分类 **TP311** 密级

西北师范大学

硕士学位论文

COCOMO II 模型的研究与应用

姚尔果

导师姓名职称：南振岐 教授

专业名称：计算机应用与技术

研究方向：企业信息化

论文答辩日期：2012 年 5 月学位授予日期：2012 年 6 月

答辩委员会主席：

评阅人：

二零一二年 月 日

硕士学位论文
M.D. Thesis

COCOMO II 模型的研究与应用
The studying and application of COCOMO II model

姚尔果
Yao Er-guo

独创性声明

本人声明所呈交的论文是我个人在导师指导下进行的研究工作及取得的研究成果。尽我所知，除了文中特别加以标注和致谢的地方外，论文中不包括其他人已经发表或撰写过的研究成果，也不包含为获得西北师范大学或其他教育机构的学位或证书而使用过的材料。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示了谢意。

学位论文作者签名：_____ 日期：_____

关于论文使用授权的说明

本人完全了解西北师范大学有关保留、使用学位论文的规定，即：学校有权保留送交论文的复印件，允许论文被查阅和借阅；学校可以公布论文的全部或部分内容，可以采用影印、缩印或其他复制手段保存论文。

(保密的学位论文在解密后应遵守此协议)

签名：　　导师签名：　　日期：

摘 要

软件是几乎所有的基于计算机的系统中最昂贵的元素，因而，一个复杂的软件项目，可能会因工作量估算的误差而出现软件项目延期、质量不能得到保证、甚至项目失败等严重后果，这对开发商的打击是灾难性的。项目规模作为软件工作量估算基本输入，虽在一些模型中提出了构建软件规模与工作量之间的关系，但因输入的信息中存在着不确定性，软件估算仍然存在很多问题，因此，准确的软件估算成为软件行业的一大挑战。本文首先对软件估算做了综合与全面的阐述，通过对软件估算误差来源分析，研究了项目规模、项目开发类型及技术人员等影响软件估算的因素。其次，本文介绍了业界广泛使用的 COCOMO II 模型，对该模型中的估算因子和函数分别做了详细的分析与研究。基于此，本文提出了改进的软件估算模型，该模型利用粒子群算法对估算因子进行调整，较之于传统 COCOMO II 模型，后者使估算数据更准确。最后，通过项目实例的应用，客观验证了改进的软件估算模型的可行性，同时直观的展示了改进估算模型的真实使用情况。

本文研究的意义在于通过对 COCOMO II 模型的分析、研究和改进，对 COCOMO II 模型有一个更深刻的认识，接着用一个项目实例验证了改进 COCOMO II 模型的可行性，对项目实践具有指导意义，在国内的软件开发成本估算方面成为一个可参考的案例。

关键词：软件估算; COCOMO II 模型; 加权粒子群算法; 分析与应用

Abstract

Software is the most expensive element of almost all the computer-based systems. As a result, due to a large estimation error of workloads, a complicated software project can make the difference between profit and loss. Cost Overruns would result in disastrous consequences for the developers. The basic input of effort estimation is the program scales. A number of models establish a relationship between software scales and Effort, however, owing to the uncertainty of the entered information, effort estimation still have many problems. Thus, accurate software effort estimation has already been one of the greatest challenges in computer industry. Firstly, this paper made a comprehensive explanation on the software estimation. By analysing the source of software estimation error, it studied some factors that influence software estimation from the project scope, types of project development and technical personnel and other factors. Then, this paper introduces the widely used COCOMO II model. The model made a detailed analysis and research on estimating factor and function respectively. Based on this, this paper puts forward the improved software estimation model, and the model using particle swarm optimization algorithm adjusted estimating factor. Compared with traditional COCOMO II model, the latter is more accurate. Finally, the project application objectively approved the improved software estimation model is feasible, and at the same time it vividly displayed the usage of the improved estimation model.

This thesis aimed to get a better understanding of COCOMO II model by analyzing, studying and improving it. Meanwhile, it verified the feasibility of improving COCOMO II model with a project instance, which had instructive suggestions for project practice and could be a reference case in the aspect of software cost estimation in China.

Key words: Software Effort Estimation; COCOMO II Model; Particle Swarm Optimization With Inertia Weight; Analysis And Application

目 录

第一章　绪论

1.1 课题研究的背景与意义

软件是几乎所有的基于计算机的系统中最昂贵的元素[1]，因而，一个复杂的软件项目会因一个大的工作量估算误差而出现获得利润或可能亏损的估算结果差别。成本（工作量）超出预计[2]，会出现软件项目进度延期、预算超支和质量缺陷等问题，可以给开发商带来灾难性的后果。项目规模作为软件工作量估算基本输入[3]，虽在一些模型中提出了构建软件规模与工作量之间的关系，但因输入的信息中存在着不确定性，工作量估算仍然存在很多问题，因此，准确的软件工作量估算成为行业中的一大挑战。

根据 Standish 组织 2004 年公布的 CHAOS 报告中显示[2][4]，在来自 350 多个组织的 50000 多个项目中，只有 29%是成功的(succeeded)，即项目在预算和限期内完成；18%是失败的(failed)，即项目没有完成或者取消；其余 53%被称为被质疑的(challenged)，虽然完成，但平均预算超支 89%。相比 1995 年的数据：16.2%，31.1%，52.7%，虽然成功项目所占的比例有所提升，但被质疑项目比例仍占 53%[5](如图 1 所示)。有些研究者认为，报告中平均预算超支 89%有些夸张，实际情况应该在 30%～40%之间[6]。但有一点是肯定的：人们对软件成本估算不足和需求不稳定，是软件项目失控主要的两个原因[4]。

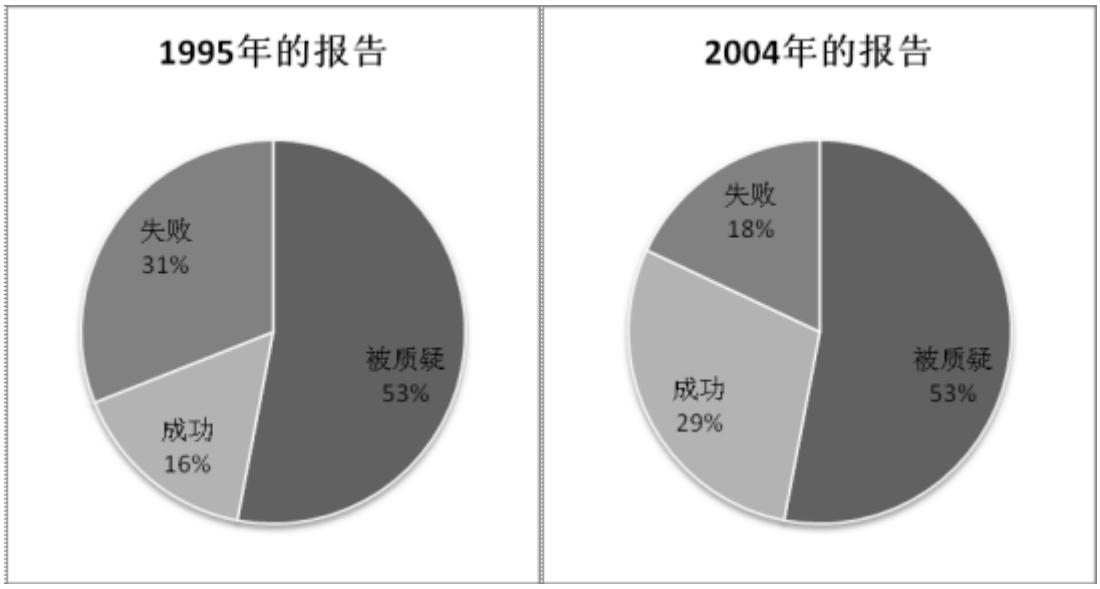

图 1.1　Standish 咨询公司对软件项目完成情况的统计数据[2] [4] [5]

“理解并控制软件成本带给我们的不仅仅是更多的软件，而会是更好的软件”[7]，

成功的软件估算是降低软件项目预算超支等问题的主要方法之一，可以帮助管理者做出如合理的投资、外包和竞标等商业决策，也是项目周期中预算和进度方面里程碑的重要参考对象。

估算，是确定项目开发时间和开发成本的过程[2]。由于当前人员成本通常是整个软件项目主要，“软件成本估算”与“软件工作量估算”在很多情况下是等同的。

软件估算研究最早可以追溯到 20 世纪 60 年代的 SDC(system development corporation)线性模型[6]，已历经了 50 年左右的发展。软件估算方法有许多分类形式，其根据是否采用算法模型分为 3 大类(如图 1.2 所示)：

⑴ 基于算法模型的方法；

⑵ 非基于算法模型的方法；

⑶ 组合方法。

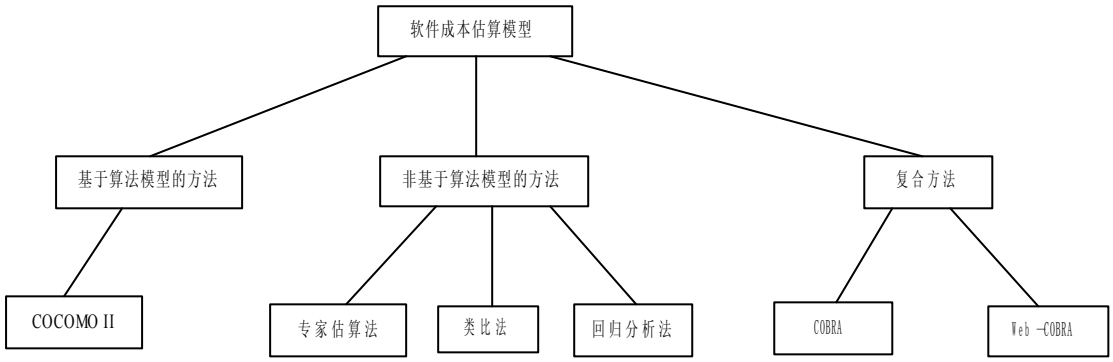

图 1.2 软件成本估算方法的分类

1981 年，由 Barry Boehm 在其经典著作“软件工程经济学”(software engineering economics)[8]中，介绍了一种软件估算模型的层次体系，称为 COCOMO(构造性成本模型，Constructive Cost Model)，在这些众多的软件估算方法中，COCOMO 模型方法脱颖而出，成为世界上应用范围最广泛的软件估算模型。Barry Boehm 在原有版本上做了大量更新[9]，以改进它对新的开发过程、方法、工具、技术和企业文化的适用性，并提高了估算的准确性，于 1995 年提出了 COCOMO II[10]，详细内容会在本文的第三章介绍。

1.2 软件估算现状

1.2.1 国内现状

目前国内软件开发管理正在向规范化发展，可是在软件估算上绝大部分还处于比较落后的状态：其一是管理层意识上没有认识到软件估算的重要性，把估算等同

于估计，而且很多还受限于商业行为；其二是缺少专门的辅助估算工具来，或没有专门对它进行研究。对于软件项目的估算基本上是依靠经验来判断，由于没有量化标准，不同经验和不同领域的人估算出的结果相差很大，而且他们之间很难沟通，因为每个人都有理由坚持自己的判断，使得最终结果往往是以某个“专家”的估算为准。因此，国内的软件估算需要的是定量估算，这样做不仅规范精确，减少主观因素，而且有助于软件行业的发展及与国际接轨。

1.2.2 国外状况

在软件估算方面国外发达国家要比国内要成熟的多，他们不仅有诸如代码行估算法、功能点估算法、人力估算法等先进方法，还有许多专业化的估算工具。微软的项目管理工具 Project，加拿大 Software Productivity Center Inc.公司的 Estimate，这些都是相对成熟的软件估算工具。其中 Project 采用了自下而上的估算方法，而 Estimate 属于专业化工具，其包含许多常用的估算方法和校正方法，并且使用了 COCOMO Ⅱ、Putnam Methodology 和 Monte Carlo Simulation 等几种成熟算法，估算结果有项目制作周期、人力成本，还包括十多种分析报告及预计缺陷图、模拟发散图、人力图、计划编制选项图和缺陷方差图等，从不同角度帮助管理人员进行分析与决策。

1.3 本文研究的内容

COCOMO II 是一个用于规划和执行软件项目的目标成本模型。它是管理软件项目或软件生产过程的重要组成部分。COCOMO II 的包括两个基本的信息模型。第一个是用于描述软件项目的框架，包括过程模型，文化，干系人，方法，工具，开发团队和软件产品的规模或复杂度。二是经验库，对 COCOMO 做了重要更新，以改进它对现代过程、方法、工具和技术的适用性。

本文做了以下工作：

1、总结和归纳了以下几点：软件估算与软件项目开发过程中各方面的关系:软件估算的价值和意义；估算误差的来源，估算过程中要注意的问题；影响估算的主要因素。

2、介绍和详细分析研究了 COCOMO II 估算模型。介绍了模型在软件开发各个周期的使用；软件估算公式的表达方式和意义；每个估算因子和工作量乘数所代表的意义和等级的取值；在实际应用中应该对每个因子怎样评定等级和取舍。

3、对 COCOMO II 模型做了一点改进。对工作量计算公式，引入偏置因子，并

运用加权粒子群算法对因子进行调整，使其对中小企业的中小规模软件项目的估算更有适应性。

4、鉴于 COCOMO II 估算模型在软件开发初期使用的局限性，通过结合 WBS 方法和未调整功能点算法，使软件估算在项目开发初期更有参考意义。

5、结合我参与和一个软件项目，跟踪整个开发过程，通过介绍改进 COCOMO II 模型在软件开发过程中的详细使用，估算过程中根据实际情况，对每一个出现的问题进行分析总结，对模型的每一个因子进行调整和取舍等的展示，使读者对改进 COCOMO II 模型有一个更直观的认识，也是一个软件估算的参考案例。最后通过数据对比，证明改进模型具有更好的适用性。另外，我在此项目中担任需求分析师和配置管理员。

第二章　软件估算

2.1 研究软件估算的意义

2.1.1 估算、目标和承诺

估算：是对项目将持续多长时间，花费多少成本的预测；

目标：描述期望达到的业务的目的；

承诺：许诺载特定的日期之前以特定的质量交付规定的功能，可以比估算激进，也可能比估算保守[11]。

2.1.2 估算、计划的关系

估算是客观的分析过程，计划是主观的目标求解过程[11]。当有人要你提供估算值时，要确定他是期望你进行估算还是期望你给出如何达到这个目标的计划。估算的目的是得到准确的结果，不是寻求特定的结果，而计划的目的是寻求特定的结果；如果估算结果与目标之间存在显著的区别，进行项目计划时就要认识到两者之间的差距，并考虑可能存在的较高的风险；如果估算结果接近于目标，在进行计划时就可以认为风险较低[11]。

工作计划中有部分问题在制定中一定程度上依赖于准确的估算：建立详细的进度表确定项目的关键路径；建立完整的工作分解结构；确定要交付的功能的优先级；将项目按多次迭代进行分解；准确的估算可以为更好完成这些工作提供支持。

2.1.3 估算、控制的关系

海森堡不确定性：仅仅对事物的观察就会改变事物的本身，因此只有看到事物的表现时才会知道它的表现是什么样的，而不可预知它的表现。

项目中会受到各种各样的因素的影响导致对最初进行项目估算时的假设失效，但如果最终提供了与估算一致的结果,那么就说明这个估算是个良好的估算，所以评价一个良好的估算不能基于它的预测能力，而是要基于估算为项目成功提供支持的能力。仅仅通过估算本身并不能获得准确的估算结果，还需要通过有效的项目控制来提供支持。

