Software Project Management (5)

Du Yugen

ygdu@sei.ecnu.edu.cn

Chapter 5 Software Effort Estimation
软件工作量估算

Roger Du@SEI of ECNU

OBJECTIVES

When you have completed this chapter you will be able to:

Avoid the dangers of unrealistic estimates;

Understand the range of estimating methods that can be used;

Estimate projects using a bottom-up approach;

Count the function points(功能点) and object points(对象点) for a system;

Estimate the effort needed to implement software using a procedural programming language;

Understand the COCOMO approach to developing effort models.

5.1Introduction

有些估算做得很仔细，而有些却只是凭直觉的猜测。大多数项目超过估算进度的25%到100%，但也有少数一些组织的进度估算精确到了10%以内，能控制在5%以内的还没有听说。

——Jones, 1994

软件工作量估算

“大多数IS人士，无论是否为管理者，从来都无权控制他们自己的进度计划。进度计划通常由市场部或高层管理部门直接下达，就像飞石从天而降（也有人称之为鸟粪）”

“就此问题，我曾与IS领域中许多人士进行过交流。大家一致认为当前IS领域面临的最大难题，既不是掌握快速更新的技术，也不是探求新型的管理哲学，而是被迫接受根本无法达到的进度计划。” **(Robert.L.Glass)**

你当初计划10万元造的房屋可能最终的实际造价为50万元。

从造房子中学到的

- **除非你确切知道“它”是什么？否则无法说明它的确切花费。**
- **盖房子时，可以盖梦想中的房子（不考虑花费），也可以按估算盖，但是功能必须具有一定的灵活性**

不确定性问题

- 客户会要求X功能吗?
- 客户要的是X功能的便宜版本还是昂贵版本呢？同一功能的不同版本的实施难度至少有10%左右的差别。
- 如果实施了X功能的便宜版本，客户会不会以后又想要昂贵的版本。
- X功能如何设计？同一功能的不同设计，在复杂度方面会有10%左右的差别。
- X功能的质量级别是什么？依据实施过程的不同，首次提交的X功能的缺陷数量会有10%的差异。
- 调试和纠正X功能实施过程中的错误要花多少时间？研究发现调试和纠正同样的错误，不同程序员所花时间会有10%左右的差异。
- 把X功能和其它功能结合起来要花多少时间?
- ……

软件工作量估算的渐进性

阶段	工作量和规模		工作量和规模	
	乐观	悲观	乐观	悲观
初始产品定义	0.25	4.0	0.60	1.60
批准的产品定义	0.50	2.0	0.80	1.25
需求说明书	0.67	1.5	0.85	1.15
产品设计说明书	0.80	1.25	0.90	1.10
详细设计说明书	0.90	1.10	0.95	1.05
摘自 Cost Models for Future Life Cycle Process: COCOMO 2.0, 1995				

估算的准确性和精确性

- 准确（accuracy）是结果与目标之间有多近，用3代表圆周率比用4更准确
- 精确（precision）是结果有多少有意义的位数，3.14比3代表圆周率更精确
- 一个结果可以不准确而精确，不精确而准确，
- 软件估算中错误的精确是准确的敌人，40～70个人月的工作量估算可能是最准确又最精确的估算，而精确到55个人月看起来更精确，但不准确。

软件工作量估算困难的原因

- 估算困难是由于软件的本质带来的，特别是其复杂性和不可见性。
- 软件开发是人力密集型工作的，因而不能以机械的观点来看待
- 传统的工程项目经常会以相近的项目做参考，不同的只是客户和地点，而绝大部分软件项目是独一无二的。
- 新技术的不断出现和应用。
- 缺少同类项目经验数据，许多组织无法提供原有项目数据，而即使提供了这些项目数据，也未必非常有用。

例子

Table 5.1 *Some project data – effort in work months (as percentage of total effort in brackets)*

	Design		Coding		Testing		Total	
Project	*wm*	*(%)*	*wm*	*(%)*	*wm*	*(%)*	*wm*	*SLOC*
a	3.9	(23)	5.3	(32)	7.4	(44)	16.7	6050
b	2.7	(12)	13.4	(59)	6.5	(26)	22.6	8363
c	3.5	(11)	26.8	(83)	1.9	(6)	32.2	13334
d	0.8	(21)	2.4	(62)	0.7	(18)	3.9	5942
e	1.8	(10)	7.7	(44)	7.8	(45)	17.3	3315
f	19.0	(28)	29.7	(44)	19.0	(28)	67.7	38988
g	2.1	(21)	7.4	(74)	0.5	(5)	10.1	38614
h	1.3	(7)	12.7	(66)	5.3	(27)	19.3	12762
i	8.5	(14)	22.7	(38)	28.2	(47)	59.5	26500

Table F.4 *Productivity rates*

Project	*Work-months*	*SLOC*	*Productivity (SLOC/month)*
a	16.7	6050	362
b	22.6	8363	370
c	32.2	13334	414
d	3.9	5942	1524
e	17.3	3315	192
f	67.7	38988	576
g	10.1	38614	3823
h	19.3	12762	661
i	59.5	26500	445
Overall	249.3	153868	617

Table F.5 *Estimated effort*

Project	*Estimated work-months*	*Actual*	*Difference*
a	6050/617 = 9.80	16.7	6.90
d	5942/617 = 9.63	3.9	−5.73

结论：很难用这些数据去估算项目

工作量估算的其它困难

- 某些人试图建立一个过去项目的全软件业的数据库，但是许多词汇意义的不明确使得这种努力没有效果，例如“测试”阶段究竟包括哪些活动就不明确。
- **估计的主观性：**人们容易低估小项目的工作量，而过分夸大大项目的工作量
- **估计的政治因素：**不同的人有不同的目标，如项目经理会高估项目工作量，许多机构采用独立的估算小组，但是将项目经理和项目成员吸收进估算小组，能够增强他们的责任感。

5.2何时需要度量

- 战略策划：选择合适的项目
- 可行性分析
- 系统规格：实现各个需求的工作量需要被衡量
- 评估供应商的建议
- 项目策划：
  - 项目进行过程中，估算越来越准确
  - 在项目开始阶段考虑的是用户需求，不考虑实现，但是为了估算，有时需要考虑一些实现方法

5.3过高估计和过低估计的问题

- **过高估计的问题**
  - **Parkinson法则：工作总是用完有效时间,即给的时间越多，工作花费的时间也越多**
  - **Brook法则：当人数增加后，项目所需的工作量将不成比例的增加。当团队规模变大后，由于管理，协调和通信的增加，将造成工作量的增加。因而“投入更多的人将使延期的工作更加延期”**
- **过低估计的问题**
  - **质量降低**
  - **Weinberg的可靠性零法则“如果系统不必可靠，那么它可以满足任何目标”。**

