一种基于模糊-插值的功能点分析法

陈庆章 李义冬 黄万德 陈巧燕 程 荣

(浙江工业大学计算机科学与技术学院 杭州 310023)

摘 要 功能点分析法是一种使用非常广泛的软件规模估算方法。主要针对IFPUG(International Function Point User Group)提出的功能点分析法在划分功能组件复杂度等级时所存在的不连续性问题,结合模糊理论和插值方法,提出了模糊-插值功能点分析法,解决了复杂度等级划分中出现的不连续、不精确等缺点和问题。实例证明,新方法不仅能更准确地估算出功能点数量,而且具有很强的实际可操作性。

关键词 功能点分析法,模糊,插值,复杂度等级

中图法分类号 TP311.5 **文献标识码** A

Study of Function Points Analysis Based on Fuzzy-Interpolation

CHEN Qing-zhang LI Yi-dong HUANG Wan-de CHEN Qiao-yan CHENG Rong

(Department of Computer, Zhejiang University of Technology, Hangzhou 310023, China)

Abstract Function Point Analysis is a widely used estimation method for measuring software size. In order to solve the problem of uncontinuity caused by complexity grade division with IFPUG(International Function Point User Group) Function Point Analysis, combined with fuzzy theory and interpolation methods, this paper presented the fuzzy-interpolation function point analysis. It is proved that the new method can more accurately estimate the number of function points, and also has excellent performance on practical operability.

Keywords Function point analysis, Fuzzy, Interpolation, Complexity grade

1 引言

精确评估软件成本是软件开发过程中最重要和最具挑战性的活动之一。评估的好坏直接影响开发的进度、人力资源的调度,甚至会影响整个项目的成败[1]。因此,软件成本评估已经日渐成为一个重要的话题。软件规模估算通常估算软件有"多大",这是通过统计功能点、代码行或对象的数目并赋予适当的权重[2],得到一个较为具体的数字来表示软件的规模。从某种意义上说,估算待开发软件的规模,是进一步估算所需工作量、开发进度和开发成本的基础,因此软件规模估算是整个评估过程中相当重要的一环。

软件估算的意义在于,通过对软件项目的规模、工作量、进度和关键计算机资源进行科学预测,从而对软件开发做出严肃、合理的承诺,指导软件开发的整个过程[2,3]。目前,软件估算方法有多种,如代码行(Lines of Code, LOC)技术、功能点分析(Function Point Analysis, FPA)方法、COCOMO模型(Constructive Cost Model)、任务估算技术、Putnam-slim估算模型和对象点模型等。其中,FPA在软件项目管理中已经得到了广泛的应用,该方法基于软件需求从功能角度和用户角度来度量一个应用程序的大小,与程序设计语言无关,既可用于传统的语言,又可用于非过程的语言,度量出来的结果可以在不同的开发方法之间进行比较,而且在项目开发初期就可以利用需求分析模型进行功能点的估算[4,5]。

软件项目的规模估算历来是比较复杂的,因为软件本身的复杂性、历史经验的缺乏、估算工具缺乏以及一些人为错误,导致软件项目的规模估算往往和实际情况相差甚远[6,7]。本文主要针对IFPUG功能点分析法在划分功能组件复杂度等级时所存在的不连续性问题,结合模糊理论和插值方法,提出了模糊-插值功能点分析法,以期解决复杂度等级划分中出现的不连续、不精确等缺点和问题。

2 IFPUG功能点分析法简介

功能点分析法历经30多年的发展,已经形成了各种不同的方法,如IFPUG功能点分析法、COSMIC功能点分析法和Mark II功能点分析法等[8],它们有许多共同点也有各自的一些特点。目前国际上最为流行的是IFPUG功能点分析法,它是由"国际功能点用户组"(International Function Point Users Group)联盟继承并发展Albrecht的功能点分析法[9]而形成的。

IFPUG功能点分析法[4]是基于系统功能的一种软件规模估算方法,它把系统按照模块的方式进行划分,用IFPUG定义的组件作为度量单位,分别对每个模块计算功能点数,从

到稿日期:2010-05-18 返修日期:2010-12-03 本文受国家自然科学基金(60872112)和浙江省重大科技优先主题项目(2007C13064)资助。