2.1.4 准确估算的价值

1、高估与低估

完全准确的估算值可以为项目计划打下很好的基础。如果估算值是准确的，就

可以有效协调不同开发人员之间的工作。但实际上，我们知道准确的估算是很少见的，如果我们在项目估算中产生偏差的话，是高估的结果好还是低估的结果好呢？

⑴ 高估的影响

① 帕金森法则的作用-如果给一个团队 6 个月时间去完成一个本来可以在 4 个月内完成的项目，那么团队会在多出的这 2 个月时间内找其他事情做，使项目在 6 个月时间做完。

②“学生综合症”，指如果给开发人员过多的时间，他们就会一直拖延到项目快结束时才匆忙赶工，结果反而很可能使项目无法按时完成。

⑵ 低估的影响

① 降低项目计划的有效性。

过低的估算会在特定活动的计划中引入不准确的假设，造成无法进行有效的计划。如果计划中采用的假设达到过高的错误程度时，根据这些假设做出的项目计划实际上也就失去了作用。

② 从统计角度降低了按时交付的可能性。

③ 未做好前期基础工作导致项目实际结果比名义上取得的结果更差。

低估会导致开发中上游活动，像需求和设计等方面花费的时间太少。如果对需求和设计缺乏足够的重视，在项目后期的工作中就要对需求和设计进行返工，这样做比一开始就很好地完成这些活动需要更高的成本和时间。

④ 项目后期破坏性变因导致项目实际结果比名义上可以取得的结果更差。

一旦项目进入“延误”状态，项目团队就要进行大量额外的活动；而如果项目按计划正常进行，就不需要大量的额外活动了。

2、准确估算的价值

⑴ 更准确地显示项目状态

如果计划实际可行，就能够根据计划来跟踪进度。良好的估算可以为项目跟踪提供重要的支持。

⑵ 更高的质量

所有软件错误中大约有 40%的错误是因为进度压力造成的，而这些错误是可以通过适当的安排进度，降低给开发人员的压力来避免的。准确的估算有助于避免产生和进度压力相关的质量问题。

⑶ 更好地与软件以外的活动进行协调

软件项目通常要与其他业务活动进行协调，如果软件开发进度计划不现实，就

会造成相关的功能不能实现，导致整个项目进度的延误。软件估算越准确，整个项目的协调也就越紧密，这些协调包括软件方面的活动和软件以外的活动。

⑷ 更好地编制预算

准确的预算需要准确的估算提供支持。

⑸ 提高开发团队的可信度

能提供准确估算的开发团队，在管理人员，市场人员销售人员中有一个高的可信度。

⑹ 更早获得风险信息

准确的估算，可以更早的发现项目目标和项目估算的不一致，从而更早进行相应措施。

2.2 软件估算的定义

2.2.1 软件估算的定义

软件估算是根据软件的开发内容、开发工具、开发人员等因素对需求分析、软件设计、编码、测试与整个开发过程所花费的时间、费用及工作量的预测[3][31]。

2.2.2 软件估算的真正目的

软件项目面临着相似的两难局面。项目计划者常常发现在项目的业务目标与进度和成本的估算值之间存在差距。如果差距不大，计划者也许可以通过精打细算或“挤压”项目进度表、预算或特性集来控制项目成功结束。如果差距很大，就要重新考虑项目的目标。

软件估算的首要目标并不是预测项目的结果，而是确定项目目标是否够现实，从而让项目在可控的状态下达成这些目标。软件估算无需非常准确而是要有用。

2.3 估算误差的来源

2.3.1 有关被估算项目的不准确信息

实施该项目的开发组织能力相关的不准确信息；项目过于混乱而无法为准确估算提供有效支持（也就是说，试图对有变动的目标进行估算）；估算本身带来的误差；估算不确定性的来源。

软件开发是一个逐步求精练的过程。从一个大致的产品概念（待构建软件的愿景）开始，然后根据产品和项目目标来细化该概念。有时候估算的目标是交付特定

量的功能所需的成本和时间。其他一些时候的目标是估算在预定的时间和确定的预算限制下能够实现多少功能。在任何一种情况下，实现软件的不同途径都会让最终的成本、进度和功能出现巨大的差异[1]。

2.3.2 混乱的开发过程

⑴ 一开始就未能很好的研究需求；

⑵ 在需求阶段缺乏最终用户的参与；

⑶ 导致代码中出现无数错误的不良设计；

⑷ 引发大量排错工作的不良编码实践；

⑸ 缺乏有经验的人员；

⑹ 不完整或不熟练的项目计划活动。

这些混乱源有两个共同之处[11]：首先是每个混乱源都会产生可变性，导致难以进行准确的估算；其次就是解决这些问题的最佳方法不是通过估算活动，而是通过更好的项目控制。

2.3.3 不稳定性需求

许多报告指出，需求变更是产生估算问题的常见来源[15]，不稳定的需求会带来各种挑战，包括两个方面：一是不稳定的需求代表了特定方式的项目混乱；二是由于没有常常对需求变更进行跟踪，项目往往没有在需要的时候被重估。

更好的估算本身并不能解决由于需求不稳定所造成的问题，如果所处的环境不能让需求稳定下来，就要考虑采用那些针对高可变性环境的其他开发方法，例如短迭代、极限编程等。

2.3.4 遗漏的活动

最常见的估算误差来源之一是在项目估算时忘记包括对某些必要的任务进行估算[5]，有研究指出开发人员很容易对他们记得要估算的工作进行准确的估算，但同样容易忽略 20%～30%必要的工作任务从而导致 20%～30%的误差。

2.3.5 主观性和偏差

主观性估算本身是一种客观的分析过程，不同于计划，计划是一种主观的目标求解过程。主观性的增加势必会影响估算结果的准确度[14]。

1、无根据的精度

在随意的谈话中，人们往往把“准确度”和“精度”当作同义词[11]，但对估算而言，两个术语之间的区别是至关重要的：准确度是指数值离真实值有多远；精度仅仅是指一个数值有多精确。

2、太过乐观

很多软件开发人员认为“一切都会按照预定的计划实现的”[14][31]，这种过于乐观的想法使得项目负责人会对估算的结果进行无根据的增加或者删减。

2.4 影响估算的因素

2.4.1 项目规模

项目规模是影响工作量、成本和进度的最具有决定性的因素。

常见违背这一事实作法：在不知道软件会有多大规模的情况下，估算工作量、成本和进度；在有意识地增加软件规模时，没有相应地调整对工作量、成本和进度的估算。

1、用代码行来表示软件规模

用代码行表示规模是常见方式，新接触估算时，有时会怀疑它是否确实为有效的方式，其原因有二：现代许多编程环境不是面向代码行；许多开发活动（如：需求、分析、测试等等）不能用代码行表示。但实际实用中，代码行表示法有很多优点：比较容易保用相关工具来收集在过去项目中的代码行数据；大多数开发组织有大量的以代码行形式存在的历史数据；在大多数编程语言中，代码行编写所需工作量，大致相近；大部商家用估算工具都使用代码行作为估算的基础，代码行具有通用性。

2、规模不经济

在软件行业之外，如果建立一个更大的制造厂，就能降低制造单位产品所需的成本——规模经济[16]；在软件行业之内，随着规模的增加，人与人之间交流沟通路径的数目会按参与人员数 n*(n-1)/2 增加；沟通路径的增加，就带来了工作量／成本会呈现指数增长——规模不经济；随着项目规模的扩大，项目开发条件会让生产率降低得比一般规模情况下更快。实际上，工作量是以指数方式增长的。

3、软件估算中规模不经济的重要性

项目原始规模才是影响估算值的最大因素，规模不经济对估算值的影响只是次要的，要将主要精力放在获得良好规模估算上。

2.4.2 待开发软件的不同类型

这是排在“规模”之后，对估算值影响第二大的因素，不同类型之间存在 10～20 倍的差距，表 2.1 显示了不同类型的项目中每个人月能够完成的代码行数。

表 2.1 常见类型项目的生产率[11]

代码行/人月 最低值～最高值（典型值）			
软件类型	10K 代码行的项目	100K 代码行的项目	250K 代码行的项目
应用系统	800~18000（3000）	200~7000（600）	100~5000（500）
嵌入式系统	100~2000（300）	30~500（70）	20~200（60）
公众因特网系统	600~10000（1500）	100~2000（300）	100~1500（200）
内部因特网系统	1500~18000（4000）	300~7000（800）	200~5000（600）
实时系统	100~1500（200）	20~300（50）	20~300（40）

2.4.3 人员因素

人员因素，也会对其项目结果产生显著的影响。COCOMO II 中[18]，在具有 10000 代码行的项目里，所有人员因素的组合效果可以让一个项目的估算结果出现高达 22 倍的变化。

个体间的差距意味着，如果没有对参与人员有一定的了解，就无法准确估算项目。

2.4.4 编程语言

它将由以下四个方面影响估算[17]：团队对将要使用编程语言的经验会对总体生产率产生 40%影响；某些编程语言的代码行效率高于其它；将使用编程语言的相关支持工具和开发环境是否丰富，最差工具会增加约 50%工作量；使用解释型语言比编译型的生产率更高。

2.4.5 其他因素

COCOMO II 模型的调节因子，详见第三章。

第三章　COCOMO Ⅱ模型的研究与改进

3.1 COCOMO II 模型简介

Barry Boehm 是第一个从经济学角度看待软件工程学的研究者，他提出的COCOMO81 模型和改进后的 COCOMO II 在第一章中有过介绍。

COCOMO II 模型由 3 个子模型组成[27]：

应用组装模型(Applications Composition Model)，用于项目规划阶段，用来估算采用集成计算机辅助软件工程工具的快速应用开发软件项目的工作量和进度；

早期设计模型(Early Design Model)用于信息还不足以支持详细的细粒度估算阶段，其基于功能点或可用代码行及 5 个规模指数因子、7 个工作量乘数因子，选择体系结构的方案或增量开发测量；

后体系结构模型(Post-Architecture Model)，用于完成顶层设计与获取项目详细信息阶段，软件体系结构定义和建立之后，其基于源代码行和功能点以及 5 个规模指数因子及 17 个工作量乘数因子。

3.2 软件规模估算方法

3.2.1 源代码行(SLOC)计算

估算新代码行，需要原始数据，其中最好的是历史数据。项目早期会有功能点，组件等可以利用的数据，能转换成代码行数。如果缺乏历史数据，可以用专家意见推导出可能的，最不可能的和最有可能的规模。

代码规模表示为源代码千行数（KSLOC），源代码行一般不包括未交付的支持软件，如测试驱动程序。可是，如果这些开发需要有自己的评审，测试计划、文档等，那么应该把它们计算在内。

3.2.2 未调整功能点(UFP)计算

功能点估算法是比较有用的方法，其基于软件项目中功能的数量和一组独立的项目因子[25][26][27]，这些信息在软件项目的生命周期早期就可以得到。

功能点的度量方式是先按需求描述对功能点计数[28]，得到未调整功能点，再按系统涉及到的技术功能点对该数值进行修正。

COCOMO II 用于确定未调整功能点过程定义如下[18]，可用于早期设计与后体系结构模型中。

按定义的类型来确定功能数。未调整功能数由技术人员依据软件需求与设计文档的信息，根据五种用户功能类型分别来统计（见表 3.1）。

表 3.1 五种用户功能类型

功能点	描述
外部输入（EI）	主要指数据输入界面
外部输出（EO）	主要指报表
内部逻辑文件（ILF）	数据库的数据表数目
外部逻辑文件（EIF）	输入和输出之间的接口文件
外部查询（EQ）	应用系统与外部之间的通信或外部消息

根据表 3.2 确定复杂性等级。

表 3.2 复杂性等级

对于内部逻辑文件和外部逻辑接口文件			
记录单元	数据单元		
	1-19	20-50	51+
1	低	平均	平均
2-5	低	平均	高
6+	平均	高	高
对于外部输出和外部查询			
文件类型	数据单元		
	1-5	6-15	16+
0 或 1	低	低	平均
2-3	低	平均	高
4+	平均	高	高
对于外部输入			
文件类型	数据单元		
	1-4	5-15	16+
0 或 1	低	低	平均
2-3	低	平均	高
4+	平均	高	高

根据表 3.3 的数据，应用复杂性权重，对每个复杂等级用相应的功能类型数加权。

表 3.3 等级的权重

		复杂等级		
		简单	一般	复杂
特征系数	EI	3	4	6
	EO	4	5	7
	EQ	3	4	6
	EIF	7	10	15
	LIF	5	7	10

（4）对所有的带权重功能计数相加，得到未调整功能点。

3.2.3 累加新的、改编的和复用的代码

从其他渠道得来的并用于被开发产品的代码也影响产品的有效规模。包括复用代码和改编代码。

复用代码，事先存在的，不经过修改直接使用的代码。

改编代码，事先存在的，经过修改后再使用的代码。

把复用代码和改编代码的有效规模调整为等价的新代码行（基于将代码包含到产品中而修改代码所花费的工作量），调整所得的代码行称为等价源代码行（ESLIC）。

1、非线性复用影响

把复用代码改编量与最终复用成本相联系的复用成本函数在两个方面是非线性的：理解要改编软件的成本和检查模块接口的相对成本。复用代码所需的工作量不是从零开始，一般大约有 5%的成本用于评价、选型和消化可复用的组件。在复用产品中做较小的改动会产生不成比例的较大的成本。

2、复用模型

COCOMO II 采用非线性估算模型处理软件复用：

$$\text{等价 } KSLOC = \text{改编 } KSLOC \times \left(1 - \frac{AT}{100}\right) \times AAM$$

$$AAF = (0.4 \times DM) + (0.3 \times CM) + (0.3 \times IM)$$

$$AAM = \begin{cases} \dfrac{[AA + AAF(1 + (0.02 \times SU \times UNFM))]}{100} & \text{当} AAF \leq 50 \text{ 时} \\ \dfrac{[AA + AAF(\times SU \times UNFM)]}{100} & \text{当} AAF > 50 \text{ 时} \end{cases}$$

它涉及到估算软件要改编的数量和三个修改程度系数：设计修改（DM）百分比，代码修改（CM）百分比和集成改编可复用软件所需的集成工作量（IM）百分比。

SU，软件理解增量；AA，用于评估与消化增量等级；UNFM，程序员不熟悉性等级

3、改编原则

对于不同的代码，COCOMO II 有不同的改编软件系数。

新代码，指从零开始开发的代码。

复用代码，指对现有代码不做任何改动。

改编代码，是对现有代码做某些改动。

COST 成品，不做改动，可能需要一些连接代码与之建立联系。

4、需求演进和易变性（REVL）

用于调整由需求演进和易变性而引起的产品有效规模，而需求演进和易变性是由任务或用户接口变化、技术更新或 COST 易变性而引起的。REVL 是因需求演进而遗弃的百分比。

使用 REVL 计算规模如下：

$$Size=\left(1+\frac{REVL}{100}\right)\times Size_D$$

$Size_D$ 是所交付的复用等价量。

3.2.4 自动转换的代码

对于软件的重构和移植，由于再构造时自动工具的效率，导致极高的代码修改的百分比和相应很少的工作量：

$$PM_{Auto}=\frac{\text{改编}SLIC\times(AT/100)}{ATPROD}$$

AT：重构时自动处理代码的转换比。

PM_{Auto}：自动转换影响估算工作量。

ATPROD：自动转换生产率，缺省值是 2400 源语句。

3.2.5 软件维护的规模

当已知基本代码规模和基本代码修改百分比时，维护规模通常由下面公式给出[8]。

$$(\text{Size})_M=\left[(\text{基本代码规模})\times\text{MCF}\right]\times MAF$$

MCF：维护改变系数，是基本代码修改的百分比。

$$\text{MCF}=\frac{\text{增加规模+修改规模}}{\text{基本代码规模}}$$

当知道在维护期内只对现有基本代码做少量增补或修改时，可以使用较简单的版本，删除的代码不统计在内。

$$(\text{Size})_M=(\text{增加规模+修改规模})\times MAF$$

MAF 用来调整有效维护规模，说明软件理解和不熟悉性带来的影响。COCOMO II 用复用模型的软件理解（SU）因子与程序员不熟悉性（UNFM）因子，反映软件结构和可理解性是否优良对维护工作量的影响。

$$MAF=1+\left(\frac{SU}{100}\times UNFM\right)$$

3.3 工作量估算方法

在 COCOMO II 中，工作量用人月（PM）表示[18]，一个人月定义为一个人在一个月之内从事软件项目开发的时间数。COCOMO II 把每人月的人时数（PH/PM）作为可调整的系数，基标称值为 152/PM。该模型可用于早期设计和后体系结构成本模型。