工作量估算对职员的影响

- 如果职员能够完成目标，那么他们将受到鼓舞
- 如果他们发现目标根本不能完成，那么他们的激情将受到极大损害
- 因而，估计不是一种简单的预测行为，而是一种管理目标

5.4软件估算的基础（1）

- **历史数据的需要**
  - **在参考历史数据时需要考虑不同的环境，如编程语言，软件工具，标准和人员的经验。**
- **工作度量**
  - **直接计算真正的成本或时间是不可能的。编写程序的时间不同的人将有显著的区别。**
  - **通常将工作量表达为工作量，如源代码的数量（source line of code，SLOC），或者千行代码量（KLOC）**

软件估算的基础（2）

- **复杂性**
  - **相同KLOC的两个程序花费的时间将会不同。因而不能简单地应用KLOC或SLOC，而要根据复杂性进行修正，但是复杂性的度量通常是主观而定的。**

基于承诺的估计

- 一些组织直接从需求出发安排进度而不进行中间的工作量估算。他们要求每个开发者作出进度承诺而非进度估算。
- 有利于开发者对进度的关注，开发者在接受承诺后士气高昂，自愿加班加点
- 问题在于开发者的估算比现实要乐观，大约低20至30个百分点（Van Genuchten, 1991）
- 承诺应该现实可行，以使你的团队会不断成功而不是不断失败。

5.5软件工作量估计技术

- 算法模型algorithmic models:用工作驱动因子(代表目标系统和实现环境的特征)来预测工作量
- 专家判断expert judgement
- 类比法analogy
- Parkinson：标识有效的能做成项目的人力并依此当作估算
- 赢的价格：这里估计是足够低的能赢得合同的价格
- 自顶向下：首先定义整个项目的工作量，然后分解到各个部分
- 自底向上：各个部分的工作量先估算出来，然后进行合成

5.5.1自底向上方法

- 该方法首先将项目分成部件任务，然后估算每个任务所需的工作量。
- 在大型的项目中，分解任务的过程是一个叠代的过程，直到最下面的任务不可分解，产生WBS。
- 该方法适合于项目规划的后期。如果应用在前期，那么必须对最终的系统作出一些假设，例如对软件模块的数量和大小进行假设。
- 如果项目是全新的或者没有历史数据，建议用该方法

练习

- 工资系统已经被安装在**Brightmouth**学院，目前有一个新的需求，需要在系统中添加一个子系统，该系统分析每节课时老师的成本。每个老师的工资可以从系统中获得，每个老师花在每个课程上的时间也可以从系统中获得。为了实现该系统，需要哪些任务，哪些任务的工作量比较难计算。

练习

- 答案
  - 获取用户需求
  - 分析系统中已有数据
  - 设计报表和编写用户建议
  - 编写测试计划
  - 编写技术描述
  - 设计软件
  - 写软件
  - 测试软件
  - 写说明书
  - 执行接受测试
- 设计，写，测试软件将最难估算工作量

5.5.2自顶向下方法和参数模型

- **自顶向下的方法和参数模型相关**
- **一般采用对比方法确定总的工作量**
- **对比是建立在一系列参数的基础上的，通过这些参数可以计算出新系统的工作量**
- **参数模型的形式：**

  effort=（系统规模）*（生产率）

  - 例如系统规模可以用**KLOC**来计算，生产率以**40**天**/KLOC**
  - 预测软件开发工作量的模型有两个部分，第一部分为估算软件大小，第二部分为估算工作效率

练习

- **学生要求每学期写一篇有关IT的报告，如果你想建立一个估算学生完成这样一份报告的模型，你用什么来衡量报告的大小，什么因素会影响学生完成报告的难度?**

参考答案

最明显的工作量驱动因子是:

- **字数**

难度因子可能包括:

- **材料的可获得性**
- **学生对主题的熟悉程度**
- **需要的宽度/深度**
- **技术难度**

5.6专家判断

- 具有应用领域或者开发环境知识的人员对任务的评估
- 该方法特别是在对原由系统进行替换时有用，评估者对影响的代码的比例进行分析，从而得到工作量评估。

5.7类比估计

- 类比方法又被称为基于案例的推理（Case-based reasoning）
- 评估者寻找已经完成的项目，这些项目与需要开发的新项目在许多特征上必须是类似的。
- 如何选择与待预测的项目相近的项目？
  - 欧几里德距离（Euclidean Distance）公式
  - distance=（（目标参数1-原参数1）^{2}+（目标参数2-原参数2）^{2}+.....）的平方根

练习

- 假定要匹配的按理基于两个参数,构造系统的输入和输出参数。新的系统有7个输入，15个输出，过去有一个项目A有8个输入，17个输出，这两个项目的欧几里的距离是多少?
- 答案：$\sqrt{(7-8)^2+(15-17)^2}=2.24$

5.8Albrecht功能点分析function point analysis

- 该方法是由Allan Albrecht在IBM工作时发明的自顶向下方法。
- 功能点法（Function Point, FP）的基本点是计算机信息系统包括五个主要部件或者使用户受益的外部用户类型，它们是：
  - 外部输入类型：指输入事务(更新内部计算机文件)
  - 外部输出：提供给用户的面向应用的信息
  - 内部逻辑文件：逻辑主文件
  - 外部接口文件：与其它系统交换信息
  - 外部查询：在线的输入以获得立即的结果

功能点方法

- **加权因子的确定**

Table 5.2 *Albrecht complexity multipliers*

External user type	*Multiplier*		
	Low	*Average*	*High*
External input type	3	4	6
External output type	4	5	7
Logical internal file type	7	10	15
External interface file type	5	7	10
External inquiry type	3	4	6

练习

- 在学院工资系统项目中需要开发一个程序，该程序将从会计系统中提取每年的工资额，并从两个文件中分别提取课程情况和每个老师教的每一门课的时间，该程序将计算每一门课的老师的成本并将结果存成一个文件，该文件可以输出给会计系统，同时该程序也将产生一个报表，以显示对于每一门课，每个老师教学的时间和这些工时的成本。
- 假定报表是具有高度复杂性的，其它具有一般复杂性

练习

- 外部输入：无
- 外部输出：报告，1
- 内部逻辑文件：财务输入文件，1
- 外部接口文件：工资文件，人员文件，课程文件，财务输入文件，4
- 外部查询：无

- 考虑加权：
  - 外部输入：无；外部输出：1×7＝7；内部逻辑文件：1×10＝10；外部接口文件：4×7＝28；外部查询：无；共：45

功能点方法：复杂性判定

- **如何判定功能的复杂性?**
- **国际功能点用户小组（IFPUG)**
  - **内部逻辑文件、外部接口文件**

Table 5.3 *IFPUG file type complexity*

Number of record types	*Number of data types*		
	<20	*20 to 50*	*>50*
1	low	low	average
2 to 5	low	average	high
> 5	average	high	high

