软件项目管理

- 项目管理过程
- 软件生产率和质量的度量
- 软件项目的估算
- 软件项目计划的目标
- 软件开发成本估算

项目管理过程

- 软件项目管理的对象是软件工程项目。它所涉及的范围覆盖了整个软件工程过程。
- 为使软件项目开发获得成功，关键问题是必须对软件开发项目的工作范围、可能风险、需要资源（人、硬件／软件）、要实现的任务、经历的里程碑、花费工作量（成本）、进度安排等做到心中有数。

- 软件项目管理可以提供这些信息。
- 这种管理在技术工作开始之前就应开始，在软件从概念到实现的过程中继续进行，当软件工程过程最后结束时才终止。

启动一个软件项目

- 在制定软件项目计划之前，必须
  - 明确项目的目标和范围
  - 考虑候选的解决方案
  - 标明技术和管理上的要求
- 有了这些信息，才能确定合理、精确的成本估算，实际可行的任务分解以及可管理的进度安排。

- 软件人员和用户是在系统工程步骤中确定项目的目标和范围。
- 目标标明了软件项目的目的但不涉及如何去达到这些目的。
- 范围标明了软件要实现的基本功能，并尽量以定量的方式界定这些功能。
- 当明确了软件项目的目标和范围后，就应考虑候选的解决方案。

- 有了方案，管理人员和技术人员就能够据此选择一种“好的”方法，给出诸如交付期限、预算、个人能力、技术界面及其它许多因素所构成的限制。

度量

- 进行度量工作，是为了了解产品开发的技术过程和产品本身。
  - 度量开发过程的目的是为了改进过程，
  - 度量产品的目的是为了提高产品的质量。
- 度量的作用是为了有效地定量地进行管理。

- 为有效地度量，常常需要考虑：对于过程和产品，
  - 合适的度量是什么？
  - 所收集的数据如何使用？
  - 用于比较个人、过程或产品的度量是否合理？
- 管理人员和技术人员可利用这些度量来了解软件工程过程的实际情况和它所生产的产品质量 。

估算

- 在软件项目管理过程中关键的活动就是制定项目计划。
- 在做计划时必须就需要的人力（以人月为单位）、项目持续时间（以年份或月份为单位）、成本（以元为单位）做出估算。
- 这种估算大多是利用以前的花费做为参考而做出的。

- 如果新项目与以前的一个项目在大小上和功能上十分**类似**，则新项目需要工作量、开发持续时间、成本大致与那个老项目相同。
- 假使项目背景完全生疏，只凭过去的经验做出估算可能就不够了。
- 现在已有了许多用于软件开发的估算技术。其共同特点是：
  - 事先建立软件范围
  - 以软件度量（以往的度量）为基础，以做出估算
  - 项目被分解为可单独进行估算的小块
- 管理人员大多使用不止一种估算技术，并用一种估算技术做为另一种估算技术的交叉检查。

风险分析

- 每当新建一个程序时，总是存在某些不确定性。
  - 用户要求是否能确切地被理解？
  - 在项目最后结束之前要求实现的功能能否建立？
  - 是否存在目前仍未发现的技术难题？
  - 在项目出现严重误期时是否会发生一些变更？等等。

- 风险分析对于软件项目管理是决定性的，然而现在还有许多项目不考虑风险就着手进行。
- 所谓风险分析实际上就是一系列风险管理步骤，其中包括风险识别、风险估计、风险优化、风险管理策略、风险解决和风险监督。这些步骤贯穿在软件工程过程中。

进度安排

- 每一个软件项目都要求制定一个进度安排，但不是所有的进度都得一样安排。
- 对于进度安排，需要考虑的是：
  - 预先对进度如何计划？
  - 工作怎样就位？
  - 如何识别定义好的任务？
  - 管理人员对结束时间如何掌握，
  - 如何识别和监控关键路径以**确保结束**？
  - 对进展如何度量？
  - 如何建立分隔任务的里程碑。
- 软件项目的进度安排与任一个工程项目的进度安排基本相同。首先识别一组项目任务，再建立任务之间的相互关联，然后估算各个任务的工作量，分配人力和其它资源，制定进度时序。

追踪和控制

- 一旦建立了开发进度安排，就可以开始着手追踪和控制活动。
- 由项目管理人员负责追踪在进度安排中标明的每一个任务。
- 如果任务实际完成日期滞后于进度安排，则管理人员可以使用一种自动的项目进度安排工具来确定在项目的中间里程碑上进度误期所造成的影响。

- 还可对资源重新定向
- 对任务重新安排
- （做为最坏的结果）可以修改交付日期以调整已经暴露的问题。用这种方式可以较好地控制软件的开发。

软件生产率和质量的度量

- 生产率与质量的度量是以投入工作量为依据的软件开发活动的度量和开发成果质量的度量。
  - 为什么要对软件进行度量
  - 面向规模的度量
  - 面向功能的度量
  - 软件质量的度量
  - 在软件工程过程中使用度量

为什么要对软件进行度量

① 表明软件产品的质量；
② 弄清软件开发人员的生产率；
③ 给出使用了新的软件工程方法和工具所得到的（在生产率和质量两方面）的效益；
④ 建立项目估算的“基线”；
⑤ 帮助调整对新的工具和附加培训的要求。

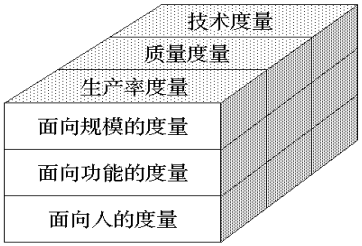
度量的方式

- 在物理世界中的度量有两种方式。
  - 直接度量（例如，度量一个螺栓的长度）；
  - 间接度量（例如，用次品率来度量生产出的螺栓质量）。
- 软件度量也同样分为两类：直接度量与间接度量。

