提出恰如其分的目标，从而给开发阶段的工作带来盲目性，使开发出来的产品无法满足用户的真正要求，甚至不得不前功尽弃、推倒重来。所以定义阶段的软件开

发目标和软件计划工作的好坏，直接决定了将要开发的软件是否能够成功。

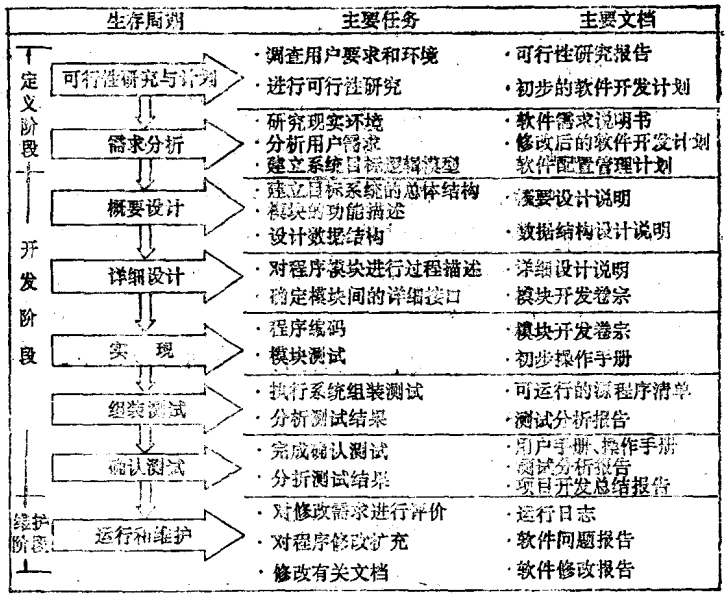


图 1-4　软件生存周期的八个阶段

1.2.1.1　可行性研究

以计算机为基础的系统开发，一般受资源和交付日期的限制。尽早对项目进行可行性研究是十分必要的。如果在可行性研究阶段能够判断一个系统不可行，就可以避免无效劳动和投资。

可行性研究涉及下列问题：

1．　现有系统分析。说明现有系统必须被一个新系统所代替的理由。

2．　所建议的新系统分析。说明新系统对现有系统的改进之处及技术上的可行性。

3．　说明可供选择的各个方案，并评价它们的优缺点及其约束条件。

4．　投资效益分析，即经济上的可行性。

5．　社会条件方面的可行性。

可行性研究以可行性研究报告为完成标志，并报请有关部门审批，决定是否进行下一步的工作。

1.2.1.2　软件计划

可行性研究报告被批准后，便可着手进行软件计划工作。

软件计划步骤应对软件作用范围、工作环境和基本功能特性加以研究，确定要做什么，不要做什么，做到什么程度。同时，估算出所需的资源、工作量、费用和进度。软件计划是以产生初步软件计划文档为完成标志。该文档应经有关管理部门审批，并据此作出项目“进行”或“停止”的决策。

初步软件计划中确定的成本和进度是进行软件需求分析的基础。但是在对系统进行需求分析时，完全可能发生超出初步估算的情况。在这种情况下，修改初步软件计划应得到管理部门的同意并经复查，做为正式软件计划。

1.2.1.3 需求分析

需求分析是定义阶段的最后一步，由开发人员和用户双方共同完成。软件需求分析依据初步软件计划中规定的软件特性、进度、成本需求，作进一步的细化工作。

软件需求分析首先对现实环境和用户要求进行深入研究，明确外部环境对系统的支持和制约程度。回答软件能做什么，不能做什么，能做到什么程度。产生一份详细的软件需求说明书，说明信息流及其界面，功能需求，设计要求和限制，测试准则和质量保证计划，软件配置管理计划。这些结果构成软件设计的基础，是软件开发取得成功的关键。

值得指出的是，我们不能把需求分析单纯的理解为这是一种从用户需求转换为面向开发人员的软件功能的手段，而是从用户需求导出系统模型，这种模型应象建筑模型一样，在着手开发之前，使用户能直接且直观地了解将来开发出来的系统是否真正满足用户的要求。

对软件需求说明进行评审之后，必须对初步软件计划重新加以评审。需求分析中得到的结果可能会影响资源、成本和进度。

1.2.2 开发阶段

开发阶段涉及到软件产品制做的各个步骤，它包括概要设计、详细设计、实现、组装测试和确认测试。这些作业均应严格按照计划阶段确定的软件计划和软件需求说明书中各项规定来进行。

1.2.2.1 概要设计

概要设计的任务是把软件需求说明转换成软件结构说明。这一步集中于软件的整体结构，即确定功能模块之间的关系、定义各功能模块间的数据接口、控制接口，设计数据结构，规定约束或限制。