陈庆章 博士,教授,主要研究方向为无线传感网络、软件工程、信息安全等,E-mail:qzchen@zjut.edu.cn;李义冬 硕士生,主要研究方向为无线传感网络、软件工程;黄万德 硕士生,主要研究方向为无线传感网络、软件工程;陈巧燕 硕士生,主要研究方向为无线传感网络、软件工程;程 荣 硕士生,主要研究方向为无线传感网络、软件工程。

而得到反映整个系统规模的功能点数。功能点分析定义了两类功能性需求，分别是数据功能性需求和事务功能性需求。数据功能性需求又包括内部逻辑文件(Internal Logical Files，ILF)和外部接口文件(External Interface Files，EIF)；事务功能性需求则包括外部输入(External Inputs，EI)、外部输出(External Outputs，EO)和外部查询(External Query，EQ)3类。这些共同构成了功能点分析的5类衡量组件。

IFPUG功能点计算实践手册对5类衡量组件分别定义了复杂度级别表以及它们与功能点值的对应关系。每个功能组件的复杂度等级都可以划分为“低”(low)、“中”(average)、“高”(high)。针对应用系统中的某个模块，ILF和EIF是由记录元素类型(Record Element Types，RET)和数据元素类型(Data Element Types，DET)所决定的，EI，EO和EQ则由引用的文件类型个数(File Type Referenced，FTR)和数据元素类型决定。有关这些功能组件和决定要素的定义参见IFPUG功能点计算实践手册[4]。

功能点分析法可用于开发型项目(Development projects)、应用程序型(Applications)和升级型项目(Enhancement projects)的规模估算。估算功能规模的过程如下：(1)确定功能点计算的类型；(2)识别计算范围和所要度量的应用程序边界；(3)识别和估算软件的各种功能要素及数量；(4)根据复杂度和加权值计算出未调整的功能点(Unadjusted Function Point，UFP)数量；(5)确定14个功能点校正因子，并计算出校正后的功能点数量[4,5]。

3 复杂度等级划分存在的问题

在功能点分析法中，复杂度等级的划分明显存在问题。复杂度等级是由每个衡量组件的复杂度矩阵决定的。例如，表1[4]给出了一个ILF的复杂度矩阵。对于功能点分析法中的每个功能要素，表2[4]给出了复杂度等级“低”、“中”和“高”分别所对应的功能点数。从表1可以明显地看出，在以下情况下是不能精确地进行转换功能点的：

假设某ILF的RET为1个，DET为19个，得到的复杂度矩阵值为“低”，对应的功能点数为7个。如果DET增加1个，该功能的复杂度矩阵值还是“低”，然而存在另外一个功能的RET为1个，DET为50个，复杂度矩阵值为“低”。如果DET增加1个，它的复杂度矩阵值变为“中”，对应的功能点数为10个。这种情况如果同时出现在同一个评估项目中，最终就会导致评估结果不正确，因为这样得到的功能点数并不能准确代表系统的功能复杂度。

表1 ILF的复杂度矩阵

RETs	DETs		
	1～19	20～50	≥51
1	低	低	中
2～5	低	中	高
≥6	中	高	高

表2 复杂度等级和功能点数量转换

复杂度等级	ILF	EIF	EI	EO	EQ
低	7	5	3	4	3
中	10	7	4	5	4
高	15	10	6	7	6

4 模糊功能点分析法

采用模糊数学理论[10]把传统的功能点分析法扩展为模糊功能点分析法的主要思想是，利用现有的数学形式和概念来扩充传统的功能点分析法理论[5]。

在各个功能复杂度矩阵中，5类衡量组件的复杂度等级可以映射到论域X中，该论域X对应于所引用的DET数量。这些矩阵都使用相同的复杂度等级(“低”、“中”和“高”)来表示相应的复杂度。对于矩阵每一行中的每个复杂度等级都会产生相应的模糊数。实验表明，梯形模糊数除了能避免前面提到的问题外，还能较好地存储功能点分析法中复杂度矩阵的数值。一个梯形模糊数可以由$\tilde{N}(a,m,n,b)$表示，其隶属度函数的分段函数为

$$\mu_{\tilde{N}}(x)=\begin{cases}0, & x<a\\ (x-a)/(m-a), & x\in[a,m]\\ 1, & x\in[m,n]\\ (b-x)/(b-n), & x\in[n,b]\\ 0, & x>b\end{cases} \tag{1}$$

