第36卷 第14期 计 算 机 工 程 2010年7月
Vol.36 No.14 Computer Engineering July 2010
·博士论文· 文章编号：1000—3428(2010)14—0010—03 文献标识码：A 中图分类号：TP311

# 功能点分析方法的一种形式化定义

顾勋梅[1]，虞慧群[2]
(1. 淮海工学院计算机科学系，连云港 222005；2. 华东理工大学信息科学与工程学院，上海 200237)

**摘 要：**针对功能点分析(FPA)方法因缺少精确化定义而导致度量结果与实际之间有一定偏差的问题，基于B方法对FPA的度量规则进行形式化定义，即为功能点计算提供一个明确的定义。实例应用表明，把B方法应用到软件度量中，能够提高软件项目管理的效率，为软件功能规模的自动化度量奠定基础。

**关键词：**功能点分析；B方法；度量；形式化定义

# Formal Definition of Function Points Analysis Method

GU Xun-mei[1], YU Hui-qun[2]
(1. Department of Computer Science, Huaihai Institute of Technology, Lianyungang 222005;
2. College of Information Science and Engineering, East China University of Science and Technology, Shanghai 200237)

**【Abstract】** Aiming at the problem that the precise definition for Function Points Analysis(FPA) is lacked, there is variation between the metric result and the practice. A clear and unambiguous definition is proposed for counting function points, that is, these counting rules for FPA must be formalized based on B method. Practical example shows that combining B method with software metrics can improve the efficiency of software project management, and provides a basis for automatic functional size measurement.

**【Key words】** Function Points Analysis(FPA); B method; measurement; formal definition

## 1 概述

软件的功能规模是一个用来合理地管理和控制软件开发项目的基本要素。功能点分析(Function Points Analysis, FPA)是一种基于软件系统的软件功能规模度量方法，用“功能点数量”来表示软件的规模。与代码行这种度量单位相比，功能点数量与编程语言无关，而且在项目开发早期确定了软件需求后即可计算。

虽然FPA方法应用很广泛，但其方法本身还是存在不足，主要体现在：(1)功能点分析的计算规则是用简单的自然语言的形式给出的，易受度量工作者的主观性影响；(2)FPA方法主要是一个手工处理过程，其代价比较高；(3)功能点分析方法比较复杂，对使用者要求较高，所需时间较长，还要求较高的需求完整性和准确性。而导致这些缺点的主要原因之一在于缺少一个精确的定义。

B方法是一个基于模型的形式化方法，对于建模和软件验证是一种有效的方法。由于功能点分析方法提出得比较早，难以适应当今软件规模和网络的快速发展，因此在大规模软件项目中，把B方法和软件度量结合起来能够提高软件项目管理的效率。

本文采用B规格说明语言来定义和描述FPA，一方面能够更好地利用FPA的优势合理有效地度量软件，为功能点计算提供一个明确的定义；另一方面为软件功能规模的自动化度量奠定基础。

## 2 B方法和功能点分析

### 2.1 B方法

B方法是一种对软件系统进行描述、设计和编码的形式化方法，产生于20世纪80年代早期，已在很多的工业领域得到成功应用。它以一种基于Zermelo-Frankle集合论的符号表示法来书写，使程序规格说明处于一个统一的数学框架下，减少了出现语义错误的可能性。B规格说明语言使用3种记法来描述软件开发过程：数学，广义代换语言和抽象机记法[1]。

B方法较其他形式化方法的优点在于它支持整个软件开发的生命周期，首先从需求分析建模到规格说明的书写，然后对模型逐步求精，直到代码自动生成。

### 2.2 功能点分析

功能点分析[2]是在20世纪70年代初期由IBM委托Allan Albrecht工程师及其同事为解决LOC度量所产生的问题和局限性而研究发布的，随后被国际功能点用户协会(IFPUG)提出的IFPUG方法继承，并从单纯的规模度量发展到倾向于软件工程整个生命周期中的应用。2004年，IFPUG发布了4.2版本，简称IFPUG FPA 4.2。

(1)功能点分析的要素

功能点分析将软件的功能归结为5个基本的功能要素：内部逻辑文件(ILF)、外部接口文件(EIF)、外部输入(EI)、外部输出(EO)和外部查询(EQ)。有关这些功能要素的定义参见文献[2]。

