软件生产的宏观优化模型

陈道斌　王浣尘　（上海交通大学系统工程研究所）

摘　要

本文分析了软件工程目前存在的问题，将系统工程的原理和方法应用到软件经济学的研究中，以开发项目总成本最小为准则，建立了大型软件项目开发的宏观控制优化模型，并对模型的应用进行了讨论和模拟计算。计算结果表明，在开发过程各阶段软件项目工作量的分配比例对人员计划安排有很大影响；纠正各阶段产生错误的费用加权系数对各阶段人员安排有较大影响，因此这种分配比例和加权系数是软件生产宏观控制的重要参数。

一、问题的提出

自1968年NATO在西德加尔密斯召开的软件可靠性国际会议上正式提出“软件工程”以来，软件工程这门学科已取得了重大进展[1,2,3,4,6]，但它的产生并没有消除软件危机[5,6]，目前软件工程中尚存在一系列有待解决的问题。本文现就几个方面加以分析。

1。软件维护费用特别高是目前软件工程面临的最大难题。下列几个统计资料显示了目前软件工程的发展状况和存在的问题（图1）。

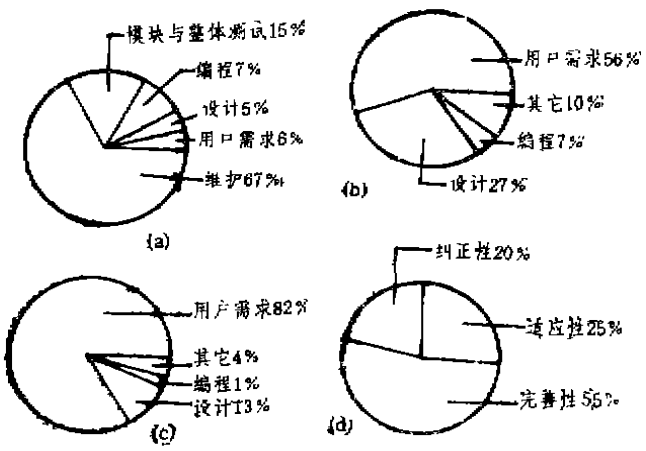

图1

图1(a)：软件生命周期各阶段投资分布，取自[5]；　(b)：软件中错误与缺陷的来源分布，取自[8]；(c)：改正错误的代价分布，取自[8]；(d)：三种维护费用的比例分配，取自[5]。

2。Putnam和Wolverton在研究软件工程的管理问题时，发现目前软件生产中人员需求和实际的人员配置不一致，这势必造成人力资源的浪费[6]。有必要根据软件生命周期各阶段工作展开的具体情况而优化安排有限的人力资源（图2）。

3。目前在软件经济学方法中，一般都把软件开发人员按同一等级划分。在计算项目工作量、工期、人数等指标时，即使考虑到人员能力的差别，也只是用一个折算因子折算到某一平均情况，没有从定量的角度考虑到由于人员水平差异而对本阶段和下阶段的生产能力和成本的影响，特别是对维护成本的影响。

基于以上三方面的分析，本文采用系统工程的原理和方法，建立了软件生产的优化模型，对软件项目中的人员配置和计划安排进行了定量探讨，应用实例说明了模型的计算结果与实际情况相符合，并具有一定指导意义。