工作量估算公式为：

$$PM = A \times Size^{E} \times \prod_{i=1}^{n} EM$$

PM：工作量；

$Size$：软件开发的规模；

EM：工作量乘数，工作量乘数 EM_i 的个数 n 的值对于后体系结构模型是 17，对于早期的设计模型是 7；

E：指数因子，当 E=1.0 时，A 接近于生产率常数，随着 E 增加，生产率会改变，它对规模大小有非线性的影响。

3.3.1COCOMO II 模型中的比例因子

指数 E 体现了五个比例因子（SF）的作用，说明了不同规模的软件项目具有的相对规模经济和不经济性。如果 E<1.0，项目表现出规模经济。即项目的规模翻倍，工作量不至于翻倍；如果 E=1.0 规模经济和不经济性是平衡的。通常小项目的成本估算运用这种线性模型；如果 E>1.0，项目就表现出规模不经济性。

$$E = B + 0.01 \times \sum^{5} SF_j$$

B=0.91，是可以校准的常数。 SF_j 代表指数比例因子。

如果一个项目所有的比例因子都为“极高”等级或者都为“很低”等级。两者的结果有着很大的差别，尤其对于大型项目，一个因子的变化，对规模产生很大的变化，比例因子的值见表 3.4。

表 3.4 COMOII 模型的比例因子值

比例因子 SF_j	很低	低	标称	高	很高	极高
PREC	全新的	绝大部分的	有一些新的	基本熟悉	绝大部分熟悉	完全熟悉
	6.20	4.96	3.72	2.48	1.24	0.00
FLEX	严格	偶尔放宽	放宽	基本一致	部分一致	通用目标
	5.07	4.05	3.04	2.03	1.01	0.00
RESL	很少	一些	常常	通常	绝大多数	完全
	7.07	5.65	4.24	2.83	1.41	0.00
TEAM	交流非常困难	交流有一些障碍	基本的交流协作	广泛的协作	高度协作	无缝合作
	5.48	4.38	3.29	2.19	1.10	0.00
PMAT	1 级的较低部分	1 级的较高部分	2 级	3 级	4 级	5 级
	7.80	6.24	4.68	3.12	1.56	0.00

早期设计模型，A，B，C，D，SF_1…，和 SF_5，与后体系结构模型的值相同。

1、先例性（PREC）

先例性指所开发项目与以前开发项目的相似程度（表 3.5）。

表 3.5 先例性等级

特征	很低	标称 / 高	超高
产品目标有组织的理解	一般	相当多	完全
相关软件系统工作中的经验	中等	相当多	广泛
与相关新硬件和操作程序 的协同开发	广泛	中等	一般
对创新的数据处理体系结构、算法的需求	相当多	一些	极少

2、开发灵活性（FLEX）

开发灵活性与先例性基本上是一个项目固有的、不可控制的属性（表 3.6）。

表 3.6 开发灵活性等级

特征	很低	标称/高	极高
软件的性能与已建立的需求一致	完全	相当	基本
软件性能与外部接口规范一致	完全	相当	基本
以上均一致	高	中等	低

3、体系结构/风险化解(RESL)

体系结构/风险化解(RESL)是通过两个比例因子合并：产品设计评审（PDR）得到的设计彻底性和通过 PDR 得到的风险消除。同时也将其等级与生命周期中的里程碑联系起来。所得等级 RESL 是所列特征的主观加权平均值。

4、团队凝聚力（TEAM）

用于说明项目混乱和熵的原因。混乱和熵是因为项目干系人的难以协调。项目干系人主要包括：用户、客户、开发人员、维护人员、协调人员等。此外，干系人之间目标和文化的差异、目标调和中的困难、干系人缺乏作为一个团队运作的经验和熟悉性也会造成混乱和熵。

5、过程成熟度（PMAT）

确定 PMAT 的过程是围绕软件工程研究所（SEI）的能力成熟度模型而组织的，确定过程成熟度的时间正是项目开始的时间。其评定是基于 CMMI 的组织评估结果，见表 3.7。

表 3.7 过程成熟度等级

等级	CMMI 等级	级别
很低	无	0
低	CMMI 1	1
标称	CMMI 2	2
高	CMMI 3	3
很高	CMMI 4	4
极高	CMMI 5	5

3.3.2COCOMO II 工作量乘数

为了反映正在开发软件项目的特征，COCOMO II 模型采用 17 个后体系结构工作量乘数（EM），以调整标称的工作量 PM。由一组等级和一组与之相应的工作量乘数来定义每个成本驱动因子。标称等级为 1.00，它不改变所估算的工作量，非标称等级一般都改变工作量估算（表 3.8）。

不管是整个项目还是由多个模块组成的项目，COCOMO II 模型可用于估算项目的工作量和进度。对于一个项目，除了要求的开发进度（SCED）成本驱动因子和比例因子之外，每个模块和其他成本驱动因子的等级都可以不同。

表 3.8 工作量乘数

成本因素	描述	等级				
		极低	低	标称	高	极高
产品因子						
RELY	所需的软件可靠性	0.75	0.88	1.00	1.15	1.40
DATA	数据库规模		0.94	1.00	1.08	1.16
CPLX	产品复杂度	0.70	0.85	1.00	1.15	1.30
平台因子						
TIME	执行时间约束条件	-	-	1.00	1.11	1.30
STOR	主存储约束条件	-	-	1.00	1.06	1.21
VIRT	虚拟机的波动性	-	0.87	1.00	1.15	1.30
RURE	电脑周转时间	-	1.87	1.00	1.07	1.15
人员因子						
ACAP	分析师能力	1.46	1.19	1.00	0.86	0.71
AEXP	应用经验	1.29	1.13	1.00	0.91	0.82
PCAP	程序员能力	1.42	1.17	1.00	0.86	0.70
VEXP	虚拟机的波动性	1.21	1.1	1.00	0.90	-
LEXP	程序语言的熟练程度	1.14	1.07	1.00	0.95	-
项目因子						
MODP	现代编程实践	1.24	1.10	1.00	0.91	0.82
TOOL	软件工具	1.23	1.08	1.00	1.04	1.10
SCED	进度表	1.23	1.08	1.00	1.04	1.10

1、产品因子，用于说明以由于所开发项目产品的特性导致的工作量的变化。

⑴ 软件的可靠性（RELY），是对软件预期功能的度量，其分组方案见表 3.9。

表 3.9 软件的可靠性等级

RELY 描述符	有点不方便	低、易弥补的损失	中等、易弥补的损失	高财政损失	对人类生命的风险	
等级	很低	低	标准	高	很高	超高
工作量乘数	0.82	0.92	1.00	1.10	1.26	n/a

⑵ 数据库规模（DATA），测试和维护数据对工作量的影响（表 3.10）。

表 3.10 数据库规模等级

DATA 描述符		测试 DB bytes /pgmSLOC<10	10≤D/P<10 0	100≤D/P<1 000	D/P≥1000	
等级	很低	低	标准	高	很高	超高
工作量乘数	a/n	0.90	1.00	1.14	1.28	n/a

⑶ 产品复杂性（CPLX），用来描述组件和产品的复杂等级（表 3.11）。

表 3.11 产品复杂性等级

	控制操作	计算操作	设备相关操作	数据管理操作	用户接口管理操作
很低	很低线性代码，并存在少许嵌套编程 符: Do、CASE、IF-THEN—ELSE。由过程调用形成简单的模块组成并且书写简练。	评价表达式简单：A=B+C×(D-E)	易于阅读，简单的书写语句格式	主存中简单的数据安放，简单的COTS—DB 查询、更新	简单的输入格式，报表生成器
低	结构化编程符的简单嵌套。判定 语句绝大多数简	评价表达式中等难度：D=SQRT(B2 4×A×C)	无须认识特定的处理器或 I／O 设备特征。I／0 处于 GET／PUT 级。	不存在数据结构变化，没有修订，无须中间文件的单文件子集。中等复杂的COTS—DB查询、更新。	使用简图形单用户化，没有修界面生成器。
标准	大多数简单的嵌套。一些模块间的控制，决策表，简单的回调和消息传递，包括支持中间件的分布式处理。	使用标准的数学和统计程序，基本的矩形/向量运算	I/O 处理包括设备选型、状态校核和错误处理。	多文件输入和单文件输出，简单的结构变更，简单的校正，复杂的COTS—DB 查询、更新。	使用简单的工具集。
高	存在许多复合谓词的高嵌套结构编程符；排序和堆栈控制；均匀分布的处理；单处理机软件实时控制。	基本的数值分析，多变量插值、常微分方程，基本的截断舍位	对物理 I/O 层的操作，优化 I/O 覆盖	由数据流内容引起的单独触发激励、复杂的数据结构再调整。	工具集的开发和扩展，简单音频 I/O，多媒体。
很高	可重入和递归代码，优先级固定的中断处理，任务回调、多相分布处理，单处理器实时控制	困难但结构化的数值分析，降秩矩阵方程，偏微分方程，简单并行分析	中断诊断、服务、屏蔽程序，通信链路处理，性能增强嵌入式系统	分布式数据库协调，复杂激励，查询优化	复杂性适度的 2D/3D，运动图形，多媒体
超高	动态变更优先级的多资源调度，微指令集控制，分布式硬实时控制	困难且未结构化的数据值分布，对有噪音的随即数据进行高精确度分析，复杂的并行分析	设计依赖时间的代码、微程序控制符，性能关键的嵌入式系统	高耦合，动态相关的对象结构，自然语言数据管理	复杂的多媒体、虚拟现实和自然语言界面

⑷ 可复用开发（RUSE），指为了使构造的组件在将来或者其他项目中使用，所付

出的额外的工作量（表 3.12）。

表 3.12 可复用开发等级

RUSE 描述符		无	贯穿于项目	贯穿于程序	贯穿于产品线	贯穿于多产品线
等级	很低	低	标准	高	很高	超高
工作量乘数	a/n	0.95	1.00	1.07	1.15	1.24

⑸ 匹配生命周期的文档编制（DOCU），减少文档似乎可以减少开发的工作量，但实际上则导致维护工作的大量增加（表 3.13）。

表 3.13 匹配生命周期的文档编制等级

DOCU 描述符	未包含大量生命周期需求	未包含一些生命周期需求	符合生命周期需求规模	超出生命周期需求	极大超出生命周期需求	
等级	很低	低	标准	高	很高	超高
工作量乘数	0.81	0.91	1.00	1.11	1.23	n/a

2、平台因子，由于硬件和软件环境的飞速发展，平台因子对项目的影响越来越小，所以在实际应用中，可以忽略，使用标称值。

⑴ 执行时间约束（TIME），对系统执行时间的强制约束（表 3.14）。

表 3.14 执行时间约束等级

TIME 描述符			使用不到 50%可用执行时间	使用 75%可用执行时间	使用 85%可用执行时间	使用 95%可用执行时间
等级	很低	低	标准	高	很高	超高
工作量乘数	a/n	a/n	1.00	1.11	1.29	1.63

⑵ 主存储约束（STOR），对存储空间的强制约束（表 3.15）。

表 3.15 主存储约束等级

STOR 描述符			使用不到 50%可用存储空间	使用 75%可用存储空间	使用 85%可用存储空间	使用 95%可用存储空间
等级	很低	低	标准	高	很高	超高
工作量乘数	a/n	a/n	1.00	1.05	1.17	1.46

⑶ 平台易变性（PVOL），包括硬件和软件(表 3.16)。

表 3.16 平台易变性等级

PVOL 描述符		每 12 个月主要的变化，每个月次要的变化	每 6 个月主要的变化，每 2 个月次要的变化	每 2 个月主要的变化，每周次要的变化	每 2 周主要的变化，每周要的变化	
等级	很低	低	标准	高	很高	超高
工作量乘数	a/n	0.87	1.00	1.15	1.30	无法获取

3、人员因子

⑴ 分析员能力(ACAP),指需求分析与概要设计、详细设计的人员的能力(表 3.17)。

表 3.17 分析员能力等级

ACAP 描述符	第 15 百分点	第 35 百分点	第 55 百分点	第 75 百分点	第 90 百分点	
等级	很低	低	标准	高	很高	超高
工作量乘数	1.42	1.19	1.00	0.85	1.71	n/a

⑵ 程序员能力（PCAP），开发人员的能力（表 3.18）。

表 3.18 程序员能力等级

PCAP 描述符	第 15 百分点	第 35 百分点	第 55 百分点	第 75 百分点	第 90 百分点	
等级	很低	低	标准	高	很高	超高
工作量乘数	1.34	1.15	1.00	0.88	0.76	n/a

⑶ 人员的连续性（PCON），指每年人员的周转率（表 3.19）。

表 3.19 人员的连续性等级

PCON 描述符	48%/年	24%/年	12%/年	6%/年	3%/年	
等级	很低	低	标准	高	很高	超高
工作量乘数	1.29	1.12	1.00	0.90	0.81	

⑷ 应用经验（APEX），对类似项目开发的经验（表 3.20）。

表 3.20 应用经验等级

APEX 描述符	≤2 个月	6 个月	1 年	3 年	6 年	
等级	很低	低	标准	高	很高	超高
工作量乘数	1.22	1.10	1.00	0.88	0.81	n/a

⑸ 平台经验（PLEX），对开发平台的使用经验（表 3.21）。

表 3.21 平台经验等级

PLEX 描述符	≤2 个月	6 个月	1 年	3 年	6 年	
等级	很低	低	标准	高	很高	超高
工作量乘数	1.19	1.09	1.00	0.91	0.85	n/a

⑹ 语言和工具经验（LTEX），对使用的编程语言和相应开发工具的使用经验（表 3.22）。

表 3.22 语言和工具经验等级

LTEX 描述符	≤2 个月	6 个月	1 年	3 年	6 年	
等级	很低	低	标准	高	很高	超高
工作量乘数	1.20	1.09	1.00	0.91	0.84	

4、项目因子

⑴ 软件工具的使用（TOOL），软件开发生命周期中相应的各种工具的使用（表 3.23）。

表 3.23 软件工具的使用等级

TOOL 描述符	编程，编码，调试	简单，前端，后端 CASE，很少集成	基本的生命周期工具，中等集成	健壮，成熟的生命周期工具，中等集成	健壮，成熟、主动的生命周期工具，过程、方法、复用的良好集成	
等级	很低	低	标准	高	很高	超高
工作量乘数	1.17	1.09	1.00	0.90	0.85	n/a

⑵ 要求的开发进度（SCED），对软件开发过程的进度要求（表 3.24）。

表 3.24 要求的开发进度等级

SCED 描述符	标称的 75%	标称的 85%	标称的 100%	标称的 130%	标称的 160%	
等级	很低	低	标准	高	很高	超高
工作量乘数	1.20	1.09	1.00	0.91	0.84	n/a