这5个基本要素又可分为2类：数据功能和事务功能。数据功能包括*ILF*和*EIF*，事务功能包括*EI*, *EO*, *EQ*。

(2)功能要素的复杂度等级

每个功能要素的复杂度等级可以划分为“低”、“平

**基金项目：**国家自然科学基金资助项目(60773094)；江苏省高校自然科学基础研究基金资助项目(09KJD520002)
**作者简介：**顾勋梅(1976－)，女，讲师、博士，主研方向：形式化方法，软件工程，软件度量；虞慧群，教授、博士生导师
**收稿日期：**2010-03-09 **E-mail：**gu790219@163.com

均”和“高”。事务功能复杂度等级是由数据元素类型(DET)和引用文件类型(FTR)决定的 数据功能复杂度等级是由 *DET* 和记录元素类型(RET)决定。*DET*, *RET* 和 *FTR* 的定义可参照文献[2-3]。

(3)功能点计算的过程

估算功能规模的过程如下：

(1)确定功能点计算的类型。

(2)识别计算范围和应用程序边界。

(3)识别和估算系统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过的功能点数量 *UFP*。

(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[2-3]。

## 3 基于 B 方法的 FPA

### 3.1 FPA 的抽象数据模型

图 1 给出了功能点分析的元模型，从图中可以看出功能点计算涉及的基本要素及步骤。

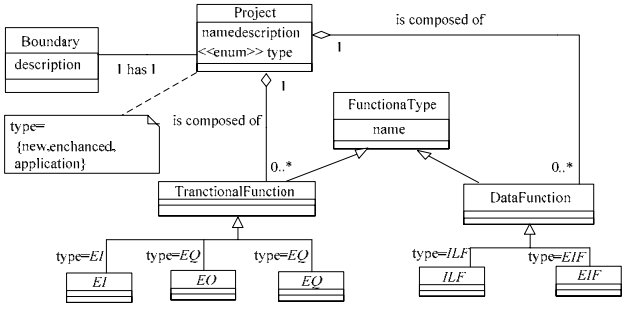


**图 1　功能点分析的元模型**

一个应用系统的功能性是由一组事务(Transaction)和数据组(Data Group)类型来共同描述的[4-5]。因此，一个应用 $\boldsymbol{a}$ 就是由事务类型 $\boldsymbol{t}_i$ 和存储的数据组类型 $\boldsymbol{f}_i$ 组成的向量：

$$\boldsymbol{a}=(\boldsymbol{t}_1,\boldsymbol{t}_2,\cdots,\boldsymbol{t}_\tau,\boldsymbol{f}_1,\boldsymbol{f}_2,\cdots,\boldsymbol{f}_\sigma)$$

其中，$\boldsymbol{t}_i$ 是由逻辑事务组成的向量； $\boldsymbol{f}_i$ 是由数据元素组成的向量。FPA 方法定义了一个度量函数 *FPC* ，这样 $FPC(a)$ 就是应用系统 $\boldsymbol{a}$ 的功能规模，具体的描述如下：

$$FPC(\boldsymbol{a})=\sum_i FPC_1(\boldsymbol{t}_i)+\sum_j FPC_2(\boldsymbol{f}_j)$$

$$\sum_i FPC_1(\boldsymbol{t}_i)=\sum_x W_{EI}(N_{DET},N_{FTR})+\sum_y W_{EO}(N_{DET},N_{FTR})+\sum_z W_{EQ}(N_{DET},N_{FTR})$$

$$\sum_j FPC_2(\boldsymbol{f}_j)=\sum_m W_{EIF}(N_{DET},N_{RET})+\sum_n W_{ILF}(N_{DET},N_{RET})$$

其中，$W_{EIF}$ ， $W_{ILF}$ ， $W_{EI}$ ， $W_{EO}$ 和 $W_{EQ}$ 表示度量过程中根据相应功能要素的复杂度等级所得到的功能点数量，而其值的决定要素是 *DET* 和 *FTR* 或者 *DET* 和 *RET* 的数量。