图1显示了中间型梯形隶属度函数分布情况，a和b分别表示梯形下底所在的下限和上限，此时$\mu_{\tilde{N}}(x)=0$；m和n分别表示梯形上底所在的下限和上限，此时$\mu_{\tilde{N}}(x)=1$；k是线段$\overline{am}$的中点，此时$\mu_{\tilde{N}}(x)=0.5$。

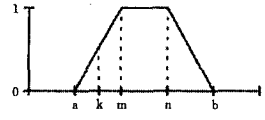

图1 中间型梯形隶属度函数

4.1 复杂度矩阵的模糊化

在每个功能要素的复杂度矩阵中，每个复杂度等级T_i(low，average，high)都要产生一个梯形模糊数$\tilde{N}(a,m,n,b)$。其中，参数low表示“低”复杂度；average表示“中”；high表示“高”。k_i表示复杂度矩阵中第i个复杂度等级的下限；n_i是m_i和m_{i+1}的数学平均值，该平均值必须是一个四舍五入的整数；n_{i-1}和m_{i+1}分别赋值给a_i和b_i。在处理第一个和最后一个复杂度等级时可以根据式(1)做一些必要的调整。

我们以包含2～5个RET的ILF为例，分别计算3个复杂度等级的梯形模糊数。在ILF的复杂度矩阵(见表1)中，根据式(1)定义的梯形隶属函数模型，可计算出包含2～5个RET的复杂度模糊数表示，如图2(a)所示。其中有3个梯形，分别代表“低”、“中”、“高”3个复杂等级的梯形隶属度函数。具体的隶属度函数表达式如下：

(1)隶属于“低”复杂度等级的隶属度函数表示

$$\mu_{\tilde{N}}(x)=\begin{cases}0, & x<m_1\\ 1, & x\in[m_1,n_1]\\ (b_1-x)/(b_1-n_1), & x\in[n_1,b_1]\\ 0, & x>b_1\end{cases} \tag{2}$$

(2)隶属于“中”复杂度等级的隶属度函数表示

$$\mu_{\tilde{N}}(x)=\begin{cases}0, & x<a_2\\ (x-a_2)/(m_2-a_2), & x\in[a_2,m_2]\\ 1, & x\in[m_2,n_2]\\ (b_2-x)/(b_2-n_2), & x\in[n_2,b_2]\\ 0, & x>b_2\end{cases} \tag{3}$$

(3)隶属于“高”复杂度等级的隶属度函数表示

$$\mu_{\tilde{N}}(x)=\begin{cases}0, & x<a_3\\ (x-a_3)/(m_3-a_3), & x\in[a_3,m_3]\\ 1, & x>m_3\end{cases} \tag{4}$$

在这个例子中，有 $m_1=1, k_2=20, n_1=a_2, m_2=b_1, n_2=a_3, k_3=51, m_3=b_2$。由等式组 $m_1+m_2=2n_1$ 和 $n_1+m_2=2k_2$，可求出 $n_1=a_2=14, m_2=b_1=27$；由等式组 $m_2+m_3=2n_2$ 和 $n_2+m_3=2k_3$，可求出 $n_2=a_3=43, m_3=b_2=59$。将所得的参数带入式(2)—式(4)，可以计算出包含 2～5 个 RET 的复杂度模糊数表示，如式(5)—式(7)所示。其中，式(5)—式(7)分别为隶属于“低”复杂度等级、“中”复杂度等级和“高”复杂度等级的隶属度函数。对应的隶属度函数图描述如图 2(b)所示。

$$\mu_{\tilde{N}}(x)=\begin{cases}0, & x<1\\ 1, & x\in[1,14]\\ (27-x)/13, & x\in[14,27]\\ 0, & x>27\end{cases} \tag{5}$$

$$\mu_{\tilde{N}}(x)=\begin{cases}0, & x<14\\ (x-14)/13, & x\in[14,27]\\ 1, & x\in[27,43]\\ (59-x)/16, & x\in[43,59]\\ 0, & x>59\end{cases} \tag{6}$$

$$\mu_{\tilde{N}}(x)=\begin{cases}0, & x<43\\ (x-43)/16, & x\in[43,59]\\ 1, & x>59\end{cases} \tag{7}$$