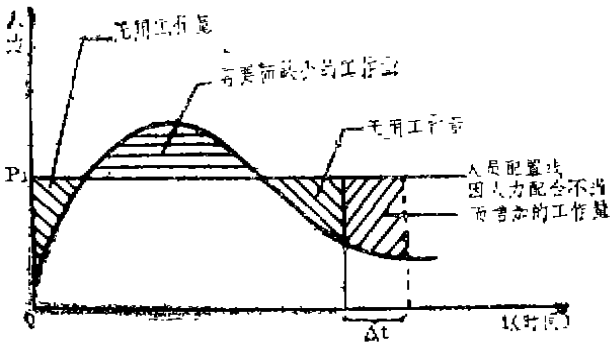

图2

二、模型建立

1. 基本假设

为了便于问题的解决和描述，这里假定：

(1) 软件开发部门在开发某一软件项目前已经通过软件经济学方法估算出了项目的规模[3,7]。

(2) 已经预知软件生命周期中每一阶段所需工作量占总工作量的比例[3,6]，生命周期划分见图3(a)。

(3) 开发部门软件技术人员可按能力进行分类（本文将其按能力从高到低分为A、B、C三类）。

(4) 暂不考虑支持环境对人员和费用等的影响。

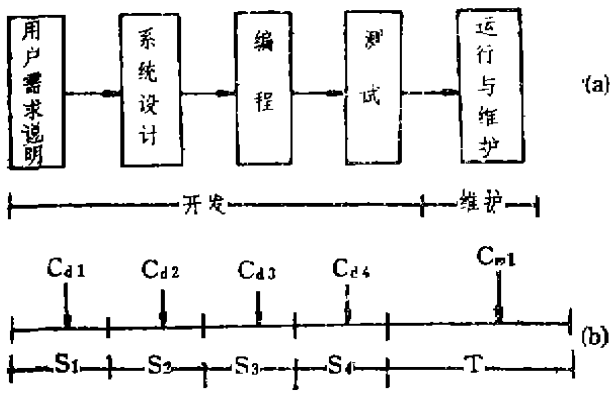

图3

2. 成本函数(C)

成本(C)=开发成本(Cd)+维护成本(Cm)

(1) **开发成本（Cd）的计算** 开发成本的构成可由项目工作量与单位工作量费用的乘积表示。开发部门技术人员单位时间里的成本（工资及其它费用）为P_{ij}，$i=1,2,3,4$对应于图3开发阶段的不同时期，$j=1,2,3$对应于A、B、C三类软件技术人员。则第i时期的开发成本Cd_i为：

$$Cd_i=\sum_{j=1}^{3}P_{ij}\cdot P_{ij}\cdot S_i$$

$i=1,2,3,4,$ (2.1)

其中：P_{ij}为i时期j类技术人员数目，单位：人；

S_i为i时期长度，以年计。

(2) **维护成本(Cm)的计算**

维护成本(Cm)=纠正性维护成本(Cm_1)+适应性维护成本（Cm_2）

i) 纠正性维护成本Cm_1。这部分成本由开发部门承担。一方面，软件中的错误来自开发阶段的不同时期，且在维护阶段纠正不同时期留下的错误所需的费用是不等的，越是开发早期留下的错误对其纠正的费用越高。另一方面，能力高的技术人员完成的软件质量高，容易维护，维护费用也相应降低。设e_{ij}为i时期j类人员单位时间里产生错误的维护费用，单位为元/年（其数据取法见[6,7]），W_i表示对i阶段的错误纠正费用的加权系数，则

$$Cm_1=\sum_i W_i\sum_j e_{ij}\cdot P_{ij}\cdot S_i \quad (2.2)$$

ii) 适应性和完善性维护成本Cm_2。这部分成本由用户单位承担。Cm_2受到两方面的影响：一是开发阶段完成的软件质量和可靠性的影响，二是所提出的新问题的难易程度的影响。开发部门并不需要计算Cm_2，此处不再深入讨论。

(3) **总成本（C）的计算** 软件开发部门在规划和决策时，只考虑Cd_i和Cm_1。设投资效果系数为r，基本以开发期开始年年初计，为简化处理，开发阶段每一时期投资

在时期中间一次投入，T为软件完成并投入运行后其使用年限。则可得开发成本函数表达式为(图3(b))

$$C=\sum_{i=1}^{4} Cd_i\cdot(1+r)^{(-\sum_{1}^{i}S_i+\frac{1}{2}S_i)}$$

$$+Cm_1\cdot(1+r)^{-(\sum_{i=1}^{4}S_i+\frac{T}{2})}$$

将(2.1)，(2.2)式代入上式，并整理后可得

$$C=\sum_{i=1}^{4} S_i(1+r)^{-(\sum_{i=1}^{i}S-i\frac{1}{2}S_i)}$$

$$\cdot\sum_{i=1}^{8}p_{ij}P_{ij}+(\sum_{i=1}^{4}W_iS_i\sum_{j=1}^{n}e_{ij}\cdot P_{ij})\cdot$$