⑶ 多点开发（SITE），主要从地点分布和交流支持两方面来描述（表 3.25）。

表 3.25 多点开发等级

SITE：配置描述符	国际	多城市或多企业	多城市或多企业	同一城市或大区域	同一建筑或企业	完全同处一地
SITE：通信描述符	电话、邮件	专用电话	窄带电子邮箱	宽带电子通信	宽带电子通信、偶然视频会议	多媒体，交互
等级	很低	低	标准	高	很高	超高
工作量乘数	1.22	1.09	1.00	0.93	0.86	0.80

3.3.3 多模块的工作量估算

COCOMO II 模型可以为多组件系统估算工作量和进度。步骤如下：

1、对所有组件的规模 $Size_i$ 求和。

$$Size_{Aggregate} = \sum_{i=1}^{n} Size$$

2、将比例因子和 SCED 成本驱动因子应用于总规模，以导出项目的总工作量。

$$PM_{Basic} = A \times (Size_{Aggregate})^{E} \times SCED$$

3、根据每个组件对总规模的比重，确定每个组件的基工作量 $PM_{Basi(1)}$。

$$PM_{Basi(1)} = PM_{Basic} \times (\frac{Size_i}{Size_{Aggregate}})$$

4、将组件级成本驱动因子应用于组件的基本工作量。

$$PM_i = PM_{Basic(i)} \times \sum_{j=1}^{16} EM_j$$

5、求和，得到项目的总工作量， $PM_{Aggregate}$。

$$PM_{Aggregate} = \sum_{i=1}^{n} PM_i$$

3.4 进度估算

COCOMO II 模型对早期设计阶段和后体系结构阶段初始进度估算方程如下：

$$\text{TDEV}=[\text{C}\times(\text{PM}_{NS})^{(D+0.2\times(E-B))}]\times\frac{SCED\%}{100}$$

公式中，C=3.67，D=0.28，B=0.91。其中 C，D 是一个可校准的系数； PM_{NS} 不考虑 *SCED*因子；E 是从比例因子之和导出的规模指数；B 为校准后的比例因子指数；*SCED*%是 *SCED*等级量表中的压缩/扩展比。

3.5 对模型的改进

对工作量计算公式，引入偏置因子，并运用加权粒子群优化算法对因子进行调整，使其更具有适用性。

3.5.1 基本工作模型

估算软件工作量的一般做法是将其表示为项目规模的一个可变函数，工作量公式如下：

$$PM = A \times \text{Size}^E \times \prod_{i=1}^{n} EM$$

$$E = B + 0.01 \times \sum_{j=1}^{5} SF_{\text{j}}$$

其中 A，B 是常数。

3.5.2 带惯性权重的标准 PSO

首先将 PSO 初始化为一群随机粒子，通过迭代找到最优解。所有的粒子都有一个函数适应值(fitness value)和速度向量决定收敛的方向和距离，粒子群追随当前最优粒子的位置在解空间中搜索。每次迭代过程，粒子通过跟踪两个"极值"来更新自己的位置，一个极值是粒子本身所找到的最优解为局部最优解，记为 *Pbest*；另一个极值是整个种群目前找到的最优解为全局最优解，记为 *Gbest*。

PSO 的基本概念在于每个粒子加速实现其 *Pbest* 和 *Gbest* 位置更新，惯性权重每次通过一个随机数对加速度加权。可以通过使用下列数学建模公式修改粒子的位置：

$$V_i^{k+1} = w * V_i^k + c_1 * rand()_1 * (Pbest - S_i^k) + c_2 * rand()_2 (Gbest - S_i^k)$$
$$S_i^{k+1} = S_i^k + V_i^{k+1}$$

基中，c_1，c_2，S_i^k 是当前位置，S_i^{k+1} 是修改后的位置，V_i^k 是当前速度，V_i^{k+1} 修改后的速度，V_{Pbest} 是基于对 *Pbest* 的速度，V_{Gbest} 是基于对 *Gbest* 的速度，W 是权重函数，c_j 是权重因子，*rand*() 是 0 和 1 之间的随机数。

3.5.3 带惯性权重的标准 PSO 软件工作量估算

参数 A，B 通过历史数据的回归分析来测量。我们使用加权的标准 PSO 来调整这些参数，用下面的公式进行加权函数的更新。

$$W_{new} = [(T_{mi} - T_{ci}) * (W_{iv} - W_{fv})] / T_{mi} + W_{fv}$$

W_{new} 是新的加权因子，T_{mi} 是最大迭代次数，T_{ci} 当前迭代次数，W_{iv} 权重的初始值，W_{fv} 权重的最终值。

在实验过程中，惯性权重从 0.9 线性下降至 0.4。在第一个实验中，保持参数 C_1 和 C_2（加权因子）固定，而随后的实验中，迭代中采用下列公式改变 C_1 和 C_2（加权因子）的值。

认知学习因子：$C_1(t) = 2.5 - 2 * (t / \max\_iter)$ 。

社会学习因子：$C_2(t) = 0.5 - 2 * (t / \max\_iter)$。

3.5.4 模型描述

在此模型中，我们已经考虑了“带惯性权重的标准 PSO”有/无改变加权因子（C_1, C_2）。

PSO 是一个基于成群运动的智能学习算法，本文利用这种成群的行为是用于调整软件成本/工作量参数。

1、方法

输入：软件项目的规模，工作量，工作量乘数（EAF，Effort Adjustment factor）。

输出：工作量估算的优化参数。

以下是用于调整软件工作量估算在改进模型参数的方法。

步骤 1：初始化粒子群。用随机位置 P_i 和速度矢量 V_i 初始化粒子群，粒子群规模为 n，粒子群速度取值范围[$-V_{\max}$， $V_{\max}$]，每个粒子的局部最佳初始位置。

步骤 2：初始化权重函数 w 赋值 0.5，权重参数的认知学习因子 C_1，社会学习因子 C_2 赋值 2.0。

步骤 3：重复步骤 4-9 直到达到指定次数或者得到最优值。

步骤 4：for i =1，2，……，n do//对每个粒子赋值并调整参数。对改进函数求值，改进函数是相对误差的绝对值的平均（MARE），这种方法的目的是从第 1 步中指定的范围内选择适当的值，以尽量减少 MARE。

步骤 5：*Pbest* 是通过计算和比较工作量值，估算当前的工作量值和参数值，来确定每个粒子的位置。如果改进（p）比改进（*Pbest*）好，那么：*Pbest*=p。

步骤 6：设定'*Pbest*'的最优值为全局最优值---*Gbest*。估算和测量的工作量中粒子值变化最小的为 *Gbest* 粒子。

步骤 7：通过以下公式更新权重函数。

$$W_{new} = [(T_{mi} - T_{ci}) * (W_{iv} - W_{fv})] / T_{mi} + W_{fv}$$

步骤 8：通过下列公式更新加权因子以实现更快收敛。

$$C_1(t) = 2.5 - 2 * (t / \max\_iter)$$

$$C_2(t) = 0.5 - 2 * (t / \max\_iter)$$

步骤 9：用下面的公式更新调整参数的速度和位置。for j=1，2，……m do//参数数量，我们例子中取 m=2，3 或 4。

$$V_{ji}^{k+1} = w * V_{ji}^{k} + c * rand()_1 * (Pbest - S_{ji}^{k}) + c_2 * rand()_2 (Gbest - S_{ji}^{k})$$

$$S_{ji}^{k+1} = S_{ji}^{k} + V_{ji}^{k+1}$$

步骤 10：*Gbest* 值作为最佳解决方案。

步骤 11：结束。

2、构建基于粒子群算法的软件工作量估算模型

一些项目规模小，但工作量乘数和复杂度较高，而有一些项目规模庞大，但复杂度低，这些复杂因素造成输入参数的不确定性和非线性。模型的规模和工作量是成正比的。但是有时因为输入项的偏离，并不能满足这种情况，这时可引入偏置因子（D）进行统

计。所以工作量估算公式是：

$$PM = A \times Size^{E} \times \prod_{i=1}^{n} EM + C \times \prod_{i=1}^{n} EM + D$$

$$E = B + 0.01 \times \sum_{j=1}^{5} SF_{j}$$

现在我们用上述 PSO 的方法调整模型的参数。

更新参数“C”的速度和位置

$$V_{Ci}^{k+1} = w * V_{Ci}^{k} + c_1 * rand()_1 * (Pbest - S_{Ci}^{k}) + c_2 * rand()_2 (Gbest - S_{Ci}^{k})$$

$$S_{Ci}^{k+1} = S_{Ci}^{k} + V_{Ci}^{k+1}$$

更新参数“D”的速度和位置

$$V_{Di}^{k+1} = w * V_{Di}^{k} + c_1 * rand()_1 * (Pbest - S_{Di}^{k}) + c_2 * rand()_2 (Gbest - S_{Di}^{k})$$

$$S_{Di}^{k+1} = S_{Di}^{k} + V_{Di}^{k+1}$$

3.5.5 模型分析

1、实施

已经实现了在“JAVA”语言中用上述方法调整参数 C 和 D。参数“C”粒子的速度和位置更新应用下列公式：

$$V_{Ci}^{k+1} = w * V_{Ci}^{k} + c_1 * rand()_1 * (Pbest - S_{Ci}^{k}) + c_2 * rand()_2 (Gbest - S_{Ci}^{k})$$

$$S_{Ci}^{k+1} = S_{Ci}^{k} + V_{Ci}^{k+1}$$

同样得到参数 D 的值，通过第一个试验获得，通过迭代修正权重因子 w，c_1，c_2是常数。第二个实验，我们通过公式来改变c_1，c_2的加权因子。

2、性能措施

我们考虑三个方面的性能标准：

方差的百分比(VAF)如下：

$$\%VAF = [1 - \frac{\mathrm{var}(ME - EE)}{\mathrm{var}(ME)}] \times 100$$

相对误差绝对值的均值(MARE)如下：

$$\%MARE = mean[\frac{abs(ME - EE)}{(ME)}] \times 100$$

相对误差的方差(VARE)如下：

$$\%VARE = \mathrm{var}[\frac{abs(ME - EE)}{(ME)}] \times 100$$

ME 代表测量工作量，EE 代表估计工作量。

3.5.6 模型试验

实验 1：

对于这些模型的研究中，我们应用 6 个项目的数据进行了实验，项目数据见表 3.26。

表 3.26 项目数据

项目编号	规模（人月）	工作量测量（月）
1	2.1	5
2	3.1	7
3	4.2	9
4	1.25	2.39
5	4.65	7.9
6	5.45	9.08

通过实验，我们得出以下值。

C=0.045208 和 D=-2.020790。范围是 C[-1，1]和 D[1，20]。

实验 2：

下面结果是通过每次迭代都改变加权因子 C 的 PSO 算法得到。

C=-0.020757 和 D =0.767248。范围是 C[-1，1]和 D[1，20]。

3.5.7 结果与分析

表 3.27 显示了我们的改进模型估算的工作量。

表 3.27 改进模型估算的工作量

项目	规模	实际工作量	工作量估算，在迭代中 C1，C2 是常数（实验 1）	工作量估算，在迭代中 C1，C2 变化（实验 2）
1	2.1	5	5.000007	5.000001
2	3.1	7	7.17543	6.975912
3	4.2	9	8.999259	9.054642
4	1.2.	2..39	2.405549	2.282118
5	4.6.	7.9	7.184614	7.33909
6	5.4.	9.08	8.204368	8.344935

表 3.28 显示了改进模型与其他估算方法的比较结果。

表 3.28 与其他估算方法的比较

模型工作量	未调整的模型	实验-1 模型	实验-2 模型
5	7.226	5.000007	5.000001
7	8.212	7.17543	6.975912
9	10.357	8.999259	9.054642
2.39	1.916	2.405549	2.282118
7.9	6.824	7.184614	7.33909
9.08	8.093	8.204368	8.344935

3.5.8 性能分析

表 3.29 显示了改进前后的效果对比。

表 3.29 效果比较

模型	差额统计(VAF)	相对误差绝对值的均值 (MARE)	相对误差的方差 (VARE)
未调整的模型	98．41	26.488	6.07
实验 1 模型	99.15	10.803	2.25
实验 2 模型	93.147	17.325	1.21

本文用两种方法改进了 COCOMO II 模型，一是粒子群算法中的参数因子 C1，C2 不改变时，工作量估算结果与实际工作量结果进行比较，可从纵观分析知，工作量估算数据可精确到（10 的负 6 次方）；二是当粒子群算法中的参数因子 C1，C2 改变时，与实际工作量和参数因子不变的情况下比较，数据更为准确，误差有明显下降趋势。我们用第二种算法作为改进算法，应用到一个项目实例中检测其适用性。

第四章　应用实例

4.1 项目介绍

NTIT 教育网是把视频技术、视频和音频数据的压缩及解压处理技术、互联网应用技术相结合，把影视、图形、图像、声音、动画以及文字等各种多媒体信息及控制实时动态地引入教学过程的一种专用电脑网络教学平台，是利用计算机技术、网络技术、多媒体技术进行现代化教学活动的一个系统概念。有了它，教师可以通过电脑，利用图形、动画乃至声音、音乐以及多媒体教学软件等先进手段在广域网上进行远程教学活动，具有多媒体广播教学（屏幕广播、音视频广播）。它较传统教学方式更加现代社会对教育的高效、方便、频繁交流的需要。随着如今网络的高速发展，NTIT 教育网将是人们学习的一种重要手段。

4.1.1 子系统介绍

该系统主要由 6 个子系统组成，其主要描述见表 4.1

表 4.1　子系统介绍

子系统编号	子系统名称	功能说明
01	用户管理子系统	用户信息管理是实现用户的添加、修改和删除等功能，主要是用户信息的维护。
02	帐号管理子系统	帐号管理子系统是帐号进行管理和规划，并为帐号与财务信息提供原始依据。
03	我的课堂子系统	我的课堂子系统是对已经注册的用户的个人信息、资料、课程包及学习进度的管理。
04	会员管理子系统	会员可查看自己的信息，管理员可查看所有会员信息。
05	视频管理子系统	包括前台注册会员以及游客对视频的播放和后台操作人员对视频文件的相关信息的维护
06	课程包管理子系统	包括对课程包，课程类型以及课程的管理

4.1.2 模块汇总表

提示：这里模块是指相对独立的软件设计单元，例如对象类、函数包等等，系统模块汇总见表 4.2。

表 4.2 模块汇总表

用户管理	
模块名称	功能简述
用户信息管理	系统用的基本信息的管理，包括添加新用户、修改用户信息、删除用户功能。
用户权限管理	用户所属角色分组，如管理员组、帐号管理组、游客组等，另外是权限管理，如查询、修改、删除、浏览等权限的设置。
帐号管理	
模块名称	功能简述
帐号信息管理	系统对所有帐号信息进行查询和维护。
帐号情况统计	主要对所有帐号使用情况进行统计。
我的课堂	
模块名称	功能简述
课程试听	对网校中的课程进行试听。
学习进度管理	对学习过程中的进度进行管理。
我的课程包管理	对当前用户的课程包进行维护和查询。
我的个人账户管理	对个人帐号进行维护查询，并详细了解个人信息情况。
我的资料管理	对网校学习中的学习资料进行管理。
我的消息管理	对网络消息进行查询维护。
会员管理	
模块名称	功能简述
会员信息审核	对已经注册的网校会员信息进行审核。
会员密码重置并发送邮件	修改会员密码并可以发送邮件告知会员修改信息。
会员信息修改	修改会员信息。
视频管理	
模块名称	功能简述
视频信息管理	对视频进行维护查询。
视频文件上传	对视频文件进行上传。
课程管理	
模块名称	功能简述
课程信息管理	对网校课程进行维护查询，建立课程体系结构。
设置课程所属的课程包	对课程包所包含的课程进行设置。
课程包管理	
模块名称	功能简述
课程包信息管理	对课程包进行维护查询。
课程包类型信息管理	对课程包的类型进行维护。