用这样的方法可以得到 ILF 复杂度矩阵中 3 类 RET 的模糊数表示，如图 3 所示。同样，我们可以得到另外 4 个功能性组件 EIF，EI，EO 和 EQ 复杂度矩阵的相关等级模糊数表示。

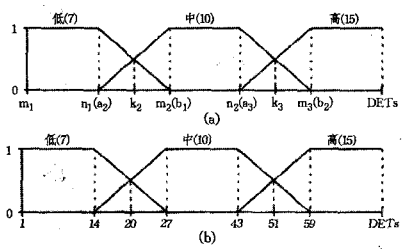

图 2　包含 2～5 个 RET 的复杂度模糊数(ILF)

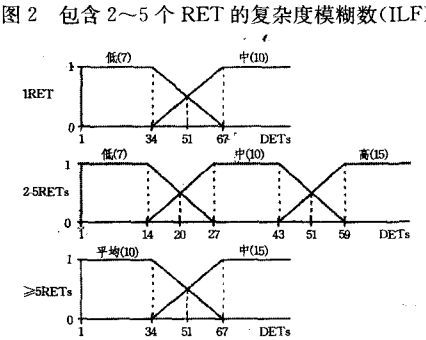

图 3　ILF 的 3 类 RET 的模糊数表示

4.2　消除模糊性

根据以上方法，得到了模糊化复杂度矩阵的梯形隶属度函数。在模糊功能点分析法中，由于引入了模糊矩阵隶属度的概念，要获得当 $\mu_{\tilde{N}}(x)<1$ 时的功能点数量，需进行如下的模糊化消除过程。

$$f(x)=\mu_{\tilde{N}}(x)\cdot p_i+\bar{\mu}_{\tilde{N}}(x)\cdot p_{i+1} \tag{8}$$

式中，$\bar{\mu}_{\tilde{N}}(x)=1-\mu_{\tilde{N}}(x)$，$f(x)$为包含有 x 个 DET 的复杂度权值，p_i 为第 i 个复杂度等级的权值。

例如，把式(8)应用到包含 3 个 RET 和 45 个 DET 的 ILF 中，根据图 2(b)所示的隶属度函数，就可以得到

$\mu_{\tilde{N}}(45)=(59-45)/(59-43)=0.875$

$\bar{\mu}_{\tilde{N}}(45)=1-0.875=0.125$

$p_d=0.875\times10+0.125\times15=10.625$

这就是说，一个包含 3 个 RET 和 45 个 DET 的 ILF 可以转换成 10.625 个功能点。

5　模糊-插值功能点分析法

虽然模糊化的功能点分析法能够较好地解决前面提到的问题，但是复杂度取值的分配还是存在一定的问题。比如在包含 2～5 个 RET 的 ILF 的复杂度矩阵表中，从 27 到 43 个 DET 都取平均值 10，还是存在变化的不合理性。于是提出在模糊功能点的基础上，利用插值方法进行插值处理。这里我们采用 Lagrange 插值法。

5.1　Lagrange 插值多项式

定理 1[11]　设 $y=f(x)$的函数表为$(x_i, f(x_i))(i=0,1,\cdots,n)$，$(x_i\neq x_j$，当 $i\neq j)$，则满足插值条件 $L_n(x_i)=f(x_i)$，$(i=0,1,\cdots,n)$的插值多项式为

$$L_n(x)=\sum_{k=0}^{n}f(x_k)l_k(x)$$

式中，

$$l_k(x)=\prod_{\substack{j=0\\ j\neq k}}^{n}\frac{x-x_j}{x_k-x_j},(k=0,1,\cdots,n)$$