- 软件工程过程的直接度量包括所投入的成本和工作量。
- 软件产品的直接度量包括产生的代码行数（LOC）、执行速度、存储量大小、在某种时间周期中所报告的差错数。
- 软件产品的间接度量包括功能性、复杂性、效率、可靠性、可维护性和许多其它的质量特性。

- 只要事先建立特定的度量规程，很容易做到直接度量软件所需要的成本和工作量、产生的代码行数等。
- 软件的功能性、效率、可维护性等质量特性却很难用直接度量判明，只有通过间接度量才能推断。

软件度量域的分类



- 软件生产率度量的焦点集中在软件工程过程的输出；
- 软件质量度量则指明了软件适应明确和不明确的用户要求到什么程度；
- 技术度量的焦点则集中在软件的某些特性（如逻辑复杂性、模块化程度）上而不是软件开发的全过程。

另一种分类方法

- 面向规模的的度量用于收集与直接度量有关的软件工程输出的信息和质量信息。
- 面向功能的度量提供直接度量的尺度。
- 面向人的度量则收集有关人们开发计算机软件所用方式的信息和人们理解有关工具和方法的效率的信息。

面向规模的度量

- 面向规模的度量是对软件和软件开发过程的直接度量。
- 可以建立一个面向规模的数据表格来记录项目的某些信息。
- 该表格列出了在过去几年完成的每一个软件开发项目和关于这些项目的相应面向规模的数据。

面向规模的数据表格

项目	工作量	元	KLOC	文档页数	错误数	人数
aaa-01	24	168	12.1	365	29	3
ccc-04	62	440	27.2	1224	86	5
fff-03	43	314	20.2	1050	64	6
⋮	⋮	⋮	⋮	⋮	⋮	⋮

- 项目aaa-01
  - 规模为 114.1 KLOC（千代码行）
  - 工作量用了 24个人月
  - 成本为168,000元
  - 文档页数为365
  - 在交付用户使用后第一年内发现了29个错误，
  - 有3个人参加了项目aaa-01的软件开发工作。

- 需要注意的是，在表格中记载的工作量和成本是整个软件工程的活动（分析、设计、编码和测试），而不仅仅是编码活动。
- 对于每一个项目，可以根据表格中列出的基本数据计算简单的面向规模的生产率和质量的度量。

- 根据数据表格可以对所有的项目计算出平均值：

生产率 = KLOC / PM（人月）
质量 = 错误数 / KLOC
成本 = 元 / LOC
文档 = 文档页数 / KLOC

面向规模度量的争议

- 大多数争议是 是否使用代码行数（LOC）做为度量的依据。
- 支持者认为 LOC是所有软件开发项目的必然产物，它能够很容易地被计算；现在许多既存的软件估算模型都是使用LOC或者KLOC做为关键输入的；而且大量以LOC为根据的文献和数据已经存在。

- 反对者们认为 LOC度量与程序设计语言有关，它们不适用于设计很好且较短的程序，也不适合于非过程型语言。若在估算中使用，很难达到要求的详细程度（计划者必须在分析和设计远未完成之前就要估算出需要生产的LOC）。

面向功能的度量

- 面向功能的软件度量是对软件和软件开发过程的间接度量。
- 面向功能度量主要考虑程序的“功能性”和“实用性”，而不是对LOC计数。
- 该度量是一种叫做功能点方法的生产率度量法，利用软件信息域中的一些计数和软件复杂性估计的经验关系式而导出功能点FP。

面向功能的数据表格

信息域参数	计数		加权因数 简单	中间	复杂		加权计数
用户输入数		×	3	4	6	=	
用户输出数		×	4	5	7	=	
用户查询数		×	3	4	6	=	
文 件 数		×	7	10	15	=	
外部接口数		×	5	7	10	=	
总 计 数	→						

功能点计算

- 确定五个信息域的特征，并在表格中相应位置给出计数。

(1) 用户输入数：各个用户输入是面向不同应用的输入数据。

(2) 用户输出数：各个用户输出是面向应用的输出信息，包括报告，屏幕信息，错误信息等。在报告中的各个数据项不应再分别计数。

(3) 用户查询数：查询是一种联机的交互操作，每次询问/响应具备应计数。

(4) 文件数：每一个逻辑主文件都应计数。逻辑主文件是指逻辑上的一组数据，可以是一个大数据库的一部分，可以是一个单独的文件。

(5) 外部接口数：与系统中其他设备通过外部接口读写信息次数均应计数。

- 一旦收集到上述数据，就可以计算出与每一个计数相关的复杂性值。
- 一个信息域是简单的、平均的还是复杂的，由使用功能点方法的机构自行确定，从而计算出加权计数。
- 计算功能点，使用如下的关系式：

$$FP = 总计数 \times (0.65 + 0.01 \times SUM(Fi))$$

- 总计数是所有加权计数项的和

- *Fi*（*i*=1..14）是复杂性校正值，它们应通过逐一回答如下提问来确定。
- *Fi*的取值0..5：

  0 没有影响　1 偶然的

  2 适中的　3 普通的

  4 重要的　5 极重要的

- *SUM*（*Fi*）是求和函数。

复杂性校正值 *Fi*

1. 系统是否需要可靠的备份和恢复？
2. 是否需要数据通信？
3. 是否有分布处理的功能？
4. 是否性能成为关键？
5. 系统是否运行在既存的高度实用化的操作环境中？
6. 系统是否需要联机数据项？

7. 联机数据项是否需要建立多重窗口显示和操作，以处理输入处理。
8. 主文件是否联机更新？
9. 输入、输出、文件、查询是否复杂？
10. 内部处理过程是否复杂？
11. 程序代码是否可复用？
12. 设计中是否包括了转移和安装？
13. 系统是否设计成可以重复安装在不同机构中
14. 系统是否设计成易修改和易使用？