4.1.3 子系统关系图

图 4.1 显示了子系统之间的逻辑关系。

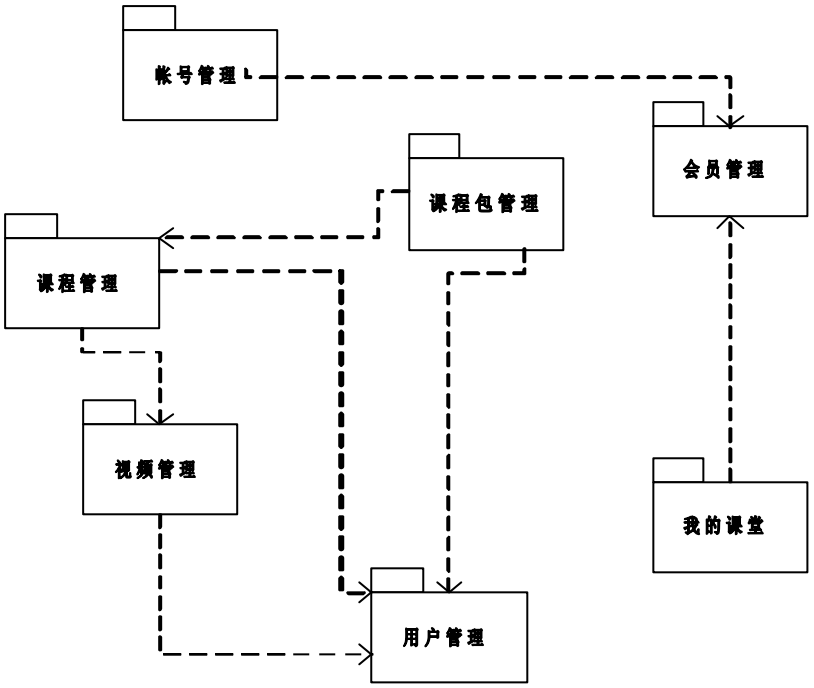

图 4.1 子系统关系图

4.1.4 系统拓扑图

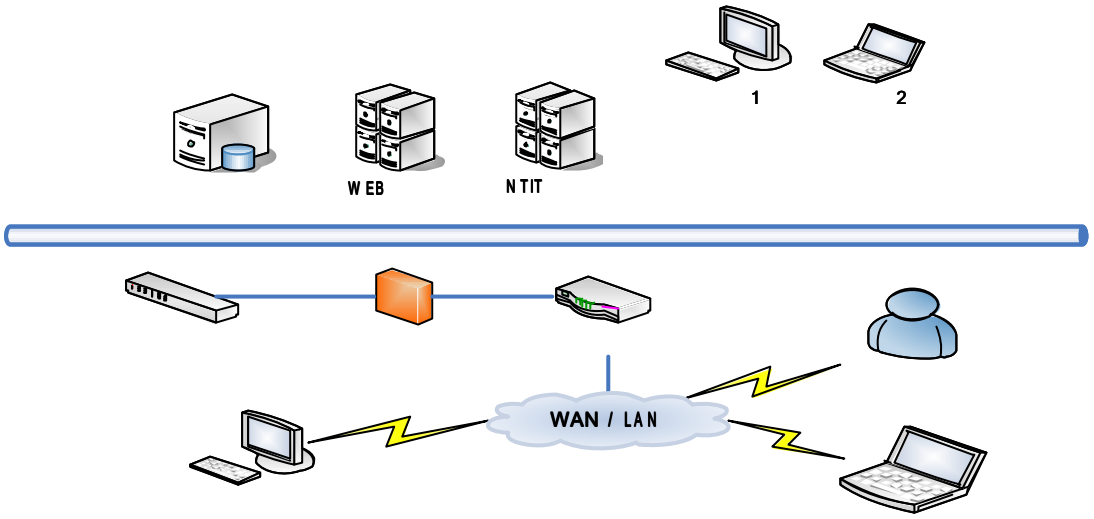

图 4.2 系统拓扑图

服务器和客户机通过网络连接在一起，通过 Intra / Internet，事务交互式地在服务器和客户机之间相互通信（见图 4.2）。客户工作站识别查询 / 命令，并且产生一个消息发送到合适的客户机/服务器执行。通信功能确定相应的协议，并对消息打包后开始传输。服务器对客户的消息请求按照预先确立的应答协议来响应。文件(包括数据、多媒体等)或消息的 applet 程序按照预先确定(如确认收到或重新传送等)传送回来。为此开发的各种软件应用程序，客户可以利用来处理所

收到的数据，以生成自定义和可编程的文本 / 图形报表。可以预定义或由用户交互地开发报表，用户应用程序可以采用各种工具(模板、框架等)以链接到处理过程中去。服务器和客户机系统双方通过内建的认证和安全控制措施来防止非授权的访问。系统双方执行同步和状态监控功能，来保证在适当的期限内处理对服务的请求。

4.1.5 Make or Buy or Reuse

表 4.3 描述了对系统组件采用哪种形式完成的决策和描述。

表 4.3 开发、购买或者复用

序号	模块名称	分析	购买	开发	复用
1	视频录制引擎	技术支持；加快项目开发速度；保证项目质量。	√		
2	用户管理子系统	用户信息管理是实现用户的添加、修改和删除等功能，主要是用户信息的维护。			√
3	帐号管理子系统	帐号管理子系统是帐号进行管理和规划，并为帐号与财务信息提供原始依据。			√
4	我的课堂子系统	我的课堂子系统是对已经注册的用户的个人信息、资料、课程包及学习进度的管理。		√	
5	会员管理子系统	会员可查看自己的信息，管理员可查看所有会员信息。			√
6	视频管理子系统	包括前台注册会员以及游客对视频的播放和后台操作人员对视频文件的相关信息的维护		√	
7	课程包管理子系统	包括对课程包，课程类型以及课程的管理		√	

4.1.6 开发团队

团队是年轻的、受过良好教育的员工，它们的大多数程序员有至少两年的编程经验，并能熟练使用 JAVA 编程语言，和熟悉相应的编程环境。由于项目有公司高层人员作为高级经理，并且由于管理部门对软件开发项目的理解，为开发工作配备设备和相应软件工具不是问题。这样，项目中每个编程人员平均有 1.7 台工作站可用。高层非常支持员工的培训，在需要的新技术的领域的视频培训和现场课程(Java、持久性数据库、专用工具、Web 编程等)。公司高层鼓励员工每年参加至少一个月的职务培训。技术课程和内部讲授的课程都在现场举办，因而有

很多机会让每个人都参加。另外，公司高层最近发起了一个讨论会计划，组织员工在内部学习讨论。

公司在 09 年启动了一个软件过程改进的工作，收集了以往采用过的最佳实践，增加了可重复的过程。过程和最佳实践及模型实例都是在线可用的。公司大多数程序员都被纳入此过程，并且该过程广泛地用于各个工作组。经过两年的努力，公司的软件能力成熟度模型(CMMI)评定达到了 SEI 第 3 级。目前公司正在对软件开发的进行持续的改进工作，目标是在计划时间内达到 CMMI 第 4 级的认证。

当然，公司的软件开发团队也存在一些问题。其初级管理人员的提拔是主要考虑他的技术才能和与同伴的关系。虽然新的管理人员可以通过相应的初步学习培训来得到管理经验，但此类培训相对较少。很多时候，优秀的技术专业人员较难转到管理工作中去。另一个问题是与估算有关。技术组得出的工作量和进度估算有两种不同倾向：太乐观和太悲观。管理都尝试纠正这些倾向，但团队中大多数成员通常相信实际工作会比估算的更简单。管理部门正在引入估算工具和通过相关的培训以试图改善这种情形。

4.2 系统开发估算

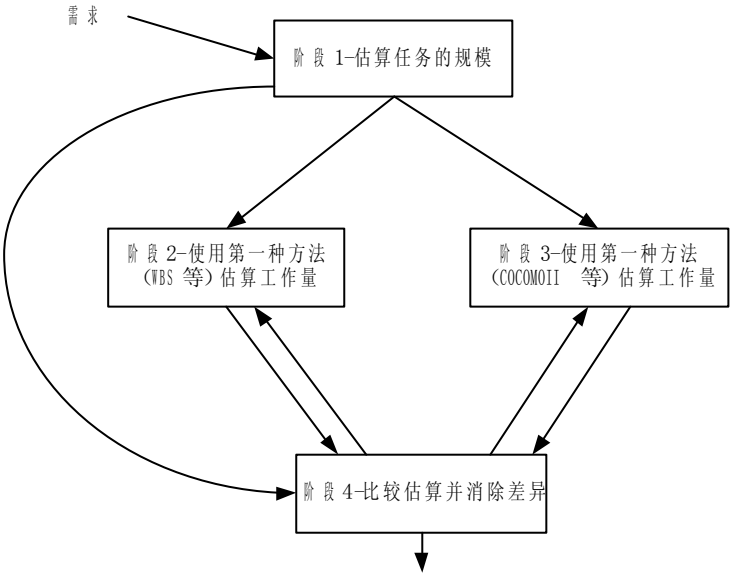

图 4.3 估算过程

估算的第一个阶段是让开发团队与客户一起确定最重要的软件功能和与之相关的软件规模（见图 4.3）。表 4.4 总结了开发团队确认的组件和与之相关的规

模。因为提供了代码行和功能点两种规模估算方法。在使用估算工具时，需要把两者联系起来。

表 4.4 所确认的功能规模

组件	功能	规模（SLOC）	注释
系统软件	视频录制引擎	4K（新）	对库的扩展和接口代码算做新代码
	会员管理子系统	1.5K（复用）	
	用户管理子系统	2K（复用）	
	帐号管理子系统	1.5K（复用）	
	我的课堂子系统	3 K（新）	
	课程包管理子系统	2 K（新）	
	类库		
用户应用程序	屏幕和报表	800（新）	机接口开发采用 GUI 构造器和图形化工具
应用程序接口故障诊断	为用户开发的内建测试	80 功能点（新）	
	故障隔离逻辑	20 功能点（新）	
	恢复管理	25 功能点（新）	

作为估算过程中的第二个阶段，软件开发团队使用标准工作分解结构(WBS)要为团队的每一个工作任务产生一组估算，WBS 代表工作的系统树，必须完成这些工作以实现该团队所定义估算过程的简要版本。根据对过去相似项目的经验证明团队的任务估算是合理的，估算假设使用工作表来记录。记录必须满足客户所需的各种异常情况。项目全部工作量是单个任务估算的总和，见表 4.4。

第三个阶段，过程组用改进 COCOMO II 后体系结构模型开展一个独立的估算。用于开展基于模型估算的 WBS 分解结构，开发团队所作的与模型参数相关的假定将在后文中说明。在估算过程的最后阶段会依据项目风险和客户需求检查和调整估算之间的差异。调整过程也许会使估算小组产生额外的估算。

因为估算需要反映校准模型所依据的数据，下表中列出的活动经确认是不是在估算范围之内，这是一个重要的考虑因素，估算该范围以外的任务，一些 COCOMO II 软件包采用所估算的软件开发工作量的百分比（见表 4.5）。

表 4.5 COCOMO II 模型估算范围

活动	在范围内或外
需求（RQ）	外
产品设计(PD)	内
实现(IM)	内
集成和测试(I&T)	内
项目管理(PM)	内
项目配置管理(CM)	内
项目质量保证(QA)	内
在项目级别上进行的项目管理活动	外

4.2.1 第 1 阶段——估算任务的规模

估算规模，我们结合源代码行法和未调整功能点法，使得估算更准确。

1、首先是要检查最初给出规模估算合理性。本文用相应的公式把这些估算的规模转换成等价的源代码行(SLOC)，以下四类软件制定规范并分别进行处理：新的；修改的；复用的；COTS。

复用软件转换为新代码行，我们用以下公式完成[8]：

$$AAF = (0.4\times DM) + (0.3\times CM) + (0.3\times IM)$$

DM=设计修改%=0%

CM=代码修改%=0%

IM=集成修改%=50%

根据定义[31]，设计的可复用软件是不需做任何改变就可以实现实例化，可是，库如果被集成和包装到系统软件组件中，是需要广泛地测试。库是可以扩展的，如果开发软件时用户可能会增加功能时，新库和接口软件就当作新代码来处理。因此对于所复用的软件，计算出的 AAF 位于 0 与 100 之间，计算如下：

$$AAF=0.4(0)+0.3(0)+0.3(50)=15$$

根据公式计算，每一行代码当作 0.15 行新代码来计算，另外，用于计算理解和消化库所需的工作量作为因子用下面公式计算（当$AA \le 50$时）：

$$ESLOC=AA+AAF(1+(0.02\times SU\times UNFM))/100$$

2、下表列出了评估者为软件理解增量(SU)、评估与消化过程(AA)、不熟悉性级别(UNFM)选择下列值（表 4.6）：

表 4.6 取值与描述

参数	值	描述
软件理解增量(SU)	20	高内聚，优质代码
评估与消化过程(AA)	4%	部分模块测试和评估，文档化
不熟悉性级别(UNFM)	0.8	大部分不熟悉

将表中的值和 AAF 值代入公式中，得到下列等价的软件规模：

ESLOC=5000[0.04+0.15(1+0.02(20) (0.0))] =950 SLOC

所以系统组件的规模估算值为：

9000+950=9950 SLOC。

表 4.7 故障诊断组件功能点估算明细

开发类型		新开发		
组件名称		故障诊断组件		
开发基础		需求规约，系统结构和数据报表		
规模估算规约				
功能类型	复杂度	权重	数量	FP 数（数量与权重的积）
外部输入 EI	简单	3	7	21
	一般	4	1	4
	复杂	6	0	0
小计				25
外部输出 EO	简单	4	7	28
	一般	5	1	5
	复杂	7	0	0
小计				33
外部查询 EQ	简单	3	5	15
	一般	4	1	4
	复杂	6	0	0
	小计			19
内部逻辑文件 ILF	简单	7	4	28
	一般	10	0	0
	复杂	15	0	0
	小计			28
外部接口文件 EIF	简单	5	4	20
	一般	7	0	0
	复杂	10	0	0
	小计			20
FP 总计		125UFPs		

3、用户应用程序组件的规模估算比较复杂，由于使用 GUI 构造器，虽然产生成千上万行代码，但是，实际只需要编写 800 行新脚本，所以成本计算时，只

需要计算 800 行代码。

4、故障诊断组件规模是用功能点表示的，COCOMO II 模型把未调整功能点转换成新源代码行数目，采用以下逆向公式：

ESLOC=FP(语言转换系数)

假定语言转换系数为 80 SLOC／FP，我们将 125 个功能点转换为 10000 行源代码，见表 4.7。

4.2.2 第 2 阶段——用 WBS 分解估算工作量

项目确定了任务的规模之后，接下来是为客户端软件开展第一个成本估算。将估算上图所示的工作分解结构(WBS)中的每个任务成本[32]，并对各行中所有的项目求和结果总结表 4.8 中。

表 4.8 WBS 估算总结

WBS 任务	估算（人时）	估算的基础
确定软件的需求	680	需求数乘以生产率数值
开发软件	10375	源代码行乘以生产率数值
执行任务管理	1115	假定追加 10%来说明其工作量
保持配置控制	620	假定为任务分配一名专职人员
执行软件质量保证	620	假定为任务分配一名专职人员
总计	13410	

1、任务一：定义软件需求

为了量化系统所涉及的工作量，对系统做一些细化：开发一个快速用户界面原型，把它作为一种方法让客户把他们在屏幕界面上的想法实例化，该项目工作量估算大约为 700 小时，即要在 5 周内完成则需要 3.5 个人。