  - **外部输入文件**

Table 5.4 *IFPUG External input complexity*

Number of file types accessed	*Number of data types accessed*		
	<5	*5 to 15*	*>15*
0 or 1	low	low	average
2	low	average	high
> 2	average	high	high

功能点方法：复杂性判定

- 外部输出文件

Table 5.5 *IFPUG External output complexity*

Number of file types	*Number of data types*		
	<6	*6 to 19*	*>19*
0 or 1	low	low	average
2 or 3	low	average	high
> 3	average	high	high

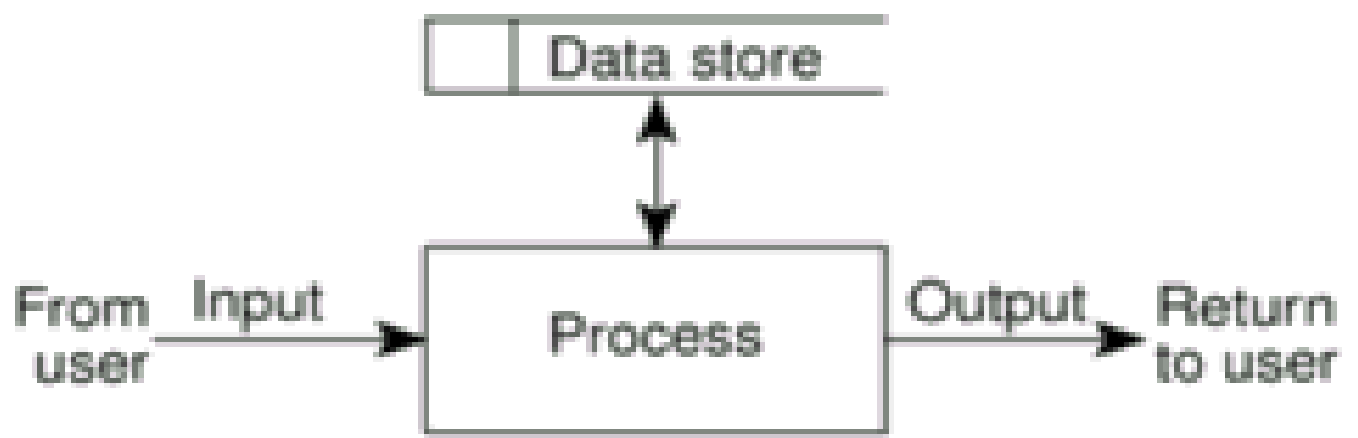
- 如何确定记录个数和数据个数
  - 如某系统内部逻辑文件：订单文件，包含订单信息（包括订单号，供应商名称，订单日期）和订单项（包括商品号，价格和数目），则记录个数为2，数据个数为6，在表中可以确定该功能点复杂性为低。

功能点方法：转换为代码行

- **通过定义各个功能点对应各种语言的代码行数，则功能点可以转化为代码行**
- **一些数据：**
  - **Cobol: 91**
  - **C: 128**
  - **Quick Basic: 64**
  - **Object Oriented Languages: 30**

5.9MarkII功能点

- 该方法被作为英国政府项目实施中采用的标准
- 基本原理：对于一个处理事务



- 计算方法：w_i×输入数据元素类型数 + w_e×引用的实体类型数 + w_o×输出数据元素类型数
- 系数总和为2.5，标准设置为w_i=0.58, w_e= 1.66, w_o=0.26

MarkII功能点

- 系数调整，考虑因素：
  - 与其它应用的接口
  - 特殊的安全特征
  - 与第三方的直接交互
  - 用户训练特征
  - 文档需求

功能点的其它扩展

- 功能点方法起源于业务信息系统应用，因而强调了数据方面的因素而没有考虑功能和行为（控制）方面的因素。
- 特征点（Feature Points):除了考虑普通功能点的内容外，还考虑了算法的特征（矩阵转换，字符串解析，处理中断等都是算法的例子）
- Boeing提出了一个三维功能点方法（3D）其中三维为数据维，功能维（输入转化为输出的步骤）和控制维（状态之间的转换数）。

功能点转化为工作量

- **对于原来的项目，计算生产率：**
  - **生产率 = 功能点数目/工作量（人日）**
  - **则，对于新项目，功能点计算出来后，工作量为：**
  - **工作量 = 功能点数目/生产率**
- **更复杂的方法：最小二乘法**
- **即工作量 = 系数1 + 功能点数×系数2**

5.10对象点

- Object Points起源于纽约大学的Leonard N.Stern商学院，它类似于功能点方法，但是更容易计算。
- 对象点方法与面向对象方法并无直接联系。
- 该方法计算应用所需要处理的屏幕，报告和部件，这些都被称为对象。每一对象需要被确定为简单的，中等的，困难的三个层次。

对象点方法

Table 5.6 *Object Points for screens*

Number of views contained	*Number and source of data tables*		
	Total < 4 (<2 servers <3 clients)	*Total < 8 (<3 servers 3 to 5 clients)*	*Total > 7 (>3 servers > 5 clients)*
< 3	simple	simple	medium
3 to 7	simple	medium	difficult
> 7	medium	difficult	difficult

Table 5.7 *Object Points for reports*

Number of sections contained	*Number and source of data tables*		
	Total < 4 (<2 servers <3 clients)	*Total < 8 (<3 servers 3 to 5 clients)*	*Total > 7 (>3 servers > 5 clients)*
< 2	simple	simple	medium
2 or 3	simple	medium	difficult
> 3	medium	difficult	difficult

Table 5.8 *Object Points complexity weightings*

Object type	*Complexity weighting*		
	Simple	*Medium*	*Difficult*
Screen	1	2	3
Report	2	5	8
3GL component	—	—	10

对象点转换为工作量

- 首先考虑已经存在的对象应该排除在工作量计算内。即计算新的对象点（NOP）
- 根据原来从事过的项目计算在不同情况下的项目的生产率，例如下表：

Table 5.9 *Object point effort conversion*

Developer's experience and capability/ICASE maturity and capability	*Very low*	*Low*	*Nominal*	*High*	*Very high*
PROD	4	7	13	25	50