- 一旦计算出功能点，就可仿照LOC的方式度量软件的生产率、质量和其它属性：

  生产率 = FP / PM（人月）

  质量 = 错误数 / FP

  成本 = 元 / FP

  文档 = 文档页数 / FP

- 功能点度量是为了商用信息系统应用而设计的。
- 特征点度量（Feature Points）可以用于系统和工程软件应用
- 特征点度量适合于算法复杂性高的应用。而实时处理、过程控制、嵌入式软件应用的算法复杂性都偏高，因此适合于特征点度量。

- 为了计算特征点，可以象功能点计算那样，对信息域值进行计数和加权。此外，特征点度量要对一个新的软件特征“算法”进行计数。
- 计算特征点可使用一个计算表格。对于每一个度量参数只使用一个权值，并且使用 $FP = 总计数 \times (0.65 + 0.01 \times SUM(Fi))$ 来计算总的特征点值。

特征点度量计算表格

度量参数	计数		权值		加权计数
用户输入数		×	4	=	
用户输出数		×	5	=	
用户查询数		×	4	=	
文 件 数		×	7	=	
外部接口数		×	7	=	
算 法		×	3	=	
总 计 数	→				

对功能点(或特征点)度量的争议

- 对于传统的工程计算或信息系统应用，功能点和特征点度量得出相同的FP值。在较复杂的实时系统中，特征点计数常常比只用功能点确定的计数高出20%到35%。
- 功能点（或特征点）度量的支持者认为FP与程序设计语言无关，它所依据的是在项目评估早期就可能知道的数据。

- 反对者认为这种方法需要某些“魔术手法”：
  - ◆ 在其计算中依赖的是主观因素而不是客观实际。
  - ◆ 信息域的数据事后很难收集，而且FP没有直接的物理意义，它只不过是一个数字。

软件质量的度量

- 质量度量贯穿于软件工程的全过程中以及软件交付用户使用之后。
- 在软件交付之前得到的度量可作为判断设计和测试质量好坏的依据。这一类度量包括程序复杂性、有效的模块性和总的程序规模。
- 在软件交付之后的度量则把注意力集中于还未发现的差错数和系统的可维护性方面。

- 使用得最广泛软件质量的事后度量包括正确性、可维护性、完整性和可使用性。

(1) 正确性：一个程序必须正确地运行，并为它的用户提供某些输出。正确性要求软件执行所要求的功能。正确性的度量是每千代码行(KLOC)的差错数，其中将差错定义为已被证实是不符合需求的缺陷。

(2) 可维护性：软件维护比其它的软件工程活动需要更多的工作量。还没有一种方法可以直接度量可维护性，因此必须采取间接度量。

有一种简单的面向时间的度量，叫做平均变更等待时间MTTC。

这个时间包括分析变更要求、设计适当的修改、实现变更并测试、及把变更发送给所有的用户。

一个可维护的程序与不可维护的程序相比，应有较低的MTTC。

(3) 完整性：完整性度量一个系统抗拒对它的安全性攻击（事故的和人为的）的能力。软件的所有三个成分程序、数据和文档都会遭到攻击。

度量完整性，需要定义两个附加的属性：危险性和安全性。

危险性是特定类型的攻击将在一给定时间内发生的概率，安全性是排除特定类型攻击的概率。

一个系统的完整性可定义为 完整性＝∑(1－危险性×(1－安全性))

其中，对每一个攻击的危险性和安全性都进行累加。

(4) 可使用性：如果一个程序不具有“用户友好性”，即使它所执行的功能很有价值，也常常会失败。可使用性量化“用户友好性”，并依据以下四个特征进行度量：

- 为学习系统所需要的体力上的和智力上的技能；
- 为达到适度有效使用系统所需要的时间；
- 当软件被某些人适度有效地使用时所度量的在生产率方面的净增值；
- 用户角度对系统的主观评价（可以通过问题调查表得到）。

协调不同的度量方法

- 代码行数和功能点之间的关系依赖于用来实现软件的程序设计语言和设计质量。
- 下面给出使用各种程序设计语言建立一个功能点所需要的平均代码行数的粗略估算。

建立一个功能点所需平均代码行数

程序语言	汇编语言	COBOL	FORTRAN	Pascal
LOC/FP (平均值)	300	100	100	90

程序语言	Ada	OOPL	4GL	代码生成器
LOC/FP (平均值)	70	30	20	15

影响软件生产率的重要因素

- 人的因素：软件开发组织的规模和专长；
- 问题因素：问题的复杂性和对设计限制，以及需求的变更次数；
- 过程因素：使用的分析与设计技术、语言和CASE工具的有效性，及评审技术；
- 产品因素：计算机系统的可靠性和性能；
- 资源因素：CASE工具、硬件和软件资源的有效性。

在软件工程过程中使用度量

- 建立基线
  - 为了将LOC和FP用于软件估算技术中，必须建立历史数据基线。
  - 根据历史经验，在软件工程过程的衔接处划出一条基线，在此基线上附有一些用于度量的经验目标信息，作为工程过程评估的依据，判断工程过程的完成是否达到预想的要求。

- 质量度量数据一旦收集到，软件开发组织就可以根据它们来调整其软件工程项目，以消除那些对软件开发有重大影响的差错产生的根源。
- 大多数软件开发人员都希望了解：哪些用户需求可能会变更？系统中哪些模块容易出错？对每一个模块要做多少测试？在测试时能够预计多少错误？如果能收集到相关的度量数据，就能确定这些问题的答案。