概要设计以交付概要设计说明和数据结构设计说明文档为完成标志。对这些文档应进行严格评审，以检查设计质量、设计与需求的一致性和可追溯性。

概要设计提供了一个完整的软件系统框架，这是整个开发阶段的基础。

1.2.2.2 详细设计

详细设计是概要设计的细化，采用设计工具详细描述功能模块的内部过程。

详细设计是一个逐步细化的过程，每遍细化的结果应经概要设计人员予以评审，以保证设计满足规定的各项要求。

值得注意的是，设计人员不要在这一阶段随便引入未经许可的修改和增加。这个阶段的任何变动均是要追溯到概要设计甚至软件需求，必须报请有关人员审查批准，并对有关软件配置进行修改评审。

在一个模块的详细设计过程中，应建立相应的软件开发卷宗文档。设计完成后，应进行设计评审，以发现功能错误和逻辑错误。

1.2.2.3 实现

详细设计得到批准后，便可开展实现阶段的工作。实现以模块为单位来进行，任务包括编码和模块测试。

所谓编码是将详细设计说明转化为所要求的程序设计语言或数据库语言书写的源程序。

模块测试应按测试计划中规定的标准来测试模块的性能，验证模块接口与设计说明的一致性。

各模块测试一经完成，必须对最终源程序清单、测试代码及测试结果进行评审。评审通过后，这些文档应纳入模块开发卷宗。

1.2.2.4 组装测试

组装测试的任务是将经过模块测试的各个模块进行装配并测试，从而形成一个完整的系统。

组装测试应严格按照软件测试计划的要求及各个模块的完成日期，制定出测试的策略、步骤。

测试可以采用“自顶向下”或“自底向上”的方法，也可以把两种方法结合起来，有步骤地装配模块、测试模块间的界面、数据结构的完整性、模块顺序的正确性。

组装测试应由独立于编码小组的测试小组人员来进行。当所有模块都已组装起来，修改了所发现的错误，成功地达到了测试计划中规定的要求，并形成测试分析报告，经评审人员认可后，组装测试便告结束。

1.2.2.5 确认测试

确认测试是软件开发中最重要、最困难的一步。

确认测试的任务是要证明所开发的软件符合软件需求说明中所定义的全部功能及性能要求。

测试之前应准备一个确认测试计划，这一计划是根据软件需求制定的。计划中规定所要测试的功能、预期的结果及测试每种功能的进度。

测试小组应由设计和实现该软件以外的人员组成，最好有用户单位的领导人员参加。测试应从最终的用户角度出发，在现场进行。测试过程应该进行监督，监督和错误纠正措施应该在现场测试开始之前就确定下来，以避免执行过程中含糊不清引起混乱。

测试完成后，由负责的管理人员对记录下来的测试性能数据和结果进行仔细分析，并写成测试分析报告。测试分析报告经有关方面评审确认后，便结束了软件开发工作，这些软件则可作为一个产品投入运行。

在确认测试过程中要对设计文档、用户手册和操作手册做最后修改，这些资料做为“最终设计”文档，是软件生存周期的最后阶段——软件维护的基础。

1.2.3 维护阶段

当软件产品制做完成交付使用后，便开始了软件生存周期的最后阶段——维护阶段。这是一个极为重要的阶段。

软件维护是软件生存周期中最不被人重视的一个阶段。造成这种情况有以下几方面原因：

1. 心理上的问题：在人们心目中，维护是二流工作，不象开发工作那样吸引人。

2. 管理上的问题：维护工作不象开发工作那样，当作一个正式项目列入计划，作为成果进行鉴定，记入个人技术档案。往往安排新手承担。

3. 技术上的问题：没有可用的工具，缺乏完整可靠的文档资料作为依据。

4. 时间上的问题：一旦用户部门提出需求变更或运行中出现故障，往往需要技术人员放下手中的其它工作立即去解决，事前无准备，临时应付。

5. 开发上的问题：开发的软件产品未经严格评审、测试，文档不全。为了赶进度，把

一切问题推至维护阶段。

6. 维护上的问题：维护工作缺乏严格管理，只改代码，不改文档，造成二次以后的维护更加困难。

上面所述的所有问题，均可归结于贯彻软件工程不力造成的。尤其是软件生存周期作为一个整体，决不能把维护阶段割裂开来。

对大型软件分析表明，软件运行中发现的错误，最多三分之一是由于编码失误造成的，大多数错误来自需求分析、软件设计。但是维护阶段纠正错误所花代价极高，所以说软件定义和开发阶段必须严格执行软件工程规定的工序和管理方法，着眼于软件质量和维护，不能把问题带到维护阶段来。当然，软件维护期间的管理不当也会增加维护方面的问题。