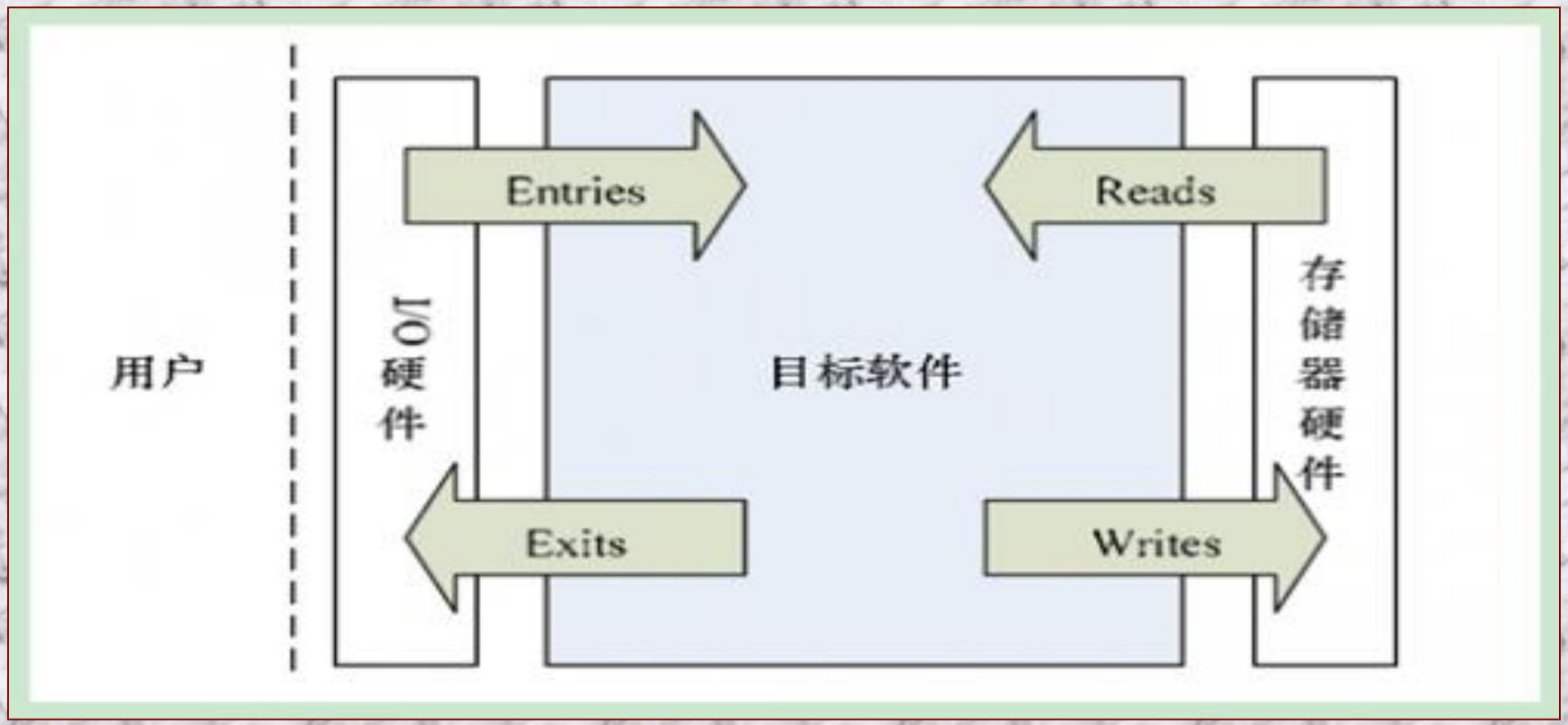
- 假定有672个对象点要开发，开发者的经验和工具使用都是正常性的，则需要672/13 = 52个月

5.11COSMIC-FFP全功能点

- 通用软件度量国际协会（Common Software Measurement International Consortium，COSMIC)提出的全功能点分析方法（COSMIC-FFP）
- COSMIC-FFP 通过入口(Entry)、出口(Exit)、读取(Read)和写入(Write)4 种数据流类型来确定软件系统功能的规模。它的度量标准单位是CFP(COSMIC Function Point)，等同于一个数据流。

模型数据移动的4 种类型

模型把数据移动分为4 种类型：Entry(将数据组从用户移到目标软件)，Exit(将数据组从目标软件移到用户)，Read(将数据组从存储设备移到目标软件)和Write(将数据组移到存储设备)。其中，Entry 和Exit 属于用户接口，Write 和Read 属于存储器接口。每个有效的数据传送都被看成一个CFP



COSMIC-FFP主要适用于的领域

在COSMIC-FFP中，将系统的功能处理分解为“数据计算”和“数据移动”2种类型，该方法只统计了“数据移动”的个数，没有对“数据计算”进行度量，所以，COSMIC-FFP方法主要适用于如下的领域：

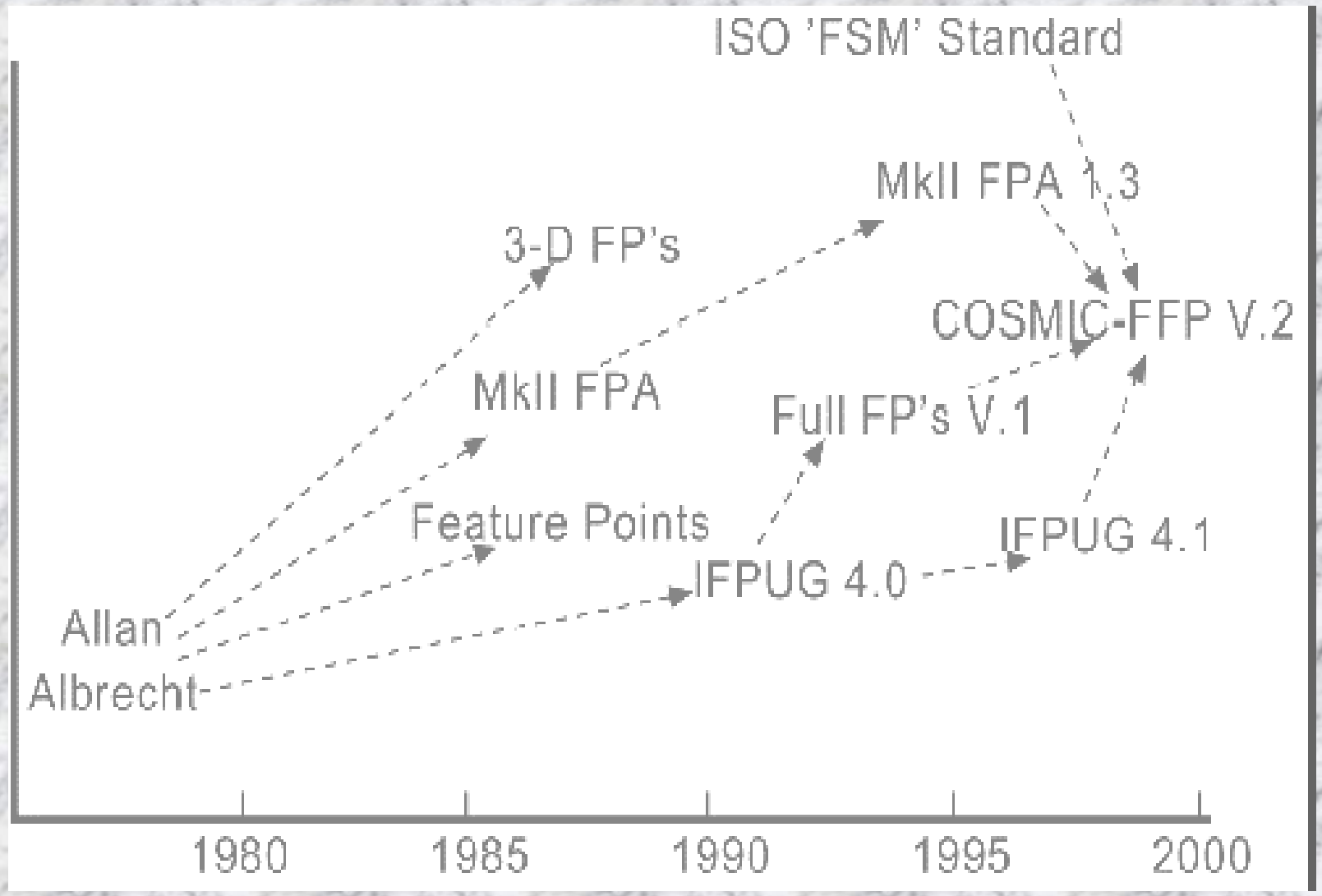
- 以数据处理为主的商务应用软件，如银行、财务、保险、个人、采购、分销、制造等领域的信息系统；
- 实时系统，如电话交换系统、嵌入式控制软件（家电中的控制软件、汽车中的控制软件、过程控制中的自动数据采集系统等）；
- 上述两种类型的混合，如飞机售票系统、旅馆预订系统等。