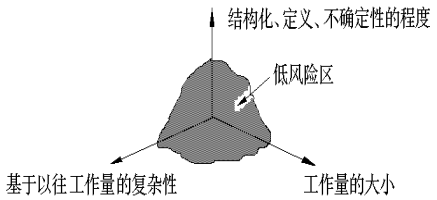
- 为了帮助计划、成本和工作量估算，基线的数据应当具有下列属性：
  - 数据必须合理、精确，应避免单纯根据以往项目进行“盲目估算”；
  - 应从尽可能多的项目中收集数据；
  - 数据必须一致；
  - 基线数据的应用必须与要做估算的工作类似。

软件项目的估算

- 软件项目管理过程开始于项目计划。
- 在做项目计划时，第一项活动就是估算。
- 在做估算时往往存在某些不确定性，使得软件项目管理人员无法正常进行管理而导致产品迟迟不能完成。
- 现在已使用的实用技术是时间和工作量估算。

- 因为估算是所有其它项目计划活动的基石，且项目计划又为软件工程过程提供了工作方向，所以我们不能没有计划就开始着手开发，否则将会陷入盲目性。
- 估算资源、成本和进度时需要经验、有用的历史信息、足够的定量数据和作定量度量的勇气。估算本身带有风险。

估算对风险的影响



- 项目的复杂性对于增加软件计划的不确定性影响很大。复杂性越高，估算的风险就越高。
- 复杂性是相对度量，它与项目参加人员的经验有关。
- 复杂性度量一般用在设计或编码级，在软件计划时使用就很困难。因此，可以在计划过程的早期建立其它较为主观的复杂性评估，如功能点复杂性校正因素。

- 项目的规模对于软件估算的精确性和功效影响也比较大。
- 随着软件规模的扩大，软件相同元素之间的相互依赖、相互影响程度也迅速增加，因而问题分解也会变得更加困难。项目的规模越大，开发工作量越大，估算的风险越高。
- 项目的结构化程度也影响项目估算的风险。随着结构化程度的提高，进行精确估算的能力就能提高，而风险将减少。

- 历史信息的有效性也影响估算的风险。对过去的项目进行综合的软件度量，可借用来比较准确地进行估算，安排进度以避免重走过去的弯路，而总的风险也减少了。
- 风险靠对不确定性程度定量地进行估算来度量，如果对软件项目的作用范围还不十分清楚，或者用户的要求经常变更，都会导致对软件项目所需资源、成本、进度的估算频频变动，增加估算的风险。

- 计划人员应当要求在软件系统的规格说明中给出完备的功能、性能、接口的定义。
- 更重要的是，计划人员和用户都应认识到经常改变软件需求意味着在成本和进度上的不稳定性。

软件项目计划的目标

- 软件项目管理人员在开发工作一开始需要进行定量估算。
- 软件项目计划的目标是提供一个能使项目管理人员对资源、成本和进度做出合理估算的框架。
- 这些估算应当在软件项目开始时的一个有限的时间段内做出，并且随着项目的进展定期进行更新。

66

软件的范围

- 软件范围包括功能、性能、限制、接口和可靠性。
- 估算开始时，应对软件的功能进行评价，对其进行适当的细化以便提供更详细的细节。由于成本和进度的估算都与功能有关，因此常常采用某种程度的功能分解。
- 性能的考虑包括处理和响应时间的需求。

67

- 约束条件则标识产品成本、外部硬件、可用存储或其它现有系统对软件的限制。
- 功能、性能和约束必须在一起进行评价。当性能限制不同时，为实现同样的功能，开发工作量可能相差一个数量级。
- 此外还要叙述某些质量因素（例如，给出的算法是否容易理解、是否使用Ada语言等）。

68

- 软件与其它系统元素是相互作用的。要考虑每个接口的性质和复杂性，以确定对开发资源、成本和进度的影响。接口的概念可解释为：
  - 运行软件的硬件（如处理机与外设）及间接受软件控制的设备（如机器、显示器）；
  - 必须与新软件链接的现有的软件（如数据库存取例程、子程序包、操作系统）；

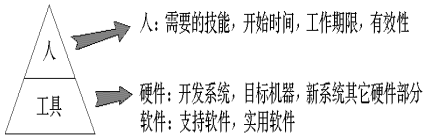
69

  - 通过终端或其它输入／输出设备使用该软件的人；
  - 该软件运行前后的一系列操作过程。
- 对于每一种情况，都必须清楚地了解通过接口的信息转换。

软件开发中的资源

- 软件项目计划的第二个任务是对完成该软件项目所需的资源进行估算。
- 软件开发所需的资源有
- 现成的用以支持软件开发的工具
  ——硬件工具及软件工具
- 最基本的资源
  ——人

软件开发中的资源



- 通常，对每一种资源，应说明以下四个特性：
  （1）资源的描述；
  （2）资源的有效性说明；
  （3）资源在何时开始需要；
  （4）使用资源的持续时间。
- 最后两个特性统称为时间窗口。对每一个特定的时间窗口，在开始使用它之前就应说明它的有效性。

73

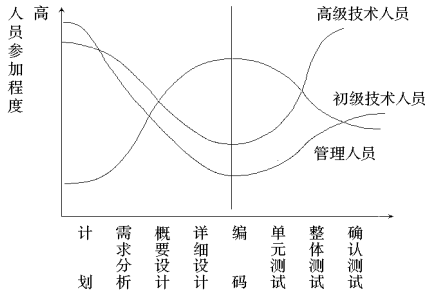
1. 人力资源

- 在考虑各种软件开发资源时，人是最重要的资源。在安排开发活动时必须考虑人员的技术水平、专业、人数、以及在开发过程各阶段中对各种人员的需要。
- 计划人员首先估算范围并选择为完成开发工作所需要的技能。还要在组织状况（如管理人员、高级软件工程师等）和专业（如通信、数据库、微机等）两方面做出安排。

