目录

**[（1）原理 2](#_Toc3584)**

**[（2）公式 2](#_Toc22454)**

**[（3）优点 2](#_Toc3840)**

**[（4）缺点 2](#_Toc9485)**

**[（5）适用场景 2](#_Toc920)**

[2.COCOMO II 3](#_Toc18581)

**[（1）原理 3](#_Toc20406)**

**[（2）公式 3](#_Toc4155)**

**[（3）优点 3](#_Toc1126)**

**[（4）缺点 3](#_Toc8291)**

**[（5）适用场景 3](#_Toc21981)**

[3. 专家判断（Delphi估算法） 3](#_Toc79)

**[（1）原理 3](#_Toc10954)**

**[（2）公式 3](#_Toc20117)**

**[（3）优点 3](#_Toc22376)**

**[（4）缺点 4](#_Toc1669)**

**[（5）适用场景 4](#_Toc27106)**

**[（6）优化改进 4](#_Toc17764)**

[4. 类比估计 4](#_Toc23298)

**[（1）原理 4](#_Toc14060)**

**[（2）公式 4](#_Toc19789)**

**[（3）优点 4](#_Toc19066)**

**[（4）缺点 4](#_Toc5618)**

**[（5）适用场景 5](#_Toc7869)**

[5.功能点方法（FP） 5](#_Toc26165)

**[（1）原理 5](#_Toc16370)**

**[（2）公式 5](#_Toc24898)**

**[（3）优点 5](#_Toc29932)**

**[（4）缺点 5](#_Toc2444)**

**[（5）适用场景 5](#_Toc31103)**

[6.COSMIC功能点方法 5](#_Toc2592)

**[（1）原理 5](#_Toc14590)**

**[（2）公式 6](#_Toc17301)**

**[（3）优点 6](#_Toc29163)**

**[（4）缺点 6](#_Toc388)**

**[（5）适用场景 6](#_Toc13338)**

[7.对象点方法 6](#_Toc9383)

**[（1）原理 6](#_Toc29892)**

**[（2）公式 6](#_Toc12051)**

**[（3）优点 6](#_Toc4194)**

**[（4）缺点 7](#_Toc19246)**

**[（5）适用场景 7](#_Toc4178)**

[8.NESMA功能点估算方法 7](#_Toc27460)

**[（1）原理 7](#_Toc31364)**

**[（2）公式 7](#_Toc20210)**

**[（3）优点 7](#_Toc25547)**

**[（4）缺点 7](#_Toc25793)**

**[（5）适用场景 7](#_Toc395)**

[9.FiSMA功能点方法 8](#_Toc29084)

**[（1）原理 8](#_Toc7626)**

**[（2）公式 8](#_Toc29968)**

**[（3）优点 8](#_Toc10492)**

**[（4）缺点 8](#_Toc14454)**

**[（5）适用场景 8](#_Toc16126)**

1. COCOMO（Constructive Cost Model）

（1）原理

COCOMO模型是由Barry Boehm提出的，用于估算软件开发项目成本的算法模型。最初的COCOMO模型基于三个不同的模式：有机（organic）、半分离（semi-detached）和嵌入式（embedded），分别对应不同类型的软件项目。COCOMO模型通过确定项目的规模和选择合适的模式来估算工作量。常数A和B根据不同的项目类型和经验数据来确定。

（2）公式

Effort = A \*（ Size ） ^ B

- Effort是所需的工作量，通常以人月为单位。

- A和B是根据项目类型确定的常数。

- Size是软件项目的大小，可以是代码行数（LOC）或功能点（FP）。

（3）优点

- 易于理解：COCOMO模型的基本原理和公式相对简单，易于理解和应用。

- 快速估算：可以快速提供一个初步的成本估算，适合初步预算和计划。

- 可扩展性：模型可以根据新的数据和经验进行调整，以提高未来估算的准确性。

（4）缺点

- 数据依赖性：需要大量的历史数据来确定参数，对于新项目或缺乏数据的情况不适用。

- 低灵活性：对于需求变化频繁的项目，COCOMO模型可能不够灵活，难以适应变化。

- 低时效性：随着软件开发方法和技术的发展，COCOMO模型可能不再适用于现代的软件开发环境。

（5）适用场景

COCOMO模型适用于那些需要快速估算软件开发成本的大型项目，尤其是在项目的早期阶段。它适用于结构化、功能明确的软件项目，尤其是在项目的早期阶段需要进行成本和进度估算时。适用于那些有明确需求和相对稳定的开发环境的项目。微软Windows操作系统的开发。Windows操作系统是一个大型、结构化、功能明确的软件项目，需要在项目的早期阶段进行成本和进度估算。COCOMO模型可以用来估算开发这样一个复杂系统的所需工作量和成本。