与定义和开发阶段相比，维护阶段要求对管理、制定技术计划和系统的评审需给予更大的注意。

维护分为三种类型：

1. 正确性维护——改正在开发阶段产生、在测试阶段又没有发现的错误。

2. 适应性维护——为适应软件的外界环境的变化（如新的操作系统、不同的硬件）引起的软件修改。

3. 完善性维护——为扩充软件功能或改善性能而进行的修改。

这三类维护所占的百分比分别为20%、25%和55%。

维护工作应象开发工作那样事先作好计划。报告软件问题的工序应在软件投入运行之前就建立，明确工作人员和管理人员的职责，并应设立一个机构来评价其改变对全局的影响。软件修改必须对软件配置做出相应准确的修改和评审。

1.3 软件质量

软件产品与其它工业产品一样，产品的价值取决于产品的质量。

软件质量特性是多方面的。一个正确执行了用户指定功能的软件系统不一定是质量很高的软件系统。因为该软件可能难于读懂和修改，这样将增加软件维护费用；该软件可能不易使用，不小心就用错；该软件可能依赖于所实现的机器，难以与其它系统集成等等。因此，软件开发必须考虑软件产品的整体质量，不能单纯追求正确性或效率等一、两个特性而忽视了其它质量特性。

为了保证软件产品的整体质量，应该在软件开发和管理过程中遵循一套完整的质量评价准则，定量的评价每一阶段的工作，采用各种行之有效的度量方法，以便定性管理软件项目的生产。软件质量度量（SQM）就是这样一种技术。这种技术用来管理指导软件开发，作为用户和开发人员对最终产品的评价和验收标准。

1.3.1 软件质量度量模型

国际标准化组织（ISO）于1985年建议软件质量度量模型由三层组成：

高层：软件质量需求评价准则（SQRC）

中层：软件质量设计评价准则（SQDC）

低层：软件质量度量评价准则（SQMC）

该模型的三层关系如图 1－5 所示。ISO 认为，应对高层和中层建立国际标准，以便在国际范围推广应用 SQM 技术。而低层可由使用单位视实际情况而定。

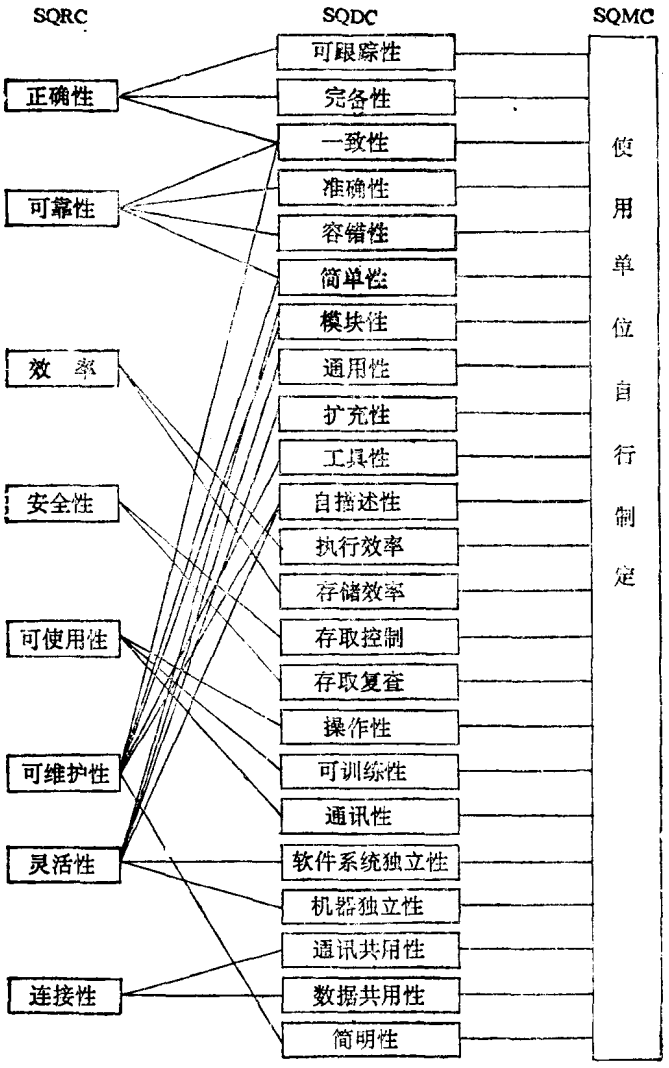


图 1-5　软件质量度量模型

SQRC由八个元素组成，其定义为：

（1）正确性：程序满足规范书及完成用户目标的程度。

（2）可靠性：程序在所需精度下完成其功能的期望程度。

（3）效率：软件完成其功能所需的资源。

（4）安全性：对未经许可人员接近软件或数据所施加的控制程度。

（5）可使用性：人员学习操作软件、准备输入和解释输出所需的努力。

（6）可维护性：在需求变更时更改软件或弥补软件缺陷的容易程度。

（7）灵活性：改变一个操作程序所需的努力。

（8）连接性：与其它系统耦合所需的努力。