$$(1+r)^{-(\sum_{i=1}^{4}S_i+\frac{T}{2}S_i)} \qquad (2.3)$$

3. 约束条件及数学模型

(1) 生产约束。每一时期应完成规定的工作任务和相应工作量，即

$$\sum_{j=1}^{8} l_{ij}P_{ij}S_i \geqslant\delta_i\cdot LOC$$

$$i=1,2,3,4, \qquad (2.4)$$

其中：l_{ij}为i时期j类技术人员单位时间里可完成的工作量，折算成单位为程序步/年；

LOC为整个项目的总规模，折算成可执行程序量；

δ_i为i时期工作量占整个项目工作量的比重，且有$\sum_{i=1}^{4}\delta_i=1$；

其余符号意义同前。

(2) 人员约束。每个时期各类技术人员数和技术人员总数不超过相应可提供的人数，即

$$\sum_{j=1}^{3}P_{ij}\leqslant P_{i0} \qquad i=1,2,3,4 \quad (2.5)$$

$$P_{ij}\leqslant P_{0j} \qquad i=1,2,3,4;\ j=1,2,3 \qquad (2.6)$$

(2.5)式表示各时期技术人员总人数的限制，而(2.6)式表示各时期各类技术人员数的限制。

(3) 工期约束。每一时期的工期长短和项目的总工期长度均有限制，它们分别对应于下列(2.7)式和(2.8)式。

$$Se_i\leqslant S_i\leqslant Sl_i \qquad i=1,2,3,4 \qquad (2.7)$$

$$\sum_{i=1}^{4}S_i\leqslant S_0 \qquad (2.8)$$

其中：Se_i，Sl_i，为i时期允许的最短和最长工期；

S_0为允许的最长开发期。

将(2.1)～(2.8)式汇总，取项目的总成本最小，则可得到指导软件项目优化设计的数学模型如下：

$$\mathrm{Min}C=\sum_{i=1}^{4} S_i(1+r)^{-(\sum_{i=1}^{i}S_i-\frac{1}{2}S_i)}$$

$$\cdot\sum_{j=1}^{8}p_{ij}P_{ij}+(\sum_{i=1}^{4}W_iS_i\sum_{i=1}^{3}e_{ij}P_{ij})\cdot$$

$$(1+r)^{-(\sum_{i=1}^{4}S_i+T/2)}$$

$$\text{s.t.}\begin{cases}\sum_{i}l_{ij}\cdot S_i\cdot P_{ij}\geqslant\delta_i\cdot LOC & i=1,2,3,4\\ \sum_{i}P_{ij}\leqslant P_{i0} & i=1,2,3,4\\ P_{ij}\leqslant P_{0j} & i=1,2,3,4;\ j=1,2,3\\ Se_i\leqslant S_i\leqslant Sl_i & i=1,2,3,4\\ \sum_{i}S_i\leqslant S_0 & \\ P_{ij}\geqslant 0 & i=1,2,3,4;\ j=1,2,3\\ S_i>0 & i=1,2,3,4\end{cases}$$

三、讨论

1. 模型求解与简化。上面所建立的模型为非线性混合整数规划模型。一般而言，其规模是可以预测的，经过适当处理后在求解方面并没有多少困难。如果直接求解时，可以采用迭代的方法，即先固定S_i求P_{ij}，然后再固定P_{ij}求S_i，直至收敛，得到最优解，每轮迭代实际上为求解非线性规划和线性整数规划问题。结合实际工作，考虑到开发期各阶段并不太长，而各阶段长度S_i变化也不会多大，因此实际求解时可假定S_i为常数，仅对P_{ij}优化，一次就可求解该问题。

2. 参数选取。软件工程和软件经济学所取得的进展，已经可以较为准确地估计出上述模型中所需的各个参数[3,6]。一旦这些参数确定后，开发部门可以参考模型优化计算和模拟的结果来配置人力和资源，从而可从宏观上有效地指导软件的开发工作。

3. 推广应用。当软件开发部门在进行开发项目的规划和决策时，如果要求开发成本最小，同时又对别的指标（如人数、工期等）提出要求时，则可以通过求解下列多目标规划问题而解决。例如：

i) 要求开发费用最小和开发工期最短时，其目标函数可表示为：

$$\begin{cases}\text{Min } C\\ \text{Min}\sum_i S_i\end{cases} \qquad (3.1)$$

ii) 要求开发费用最小和开发总人数最少时，其目标函数可表示为：

$$\begin{cases}\text{Min } C\\ \text{Min } (\text{Max}_i \sum_j P_{ij})\end{cases} \qquad (3.2)$$

iii) 要求开发费用最小，开发总工期最短和开发总人数最少时，其目标函数可表示为：

$$\begin{cases}\text{Min } C\\ \text{Min } \sum_i S_i\\ \text{Min } (\text{Max}_i \sum_j P_{ij})\end{cases} \qquad (3.3)$$