### 3.2 B 记法描述的 FPA

根据功能点分析的框架结构，表 1 列出了功能点中基本组件和 B 记法之间的映射关系。

**表 1　功能点中基本组件和 B 记法之间的映射关系**

| FPA 的基本组件 | B 记法 |
|---|---|
| 边界 | 一组操作的集合 |
| 基本处理过程 | 操作 |
| 数据组 | 一个抽象集的子集 |
| *DET* | 变量 *S* 或者从 *S* 到 *T* 的函数变量 *f* |
| *RET* | *DET* 集合分成 2 个 *RET* 集合：整体的和局部的函数变量 |

如表 1 所示，FPA 中应用程序的边界对应于 B 方法中的一组操作的集合，这是因为如果给定一组操作，就可能导出一个数据组件的集合，所以功能点计算过程的输入由 B 规格说明中的文本和要计算的操作集名称组成，用 *F* 表示应用程序边界。

### 3.3 FPA 组件的形式化定义

根据上面的映射关系就可以用 B 规格说明语言来形式化定义功能点计算的组件，下面以常用的数学记法来定义功能点计算的规则。

(1)数据组和 *DET*

FPA 中的数据组的典型代表就是逻辑上相关的数据组或者关系数据库中的一个关系，因此，数据组对应于变量 *S* ，其中， *S* 是一个抽象集合 *AS* 的一个子集，即 $S\subseteq AS$ ，同时也是部分函数 $f$ 的域， $f$ 是从 *S* 到 *T* 的部分函数[5]。

令 *AS* 为规范 *B* 定义的所有抽象集的集合,同时令 *V* 是 *B* 中所定义变量的集合。令 *FV* 为 *B* 中定义的所有函数的集合。这里用 *DG* 表示数据组的集合。因此，数据组的形式化定义如下：

$$DG\triangleq\{S\mid S\in V\wedge\exists AS\cdot\exists f\cdot\exists T\cdot (AS\in \mathrm{AS}\wedge S\subseteq AS\wedge f\in S\nrightarrow T)\} \quad (1)$$

一个 *DET* 就是一个数据组的一个属性，对应于一个变量 *S* 或 $f$ ,这里用 $DET(S)$ 来表示一个数据组 *S* 的 *DET* 的集合，具体定义如下：

$$DET(S)\triangleq\{S\}\bigcup\{f\mid f\in FV\wedge\exists T\cdot(f\in S\nrightarrow T)\} \quad (2)$$

(2)记录元素类型

根据 FPA 的定义，一个数据组的 *DET* 的集合可以分成若干 *RET* 的集合,而 *RET* 的集合可以被划分成必选和可选这 2 个部分。

显然，*DET* 的划分可以采用抽象机中的 INVARIANT 子句，但是这种划分存在一定的主观性，所以，折中的办法就是定义一个默认的 *RET* 来包含一个数据组中所有必选的 *DET*，用另一个 *RET* 包含所有可选的 *DET*[2,5]。

为了形式化定义 *RET*，令 *PFV* 为局部函数定义的所有机器变量的集合，则令 *TFV* 为全局函数定义的所有机器变量的集合。这样，定义 $RETm(S)$ 为一个数据组 *S* 的必选 *RET* 的集合，则定义 $RETo(S)$ 为 *S* 的所有可选的 *RET* 的集合，这 2 个集合的形式化定义如下：

$$RETm(S)\triangleq\{S\}\bigcup\{v\mid v\in DET(S)\wedge v\in TFV\} \quad (3)$$

$$RETo(S)\triangleq\{S\}\bigcup\{v\mid v\in DET(S)\wedge v\in PFV\} \quad (4)$$

(3)内部逻辑文件(ILF)和外部接口文件(EIF)

一个数据组可以归类为一个 *ILF* 或一个 *EIF*。在这里用 *ILF* 和 *EIF* 表示边界 *F* 的外部接口文件和内部逻辑文件的集合。结合内部逻辑文件和外部接口文件的定义， *ILF* 实际上是边界 *F* 内的操作所维护的数据组的集合，而 *EIF* 是保存在边界 *F* 外的，被边界内的应用系统引用，并由另外一个系统维护的数据组的集合。有关这 2 个集合的形式化定义如下：

$$ILF\triangleq\{S\mid S\in DG\wedge\exists o\cdot(o\in F\wedge inFmaintain(o,S))\} \quad (5)$$

$$EIF\triangleq\{S\mid S\in DG\wedge\exists o\cdot(o\in F\wedge outFmaintain(o,S))\wedge\neg(\exists o\cdot(o\in F\wedge inFmaintain(o,S)))\} \quad (6)$$