该方法不适合于复杂算法的系统与处理连续变量的系统，如：专家系统、模拟系统、自学习系统、天气预报系统、声音和图象处理系统等。

功能点标准总结

- 现在世界上有 5 个功能点的ISO标准,分别是：
- IFPUG（澳大利亚，最大的功能点组织）
- NESMA（荷兰，世界第二大功能点组织，代表欧洲）
- MarkII
- COSMIC（能用到嵌入式）
- FiSMA（芬兰）。
- 功能点有各种简化方法，但基本上都“与功能点兼容”，就是说借用了功能点的定义和数值。

功能点方法的发展历程


ISO 'FSM' Standard
MkII FPA 1.3
3-D FP's
COSMIC-FFP V.2
MkII FPA
Full FP's V.1
Feature Points
IFPUG 4.1
IFPUG 4.0
Allan
Albrecht
1980
1985
1990
1995
2000


IFPUG: International Function Point Users Group - Microsoft Internet Explorer
文件(F) 编辑(E) 查看(V) 收藏(A) 工具(T) 帮助(H)
后退 前进 停止 刷新 主页 搜索 收藏 历史 邮件 打印 编辑
地址(D) http://www.ifpug.org/
转到
链接 Internet 起始页 Microsoft Windows Update Windows 免费的HotMail 频道指南 自定义链接 最佳 Web 页
IFPUG
INTERNATIONAL FUNCTION POINT USERS GROUP
191 Clarksville Road
Princeton Junction, NJ 08550
P 609.799.4900
F 609.799.7032
A WORLD O
ABOUT IFPUG
CHAPTERS
EDUCATIONAL SERVICES & WORKSHOPS
CONFERENCES
CERTIFICATION
MEMBERSHIP INFORMATION
PUBLICATIONS & PRODUCTS
ADVERTISERS / VENDORS / RELATED SITES
MEMBERS ONLY
BULLETIN BOARD
CONTACT US
PUBLIC CFPS SEARCH
Site Map | April 30, 2002
What's New at IFPUG - Microsoft Internet Explorer
WHAT'S NEW ON THE IFPUG WEB SITE
New White Paper - Counting Third Party Software (Members Only)
ISO Press Release
Regional CFPS Exam Sites
Function Point WORKBENCH™ 5.0
CHARISMATEK Software Metrics
TRAINING - IFPUG Certified
click here to register for an online demo
Internet

COCOMO: 参数化模型

- COCOMO： Constructive Cost Model
- Boehm在二十世纪70年代采用他的模型对63个项目进行了研究，由于其中只有7个是商务系统，因而它们不仅仅能被用于信息系统。
- 基本的公式为： $Effort=c\times size^{k}$
  - 其中effort采用“人月(152个工作小时）”pm来度量，size采用kdsi即千行交付源代码指令(thousands of delivered source code instructions)

COCOMO系数

- c,k的取值根据系统的分类而定：
  - 根据系统的技术特性和开发环境可以分为：
  - 有机模式（organic mode): 相对小的团队在一个高度熟悉的内部环境中开发规模较小，接口需求较灵活的系统。
  - 嵌入式模式（Embedded Mode)开发的产品在高度约束的条件下进行，对系统改变的成本很高。
  - 半分离模式（Semi-detached Mode)上面两者之间
  - 信息系统是有机模式，而实时系统是嵌入式模式。

COCOMO系数

- 系数表：

Table 5.10 *COCOMO constants*

System type	*c*	*k*
Organic	2.4	1.05
Semi-detached	3.0	1.12
Embedded	3.6	1.20

- k的值反映了项目越大，则工作量成指数增加，因为大项目需要更多的协调和安排。

Table F.6 shows a comparison of the actual work-months from Table 5.1 and the COCOMO estimates. It illustrates the need for the calibration of COCOMO to suit local conditions.

Table F.6 *Comparison of COCOMO estimates and actual effort*

SLOC	*Actual (work-months)*	*COCOMO estimates*	*Difference (work months)*	*Difference (%)*
6050	16.7	15.9	–0.8	–4.9
8363	22.6	22.3	–0.3	–1.2
13334	32.2	36.4	4.2	13.1
5942	3.9	15.6	11.7	299.7
3315	17.3	8.4	–8.9	–51.2
38988	67.7	112.4	44.7	66.0
38614	10.1	111.2	101.1	1001.5
12762	19.3	34.8	15.5	80.2
26500	59.5	74.9	15.4	25.9

COCOMO修正

- 事实上，基本COCOMO模型对工作量的衡量不稳定，Boehm本人也发现了此问题，因而提出名义工作量(nominal effort)估算的概念。
- 首先从基本模型得到名义工作量pm_{nom}估计，然后采用开发工作量乘法算子（development effort multiplier, dem)进行修正，即：

$$pm_{est}=pm_{nom}\times dem$$

COCOMO成本因素

- dem的计算

Table 5.11 *COCOMO81 intermediate cost drivers*

Driver type	*Code*	*Cost driver*
Product attributes	RELY	required software reliability
	DATA	database size
	CPLX	product complexity
Computer attributes	TIME	execution time constraints
	STOR	main storage constraints
	VIRT	virtual machine volatility – degree to which the operating system changes
	TURN	computer turn around time
Personnel attributes	ACAP	analyst capability
	AEXP	application experience
	PCAP	programmer capability
	VEXP	virtual machine (i.e. operating system) experience
	LEXP	programming language experience
Project attributes	MODP	use of modern programming practices
	TOOL	use of software tools
	SCED	required development schedule.