因为考虑到需求的易变性，根据一些预期的／假设的返工百分比，即用 1.2(8 小时／需求)的 20％的返工系数，来调整其损耗量或易变性启发式规则。

根据经验，弥补制定一组可靠需求而所需的额外小时数，可增加 6％到 12％。如果使用 MBASE／RUP 模型时，处理需求任务时可以在起始阶段，增加大约 6％的工作量。如果采用集成产品小组定义需求时，应该增加 10％或 12％。如果需求不清时，工作量估算可能高于 10％到 12％。

2、任务二：开发软件

量化软件开发的相关工作量，需要对项目任务规模进行度量和估算。

需要把组件细分，并根据标准度量估算的每一个部分。本系统中，组件有 16 个部分，其中每个部分需要 100 小时进行设计、开发、测试的工作量，最终

估算将是 1600 小时。估算范围通常是从需求评审确定一直到项目验收测试完成。如果采用了最好的商业实践，产品发布所需的所有编程工作量也在这个范围包含的工作量之内。

用这种方法估算，需要把客户方组件划分到根一级，然后采用基于历史的度量就可以对整个任务开展估算。

作为一种可选方法，用生产率数据，作为估算基础。采用前面导出的源代码行进行规模估算，对客户方系统的三个组件源代码行求和，得 20750 等价新 SLOC。结果如下：

(20750SLOC)／(2SLOC／人时)=10375 人时

大多数决策基于过程中已知的起点和终点，包括了部分管理成本和大部分对以往任务计费的工程成本。从而，作为历史数据的一部分成本会有所不同，造成生产率数据建立不同的基础之上。

3、任务三：进行项目管理

项目管理是作为软件开发范围的一部分而包括进去，但有时在估算过程中不包括进去。如果不包括进去[32]，通常软件开发期间所涉及的任务管理工作按增加 10％的额外工作来说明。这种方法以下列假设为基础：一线的管理人员既管理该任务又从事技术上工作。根据具体环境不同，工作量高低不定。比如在扁平型结构的机构中，任务领导者也从事开发，那么管理的工作量可能相应会少一些。因为任务领导者必需用于计划、控制、激励和指挥的时间，则增加 10％以说明所完成的管理工作量。

4、任务四：保持配置控制

配置控制像任务管理一样有时也作为软件项目开发范围的一部分包括进来。一般，要为软件项目开发所涉及的工作量增加 4％～6％额外费用，尤其是对于大项目。同样，这个估算结果根据环境不同可能会有变化。有些情况下，如果组件在多个产品系列或者多条产品线间共享，即有更多人会受到变化的影响，那么可复用软件组件的变更工作会需要花更多工作量去管理。针对这种情况，假定在项目的 4 个月中专门委派一个人花费一半时间完成配置管理工作，同时管理软件库，并为该项目计算资源。

5、任务五：执行软件质量保证

软件质量保证是 WBS 中最后的任务。软件质量保证提供所需的检查和比较工作，从而以保证开发工作按标准进行，而并且所开发的产品质量是最高的。一

般来说质量保证工作由小组外的机构做，这时通常不作为软件开发估算的工作量计算。而是用固定数目的人员或工作量百分比级别来估算涉及的工作量。一般情况下，根据质量保证人员所完成的工作量，估算的变动范围可以在 4%～8%之间。针对这种估算，质量保证假定在项目的 4 个月中专门委派一人花费一半的工作时间完成。

4.2.3 第 3 阶段——改进 COCOMO II 模型估算

接下来用改进 COCOMO II 模型后体系结构模型进行第二次估算，下图是其方案。估算过程的第一阶段已经完成图中的第一步的估算规模。第二步，则需要评定比例因子和工作量乘数因子等级。

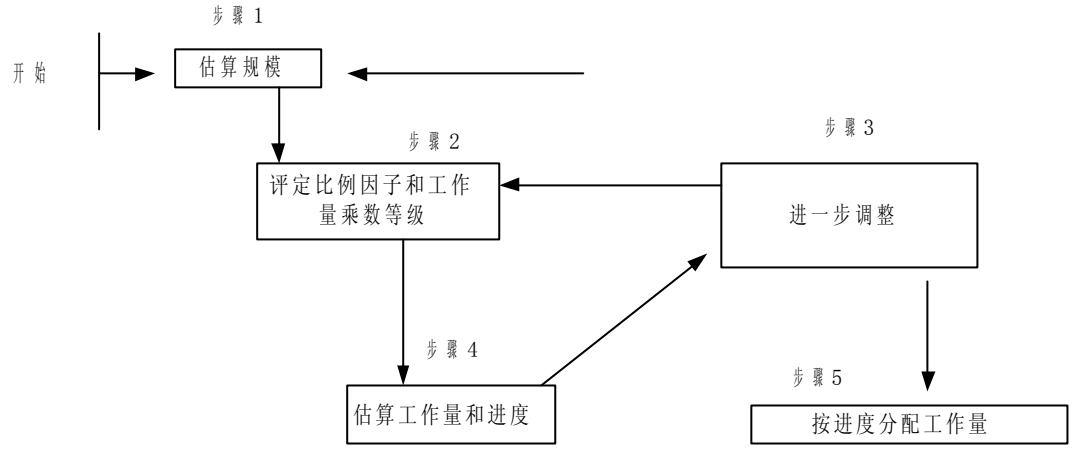

图 4.4 改进 COCOMO II 模型估算执行方案

表 4.9 总结了比例因子等级。可以看到，已经根据案例提供的一些信息作了许多假设。必须了解项目的任务、人员、平台和机构情况。接下来确定组成工作量乘数成本驱动因子的等级。

表 4.9 例因子等级及其基本原理

例因子等级及其基本原理		
比例因子	等级	基本原理
PREC	高	项目组有相关系统工作的经验
FLEX	高	企业进行 CMMI3 过程改进，按规范与标准进行开发。
RESL	高	在风险识别与消除，产品设计评审上基本一致。
TEAM	标称	团队中有新成员，新成员与老成员的磨合还需要一些时间。
PMAT	标称	企业取得 CMMI3 认证。

下面几个表（表 4.10，表 4.11，表 4.12），概述了 COCOMO II 模型每一个成本驱动因子的等级及原理，分别是：产品因子、人员因子、平台因子、项目因子。因为准备快速地开展每一次估算，所以选择在项目一级评定工作量乘数的行

者等级。先看看改进 COCOMO II 模型和 WBS 估算之间是否有较大差异。如果有则需要细化两种估算。

表 4.10 产品成本驱动因子等级和基本原理

产品成本驱动因子等级和基本原理		
成本驱动因子	等级	基本原理
RELY	标称	只有一些潜在的损失。
DATA	标称	没有数据库信息。
CPLX	低	产品复杂性较低。
RUSE	标称	假定标称。
DOCU	高	由于组织级的需要，包含一些超出项目级的文档。

平台成本驱动因子的等级和基本原理，由于硬件的快速发展，开发平台的日趋完善，对于一些小的应用项目，TIME，STOR，PVOL 已经失去了估算成本的影响，所以全部设为标称。

表 4.11 人员成本驱动因子的等级和基本系理

成本驱动因子	等级	基本原理
ACAP	高	分析员是团队的主干，有丰富的经验，良好的个人能力。
PCAP	标称	程序员能力，老程序员有丰富的经验，但由于有新的没有经验的成员的加入，取为标称。
PCON	低	人员流动性大，几乎是每个 IT 企业的通病。
APEX	标称	是平均结果。
LTEX	高	采用的是 Eclipse 每个程序员都很熟悉，平均有三年以上经验。
PLEX	高	JAVA 语言，同样很熟悉。

表 4.12 项目成本驱动因子的等级和基本原理

成本驱动因子	等级	基本原理
TOOL	高	软件工具相对较多，而且也广泛应用。
SITE	极高	完全在同一处地方开发，且容易交流。
SCED	标称	准备按项目计划开发。

由于不完全了解所需的信息，我们对项目做出一些假设，来评定所有表中每个因子的等级。我们采用高估的观点，这样会使误差偏向高的一边，如果项目失败，这种误差也可以为估算保留一些余地。

在考虑产品因子时，因为对其中几个成本驱动因子几乎一无所知：不了解其可靠性（RELY）和复杂性（CPLX）。虽然可以根据相似应用程序的经验做一个有根据的猜测，但是在此情况下还是应该选择标称等级，它是大多数专家和历史数据对因子的所描绘的准则。

对于平台成本驱动因子的评定，因为平台技术的发展，硬件资源的可扩充，

这些因子，我们选择了标称，如果在稍后的过程中，定义需求中，确认了系统和环境约束，可以调整该估算。

人员因子的评定比较复杂。在项目确定之前，许多管理者假定能得到最好的人员，但事实上假设很少能成立。对于大多数企业来说，分析员基本都是有能力的，而且对一个企业来说，其分析员基本都是确定的。程序员比较难以确定，这里假定一个标称的保守组合。另外，确定人员因子级别时，选择的是小组平均水平的级别。

对于人员的流动性，企业中很容易得到统计数据。

对于成本驱动因子，与项目有关。项目组目前使用的开发工具和软件环境比较成熟和先进的。工作区域也是相对确定的。对于 SCED，首先将得出工作量和开发周期的估算。然后，根据开发的过程，调整 SCED 因子反映所期望的进度。

如果比例因子与工作量乘数的等级全部能确定，则能进行初步的工作量和开发估算。

通过改进 COCOMO II 模型导出的结果，工作量最有可能的 54.4 人月(PM)，开发周期 7.2 个月。

4.2.4 第 4 阶段——比较各估算结果并消除误差

WBS 估算结果为 9 个月完成大约 13410 人时的项目任务，改进 COCOMO II 模型则认为，任务可以在 7 个月，用约 8262 人时完成。

两个估算之间有很大的差异，对这种差异可能有以下的原因：

⑴ 两个估算所包含的范围可能不同；

⑵ 对 WBS 估算和改进 COCOMO II 模型的简单化假设引起了广泛变化；

⑶ 其差异表示还应该考虑到其他一些因素。

对于第一个原因，很明显两个估算的范围不同。WBS 估算包括需求定义[8]、软件开发、管理、配置管理和质量保证任务。而改进 COCOMO II 模型不包括定义需求的工作量。所以改进 COCOMO II 模型必须增加到可与 WBS 估算相比较的范围。另外，7 个月的标称进度需要增加得出需求所需的时间量。这样一个定义任务很需要花费 1 到 2 个月来完成。

对于第二个原因，涉及运用模型时所作的简化假设。系统软件比 GUI 或故障诊断软件更为复杂。另外，员工的相关项目经验和能力常常作为开发软件类型的一个直接函数而产生相应变化。只对系统软件等级进行改变，将复杂性提高到“高”等级、而将程序员 / 分析员经验与能力调整为“标称”等级，改进 COCOMO

II 模型估算的工作量相应增加到 9885 人时(即 65 人月)和软件周期 8.4 个月。因此，这两个的估算之间的差距迅速缩小。

其实我们应采取更悲观的观点高估目前的情形。因为，这是软件估算的准则和 WBS 估算基础的指导原则。如果体系结构风险不是最初认为的好控制，还有一些与需求有关的不确定性。那么预测则围绕最可能的情况而不是最好的情况。为了反映可能发生的比较糟糕的事情状态，应该将比例因子级别设置为标称。这样，估算值就变为 11678 人时(即，76.8 人月)和 8.7 个月的持续时间。

对于第三个问题，所涉及的其他可能影响估算的因素。假定一个员工一个月工作时间是 152 个有效小时，其中把非生产性时间和各种假期与请假时间包括在内。但是大多数软件开发人员通常实际上每周投入 50～60 小时工作。在这种情况下，还要考虑这些义务的加班时间。然而，工作量的估算不受加班时间的影响，需要的工作量不会改变。

开展估算的重要性在于对于假设问题的选择和决策。用一个模型，艰苦的估算过程得出的数字往往不是估算的全部东西，估算实际上就是做出选择、估价风险并考虑可能潜在的影响成本和进度的因子的影响。

表 4.13 风险描述

风险	优先次序	描述
管理者是技术人员	未定	管理者是根据他们的技术技能选出来的，这将导致新的管理人员偏爱其熟悉的技术等问题。
新的过程	未定	企业投入了大量资金进行 CMMI3 过程改进，但将过程实践进行到项目中去，也需要时间去引导和完善。
缺乏可信度的估算	未定	任务总是低于或高于最初的估算，这个过程问题导致不现实的期望，并导致任务的完成难于实现。
新的规模度量	未定	选择 SLOC 或功能点来度量所涉及的工作量，为两者设置步骤会导致冗余的工作。
规模增长	未定	估算的基础是规模。但是，规模估算可能包含很大的不确定性，而我们的估算也许会低估。不正确的规模估算可能对成本有深远的影响，因为其影响。是非线性的。
人员流动	未定	通常真正优秀的人员会被安排去执行高优先级的任务。经验级别的混合使平均经验等级达不到所期望的值或满足不了完成任务的需要，而且有经验的人会共较多的时间指导接替工作的人。
易变的需求	未定	需求从不稳定，然而，当它们的变化没有得到恰当的控制时，需求变成一个主要因素。
有挑战的进度	未定	通常会告诉管理部门将试图达成他们的进度目标，即使在它并不现实的时候也会如此。这样，选择方案就可能被取消或缩小项目的范围。

1、划定风险边界

完成了估算之后，接下来可以用模型，利用已产生的成本和持续时间估算，可以量化估算过程中识别的任务风险的影响。利用这些数据作为基础，可以按优先次序，即主次来排列风险并制定降低风险的计划。

⑴ 管理者是技术人员：改进 COCOMO II 模型假定项目管理良好，该模型通常没有考虑将经验丰富的技术人员提拔为管理者的情况。该问题可以通过 ACAP、PCAP、TEAM 的等级来考虑它的影响，ACAP 和 PCAP 涉及了个人能力，也提到了开发者与客户团队的能力。

⑵ 新的过程：虽然新过程对成本和生产率有一个正面的长期的影响，但在短期内会有一些混乱，量化这个短期影响的方法是通过比例因子 PMAT 和成本驱动因子 DOCU。此外，可能通过考虑这些因子的不同等级而评估相关的影响。如果是第一次使用新的风险管理过程，则可以将 RESL 包括在影响分析中。因为认识到是在整个项目的全程中考虑这些因子的综合影响，所以为每个比例因子和成本驱动因子采用中间的等级会更好地反映想暴露的情况。

⑶ 缺乏可信度的估算：改进估算可信度的唯一方法是改变直觉。采用定义的估算过程和校准的成本模型，如改进 COCOMO II 模型，能在以后极大地改进估算的准确性。然而，它不会改变由过去的准确性而产生的看法。为了评估这个问题的影响程度，我们所能做的是确定过去的估算偏离了多远。然后可以利用过去的标准误差系数调整估算。以便对诸如规模和复杂性这样的关键驱动因子，确定拙劣估算过程对成本和进度估算的影响。

⑷ 新的规模度量：在改进 COCOMO II 模型中，源代码行是与工作量和成本规模相关的基础，功能点可以按转换系数转换成源代码行。但是，当规模度量的选择过于自由时，可能会产生工作量的混淆和重复。必须早些做出抉择。不然，缺乏清楚的定义和计数约定，会使规模估算导致很大的差异。