其中， $inFmaintain(o,S)$ 表示操作 *o* 是维护数据组 *S* 的； $outFmaintain(o,S)$ 表示 *S* 被边界内应用程序操作 *o* 所引用。

一个操作 *o* 可以维护一个数据组 *S* 的前提是存在一个基本代换 *b* ，且 *b* 的产生式系统规则的左边是 *S* 的一个 *DET*，即 $LHS(b)\in DET(S)$ 。令 $B_0$ 为操作 *o* 的定义中涉及的基本代

换的集合，则谓词 $inFmaintain(o,S)$ 的形式化定义如下：

$$inFmaintain(o,S) \Leftrightarrow \exists b \cdot \exists v \cdot (b \in B_0 \wedge v \in LHS(b)) \quad (7)$$

一个操作 $o$ 引用数据组 $S$ 的前提是 $o$ 不能维护数据组 $S$，而且 $o$ 的定义中存在 $v \in DET(S)$。这里，$occurs(v,o)$ 表示 $o$ 操作涉及数据元素类型 $v$，则谓词 $outFmaintain(o,S)$ 的定义如下：

$$outFmaintain(o,S) \Leftrightarrow \exists v \cdot (v \in DET(S) \wedge occurs(v,o) \wedge \neg inFmaintain(o,S)) \quad (8)$$

(4)外部输入(EI)

一个外部输入是维护数据或控制信息的基本处理过程，这些数据可能是从屏幕输入或从另一个应用系统得到，可用于维护一个或多个 *ILF*。因此，*EI* 对应于 B 规格说明语言中的操作。令 *EI* 为一个系统的所有外部输入的集合，$inParameter(o,p)$ 表示 $p$ 是操作 $o$ 的一个输入参数，同时 $outParameter(o,p)$ 表示 $p$ 是操作 $o$ 的一个输出参数。所以，*EI* 的形式化定义如下：

$$EI \triangleq \{o \mid o \in F \wedge \exists p \cdot \exists S \cdot (inParameter(o,p) \wedge S \in DG \wedge inFmaintain(o,S)) \wedge \neg(\exists p' \cdot (outParameter(o,p')))\} \quad (9)$$

(5)引用文件类型(FTR)

*FTR* 是指被 *EI* 事务读取或维护的 *ILF* 总数以及被 *EI* 事务读取的 *EIF* 总数。基于上面的 *DET* 和 *RET* 的定义，对于 *FTR* 的形式化定义如下：

$$FTR(o) \triangleq \{S \mid o \in F \wedge \exists v \in RET(S) \wedge \exists p \cdot (inParameter(o,p) \wedge inFmaintain(o,v))\} \quad (10)$$

(6)外部输出(EO)

外部输出是指应用系统向其边界外提供数据或控制信息的基本处理过程。一个 *EI* 至少包括一个数学公式或者计算、创建导出数据，维护一个或多个 *ILF*，或者改变系统的行为，因此，*EI* 对应于操作 $o$。这里，$derive(o,p)$ 表示 $o$ 可以导出数据 $p$，令 *EO* 为一个系统的所有外部输出的集合。所以，*EO* 的形式化定义如下：

$$EO \triangleq \{o \mid o \in F \wedge \exists p \cdot (outParameter(o,p) \wedge (derive(o,p) \vee \neg(\exists q \cdot (inParameter(o,q))) \vee \exists S \cdot (S \in ILF \wedge inFmaintain(o,S))))\} \quad (11)$$

(7)外部查询(EQ)

*EQ* 是指应用程序向其边界外提供数据或控制信息查询的基本处理过程，可以有输入和输出。*EQ* 将一个或多个 *ILF* 中的数据输出到 *EIF* 中，因此，*EQ* 也对应于边界内的操作 $o$，则令 *EQ* 为应用系统的所有外部查询的集合。所以，*EQ* 的形式化定义如下：

$$EQ \triangleq \{o \mid o \in F \wedge \exists p \cdot (inParameter(o,p)) \vee \exists p' \cdot (outParameter(o,p'))\} \quad (12)$$