74

- 对于一些规模较小的项目（1个人年或者更少），只要向专家做些咨询，也许一个人就可以完成所有的软件工程步骤。
- 对一些规模较大的项目，在整个软件生存期中，各种人员的参与情况是不一样的。下面是各类不同的人员随开发工作的进展在软件工程各个阶段的参与情况的典型曲线。

75



76

2. 硬件资源

- 硬件是作为软件开发项目的一种工具而投入的。
  （1）宿主机（Host）— 软件开发时使用的计算机及外围设备；
  （2）目标机（Target）— 运行已开发成功软件的计算机及外围设备；
  （3）其它硬件设备 — 专用软件开发时需要的特殊硬件资源；

77

- 宿主机连同必要的软件工具构成软件开发系统。通常这样的开发系统能够支持多种用户的需要，且能保持大量的由软件开发小组成员共享的信息。
- 在许多情况下，除了那些很大的系统之外，不一定非要配备专门的开发系统。因此，所谓硬件资源，可以认为是对现存计算机系统的使用，而不是去购买一台新的计算机。宿主机与目标机可以是同一种机型。

78

3. 软件资源

- 软件工程人员在软件开发期间使用了许多软件工具来帮助开发。这种软件工具集叫做计算机辅助软件工程（CASE）。
  （1）业务系统计划工具集
  （2）项目管理工具集
  （3）支援工具——文档生成工具、网络系统软件、数据库、电子邮件、通报板，以及配置管理工具。

79

（4）分析和设计工具
（5）编程工具
（6）组装和测试工具
（7）原型化和模拟工具
（8）维护工具
（9）框架工具——这些工具能够提供建立集成项目支撑环境（IPSE）的框架。

80

4. 软件复用性及软件部件库

- 为了促成软件的复用，以提高软件的生产率和软件产品的质量，可建立可复用的软件部件库。

软件项目估算

- 软件成本和工作量的估算中变化的东西太多，人、技术、环境、政治，都会影响软件的最终成本和开发的工作量。
- 软件项目的估算还是能够通过一系列系统化的步骤，在可接受的风险范围内提供估算结果。
- 成本估算必须“事前”给出。时间越久，我们了解得越多，估算中出现的严重误差就越少。

分解技术

- 当一个待解决的问题过于复杂时，我们可以把它进一步分解，直到分解后的子问题变得容易解决为止。然后，分别解决每一个子问题，并将这些子问题的解答综合起来，从而得到原问题的解答。

LOC和FP估算

- 在软件项目估算中，在两个方面使用了LOC和FP数据：
  - 把LOC和FP数据当做一个估算变量，用于量度软件每一个元素的规模。
  - LOC和FP数据作为从过去项目中收集到的基线数据，与其它估算变量联合使用，进行成本和工作量的估算。

- LOC和FP是两个不同的估算技术。两者的共性在于: 项目计划人员
  - 给出一个有界的软件范围的叙述
  - 由此叙述尝试把软件分解成一些小的可分别独立进行估算的子功能
  - 对每一个子功能估算其LOC或FP
  - 把基线生产率度量（如LOC / PM或FP / PM）用做特定的估算变量，导出子功能的成本或工作量
  - 将子功能的估算进行综合后就能得到整个项目的总估算。

- LOC或FP估算技术对于分解所需要的详细程度是不同的。
- 用LOC做为估算变量时，必须进行功能分解，且需要达到很详细的程度。而估算FP时需要的数据是宏观的量，当把FP当做估算变量时不需分解得很详细。
- LOC是直接估算的，而FP是通过估计输入、输出、数据文件、查询和外部接口的数目，以及14种复杂性校正值间接地确定的。

- 项目计划人员可对每一个分解的功能提出一个有代表性的估算值范围。
- 利用历史数据或凭实际经验（当其它的方法失效时），对每个功能分别按最佳的、可能的、悲观的三种情况给出LOC或FP估计值。记作a、m、b。
- 接着计算LOC或FP的期望值 E。

$$E = (a + 4m + b) / 6$$

- 所有子功能的总估算变量值除以相应于该估算变量的平均生产率度量得到项目的总工作量。
- 例如，若假定总的FP估算值是310，基于过去项目的平均FP生产率是5.5FP / PM，则项目的总工作量是：

  工作量 = 310 / 5.5 = 56 PM

  作为LOC和FP估算技术的实例，考察一个为计算机辅助设计（CAD）应用而开发的软件包。

- 系统定义评审指明，软件是在一个工作站上运行，其接口必须使用各种计算机图形设备，包括鼠标器、数字化仪、高分辨率彩色显示器和激光打印机。
- 在这个实例中，使用LOC做为估算变量。
- 根据系统规格说明，软件范围的初步叙述如下

“软件将从操作员那里接收2维或3维几何数据。 操作员通过用户界面与CAD系统交互并控制它，这种用户界面将表现出很好的人机接口设计特性。所有的几何数据和其它支持信息保存在一个CAD数据库内。要开发一些设计分析模块以产生在各种图形设备上显示的输出。软件要设计得能控制并与能各种外部设备，包括鼠标器、数字化仪、激光打印机和绘图仪交互。”

经过分解，识别出下列主要软件功能：

- 用户界面和控制功能
- 二维几何分析
- 三维几何分析
- 数据库管理
- 计算机图形显示功能
- 外设控制PC
- 设计分析模块

- 通过分解，可得到如下估算表

估算表

功 能	a 最佳值	m 可能值	b 悲观值	E 期望值	元／行	行／PM	成本（元）	工作量（PM）
用户接口控制	1800	2400	2650	2340	14	315	32760	7.4
二维几何造型	4100	5200	7400	5380	20	220	107600	24.4
三维几何造型	4600	6900	8600	6800	20	220	136000	30.9
数据结构管理	2950	3400	3600	3350	18	240	60300	13.9
计算机图形显示	4050	4900	6200	4950	22	200	108900	24.7
外部设备控制	2000	2100	2450	2140	28	140	59920	15.2
设计分析	6600	8500	9800	8400	18	300	151200	28.0
总 计				33360			656680	144.5