Very low	1.46
Low	1.19
Nominal	1.00
High	0.80
Very High	0.71

练习

- **在某企业中，绝大多数系统技术上，产品，计算机和项目等属性都是类似的。只有人员的属性有所差异。该企业制定了下表：**

Personnel attributes	ACAP	analyst capability
	AEXP	application experience
	PCAP	programmer capability
	VEXP	virtual machine (i.e. operating system) experience
	LEXP	programming language experience

Attribute	*Very low*	*Low*	*Nominal*	*High*	*Very high*
ACAP	1.46	1.19	1.00	0.86	0.71
AEXP	1.29	1.13	1.00	0.91	0.82
PCAP	1.42	1.17	1.00	0.80.	0.70
VEXP	1.21	1.10	1.00	0.90	—
LEXP	1.14	1.07	1.00	0.95	—

- **分析员非常优秀，编程人员也很优秀但是对该项目面向的领域不熟悉并准备用新的编程语言。他们对操作系统很熟悉。请计算dem。如果名义工作量是4人月，则估算的工作量是多少?**

练习

Table F.7 *Calculating the development multiplier*

Factor	*Rating*	*Multiplier*
ACAP	very high	0.71
AEXP	low	1.13
PCAP	high	0.80
VEXP	high	0.90
LEXP	low	1.07

$dem = 0.71 \times 1.13 \times 0.80 \times 0.90 \times 1.07 = 0.62$

final estimate = 4 person-months × 0.62 = 2.48 staff months

COCOMOII

- 针对COCOMO的不足，Boehm开始和其合作者发展了新的模型。该方法利用多种乘法算子和指数。
- 一个明显的特点是该模型适应了项目在执行过程中变得越来越确定的状态，因而是一种渐进性评估。

COCOMOII

- COCOMOII被分为三个阶段性模型：
  - 应用构成阶段（Application Composition)：系统的外部特征被设计。在该阶段经常可以采用原型。
  - 早期设计（Early Design): 基本的软件结构被设计。
  - 构造阶段（Post architecture): 构造满足要求的系统。

COCOMOII

- 在应用构成阶段，采用对象点计算的方法
- 在早期设计阶段，采用功能点计算的方法。功能点可以转换为SLOC。
- $pm = A \times size^{sf} \times em_1 \times em_2 \times \ldots\ldots em_n$
- pm为“人月”工作量，A是一个常数，size以SLOC为单位，sf是指数比例因子。
- sf = 1.01+0.01×指数驱动因子的和

COCOMOII

- 计算规模因素的质量：
- 先前经验（Precedentedness): 是否有先前的经验
- 开发的灵活性（Development Flexibility): 是否需求能够以多种方式来满足；
- 体系结构/风险解决(Architecture/Risk Resolution)：是否方案已经被确定和解决的程度
- 团队的凝聚性（Team cohesion)
- 过程的成熟性（Process Maturity)

练习

- 对于某一个软件企业，一个新的项目的新颖性一般，因而在先前经验方面给3分，开发灵活性方面很低，因而给以0分，但是需求可能会变化得比较厉害，因而风险解决指数给4分，团队很融洽，给1分，但是过程不标准，因而过程成熟性给4分，请计算规模因素sf：

Table F.8 *Exponent drivers and ratings for the project*

Exponent driver	*rating*
Precedentedness	3
Development flexibility	0
Architecture/risk resolution	4
Team cohesion	1
Process maturity	4
Total	12

Therefore, $sf = 1.01 + 0.01 \times (3 + 0 + 4 + 1 + 4) = 1.13$

COCOMOII

- 计算工作量乘法算子em
- 类似于dem的计算
- 在不同的阶段有不同的em
- 如果每一项对于项目都取1

Table 5.12 *COCOMOII Early Design effort multipliers*

Code	*Effort modifier*
RCPX	Product reliability and complexity
RUSE	Required reusability
PDIF	Platform difficulty
PERS	Personnel capability
PREX	Personnel experience
FCIL	Facilities available
SCED	Schedule pressure

Table 5.13 *COCOMOII Post Architecture effort multipliers*

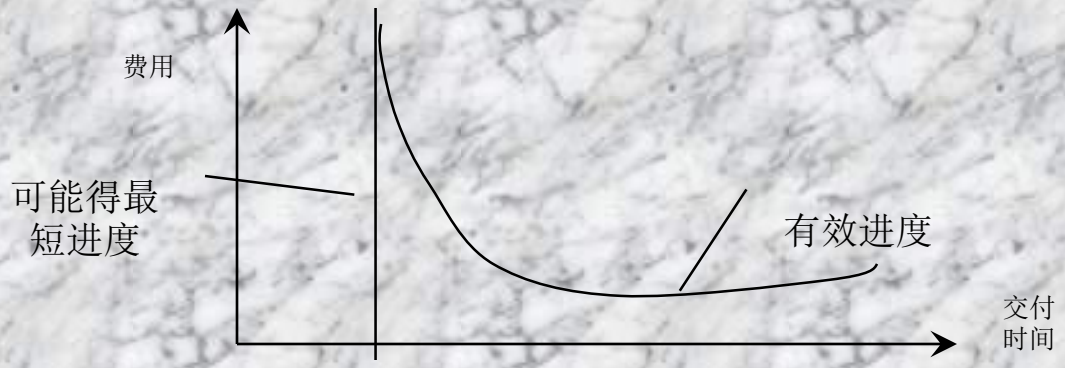
Modifier type	*Code*	*Effort modifier*
Product attributes	RELY	Required software reliability
	DATA	Database size
	DOCU	Documentation match to life-cycle needs
	CPLX	Product complexity
	REUSE	Required reusability
Platform attributes	TIME	Execution time constraint
	STOR	Main storage constraint
	PVOL	Platform volatility
Personnel attributes	ACAP	Analyst capabilities
	AEXP	Application experience
	PCAP	Programmer capabilities
	PEXP	Platform experience
	LEXP	Programming language experience
	PCON	Personnel continuity
Project attributes	TOOL	Use of software tools
	SITE	Multisite development