等等。各多目标规划问题的约束条件同第二节所述。

四、算例和分析

对第二节中建立的模型可进行仿真计算。计算时仅对各时期人员数进行优化，设各时期的长度S_i事先已确定，这时待求解的问题为全整数规划问题，采用分支定界算法求解。为了节省计算工作量和计算机内存，编程中采用了变量带上界技术的改进单纯形算法和变量带上界的对偶改进单纯形算法，使得在每次增加分支时只改变某一变量的上下界即可，不增加约束数目，从而单纯形表的规模（行列数）在迭代中始终不增大。

1. 参数

本算例参数主要取自文献[6]。根据统计资料，一个中等水平的软件工程师每日可完成的软件工作量可折算为大约20步可执行程序，所完成的每1000条可执行程序中平均有3.1个错误，检测并纠正每个错误的时间大约平均为4.4小时。根据这些数据，我们取A、B、C三类人员每日完成工作量折算成25，20和15步可执行程序，每1000步程序中出错率为3.0，5.0和10.0个，他们的工资等年费用分别为：4000，2500和1500元，开发期各阶段长度S_i为0.5、0.5、0.5和1.5年，r为10%。每天按工作六小时计算，每年工作300天。不考虑加权时纠正每个错误的平均费用为

$$4.4\text{小时/错误}\cdot\frac{1}{6.0\text{小时/天}}\cdot\frac{1}{300\text{天/年}}\times 2500\text{元/年}=6.2\text{元/错误}$$

则可得各参数值如下：

$(S_i)=(0.5,\ 0.5,\ 0.5,\ 1.5)$ 单位：年

$$(l_{ij})=\begin{bmatrix}7500 & 6000 & 4500\\ 7500 & 6000 & 4500\\ 7500 & 6000 & 4500\\ 7500 & 6000 & 4500\end{bmatrix}$$ 单位：程序步/人年

$$(p_{ij})=\begin{bmatrix}4000 & 2500 & 1500\\ 4000 & 2500 & 1500\\ 4000 & 2500 & 1500\\ 4000 & 2500 & 1500\end{bmatrix}$$ 单位：元/人年

$$(e_{ij})=\begin{bmatrix}139.5 & 186 & 279\\ 139.5 & 186 & 279\\ 139.5 & 186 & 279\\ 139.5 & 186 & 279\end{bmatrix}$$ 单位：元/人年

本文对规模为10万和20万可执行程序的两个软件项目的人力安排进行了优化计算，每一方案的各阶段工作量分配比例考虑两种情况：设为$\delta^{(1)}$和

$\delta^{(2)}$。$\delta^{(1)}$与图1（a）中比例相对应，$\delta^{(2)}$为模拟值。对每一项目的每一种工作量分配比例$\delta^{(1)}$，$\delta^{(2)}$都进行了三种加权系数W，的计算。采用的δ和W值如下。各时期各类软件技术人员允许数目为10人。

$\delta^{(1)}$= (0.182, 0.152, 0.211, 0.455)

$\delta^{(2)}$= (0.30, 0.30, 0.20, 0.20)

$W^{(1)}$= (4.0, 3.0, 2.0, 1.0)

$W^{(2)}$= (20.0, 5.0, 3.0, 1.0)

$W^{(3)}$= (100.0, 50.0, 10.0, 1.0)

因此，共有2×2×3=12种计算结果。计算结果见表1

2.分析

从以上计算结果可以得到如下几点：

（1）人员的安排与各阶段工作量的分配比例有很大的关系，随着分配比例的变化，人员配置将发生根本性改变。

（2）各阶段遗留错误维护费用加权系数不同时，会对某一阶段的人员配置产生影响，但不对别的阶段的人员配置产生影响。加权系数小时将优先安排低水平的软件人员，当还不够时再依次以中、高级的软件人员补齐。加权系数大时，本模型能根据任务量和维护成本全面衡量，合理安排高、中、低技术人员的数目。

（3）计算结果表明，由于开发阶段的早期（用户需求说明和系统设计阶段）产生的错误难维护，应尽量减少，这时应安排高水平的软件人员来完成；而开发阶段的晚期（编程和测试阶段）的错误相对较易维护和纠正，这两阶段的工作可由中等和低（初）等软件人员来完成，而将高水平的软件开发人员抽调出来从事其它更重要的工作。这一结果与实际要求很好地相符合。

（4）通过计算、分析和比较，我们认为在实际的软件项目开发中，应根据实际情况，精确估算各阶段工作量比重和各阶段错误纠正费用的加权，合理安排各阶段各类技术人员数目，应避免各阶段人员的均匀分配和无根据地指派，以减少总费用。