- 从历史的基线数据求出生产率度量，即行 / PM和元 / 行。
- 需要根据复杂性程度的不同，对各功能使用不同的生产率度量值。
- 在表中的成本 = LOC的期望值 E与元 / 行相乘，工作量 = 用LOC 的期望值 E与行 / PM相除。
- 因此可得，该项目总成本的估算值为657,000元，总工作量的估算值为145人月（PM）。

工作量估算

- 工作量估算是估算任何工程开发项目成本的最普遍使用的技术。
- 每一项目任务的解决都需要花费若干工作量 (人日、人月或人年)。
- 每一个工作量单位都对应于一定的货币成本，从而可以由此做出成本估算。

- 工作量估算开始于从软件项目范围抽出软件功能。
- 接着给出为实现每一软件功能所必须执行的一系列软件工程任务，包括需求分析、设计、编码和测试。
- 针对每一软件功能，估算完成各个软件工程任务所需要的工作量（如人月）。同时，把劳动费用率（即成本 / 单位工作量）加到每个软件工程任务上。

- 对于每个软件工程任务，劳动费用率都可能不同。高级技术人员主要投入到需求分析和早期的设计任务中，而初级技术人员则进行后期设计任务、编码和早期测试工作，他们所需成本比较低。
- 最后一个步骤就是计算每一个功能及软件工程任务的工作量和成本。

- 为了说明工作量估算的使用，考虑上面所介绍的CAD软件。
- 与每个软件工程任务相关的劳动费用率记入表中费用率（元）这一行，这些数据反映了“负担”的劳动成本，即包括公司开销在内的劳动成本。
- 在此例中，需求分析的劳动成本为5,200元 / PM，比编码和单元测试的劳动成本高出22%。

工作量估算表

软件开发成本估算

- 软件开发成本主要是指软件开发过程中所花费的工作量及相应的代价。它不包括原材料和能源的消耗，主要是人的劳动的消耗。
- 人的劳动消耗所需代价就是软件产品的开发成本。
- 软件产品开发成本的计算方法不同于其它物理产品成本的计算。

- 软件的开发成本是以一次性开发过程所花费的代价来计算的。
- 软件开发成本的估算，应是从软件计划、需求分析、设计、编码、单元测试、组装测试到确认测试，整个软件开发全过程所花费的代价作为依据的。

软件开发成本估算方法

- 对于一个大型的软件项目，由于项目的复杂性，开发成本的估算不是一件简单的事，要进行一系列的估算处理。主要靠分解和类推。
- 基本估算方法分为三类。
  - 自顶向下的估算方法
  - 自底向上的估计法
  - 差别估计法

自顶向下的估算方法

- 这种方法的主要思想是从项目的整体出发，进行类推。
- 估算人员根据以前已完成项目所消耗的总成本（或总工作量），推算将要开发的软件的总成本（或总工作量），然后按比例将它分配到各开发任务单元中去，再来检验它是否能满足要求。

软件	库存情况更新	开发者 W.Ward	日期 2 / 8 / 82
阶　段	项目任务	工作量分布(1/53)	小计(1/53)
计划和需求	软件需求定义	5	
	开发计划	1	6
产品设计	产品设计	6	
	初步的用户手册	3	
	测试计划	1	10
详细设计	详细PDL描述	4	
	数据定义	4	
	过程设计	2	
	正式的用户手册	2	12
编码与单元测试	程序编码	6	
	单元测试结果	10	16
组装与联合测试	编写文档	4	
	组装与测试	5	9
总　计			53

- 这种方法的优点是估算工作量小，速度快。
- 缺点是对项目中的特殊困难估计不足，估算出来的成本盲目性大，有时会遗漏被开发软件的某些部分。

自底向上的估计法

- 这种方法的主要思想是把待开发的软件细分，直到每一个子任务都已经明确所需要的开发工作量，然后把它们加起来，得到软件开发的总工作量。
- 它的优点是估算各个部分的准确性高。缺点是缺少各项子任务之间相互联系所需要的工作量，还缺少许多与软件开发有关的系统级工作量.

差别估计法

- 这种方法综合了上述两种方法的优点，其主要思想是把待开发的软件项目与过去已完成的软件项目进行类比，从其开发的各个子任务中区分出类似的部分和不同的部分。
- 类似的部分按实际量进行计算，不同的部分则采用相应方法进行估算。
- 这种的方法的优点是可以提高估算的准确程度，缺点是不容易明确“类似”的界限。

专家判定技术

- 由多位专家进行成本估算
- 单独一位专家可能会有种种偏见，譬如有乐观的、悲观的、要求在竞争中取胜的、让大家都高兴的种种愿望及政治因素等。
- 最好由多位专家进行估算，取得多个估算值。
- 有多种方法把这些估算值合成一个估算值。

- 一种方法是简单地求各估算值的中值或平均值。其优点是简便。缺点是可能会由于受一、二个极端估算值的影响而产生严重的偏差。
- 一种方法是召开小组会，使各位专家们统一于或至少同意某一个估算值。优点是可以摈弃蒙昧无知的估算值，缺点是一些组员可能会受权威或政治因素的影响。