5.13大致进度估算方法

- **大致的（Bullpark）进度估算**
  - **软件估算方程和系数一定程度上比较难掌握**
  - **软件工作量估算软件比较昂贵**
  - **大致的估算简单宜行**
- **三个进度表**
  - **可能的最短进度**
  - **有效的进度**
  - **普通进度**

可能的最短进度

- 非常乐观的假设（员工10%顶尖人才，管理，工具，方法）
- 两个事实：
  - 存在一个最短的进度，而且不可能突破
    - 一个人5天内能写出1000行程序，5个人一天内能完成吗？40个人1小时内能完成吗?
  - 当你把进度缩短得比普通进度短时，费用将迅速上涨。



软件开发项目最短时间下限速查表

系统规模 (代码行数)	系统软件类产品 时间 (月)	系统软件类产品 工作量 (人月)	商务类软件产品 时间 (月)	商务类软件产品 工作量 (人月)	紧包装类产品 时间 (月)	紧包装类产品 工作量 (人月)
10,000	6	25	3.5	5	4.2	8
15,000	7	40	4.1	8	4.9	13
20,000	8	57	4.6	11	5.6	19
25,000	9	74	5.1	15	6	24
30,000	9	110	5.5	22	7	37
35,000	10	130	5.8	26	7	44
40,000	11	170	6	34	7	57
45,000	11	195	6	39	8	66
50,000	11	230	7	46	8	79
60,000	12	285	7	57	9	98
70,000	13	350	8	71	9	120
80,000	14	410	8	83	10	140
90,000	14	480	9	96	10	170
100,000	15	540	9	110	11	190
120,000	16	680	10	140	11	240
140,000	17	820	10	160	12	280
160,000	18	960	10	190	13	335
180,000	19	1,100	11	220	13	390
200,000	20	1,250	11	250	14	440
250,000	22	1,650	13	330	15	580
300,000	24	2,100	14	420	16	725
400,000	27	2,900	15	590	19	1,000
500,000	30	3,900	17	780	20	1,400

有效进度与普通进度

- **有效进度**
  - **假定**
    - **人才：顶尖得前25%得人才**
    - **人员调整每年小于6%**
  - **可能的最短进度与有效进度的关系**
    - **进度变长了，但是工作量也许减少了**
    - **压缩进度是要付出代价的**
- **普通进度**
- **（具体表格参考《快速软件开发》一书）**

估算修正

- 经理和客户常问的一个问题“如果我给你额外一个星期做估算，你能修正它以减少不确定性吗？”
- 答应这种要求最后将给自己带来麻烦，因为减少不确定性必须在软件开发过程本身中进行。
- 许多项目中要求给出单点估计，这种方法的结果是估算永远不准确
- 理智的方法是先给出大的区间，逐步缩小区间。

估算的再修正

- 假定有一个6个月的进度计划，计划4周内达到第一个里程碑，而实际上花了5周，如何修正进度：
  - 再后续进度中弥补损失的一周
  - 把这一周加到进度中
  - 整个进度乘以拖延的数量，本例乘以25%

估算的再修正

- **1991年对300多个项目的调查表明，项目几乎不能弥补损失的时间（Van Genuchten, 1991）**
- **第一阶段估算不准确的原因一般会分布在整个项目中**

估算的技巧

- 避免无准备的估算
  - 不要随口说出一个估算
- 留出估算的时间，并做好计划
  - 估算本身也是一个项目
- 开发人员参与估算
- 不要忽略普通任务
- 使用几种不同的估算技术，并比较它们的结果
- 估算的群体讨论

过于乐观的进度计划

- **Microsoft Word for Windows 1.0开发**
- **包含249，000行代码，投入660人月，前后历时5年，实际花费时间为预期时间的5**

提交报告日期	预期交付日期	预期到完成交付还需要的开发时间	实际到完成交付所需花费的开发时间	估算偏差率
84 年 9 月	85 年 9 月	365 天	1887 天	81%
85 年 6 月	86 年 7 月	395 天	1614 天	76%
86 年 1 月	86 年 11 月	304 天	1400 天	78%
86 年 6 月	87 年 6 月	334 天	1235 天	73%
87 年 1 月	87 年 12 月	334 天	1035 天	68%
……	……	……	……	……
89 年 6 月	89 年 9 月	92 天	153 天	40%
89 年 7 月	89 年 10 月	92 天	123 天	25%
89 年 8 月	89 年 11 月	92 天	92 天	0%
89 年 11 月	89 年 11 月	0	0	0%

过于乐观的进度计划

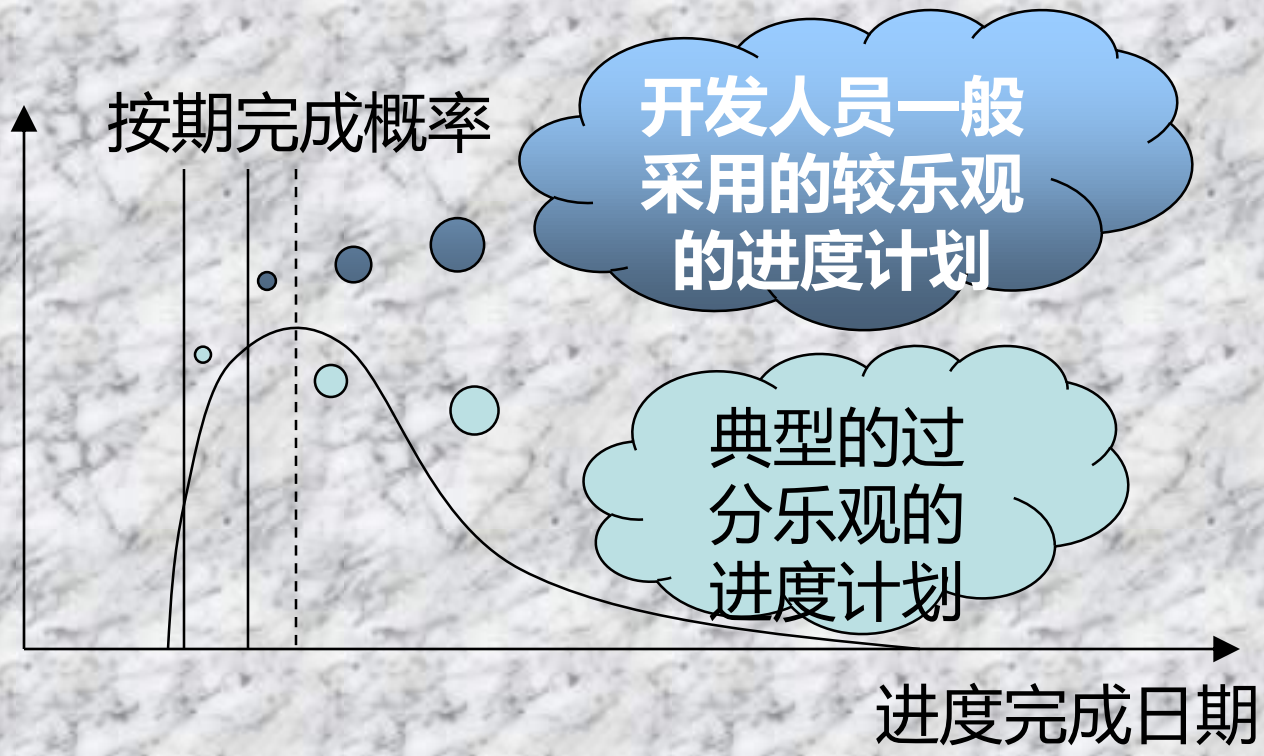
- **导致WinWord1.0开发延迟的几个主要因素：**
  - **项目初期制定的开发目标是不可实现的。盖茨下达的指示是用最快的速度开发最好的字处理软件，争取在12月内完成。实现这两个目标中的任何一个都是困难的，同时达到则是不可能的。**
  - **过紧的进度计划降低了计划的精确度。**
  - **开发过程中频繁换人。5年中共换了4个组长，其中有2人因进度压力离职，1人是出于健康的原因而离职。**
  - **迫于进度压力，开发人员匆忙写出一些低质量的和不完整的代码，然后宣称已实现某些性能。这造成了WinWord不得不将用于提高软件稳定性的时间由预计的3个月增加到12个月。**
- **由于该项目中，创新比速度更重要，因而试图缩短开发周期，反而使周期变长**