2.COCOMO II

（1）原理

COCOMO II是COCOMO模型的改进版，提供了更详细的参数化估算方法。它考虑了更多的项目和产品特性，如软件可靠性、开发团队的经验、技术成熟度等。COCOMO II模型通过考虑更多的项目特征来提高估算的准确性。每个特征都有一个相应的影响因子，这些因子被综合起来调整基础工作量。

（2）公式

Effort = C \* （ Size ） ^ E \* F15

- Effort是所需的工作量，通常以人月为单位。

- C和E是常数，分别对应于基础工作量和规模指数。

- Size是软件项目的大小，可以是代码行数（LOC）或功能点（FP）。

- F15是一个因素，考虑了15个不同的项目特征，每个特征都可能影响工作量。

（3）优点

- 精确估算：考虑了更多的项目特征，可以提供更精确的成本估算。

- 适应性：能够适应不同项目的特点，提供定制化的估算结果。

- 全面性：考虑了项目和产品的多个特性，提供了一个全面的估算框架。

（4）缺点

- 复杂性：模型较为复杂，需要专业知识才能正确应用。

- 高数据需求：需要大量的输入数据，这些数据可能难以获取或准确估计。

- 高时间消耗：由于模型的复杂性，进行估算可能需要较长的时间。

（5）适用场景

COCOMO II适用于大型、复杂的软件项目，尤其是在项目的早期阶段需要进行详细成本和进度估算时。它适用于那些需要考虑多种项目和产品特性的项目，如软件可靠性、开发团队的经验、技术成熟度等。波音公司的飞机控制系统开发。飞机控制系统是一个大型、复杂的软件项目，需要精确的成本估算和考虑多种项目特性。COCOMO II模型可以用来提供详细的成本估算和进度计划。

3. 专家判断（Delphi估算法）

（1）原理

Delphi方法是一种专家评估技术，通过匿名问卷的方式收集一组专家对某个问题的意见，然后通过多轮迭代，专家们根据反馈调整自己的估计，直到达成共识或结果稳定。Delphi方法通过多轮的反馈和调整，减少了个人偏见的影响，使得最终的估算结果更加可靠和一致。

（2）公式

Delphi方法没有固定的数学公式，它更多依赖于专家的经验和判断。

（3）优点

- 客观性：通过匿名方式收集意见，减少了个人偏见和社会压力。

- 集体智慧：能够集中多位专家的智慧，提高决策的质量。

- 迭代改进：通过多轮迭代，可以逐步改进和精确估算结果。

（4）缺点

- 高时间消耗：多轮迭代可能导致整个过程耗时较长。

- 专家依赖：过于依赖专家的质量，如果专家选择不当，结果可能不准确。

- 低稳定性：在某些情况下，可能难以达成共识，导致结果的不稳定。

（5）适用场景

Delphi方法适用于那些需要专家集体智慧来解决复杂问题的情况，特别是在缺乏历史数据或面对新颖问题时。它适用于那些需要通过多轮迭代和反馈来达成共识的项目。火星探测器的开发。火星探测器的开发是一个需要专家集体智慧来解决复杂问题的项目，尤其是在缺乏火星环境直接数据的情况下。Delphi方法可以用来收集专家意见并达成共识。

（6）优化改进

Wideband Delphi：Wideband Delphi是一种调整和简化的Delphi方法，它通过多轮迭代和反馈来达成专家间的共识。Wideband Delphi通过为专家提供更宽泛的范围，然后通过迭代来缩小范围，最终达成共识。结合了专家的知识和经验，提高了估算的准确性，比传统Delphi方法更快速和灵活。适用于需要快速估算的项目，尤其是在项目初期或专家意见分散时。

4. 类比估计

（1）原理

类比估计法通过将当前项目与历史上已完成的类似项目进行比较，基于相似项目的规模、复杂度和成本来估算当前项目的工作量和成本。类比估计法假设项目的工作量与其规模成正比，通过调整规模因子\( k \)来反映不同项目之间的差异。