Deiphi技术

- 标准Deiphi技术

① 组织者发给每位专家一份软件系统规格说明书和一张记录估算值的表格，请他们进行估算。

② 专家详细研究软件规格说明书的内容，对该软件提出三个规模的估算值，即：

a_i（最小） m_i（可能） b_i（最大）

无记名地填写表格

在填表的过程中，专家互相不进行讨论但可以向组织者提问。

③ 组织者对专家们填在表格中的答复进行整理：

a. 计算各位专家估算的期望值 *Ei*;

b. 对专家的估算结果分类摘要。

专家对此估算值另做一次估算。

④ 在综合专家估算结果的基础上，组织专家再次无记名地填写表格。比较两次估算的结果。若差异很大，则要通过查询找出差异的原因。

⑤ 上述过程可重复多次。最终可获得一个得到多数专家共识的软件规模（源代码行数）。在此过程中不得进行小组讨论。

- 最后，通过与历史资料进行类比，根据过去完成软件项目的规模和成本等信息，推算出该软件每行源代码所需要的成本。然后再乘以该软件源代码行数的估算值，就可得到该软件的成本估算值。

软件开发成本估算的经验模型

- 软件开发成本估算是依据开发成本估算模型进行估算的。
- 开发成本估算模型通常采用经验公式来预测软件项目计划所需要的成本、工作量和进度数据。
- 用以支持大多数模型的经验数据都是从有限的一些项目样本中得到的。还没有一种估算模型能够适用于所有的软件类型和开发环境。

IBM模型

$E = 5.2 \times L^{0.91}$

$D = 4.1 \times L^{0.36} = 14.47 \times E^{0.35}$

$S = 0.54 \times E^{0.6}$

$DOC = 49 \times L^{1.01}$

- L 是源代码行数（KLOC），E 是工作量（PM），D 是项目持续时间（月），S 是人员需要量（人），DOC是文档数量（页）。

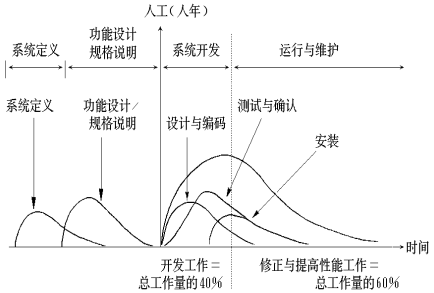
- IBM模型是静态单变量模型。
- 在此模型中，一般指一条机器指令为一行源代码。
- 一个软件的源代码行数不包括程序注释、作业命令、调试程序在内。
- 对于非机器指令编写的源程序，例如汇编语言或高级语言程序，应转换成机器指令源代码行数来考虑。
- 定义: 转换系数＝机器指令条数 / 非机器语言执行步数。

转换系数表

语　　言	转换系数
简单汇编	1
宏 汇 编	1.2～1.5
FORTRAN	4 ～ 6
PL/I	4 ～ 10

Putnam模型

- Putnam模型是一种动态多变量模型。适用于大型项目，但也可以应用在一些较小的软件项目中。
- 它是假定在软件开发的整个生存期中工作量有特定的分布。
- 大型软件项目的开发工作量分布可以用Rayleigh-Norden曲线表示。
- 这个曲线把已交付的源代码行数与工作量和开发时间联系起来。



- 用Rayleigh-Norden曲线可以导出一个“软件方程”

$$L = Ck \cdot K^{\frac{1}{3}} \cdot td^{\frac{4}{3}}$$

- *td* 是开发持续时间（年），*K*是软件开发与维护在内的整个生存期所花费的工作量（人年），*L*是源代码行数（LOC），*Ck*是技术状态常数，因开发环境而异。

技术状态常数C_k的取值

C_k的典型值	开发环境	开 发 环 境 举 例
2000	差	没有系统的开发方法，缺乏文档和复审，批处理方式。
8000	好	有合适的系统开发方法，有充分的文档和复审，交互执行方式。
11000	优	有自动开发工具和技术。

COCOMO模型（COnstructive COst MOdel）

- 结构型成本估算模型是一种精确、易于使用的成本估算方法。在该模型中使用的基本量有以下几个：
- *DSI*（源指令条数）定义为代码的源程序行数。若一行有两个语句，则算做一条指令。它包括作业控制语句和格式语句，但不包括注释语句。*KDSI＝1000DSI*。

- MM（度量单位为人月）表示开发工作量。
- TDEV（度量单位为月）表示开发进度。它由工作量决定。
- 软件开发项目的分类
  软件开发项目的总体类型:
  - 组织型
  - 嵌入型
  - 半独立型

- COCOMO模型的分类
  COCOMO模型按其详细程度分成三级：
  - 基本COCOMO模型
  - 中间COCOMO模型
  - 详细COCOMO模型
- 基本COCOMO模型是一个静态单变量模型，它用源代码行数（LOC）为自变量的（经验）函数来计算软件开发工作量。

- 中间COCOMO模型则在用LOC为自变量的函数计算软件开发工作量（此时称为名义工作量）的基础上，再用涉及产品、硬件、人员、项目等方面属性的影响因素来调整工作量的估算。
- 详细COCOMO模型包括中间COCOMO模型的所有特性，但用上述各种影响因素调整工作量估算时，还要考虑对软件工程过程中每一步骤（分析、设计等）的影响。

基本COCOMO模型

- 基本COCOMO模型的工作量和进度公式

总体类型	工 作 量	进　　度
组织型	MM＝ $2.4 \times (KDSI)^{1.05}$	TDEV＝ $2.5 \times (MM)^{0.38}$
半独立型	MM＝ $3.0 \times (KDSI)^{1.12}$	TDEV＝ $2.5 \times (MM)^{0.35}$
嵌入型	MM＝ $3.6 \times (KDSI)^{1.20}$	TDEV＝ $2.5 \times (MM)^{0.32}$