产生过于乐观的进度计划的根源

- 为了赶在某些特定时间前展示或出售产品。如计算机交易展示会。
- 管理人员和客户拒绝接受仅给出范围的估算，而是按照他们认为的“最佳”估算来制定进度计划。
- 项目管理人员和开发人员为了享受挑战的乐趣或在压力下工作的刺激，而故意缩短进度计划。
- 管理部门和市场部门为了迎合客户而缩短进度计划。
- 项目管理人员认为较紧的进度计划能够使员工勤奋工作。
- 原先的估计是合理的，但是在项目进行过程中又加进了很多新特性，从而变成了过于乐观的进度。
- ……

过于乐观的进度计划的后果

- **进度计划准确性**



过于乐观的进度计划的后果

- **计划的质量：错误的假设必将导致无效的项目规划。**
- **抛弃计划：当面临进度压力时，大多数开发组织会抛弃原有规划，而走入盲目开发的歧途。**
- **如果试图在关键阶段节省时间，必将在后续阶段加倍补偿。**
- **分散了管理者的注意力：管理者将分散经历在不断调整计划上。**
- **客户关系：项目一拖再拖，客户将失去耐心。**

过于乐观的进度计划的后果

- **仓卒收尾**
  - **要排错只能将系统拆分后再进行，一个小的变动要花很长时间。**
  - **开发人员清楚地知道系统中存在一些应作修改却未做的地方。**
  - **测试人员发现错误的速度大于开发人员排错的速度。**
  - **排除已发现错误的同时，产生了大量新的错误。**
  - **由于软件变化频繁，难以保证用户文档的同步更新。**
  - **项目估算多次调整，软件交付日期一拖再拖。**

超负荷的进度压力

- **产品质量的降低**
- **赌博心理导致风险管理的忽略**
- **激励效应不再起作用**
- **创造性思维受损**
- **精疲力竭：在后面的项目中难以提起热情**
- **中途退出：这些人通常是有能力的人**
- **长期地进行快速开发：难以补充新知识**
- **开发人员与管理人员的关系：不切实际的进度压力会使开发人员丧失对管理人员的尊重。**

有原则的谈判

- **该谈判的策略的中心是致力于探求一种双赢的解决方案。**
- **将人从困境中解脱**
  - **站在他人的立场上加以考虑（各人有各人的烦恼）**
  - **以合作的态度来努力改善与管理人员和客户的关系，制定比较现实的进度估算，尽量使各方都能理解进度估算的意义。**
  - **不要针锋相对**

有原则的谈判

- **关注共同利益，不要过分坚持立场**
  - **有时候必须作出让步**
  - **可以从下列几个方面加以说服：**
    - **真正提高开发速度**
    - **增加成功的机会**
    - **引用以前类似项目的失败教训**

有原则的谈判

- **提出多种备选方案**
  - **1、与产品有关的灵活变通：**
    - **将一些设计功能放到下一版本实现，大多数人在提出需求时，并不清楚这些需求是否必须全部在当前版本被满足。**
    - **分阶段交付产品。如版本0.7.0.8,0.9,1.0，每版实现最迫切的功能。**
    - **砍去某些实现起来费时或者需要谈判后才能确定的特性，包括与其它系统的整合能力。与旧版本兼容的能力，以及产品性能等。**
    - **对某些特性不必精雕细琢，只需实现到某种程度即可。**
    - **尽量轻松地实现各特性地详细功能需求。可以通过一些商业组件来实现。**

有原则的谈判

- 与项目资源有关的灵活变通
  - 如果处于项目初期，则增加更多的开发人员
  - 增加高层次的开放人员（如领域专家）
  - 增加更多的测试人员
  - 在管理方面给予更多的支持
  - 提高开发的支持力度，如更安静，更独立的工作间，速度更快的计算机，技术人员随时维护环境
  - 提高最终用户的参与度，最好在项目组中配备一个能够对特性设置最终拍板的用户
  - 提高主管人员的参与度

有原则的谈判

- 3.与进度计划有关的灵活变通
  - 在详细设计，产品设计，或至少是需求分析完成之前，只提出一个进度目标，但不设定一个确切期限
  - 如果是在项目初期，则在提炼产品概念，功能要求合详细设计时，可以探求缩短开发时间的方法
  - 先给出进度估算范围或大概的进度估算值，然后随着项目的进展逐步精确
- 4.其他
  - 为开发人员提供额外的支持，以保证他们能集中精力于项目的开发，例如购物服务，供餐，洗衣，清洗住所等
  - 采取更多的激励措施，如休假旅游，加班工资等

有原则的谈判

- **坚持客观标准**
  - **只能向原则低头，而不能向压力屈服**
  - **谈判不要局限于估算本身，即可以考虑项目的输入条件**
  - **有条件的话，可以由专业估算机构进行估算**
  - **坚持科学的估算过程**
  - **顶住压力**