## 4 基于 B 方法的功能点计算规则

要基于 B 方法计算未调整过的功能点数量，唯一需要的信息就是边界 *F* 内的操作列表。该操作列表应该包含机器的名称以及边界 *F* 内的操作。

根据式(1)~式(12)，可以得出功能点的计算步骤如下：

(1)计算集合 *DG* ；

(2)计算 *DG* 中的每个数据组的 *DET* 和 *RET* 集合；

(3)计算集合 *EI* , *EO* 和 *EQ* ；

(4)计算集合 *ILF* 和 *EIF* ；

(5)根据复杂度等级，分别计算 5 大功能要素(*EI*, *EQ*, *EO*, *ILF* 和 *EIF*)的功能点数量；

(6)简单求和就能得到应用程序的未调整过的功能点数量。

## 5 实例研究

本节以实际开发的网上书店系统为例来具体说明改进后的功能点分析方法的客观性和有效性。顾客通过该系统在线进行注册，然后查询和订购书籍；书店员工可借助该系统管理订单和顾客信息。这里分别采用 IFPUG FPA 4.2 和形式化方法规则来度量该系统。

系统包含 6 个数据组：图书，出版者，会员，请求，店铺和状态。表 2 和表 3 给出了 2 种度量方法的部分结果。

**表 2 基于形式化方法和 IFPUG FPA 的数据功能点的度量**

| 数据功能 | 类别 | *DET* | *RET* | 复杂度等级 |
|---|---|---|---|---|
| 图书信息 | *EIF* | 6 | 1 | 简单 |
| 会员信息 | *ILF* | 12 | 1 | 简单 |
| 请求信息 | *ILF* | 20 | 2(1) | 一般(简单) |

**表 3 基于形式化方法和 IFPUG FPA 的事务功能点的度量**

| 事务功能 | 类别 | *DET* | *FTR* | 复杂度等级 |
|---|---|---|---|---|
| 显示个人信息 | *EQ* | 10 | 1 | 一般 |
| 修改密码 | *EI* | 4 | 1 | 一般 |
| 显示个人订单 | *EO* | 14 | 4 | 复杂 |
| 订单确认 | *EI* | 18 | 5 | 复杂 |
| 审核订单状态 | *EI* | 18 | 6(5) | 复杂 |
| 查询会员信息 | *EQ* | 11 | 1 | 简单 |
| 删除订单信息 | *EI* | 1 | 1 | 简单 |

表 2 中括号里面的数据表示 IFPUG FPA 4.2 得到的值，没有括号表示 2 种方法的值是一致的。很显然，请求信息的度量结果存在一定偏差，这主要是因为对于该数据功能的度量是从开发者的角度而不是用户的角度，还有对于 *RET* 的形式化定义分为必选的和可选的。

从表 3 中可以看出，事务功能点的度量结果和标准也很接近，偏差很小。由于审核订单状态事务涉及的操作是维护内部系统状态，但没有输入输出，因此 IFPUG FPA 方法没有相关的计算规则。而根据上面的形式化方法，可以认为这是一个 *EI*。例如，可以把下了订单后 10 天未付款的会员的状态设置特殊标记，该操作没有输入，但修改了系统状态，但该操作能够被用户识别，这就可以划为一个 *EI*。为了计算这类操作的功能点，可以把规则 9 改为

$$EI \triangleq \{o \mid o \in F \wedge \exists S \cdot S \in DG \wedge inFmaintain(o,S) \wedge \neg (\exists p \cdot (outParameter(o,p)))\}$$

从上面的分析中可以看出，形式化的功能点分析方法的度量结果和标准值很接近，而且在某些功能上比标准方法更精确。

## 6 结束语

本文把 B 方法和软件度量有效地结合起来，通过对 FPA 元模型的抽象，对 FPA 的计算规则进行了形式化定义，减少了功能点计算过程中主观性因素对度量结果的影响，同时为功能点的计算规则提供了一个比较明确的描述，提高了度量结果的精确性。

基于形式化定义，本文还给出了基于 B 方法的功能点计算的步骤，并以网上书店为例，给出了标准 FPA 方法和形式化的 FPA 方法的部分度量结果。实例表明，度量结果和标准 FPA 方法很接近，提高了度量结果的准确性。但如果要度量其他的应用系统，应根据具体情况扩充以上提供的计算规则，因为 FPA 在 Web 应用系统中的使用还不够成熟，所以运用基于 Web 的应用系统需要对局部细节进行一定的修正。

(下转第 15 页)