表1　　优化计算结果表

项目	参数 \ 自变量	P_{11}	P_{12}	P_{13}	P_{21}	P_{22}	P_{23}	P_{31}	P_{32}	P_{33}	P_{41}	P_{42}	P_{43}
第一项目	$\delta^{(1)},W^{(1)}$	0	0	9	0	0	7	0	0	10	0	0	7
	$\delta^{(1)},W^{(2)}$	5	0	0	0	0	7	0	0	10	0	0	7
	$\delta^{(1)},W^{(3)}$	5	0	0	1	4	0	0	8	0	0	0	7
	$\delta^{(2)},W^{(1)}$	0	3	10	0	3	10	0	0	9	0	0	3
	$\delta^{(2)},W^{(2)}$	5	4	0	5	4	0	0	0	9	0	0	3
	$\delta^{(2)},W^{(3)}$	5	4	0	5	4	0	0	7	0	0	0	3
第二项目	$\delta^{(1)},W^{(1)}$	0	5	10	0	3	10	0	7	10	0	3	10
	$\delta^{(1)},W^{(2)}$	9	1	0	0	5	7	1	8	7	0	3	10
	$\delta^{(1)},W^{(3)}$	10	0	0	8	1	0	3	9	2	0	3	10
	$\delta^{(2)},W^{(1)}$	3	9	10	3	9	10	0	7	9	0	0	7
	$\delta^{(2)},W^{(2)}$	9	9	0	9	9	0	3	9	1	0	0	7
	$\delta^{(2)},W^{(3)}$	10	8	0	9	9	0	3	9	1	0	0	7

参考文献

[1] Abfott, R.J."An integrated approach to software development" John Wiley Sons, Inc. 1986

[2] Boehm, B.W."software Engineering" IEEE Trans. on computer, Dec. 1976

[3] Boehm, B.W."Software Engineering" McGraw-Hill Book Co. 1985

[4] Jackson, M.A."Principle of Program Design"Academic Press, 1975

人工智能用于软件原型化方法的探讨

黄 玲 （武汉大学计算机科学系）

马 江（武汉大学软件工程国家实验室）

摘 要

Applying the methods and techniques of artificial intelligence to software engineering is becoming a general trend. Can we use artificial intelligence to software prototyping? In this paper, we try to answer these questions and make some discussions for exchanging views.

一、引 言

传统生命周期方法的缺陷，促进了软件工程中原型化方法的发展。其宗旨是用较小的代价，较快的速度生成可供人们对需求和目标系统性能进行审定的系统模型或“例子”。但是由于软件原型化方法本身并不完善，加上涉及原型开发本身的开销，目前，利用原型化方法成功地开发实际产品系统的例子并不多见。与之相反，人工智能却在许多实际领域取得了成功，如：各种实用专家系统和专家系统工具的建造，并已逐步形成了较完整的方法和技术。那么，能否将人工智能用于软件原型化方法，加快软件原型的开发呢？本文在分析原型化方法的基础上，从人工智能程序设计方法和知识表示方法的角度，阐明了人工智能用于软件原型化方法的可能性，并作了一些探讨。

二、软件原型化方法

软件原型化方法是针对传统生命周期方法的缺陷而产生的，它采用演示原型的方式来启发，判断，揭示系统的需求。一般地，软件原型化方法可分为两类：抛弃式原型方法和进化式原型方法。

抛弃式原型方法，把目标放在原理的证明上，因此着眼点是快速开发一个演示系统，用于明确需求及检验系统设计的可行性。一旦用户认可，原型即被搁置一边，“真实”的系统按生命周期方法重新开发。在该方法中，代码的编写不考虑其可维护性或效率。

进化式原型方法，是通过增量式开发方法逐步达到产品的要求。这种方法比抛弃式原型方法用得少些，但对建立信息系统极为有用。由于信息系统的逻辑设计与物理实现之间存在差别，使得进化式原型方法成为可能。进化式原型方法强调系统逻辑设计的合

[5] Ramamorthy, C.V. "the Problems and Future of Software Engineering" IEEE Centennial The State of Comhuter, Oct.1984, Vol 17, No.10

[6] Shooman, M.L. "Software Engineering: Design, Reliability and Management" McGraw-Hill Inc.1983

[7] 孙振飞等，“软件工程概论”湖南科技出版社 1987.8

[8] 胥正辉，“软件开发原型化方法”《软件产业》1988.8