插值多项式 $L_n(x)$称为 Lagrange 插值多项式，而$(x_i, f(x_i))$即为所选取的插值节点。

5.2　模糊-插值功能点分析法

在插值方法中，插值结果好坏的关键在于插值节点的选取是否合理。这里我们根据模糊化后的结果选取插值节点，如表 3 所列。插值节点的选取分配方案如下(以 ILF 为例进行介绍)：包含 1 个 RET 取 3 个插值节点；包含 2～5 个 RET 取 5 个插值节点；包含 6 个以上 RET 取 3 个插值节点。具体插值节点的选取方法我们以包含 1 个 RET 的 ILF 进行介绍。为了方便起见，取各分段的中间值为相应的插值点。比如根据图 3 中 1 个 RET 的情况，DET 为 17.5((1+34)/2=17.5)时取 7，DET 为 51((34+67)/2=51) 时取 8.5((7+10)/2=8.5)，DET 为 83.5((67+100)/2=83.5) 时取 10。

表 3　ILF 权值表插值节点

1RET	DETs(x_i)	17.5		51		83.5
	Complexity values f(x_i)	7		8.5		10
2～5 RETs	DETs(x_i)	7.5	20	35	51	67
	Complexity values f(x_i)	7	8.5	10	12.5	15
≥6RETs	DETs(x_i)	7.5		20		33.5
	Complexity values f(x_i)	10		12.5		15

当所有插值节点都确定以后，下一步就要计算每个组件包含不同个 DET 时的复杂度权值。使用 Lagrange 插值法，可计算出插值节点范围内的复杂度权值。如果 DET 数大于插值节点最大值，就取该行最大插值节点的值。如包含 1 个 RET 的 ILF 权值表中，DET 数超过 83.5 个的，就取 83.5 处的权值。这样，利用 MATLAB 软件进行 Lagrange 插值计算，可得到新的 ILF 复杂度矩阵权值表。

例如，对于包含 1 个 RET 的 ILF，根据表 3 中的插值节点，利用 Lagrange 插值法得到的插值多项式为

$f(x)=0.000056x^2+0.04x+6.26$

那么，对于包含 1 个 RET 的 ILF 的复杂度权值取值可描述如下

$$f(x)=\begin{cases}0.000056x^2+0.04x+6.26, & 1\leqslant x\leqslant 83\\ 10, & x\geqslant 84\end{cases}$$

式中，x 表示 DET 数，$f(x)$表示 DET 数为 x 时的复杂度值。

经过插值以后，解决了功能点法中出现的权值断续问题，实现了权值的精确连续取值。

6 案例分析

使用本文提出的模糊-插值功能点分析法，对本项目组过去所开发的其中两个实际系统进行了功能点计算测试。被测的两个系统分别是某市公安机关二维条码管理系统(案例 1)和学校网络视频直播点播系统(案例 2)。被测系统具体的功能模块及其功能点数如表 4 和表 5 所列。

表 4 案例 1 开发工时比较

功能模块	改进前功能点数	改进后功能点数	改进前工时(人时)	改进后工时(人时)	实际工时(人时)
用户管理	41	44.38	20.5	22.19	24
收文管理	83	90.57	41.5	45.285	48
数据管理	51	47.14	25.5	23.57	24
系统操作	46	51.74	23	25.87	24
发文管理	116	123.49	58	61.745	60
单位管理	45	50.71	22.5	25.355	24
系统管理	53	49.08	26.5	24.54	24
总计	435	457.11	217.5	228.555	228

表 5 案例 2 开发工时比较

功能模块	改进前功能点数	改进后功能点数	改进前工时(人时)	改进后工时(人时)	实际工时(人时)
视频新闻	40	43.12	20	21.56	24
校园栏目	38	42.08	19	21.04	22
活动记录	53	55.64	26.5	27.82	28
DV 影像	40	43.12	20	21.56	20
纪录片	40	43.12	20	21.56	18
影视前沿	40	43.12	20	21.56	18
评论系统	45	42.96	22.5	21.48	22
访问记录	8	7.74	4	3.87	5
文件上传	28	30.21	14	15.105	16
管理权限	30	28.25	15	14.125	14
总计	362	379.36	181	189.68	187

由于功能点数量都是计算出来的，没有可比较的实际值，而利用功能点数能进一步换算得出被测系统的开发工时和代码行数，于是，我们可以通过比较开发工时和代码行数来验证改进方法的有效性。

6.1 开发工时比较

根据对开发类似项目的历史数据的统计，在基于 B/S 结构的系统开发中，编程人员的一般开发效率为 2 个功能点/(人时)[12]，即平均每人每小时可完成 2 个功能点的开发。