⑸ 规模增长：为了利用原始的 COCOMO II 模型去调整产品的有效规模以反映可能的增长，我们可以使用 PERT 规模估算公式：

$$Size = (a_i + 4m_i + b_i)/6$$

$$\phi_i = (b_i - a_i)/6$$

其中，a_i 指软件组件中可能最小的规模；m_i 指软件组件中可能最有可能的规模；b_i 是软件组件中可能最大的规模；ϕ_i 是估算标准偏差。

⑹ 人员流动：COCOMO II 模型有一个成本驱动因子，人员连续性即 PCON，

它考虑到了每年的人员流动，增加的流动性影响可以通过调整这个因子的等级来计算。但单独考虑这个因子并不够，还要调整相关的经验因子 APEX、PLEX 和 LTEX 的等级，以更真实地评估由于用经验不足的人员替换经验丰富的人引起的工作量和持续时间的变化。

⑺ 易变的需求：COCOMO II 模型中的一个因子需求演化和易变性 REVL，来调整产品的有效规模，并反映由于需求不稳定而必须丢弃的代码百分比。

⑻ 有挑战的进度：COCOMO II 模型提供了最可能的持续时间，有效地使用 COCOMO II 模型要求调整进度 SCED，以反映所希望的端点。

2、执行权衡研究

完成了估算之后，接下来可以进行许多有价值的权衡研究。可以使用 COTS 软件包和不同开发规范对成本和进度估算的影响。下面以改进 COCOMO II 模型估算作为基础进行以下两种权衡研究：

⑴ COTS 软件权衡研究

COTS 软件使用会产生明显的成本和进度的杠杆作用。COTS 软件是已经存在的软件而且无需修改就可以用于实现应用程序中的功能。COTS 软件维护由第三方来提供。因此，开发者的工作可以集中在如何把软件包集成到应用程序中去并进行测试。如果要对 COTS 增加功能，须要付费让供应商去完成。但不得不修改 COTS 时就失去了许多优势。为也评估与 COTS 相关的成本/效益，构造下表来跟踪。

表 4.14 COTS 跟踪

因子	COTS	新开发
购买组件成本	10000	92400
集成和测试成本	10000	已包括
运行许可成本	5000	不适用
维护成本	5000	9240
总计	30000	101640

⑵ 方法权衡研究

接下来对比增量开发产品和一次性构造之间的成本差异。风险处理中，对于有挑战的进度，最常用的解决方法是增量开发。把软件开发过程进行细分，从而划分成成多可交付部分，每部分分别完成一定功能。开发人员可以采用多种方式完成这些可交付部分。最常用的两种方法是增量交付和增量开发。增量交付，每个构件作为独立的实体开发、测试和交付。对于这种情况，估算中包括每个构件

的测试和包装的成本。而对于增量开发，可以在最后交付的部分进入集成和测试阶段才进行完全测试。这两种方法都是提供部分产品。但是，增量开发通常花费更小，因为增量开发减少了部分交付所需测试与包装的时间和工作量[32]。

在 NTIT 教育网中，分析每个增量开发方法所需增加的时间与工作量。以分配到每个构件的功能为基础来估算增量的规模。下表我们把规模分为如下两个部分系统软件的规模和应用程序的规模。

表 4.15 系统软件的规模和应用程序的规模

构件号	模块	规模（SLOC）
1	系统软件	9000（新开发） 5000（复用造价为 950）
2	应用软件	构件 1（9950 SLOC） 用户应用程序（800 SLOC） 故障诊断（10000 SLOC）

系统软件是产生构件 2 的基础，表中把构件 1 作为构件 2 的一部分。应用程序所需的功能的于实现，是作为增量开发或增量交付的第二次增量开发的一部分，要对该软件进行集成和测试或者修改。

对于增量交付和增量开发，把两次提交看作两个分离的子项目。第一个子项目的有效规模是 9950SLOC。可估算构件 2 的规模如下：

有效规模=规模（0.4（%DM）+0.3(%CM)+0.3(%IM)）

=9950 (0.4(0.20)+0.3(0.30)+0.3(0.30))

=9950 (0.26)

=2587 SLOC

规模（构件 2）=2587+800+10000=13387 SLOC

其中：%DM 是设计修改的百分比[8]；%CM=代码修改的百分比；%IM=集成修改的百分比。

用计算修改百分比的因子来说明两种规范之间的差异。增量交付由于承受更多测试，所假设的百分比会更大。接下来，我们将用 COCOMO II 模型对每个构件计算其工作量和持续时间，并且采用不同的规模估算来驱动模型的输出。整个任务的估算是两个构件估算之和。增量开发的总估算虽然要比一次性实现的估算要高，因为可以实现第二个构件的开发与第一个构件的部署并行地的进行，因此它更容易得到较早的初始运行能力。

上面两个权衡研究运用成本模型来评估项目方案的成本／效益。要实现一个简单的初步方案，可以通过改变比例因子和成本驱动因子的等级来导出。通过使

用 COCOMO II 附带的软件包能力，使方案的分析更为成熟和详细。

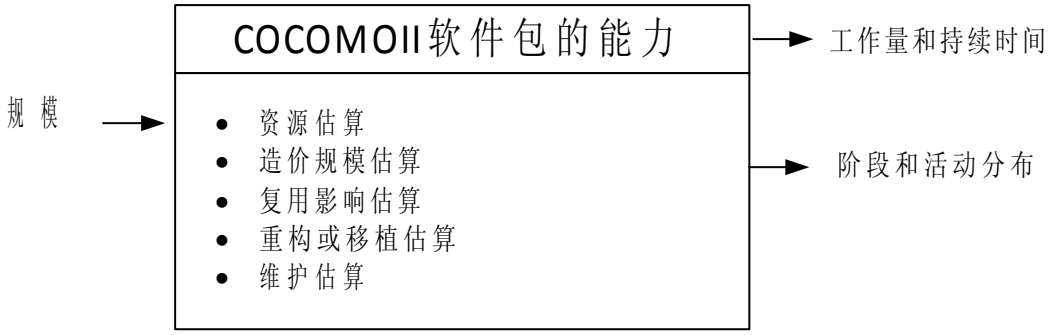

图 4.5 COCOMO II 软件包能力

表 4.16 详细分析

非经常性成本	**有效的效益**
● 购买	● 成本避免
– 资产	– 减少时间
– 方法/工具	– 减少工作量
● 过程适应	● 增加性能
● 文档化	● 减少周期时间
● 降低风险	● 降低质量成本
● 学习和培训	● 成本节约
– 开发	– 减少人员
– 管理	– 减少设备
– Beta 测试	– 减少管理费用
● 其他	● 其他
总计	总计
经常性成本	**无形的效益**
● 管理	● 更好的顾客满意度
● 重构	● 更好的使用适应性
● 维护	● 降低上市时间
● 运行	● 增加市场占有率
● 持续学习	● 提升形象
● 其他	● 其他
总计	总计
总成本	**总效益**

3、评估生命周期

生命周期成本至少包括开发成本和维护成本。COCOMO II 模型可以进行生命周期成本分析。COCOMO II 软件包提供了一个维护模型，用来估算软件交付和运行后最新版本的相关成本。COCOMO II 的维护模型，假定增加或修改的代码量超出所新开发的 20%。用复用模型估算新版本的公式如下。

$$PM_M = A(\mathrm{Size}_M)^E \times \prod_{i=1}^{17} EM_i$$

$$\mathrm{Size}_M = [(\text{基本代码规模})\mathrm{MCF}]\mathrm{MAF}$$

$$\mathrm{MAF} = 1 + [\mathrm{SU}/100]\mathrm{UNFM}$$

$$MCF = \frac{\text{增加规模} + \text{修改规模}}{\text{基本代码规模}}$$

复用模型常量 A 和 E 与 COCOMO II 模型形式相同。除 RELY（需求的可靠性）之外，其他的工作量乘数 EM_i 是相同的。维护可靠性较低的软件花费更大。此外，在维护计算中 REUSE 成本驱动因子和 SCED 忽略不计。

COCOMO II 维护模型用所修改的代码百分比来估算工作量，在应用中应该修改工作乘数和比例因子来反映软件维护情况，以下是详细分析。

表 4.16 用来检查是否界定了在生命周期中所使用的 COTS 和重新开发软件的所有成本。但是，其对前面的分析并不是真正很全面，主要是没有考虑以下成本（表 4.17）：

表 4.17 未考虑到的成本对比

因子	COTS	新开发
连接代码开发(2KSLOC，每行 50 元)	10000	不适用
集成和测试成本——测试接口代码与 COTS 间的接口	5000	不适用
学习、理解 COTS 和它的能力，包括程序员的时间	5000	不适用
维护成本(5 年)——假定每年 10%用于新软件和每年 25%用于接口软件	5000	9240
购买成本	10000	92400
总计	35000	101640

如果再考虑到未来情况时，这两种选择方案的成本差别便会迅速减小，这样，必须对采用风险和无形因素做出决策。例如，如果开发的企业生存状况受到怀疑，或者应用程序是关系到核心竞争力，那么应该自己开发应用程序。但如果是注重减少上市时间或注重增加市场占有率，这时可能适合用 COTS。根据估算做出决策，那么分析地越彻底，决策就会越好。通过运用 COCOMO II 模型，可以进行各种权衡研究和经济分析。