（2）公式

Ecurrent = Epast \* （ Scurrent / Spast ） ^ k

- Ecurrent是当前项目的估算工作量。

- Epast是类似项目的已知工作量。

- Scurrent是当前项目的规模。

- Spast是类似项目的规模。

- k是根据历史数据确定的规模因子。

（3）优点

- 快速估算：快速得出估算结果，适用于需要快速响应的情况。

- 历史数据利用：利用历史数据进行估算，可以减少完全基于假设的不确定性。

- 适应性：可以根据新项目的特点调整规模因子，以适应不同的项目需求。

（4）缺点

- 历史数据限制：依赖于历史项目的准确性和相关性，如果历史数据不准确或项目差异较大，估算结果可能不可靠。

- 片面性：可能忽视了项目之间的差异，导致估算偏差。

- 粗略性：对于项目的具体细节和复杂性考虑不足，可能导致估算不够精确。

（5）适用场景

类比估计法适用于有类似项目可以参考，且项目需求和环境相似度较高的情况，尤其是在初步估算阶段或预算有限时。它适用于那些可以基于历史项目数据进行快速估算的项目。亚马逊的电子商务平台开发。亚马逊的电子商务平台开发可以基于之前类似电商平台的开发经验进行类比估计，以快速估算新平台的开发成本和时间。

5.功能点方法（FP）

（1）原理

功能点方法（FPA）是一种基于软件功能来估算项目规模和工作量的技术。它通过计算软件的基本信息量（如外部输入、输出、查询等）来衡量软件的复杂度，并结合环境复杂性因子来估算工作量。功能点方法通过计算软件的基本信息量来衡量软件的复杂度。每个功能点类型（如外部输入、输出、查询）都有一个对应的权重，这些权重反映了不同功能点类型的复杂度。未调整功能点数乘以相应的权重，然后求和得到总功能点数，再乘以一般系统特征因子GSF得到最终的功能点数。

（2）公式

FP = UFP \* GSF

- FP是总功能点数。

- UFP是未调整功能点数，根据外部输入、输出、查询的数量计算得出。

- GSF是一般系统特征因子，考虑了14个影响软件复杂度的因素。

（3）优点

- 功能导向：强调软件的功能和业务价值，与技术实现细节无关。

- 早期估算：可以较早地进行估算，适用于项目规划阶段。

- 标准化：遵循一定的标准，有助于提高不同项目间估算的一致性。

（4）缺点

- 高成本：需要专业的功能点分析师进行评估，增加了成本。

- 片面性：对于非功能性需求（如性能、安全性）的考虑不足。

- 复杂性：对于复杂的项目，确定功能点和权重可能变得复杂和耗时。

（5）适用场景

功能点方法适用于业务逻辑复杂、功能需求明确的项目，特别是商业应用软件，以及需要基于功能进行成本估算的情况。它适用于那些可以较早地进行估算的项目规划阶段。Salesforce的CRM系统开发。Salesforce的CRM系统是一个业务逻辑复杂、功能需求明确的商业应用软件，功能点方法可以用来估算开发这样一个系统的所需工作量和成本。

6.COSMIC功能点方法

（1）原理

COSMIC（Common Software Measurement International Consortium）功能点方法是一种基于数据移动的软件规模度量方法。它通过识别和计数软件中的数据移动（输入、输出、读取、写入）来度量软件的功能规模。COSMIC方法认为每个数据移动代表了一个功能规模单位。数据移动是指数据从一个组件移动到另一个组件的过程，无论是在软件内部还是在软件与外部环境之间。这种方法强调了数据的移动和处理，而不是代码的行数或功能点的数量。

（2）公式

CFP = Σ （ Data Movements ）

- CFP是COSMIC功能点数。

- Data Movements是数据移动的次数，包括输入（Entry）、输出（Exit）、读取（Read）和写入（Write）。

（3）优点

- 技术无关性：与技术无关，适用于各种类型的软件系统。

- 客观性：强调数据移动，更加客观和一致。

- 全面性：考虑了所有类型的数据移动，提供了一个全面的度量方法。

（4）缺点

- 复杂性：对于复杂的业务逻辑和算法处理可能不够敏感。

- 高时间消耗：需要对数据移动进行详细识别和计数，可能耗时。

- 培训需求：需要对团队进行培训，以确保正确理解和应用数据移动的概念。

（5）适用场景

COSMIC功能点方法适用于数据密集型和实时系统，特别是那些以数据处理为主的软件项目。它适用于那些需要强调数据移动和处理的项目。银行的交易处理系统。银行的交易处理系统是一个数据密集型和实时系统，COSMIC功能点方法可以用来度量软件的功能规模，从而估算开发成本和时间。