下面给出两个被测系统在改进前后的开发工时对比(按每日工作 8h 计算)。

从表 4 和表 5 可以看出，采用模糊-插值功能点分析法进行功能点估算，被测系统大多数功能模块的估算结果换算出来的开发工时比原始的功能点分析更接近实际情况。少数功能模块的估算结果不理想，是因为开发这些功能模块之前已经有大量可复用的代码，所以实际开发时间较短。但是，总体上看，改进后的方法由于解决了复杂度等级划分中出现的不连续、不精确的缺点，使得估算结果更加准确、更加合理。

6.2 代码行数比较

用来衡量软件规模大小的除了功能点数外，还有一个重要的指标就是代码行数。如果不考虑其他客观因素的影响，我们可以将功能点数转化为代码行数，来比较改进前后功能点分析的精确性。部分开发语言的代码行数与功能点数的转化率如表 6[4,13]所列，改进前后代码行数对比如表 7 所列。从表 7 我们可以看出，使用改进后的功能点分析法换算出的代码行数更接近于实际代码行数。

表 6 功能点与代码行转换表

语言	代码行数/功能点数	语言	代码行数/功能点数
Access	38	Modula 2	80
Ada 95	49	Pascal	91
ASP	69	PERL	27
APL	32	PHP	64
Basic-ANSI	64	Prolog	64
Basic-Visual	32	Spreadsheet	6
C	128	USR_1	1
C++	55	USR_2	1
Fortran 95	71	USR_3	1
Fortran 77	107	USR_4	1
HTML 3.0	15	USR_5	1
Jovial	107	VB 5.0	29
Lisp	64	Visual C++	34

表 7 改进前后代码行数对比

案例	语言	改进前功能点数	改进后功能点数	改进前代码行数	改进后代码行数	实际代码行数
案例 1	PHP	435	457.11	27840	29255	29927
案例 2	ASP	362	379.36	24978	26176	26573

结束语 本文主要针对 IFPUG 功能点分析在划分功能组件复杂度等级时存在的不连续性问题，利用现有的数学形式和概念(模糊理论和插值方法)来扩充传统的功能点分析法理论，提出了模糊-插值功能点分析法，解决了复杂度等级划分中出现的不连续、不精确等缺点。

由于时间限制和种种原因，本文提出的方法只对少数实际系统进行过实际的规模度量。虽然估算结果精度有所提高，但由于有效的实际数据积累较少，还不足以调整本方法中的一些参数。所以模糊方法中隶属函数中的参数确定和插值方法中插值节点的选择都有待今后进一步研究。

参考文献

[1] Douglas M J. The impacts of the handoffs on software development: a cost estimation model[D]. USA: University of South Florida, 2006

[2] Kishore S, Naik R. 软件需求与估算[M]. 北京：机械工业出版社，2004

[3] Boehrn B W. 软件工程经济学[M]. 北京：机械工业出版社，2004

[4] IFPUG. Function Point Counting Practices Manual, Release 4.1 [Z]. International Function Point Users Group, 2003

[5] 顾勋梅，宋国新，邵志清. 一种改进的功能点分析方法[J]. 计算机工程，2007，33(22)：12-14

[6] 余方，李娟，王晓程，等. 功能点分析方法研究[J]. 计算机科学，2007，34(11)：245-251

[7] 罗光春，聂坤苗，温川彪，等. 功能点分析法的研究和改进[J]. 电子科技大学学报，2009，38(6)：983-986

[8] 蒋辉，尹俊文，何鸿君，等. 功能点方法的分析和比较[J]. 计算机工程与科学，2009，31(5)：87-89

[9] 张云帆. 软件规模度量方法比对研究[D]. 上海：同济大学，2009

[10] 李鸿吉. 模糊数学基础及实用算法[M]. 北京：科学出版社，2005

[11] 易大义，陈道琦. 数值分析引论[M]. 杭州：浙江大学出版社，2005

[12] 蔡军. 基于 FPA 方法的软件规模度量改进研究[D]. 南京：南京理工大学，2004

[13] Quantitative Software Management Inc. QSM Function Point Programming Language Table[Z]. Version 2.0, 2002