4.3 改进后的 COCOMO II 模型与未改进的 COCOMO II 模型比较分析

4.3.1 项目参数图表分析

图 4.6 到 4.10 显示了项目中估算结果与实际工作量的对比分析图表。

图 4.6 可以看出项目各个阶段开发工作量的分布情况。

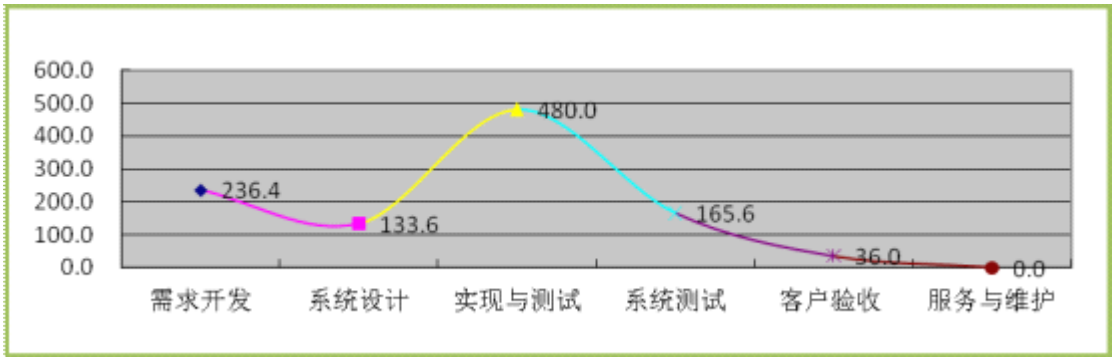

图 4.6 项目工程实际工作量按阶段分布图

图 4.7 可以看出项目各阶段的工作量估算偏差情况及其趋势。

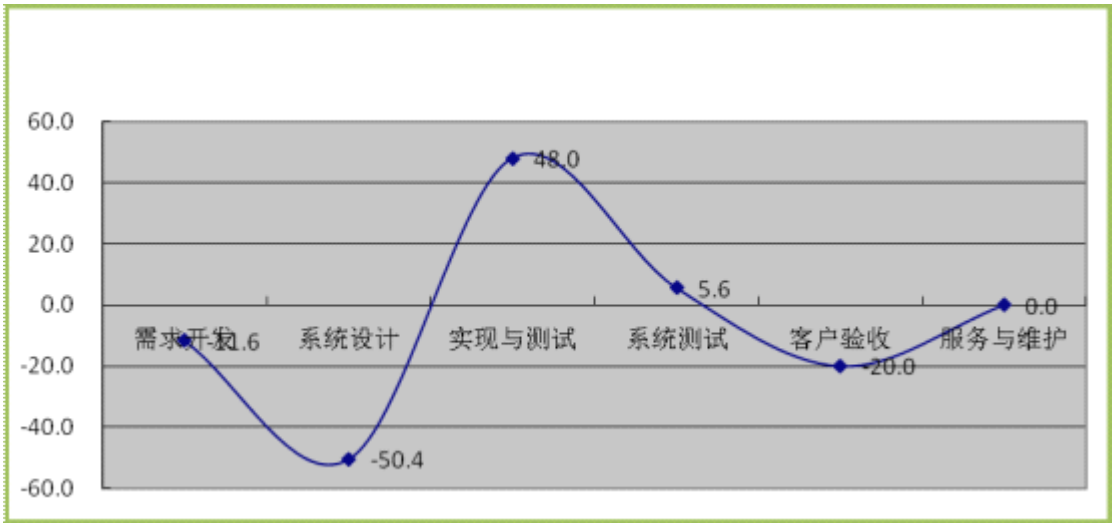

图 4.7 工作量偏差率趋势分析

图 4.8 可以看出项目各阶段的进度偏差情况。

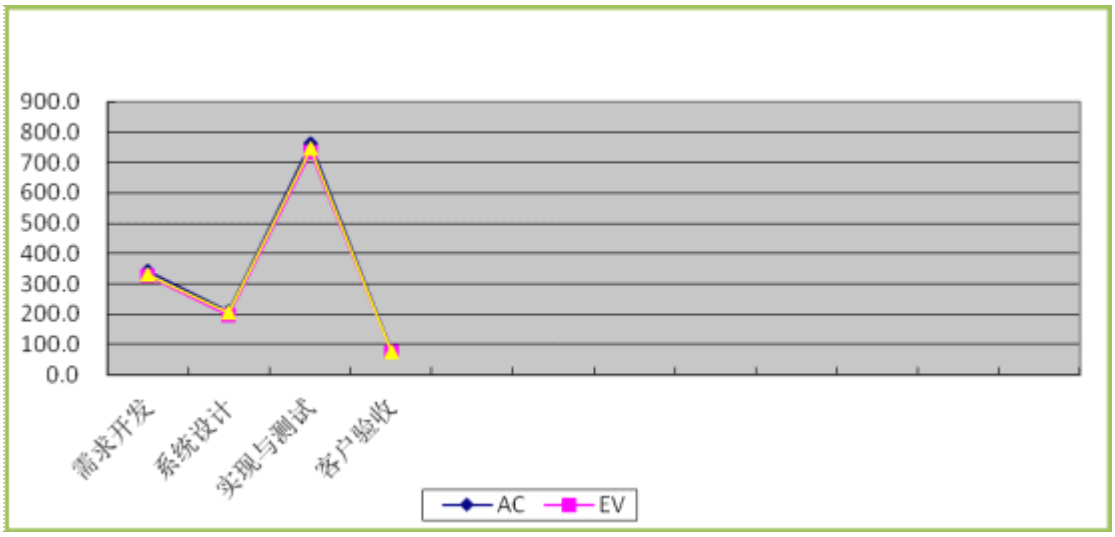

图 4.8 项目进展盈余分析图

图 4.9 可以看出项目进度、成本偏差趋势。

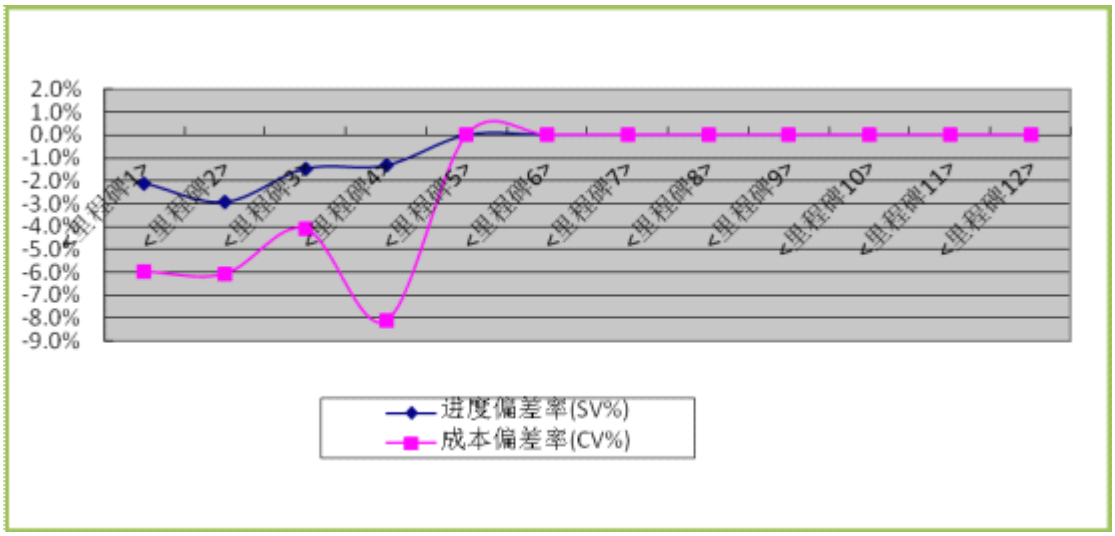

图 4.9 项目进度、成本偏差(SV%,CV%)趋势图

图 4.10 可以看到项目进度、成本性能指标趋势。

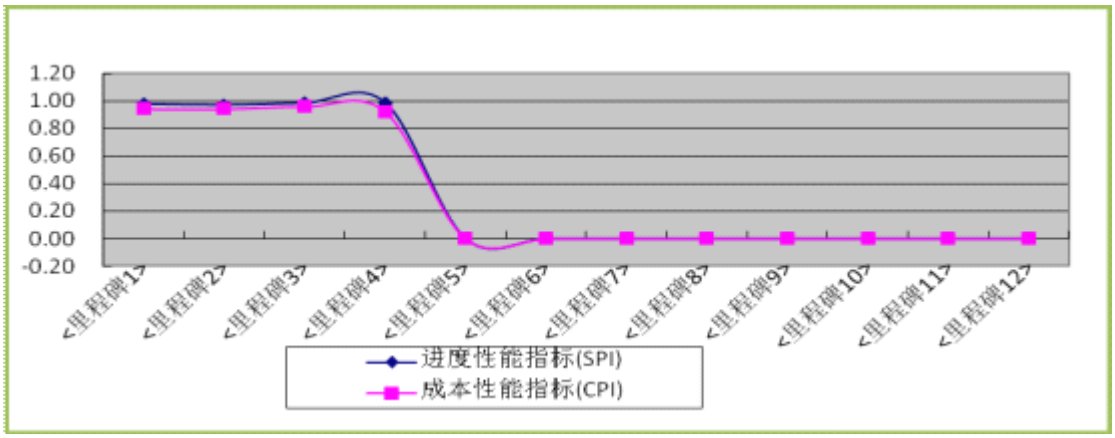

图 4.10 项目进度、成本性能指标(SPI,CPI)趋势图

4.3.2 模型数据对比

表 4.18 显示了项目中的实际值和估算值的对比。

表 4.18 估算结果对比

	实际值	原 COCOMO II 模型	改进 COCOMO II 模型
规模（SLOC）	31076	28050	30870
工作量(人时)	13764	11067	12678
进度（月）	10.3	9	9.7

表 4.19 显示了改进前后模型的工作量估算误差对比。

表 4.19 工作量估算误差对比

	原 COCOMO II 模型	改进 COCOMO II 模型
差额统计(VAF)	85.2	98.94
相对误差绝对值的均值(MARE)	89.151	98.94
相对误差的方差 (VARE)	8.151	2.4

通过对比，改进的估算模型，估算结果更为准确。

4.3.3 总结

新的模型相比于原模型，在项目估算中有更优秀的表现，但相比于实验结果，表现没有预期的好，原因是规模相比于实验项目较大，而参数没有相应的调整。所以新的模型要扩大其适应范围，还要加大不同规模和不同类型的项目实验，以得到更好的结果。

第五章　总结与展望

5.1 总结

软件成本估算是基于概率模型的估算，因此它不会产生精确值。但是，如果提供良好的历史数据和采用系统的技术，就可以得到更好的结果。模型的精确度是衡量其错误率的，可以实现尽可能的与实际值接近。

本文概述了软件估算的基本步骤，条件和使用的工具。COCOMO II 模型使得不仅容易得到项目的成本估算和工期的估算，而且还提供清晰，结构紧凑，简洁的方法，运用于整个项目的生命周期，从而减少项目风险，并为决策提供合理的理由。

软件估算虽然重要，但仅仅靠单独的软件估算过程是不够的，还要和项目管理的其他过程互相配合，才能得到真正有用的估算结果，并且在软件开发过程中和项目管理过程结合使用，才能发挥软件估算的作用。

本文通过对 COCOMO II 模型的详细分析研究，运用加权粒子群算法对 COCOMO II 模型进行了改进，再通过一个实例的应用，全面展示了改进 COCOMO II 模型的使用过程，以及在软件项目中的重要作用。但 COCOMO II 模型也不是万能的，工具再强大，也需要应对软件项目中会出现的种种可能，并且估算也只是一种科学的预测，并不是真实的结果。所以，对软件估算的理解才量关键，估算只是对工作人员提供参考。估算的首要目标不是预测项目的结果[11]，而是确定项目目标是否足够现实，从而让项目在可控的状态下达成这些目标。软件估算是一个动态的过程，要真正发挥估算的作用，还需要管理者在项目中结合项目的其他过程，让项目的过程可预测、可控。

5.2 展望

本文通过改进 COCOMO II 模型在实例中的应用，展示了好的软件估算对项目的巨大作用，它间接或直接地为项目完成了以下目标：项目近期完成，产品按合同交付，风险可能尽早的识别也规避，保证产品的质量，提高生产的效率，节约生产的成本，并且为其他部门如商务的谈判提供筹码等。

但我们在对软件估算作用的发挥和对 COCOMO II 模型的使用仍有许多地方需要更进一步发展。

1、软件估算是一个动态的过程，需要管理者重视这个过程，在项目的每个里程碑阶段和进行重大决策时，都要认识到这点。在项目中，由于项目的不同发展阶段，信息的逐步识别，各种不确定性因素，使的估算不能一次就得到准确的结果，估算是一个动态演进的过程。

2、COCOMO II 模型和其他估算模型和方法的结合使用。每个模型和方法的对软件项目的适用阶段不同，不同估算模型的结合使用使得估算更有效；另外不同估算模型的结合使用，如果几种估算之间的结果差异很大，这时我们会停一下思考为什么会有这样的结果，从而改正 COCOMO II 模型使用中的错误，或者注意到 COCOMO II 模型没有发现的问题。虽然这或许不是 COCOMO II 模型本身的错误，可能是使用者的疏忽，但我们的目的是为了得到准确的估算结果。

3、积累企业的经验库，基于历史数据和开发环境对 COCOMO II 模型进行校准。在组织拥有足够的历史数据时，基于本组织历史数据的估算是相对准确的估算。但是，当组织缺乏历史数据而又要选择和比较已有模型方法时，目前缺乏公认的公用数据集和评价标准的现状，使得组织很难做出正确的抉择。如何充分利用所有可能的已有信息、简单而准确地预见未来信息，无疑会对软件成本估算方法的下一步发展方向产生重要影响。

4、软件成本估算内容不断扩展。随着“软件变服务”的发展趋势, 从目前大家更多关注的直接开发成本，会逐步扩展到包括安装、培训、服务、设备、人员、维护等多种费用形式，甚至在某些场合下直接开发成本可以忽略不计。因此， 如何把握整体成本, 需要更系统化的估算视角、估算内容、规模度量方式以及规模与软件成本的关联等也会发生相应的变化。

5、对模型的改进。随着各种新技术的应用，软件开发的环境和过程也在逐渐发生变化，COCOMO II 模型也不能一成不变，在不同的企业，对不同的软件规模，其比例因子和工作量乘数也需要进行校正。

参考文献

[1] 周杰, 杜磊.COCOMO II 软件项目管理中的成本估算方法[J].计算机应用研究,2000,17(11):56-58.

[2] 李明树,何梅,杨达等.软件成本估算方法及应用[J] 软件学报,2007,4(18):775-795.

[3] John w. Bailey and victor R.Basili, (1981) “A meta model for software development resource expenditures”, Fifth International conference on software Engineering [J] CH-1627-9/81 /0000/ 0107500.75@ 1981 IEEE,PP 107-129,1981.

[4] The Standish Group.[R]CHAOS Report,1995. http://www. standishgroup.com.

[5] The Standish Group.[R]Research Report,2004 the 3rd Quarter. http://www.standishgroup.com;

[6] Kemerer CF. An empirical validation of software cost estimation models. Communications of the ACM, 1987, 30(5):417－429.

[7] Boehm BW. Understanding and controlling software costs.[J]IEEE Trans. on Software Engineering, 1988,14(10):1462－1477.

[8] Barry Boehm.Software engineering economics. [M]Prentice.Hall.1981.

[9] Boehm,B.W（美）.软件成本估算 COCOMO II 模型方法[M]李师贤等译,机械工业出版社 2005.

[10] Boehm BW, Clark B, Horowitz E, Westland C. Cost models for future software life cycle processes: COCOMO2.0 [J] Annals of Software Engineering,1995, 1:57−94.

[11] Steve McConnell.软件估算—“黑匣子”揭秘[M] 宋锐等译.电子工商出版社 2007.12.

[12] Steve McConnell. Software Estimation — Demystifying the Black Art.2007.

[13] 陈立. 多粒度面向对象软件估算模型的研究及应用[D]. 广州: 中山大学, 2008.

[14] 丁岳伟,马亦舟. 贝叶斯校正算法在软件估算模型 COCOMOII 中的应用[J]. 计算机应用与软件,2007,1(24):151-159.

[15] Lederer, Albert L. "Nine Management Guidelines for Better Cost Estimating,

“Communications of the ACM,[J].February 1992,pp.51-59.

[16] Fredrick P. Jr. The Mythical Man-Month; Essays on Software Engineering，Anniversary Edition(2dEd)[M]，Reading MA: Addison -Wesley. 1995.

[17] Boehm, Barry. Software Cost Estimation with Comoro II[M],Reading,MA: Addison- Wesley.2000.

[18] Barry Boehm．Software engineering economics[M].Prentice.Hall.1981.

[19] David Grams,David Herron.功能点风险---成功软件项目的测量实践[M],清华大学出版社,2003.12.

[20] Barry Boehm. Software Cost Estimation with COCOMO II [M].Prentice. Hall.2000.

[21] Frederick. Brooks.The My thickly Man-Month[J]．Addison Wesley’s.1995.

[22] 周海玲、孙涌,关于基本 COCOMO 模型参数校准的探讨[J],微电子学与计算机.2005.12.

[23] 覃征、杨利英、高勇民,软件项目管理[M],清华大学出版社,2004.7.

[24] C Behrens Measuring the Productivity of Computer Systems Development Activities with Function Points[J]．IEEE Transactions on Software Engineering，November 1983.

[25] J. Knuckler .A Cooperative Industry Study on Software Development/ Maintenance Productivity．Xerox Corporation [J].Xerox Square—XRX252A,Rochester,NY14644,Third Report, March 1985.

[26] Function Point Counting Practices：Manual Release 4.0,International Function Point Users Group,Blandon view Office Park,5008—28Pine Creek Drive,Westerville.OH4308I-4899.

[27] 周汉兵, 关 昕, 马 力. 功能点度量在软件开发中的应用[J]. 计算机工程与设计. 2006, 3(27): 525-529.

[28] Reifer 1997a.D.J.Reifer,Practical Software Reuse[M],John Whitley and Sons, 1997.

[29] M. Jorgensen. A review of studies on expert estimation of software development effort[J]. The Journal of Systems and Software, 2004, 70:37–60.

[30] 彭英. 软件成本估算模型的研究与应用[D]. 湖南:中南大学, 2007:05.

[31] 吉田宪正, 王蓉芝. 卫星指向误差的统计估算方法[J]. 控制工程,1986(02).

[32]杨平. 基于贝叶斯学习方法的软件成本估算模型研究[D]. 广东:中山大学, 2009: 05.

[33]蒋林峰，罗燕京. 基于用例分析技术的软件规模估算[J]南京大学学报, 2005, 10(41): 639-645.

[34]2004. 7. 李帜,林立新,曹亚波编著. 功能点分析方法与实践[M]. 北京：清华大学出版社, 2005,3.

[35](美)Craig Larman 著, 方梁等译. UML 和模式应用(原书第 2 版)[M]. 北京：机械工业出版社, 2005.6.

[36]卢有杰编著.现代项目管理学.北京:首都经济贸易大学出版社，2004:244.

[37]Wu F G,Lee Y J, Lin M C. Using the fuzzy analytic hierarchy process on optimum spatial allocation[J]. International Journal of Industrial Ergonomics, 2004, 33(5):553-569.

[38]毋国庆等.软件需求工程[M].北京:机械工业出版社,2008.

[39]Pressman,R.S.著.郑人杰等译.软件工程:实践者的研究方法 [M].北京:机械工业出版社,2007.

[40]Pinheiro da Silva P，Tony Griffiths，Norman W. Paton. Generating User Interface Code in a Model Based User Interface Development Environment[C].Proceeding of Advanced Visual Interfaces 2000, 2000: 155-160.

[41]微软亚洲研究院著.软件开发的科学与艺术[M].北京:电子工业出版社,2002.

[42]Martin Fowler.UML 精粹(第三版)[M].北京:清华大学出版社,2005.

[43]Van Welie, M., van der Veer, G & Eliens, A(2000), Patterns as Tools for User Interface Design, in' International Workshop on Tools for Worrking with Guidelines',Biarritz,France, PP.313-324.

[44]孙士兵,软件成本估算中的稳健回归算法研究与应用[D],湖南大学,2007(12).

[45]Wu Chong,GUO Ying-jian,XIAHan.The Model of Credit Risk Assessment in Commercial Banks on Fuzzy Integral Support Vector Machines Ensemble [J].Operations Research and Mangement Science. Ment Science.2009, (2):115-119.

[46]Ruhe M, Jeffery R, Wieczorek I. Cost estimation for Web application. In: Proc. of the 25th Int'l Conf. on Software Engineering. IEEE CS Press, 2003. 270−279.

致谢

三年的硕士学习生活即将结束，回顾我研究生学习的这三年，首先要感谢我的导师…教授。从论文的选题、拟纲、撰写到完成，都多次受到老师悉心关注和指导。在此我深深感谢导师三年来对学生的关心、信任、教导、理解、宽容、爱护和指导，学生将永远铭记于心！…一丝不苟的工作精神，对工作的高度热爱，精深的专业水平、渊博的知识、和蔼幽默的为人都给我留下了极为深刻的印象。这一切使我终生受益，并不断激励我走向更高的人生目标。在此我向尊敬的老师表示衷心的感谢。

同时，也深深感谢我的家人，以及给我很多帮助的同学、朋友们。

攻读硕士期间发表论文

[1]姚尔果, 闫秋粉, 南振岐, 薛小虎. 基于改进粒子群算法的 BP 神经网络模型研究[J]. 佳木斯大学学报（自然科学版）, 2012, 1 (30): 702-704.

[2]闫秋粉, 南振岐, 姚尔果, 薛小虎. 软件风险评估量化分析研究[J]. 计算机工程与设计, 2012, 4 (33): 1581-1585.

攻读硕士期间参与科研项目

[1]2009.7-2010.8 南特数码公司《软件能力成熟度模型三级认证 CMMI3(Capability Maturity Model Integration)》软件行业国际标准培训认证的过程改进组(EPG)核心成员，主要担任配置管理关键过程域。

[2]2010.1-2010.6 南特数码公司《南特 OA 协同办公系统》研发者，主要担任概要设计和配置管理工作。