7.对象点方法

（1）原理

对象点方法（Object Points）是一种基于面向对象分析的估算方法。它通过计算应用所需要处理的对象（屏幕、报告、部件）的数量和复杂度来估算工作量。对象点方法将每个对象的复杂度乘以其基础点数，然后求和得到总对象点数。这种方法适用于面向对象的项目，因为它直接关联到对象的复杂度和数量。

（2）公式

OP = Σ （ OPoints \* C ）

- OP是总对象点数。

- OPoints是每个对象的基础点数。

- C是对象的复杂度系数，根据对象的简单、中等、困难程度来确定。

（3）优点

- 面向对象：与面向对象技术紧密结合，适用于面向对象的项目。

- 直观性：可以直观地反映用户界面和业务对象的复杂度。

- 易于理解：对于面向对象的开发者来说，这种方法易于理解和应用。

（4）缺点

- 低适应性：对于非面向对象的项目可能不适用。

- 主观性：需要对对象的复杂度进行主观评估，可能导致估算不一致。

- 高时间消耗：对于大型项目，评估每个对象的复杂度可能非常耗时。

（5）适用场景

对象点方法适用于面向对象设计的软件项目，特别是那些用户界面和业务对象复杂的系统。它适用于那些需要计算应用所需要处理的对象的数量和复杂度的项目。特斯拉的自动驾驶软件。特斯拉的自动驾驶软件是一个面向对象设计的软件项目，对象点方法可以用来估算处理自动驾驶中涉及的复杂对象的工作量和成本。

8.NESMA功能点估算方法

（1）原理

NESMA（Netherlands Software Metrics Association）功能点方法是一种基于IFPUG（International Function Point Users Group）标准的估算方法。它通过计算软件的基本信息量通过计算不同类型的功能点（如外部输入、输出、查询等）并结合环境复杂性因子来估算软件规模衡量软件的复杂度。

（2）公式

UFP = 7 \* ILF + 5 \* EIF + 4 \* EI + 5 \* EO + 4 \* EQ

- UFP是未调整的功能点数。

- ILF是内部逻辑文件数量。

- EIF是外部接口文件数量。

- EI是外部输入数量。

- EO是外部输出数量。

- EQ是外部查询数量。

（3）优点

- 通用性：遵循国际标准，具有广泛的行业认可度。

- 一致性：提供了详细的功能点计数指南，有助于提高估算的一致性。

- 适应性：可以根据项目的具体需求调整环境复杂性因子。

（4）缺点

- 高成本： 需要专业的功能点分析师进行评估，增加了成本。

- 片面性： 对于非功能性需求的考虑不足。

- 复杂性： 对于复杂的项目，确定功能点和权重可能变得复杂和耗时。

（5）适用场景

NESMA功能点方法适用于商业应用软件，特别是那些需要遵循国际标准进行估算的项目。它适用于那些需要计算软件的基本信息量并结合环境复杂性因子来估算软件规模的项目。SAP的企业资源规划（ERP）系统。SAP的ERP系统是一个商业应用软件，NESMA功能点方法可以用来遵循国际标准来估算开发这样一个系统的所需工作量和成本。

9.FiSMA功能点方法

（1）原理

FiSMA（Finnish Software Measurement Association）功能点方法是一种面向服务的软件规模度量方法。FiSMA方法通过识别软件提供的服务和功能来估算软件规模，每个服务和功能都有一个对应的功能点因子，这些因子反映了服务的复杂度。

（2）公式

S = Σ （ BFC \* FF）

- S是软件的功能规模。

- BFC是基本功能部件的数量。

- FF是功能点因子，根据服务的复杂度确定。

（3）优点

- 服务导向：适用于互联网服务和分布式系统，强调服务和功能的识别。

- 现代架构：现代软件架构紧密相关，特别是那些以服务为中心的系统。

- 灵活性：可以根据服务的复杂度灵活调整功能点因子。

（4）缺点

- 低适应性：对于非服务导向的软件可能不适用。

- 主观性：需要对服务的复杂度进行评估，可能存在主观性。

- 复杂性：对于复杂的服务和功能，确定功能点因子可能变得复杂和耗时。

（5）适用场景

FiSMA功能点方法适用于基于服务的软件架构，特别是那些以服务为中心的互联网应用。它适用于那些需要识别软件提供的服务和功能来估算软件规模的项目。Netflix的视频流服务。Netflix的视频流服务是一个基于服务的软件架构，FiSMA功能点方法可以用来估算提供视频流服务的软件规模，从而估算开发成本和时间。