中间COCOMO模型

- 进一步考虑15种影响软件工作量的因素，通过定下乘法因子，修正COCOMO工作量公式和进度公式，可以更合理地估算软件（各阶段）的工作量和进度。
- 中间COCOMO模型的名义工作量与进度公式如下所示。

中间COCOMO模型的名义工作量与进度公式

总体类型	工 作 量	进　　度
组织型	MM＝ $3.2 \times (KDSI)^{1.05}$	TDEV＝ $2.5 \times (MM)^{0.38}$
半独立型	MM＝ $3.0 \times (KDSI)^{1.12}$	TDEV＝ $2.5 \times (MM)^{0.35}$
嵌入型	MM＝ $2.8 \times (KDSI)^{1.20}$	TDEV＝ $2.5 \times (MM)^{0.32}$

15种影响软件工作量的因素 *fi*

- **产品因素**：软件可靠性、数据库规模、产品复杂性
- **硬件因素**：执行时间限制、存储限制、虚拟机易变性、环境周转时间
- **人的因素**：分析员能力、应用领域实际经验、程序员能力、虚拟机使用经验、程序语言使用经验
- **项目因素**：现代程序设计技术、软件工具的使用、开发进度限制

工作量因素 f_i		非常低	低	正常	高	非常高	超高
产品因素	软件可靠性	0.75	0.88	1.00	1.15	1.40	
	数据库规模		0.94	1.00	1.08	1.16	
	产品复杂性	0.70	0.85	1.00	1.15	1.30	1.65
计算机因素	执行时间限制			1.00	1.11	1.30	1.66
	存储限制			1.00	1.06	1.21	1.56
	虚拟机易变性		0.87	1.00	1.15	1.30	
	环境周转时间		0.87	1.00	1.07	1.15	
人员的因素	分析员能力		1.46	1.00	0.86	0.71	
	应用领域实际经验	1.29	1.13	1.00	0.91	0.82	
	程序员能力	1.42	1.17	1.00	0.86	0.70	
	虚拟机使用经验	1.21	1.10	1.00	0.90		
	程序语言使用经验	1.41	1.07	1.00	0.95		
项目因素	现代程序设计技术	1.24	1.10	1.00	0.91	0.82	
	软件工具的使用	1.24	1.10	1.00	0.91	0.83	
	开发进度限制	1.23	1.08	1.00	1.04	1.10	

这里所谓虚拟机是指为完成某一个软件任务所使用硬、软件的结合。

- 此时，工作量计算公式改成

$$MM = r \cdot (KDEV)^c \cdot \prod_{i=1}^{15} fi$$

- 例1. 一个32*KDSI*的声音输入系统是一个输入原型，或是一个可行性表演模型。所需可靠性非常低。把此模型看做半独立型软件。则有

$MM = 3.0（32）^{1.12} = 146$

又查表知 f_1=0.75，其它 f_i=1.00，则最终有MM = 146×0.75 = 110.

- 例14. 一个规模为10KDSI的商用微机远程通信的嵌入型软件，使用中间COCOMO模型进行成本估算。
- 程序名义工作量

MM = 14.8 (10)$^{1.20}$ = 44.38（MM）

- 程序实际工作量

$$MM = 44.38 \times \prod_{i=1}^{15} fi$$

= 44.38×1.17 = 51.5（MM）

影响工作量因素 f_i	情　　况	取　　值
1 软件可靠性	只用于局部地区，恢复问题不严重	1.00（正常）
2 数据库规模	20000字节	0.94（低）
3 产品复杂性	用于远程通信处理	1.30（很高）
4 时间限制	使用70%的CPU时间	1.10（高）
5 存储限制	64K中使用45K	1.06（高）
6 机器	使用商用微处理机	1.00（额定值）
7 周转时间	平均2小时	1.00（额定值）
8 分析员能力	优秀人才	0.86（高）
9 工作经验	远程通信工作3年	1.10（低）
10 程序员能力	优秀人才	0.86（高）
11 工作经验	微型机工作6个月	1.00（正常）
12 语言使用经验	12个月	1.00（正常）
13 使用现代程序设计技术	1年以上	0.91（高）
14 使用软件工具	基本的微型机软件	1.10（低）
15 工期	9个月	1.00（正常）

- 开发所用时间

TDEV = 14.5 (51.5)$^{0.32}$ = 8.9（月）

- 如果分析员与程序员的工资都按每月6,000美元计算，则该项目的开发人员的工资总额为

51.5×6,000 = 309,000（美元）

- 做为对比，现在用IBM模型计算：

PM = 5.2 (10)$^{0.91}$ = 414.27（人月）

D = 4.1 (10)$^{0.38}$ =9.16（月）

S = 0.54 (414.27)$^{0.60}$ = 5.1 (人)

详细COCOMO模型

- 详细COCOMO模型的名义工作量公式和进度公式与中间COCOMO模型相同。
- 工作量因素分级表分层、分阶段给出。针对每一个影响因素，按模块层、子系统层、系统层，有三张工作量因素分级表，供不同层次的估算使用。每一张表中工作量因素又按开发各个不同阶段给出。

- 例如，关于软件可靠性（RELY）要求的工作量因素分级表（子系统层），如表所示。
- 使用这些表格，可以比中间COCOMO模型更方便、更准确地估算软件开发工作量。

软件可靠性工作量因素分级表(子系统层)

阶段 / RELY级别	需求和产品设计	详细设计	编程及单元测试	集成及测试	综合
非常低	0.80	0.80	0.80	0.60	0.75
低	0.90	0.90	0.90	0.80	0.88
正常	1.00	1.00	1.00	1.00	1.00
高	1.10	1.10	1.10	1.30	1.15
非常高	1.30	1.30	1.30	1.70	1.40