第 35 卷 第 4 期 计 算 机 工 程 2009 年 2 月
Vol.35 No.4 Computer Engineering February 2009
·博士论文· 文章编号：1000—3428(2009)04—0028—03 文献标识码：A 中图分类号：TP311.1

基于 Java 程序的功能点度量

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

摘 要：针对软件项目开发中维护阶段的成本和工作量难以估计的问题，提出一种基于 Java 程序的功能点度量方法，并介绍从目标程序中抽取数据功能点以及事务功能点的计算规则，同时以 Java 源代码为例说明具体的度量过程。实验结果表明，该方法是有效可行的。
关键词：功能点分析；度量；Java 程序

Function Point Measurement Based on Java Program

GU Xun-mei[1], SHAO Zhi-qing[2]
(1. Dept. of Computer Science, Huaihai Institute of Technology, Lianyungang 222005;
2. Dept. of Computer Science & Engineering, College of Information, East China University of Science & Technology, Shanghai 200237)

【Abstract】 Aiming at the problems that it is difficult to estimate the cost and workload during the maintenance period of software development, a function point measurement method based on Java program is presented. The counting rules for extracting data and transactional function points from object program are also introduced. An example of Java source code is given to explain the detail of the measurement process. Experimental results show this method is effective and feasible.
【Key words】 Function Point Analysis(FPA); measurement; Java program

1 概述

随着计算机应用的普及，现代应用软件系统的规模和复杂度越来越大，软件测试的难度不断增加。由于软件规模与开发软件的工作量、进度和成本关系紧密，因此早期估算软件功能规模有助于确定软件价格，并可提高策划过程中的估算能力。功能规模度量(Functional Size Measurement, FSM)方法的设计目标就是仅从用户功能需求的角度度量软件。目前常用的软件功能规模度量方法有功能点分析(Function Point Analysis, FPA)和全面功能点(Full Function Point, FFP)方法。

如果软件开发者想评估一个现有系统的规模，但是该系统还需要维护，而此时的需求文档一般是过时的，那么使用 FSM 方法将得不到维护阶段所要付出的工作量和成本。

对处于维护阶段的系统，人们度量的唯一依据是源代码。因此，本文基于 FPA 并试图从 Java 源代码的角度度量软件的功能规模，以得出维护阶段所需的工作量和成本。

2 功能点分析简介

功能点分析是 20 世纪 70 年代初由 IBM 公司委托工程师 Allan Albrecht 和他的同事为解决 LOC 度量所产生的问题和局限性而研究发布的，后被国际功能点用户协会提出的 IFPUG 方法继承，在国际软件行业范围内得到接受和推崇，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，即 IFPUG FPA 4.2。

2.1 功能点分析的要素

功能点分析使用软件提供的功能度量作为规范值，它由基于软件信息领域的可计算度量及软件复杂性的评估所导出的。它将软件功能归结为 5 个基本功能要素：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)。

这 5 个基本要素又分为 2 类：数据功能(Data Function)和事务功能(Transactional Function)。事务功能包括 EI, EO, EQ，数据功能包括 ILF 和 EIF。

2.2 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”(low)、“平均”(average)和“高”(high)。事务功能复杂度等级由数据元素类型(DET)和引用文件类型(FTR)决定；数据功能复杂度等级由 DET 和记录元素类型(RET)决定[1]。DET, RET 和 FTR 的定义如下：

(1)DET 是用户可识别的无递归、不重复的信息单元，可以认为是个数据元素、一个变量或一个字段。DET 是动态的，可以读自于文件或由 FTR 的数据单元创建。另外，一个 DET 也可以是对一个事务处理过程的唤醒或是事务的有关信息。

(2)RET 是指在 ILF 或 EIF 中，用户可识别的数据集的子集，可以通过检查数据中的各种逻辑分组来识别，通常表现为一种父子关系。

(3)FTR 是指在一个事务过程中所引用到的各种文件，可以是 ILF，也可以是 EIF。

2.3 功能点计算的过程

FPA 是用于计算应用程序型、开发型项目和升级型项目规模的。估算功能规模的过程为：(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[1]。

基金项目：国家自然科学基金资助项目(60373075)
作者简介：顾勋梅(1976－)，女，讲师、博士，主研方向：软件工程，软件度量；邵志清，教授、博士生导师
收稿日期：2008-07-10 **E-mail：**gu790219@163.com

3 基于 Java 程序的功能点度量

对 Java 程序进行功能点度量的主要处理过程是从目标程序中提取数据功能点和事务功能点。由于仅仅根据有关源代码的静态信息来判断 FPA 基本功能要素的类型较困难，因此需要收集程序执行过程中的动态信息，这些执行过程是基于一组与目标程序的功能点对应的测试用例。当然，笔者希望这些测试用例是用户可接受的测试，因为用户使用不到的功能将不会被测试，且也不会算作功能点[2-3]。

例如，从程序执行过程所收集到的动态信息如图 1 所示。图中有 4 个类：*A, B, C* 和 *D*。这里的类就是对象，展示了类与类之间交换消息的一种交互作用，类似于顺序图。这里可以把这种顺序称为方法调用序列。通过分析消息所含的内容以及方法调用序列中类的类型，就能得出功能点数。这里，类是由每个测试用例构建的。

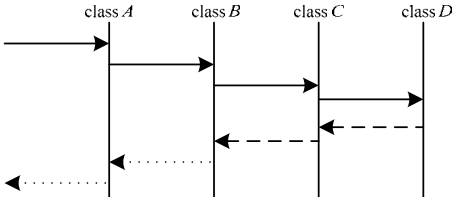

图 1　动态信息举例

3.1　数据功能点类型的计算

在 IFPUG FPA 方法中，数据功能点定义为：提供给用户且满足内部和外部数据需求的功能。同样地，面向对象需求/设计说明书中涉及的大多数类直接对应于数据功能点，当然，参与者类除外[4]。但在实际程序中，即使该程序是基于设计说明书开发的，仍然会有很多类不包含在说明书中。因此，识别数据功能点对应的类还是比较繁琐的。

假设功能点分析者从源程序中选择了数据功能点的类，那么数据功能点分为 2 种类型：内部逻辑文件(ILF)和外部接口文件(EIF)。具体解释如下：

(1)ILF：在程序执行期间，如果选定一些类的方法是由一些参数来调用的，那么这些类就视为 ILF，即参数代表数据，方法是更新类所包含的数据。

(2)EIF：在选定的类中除去 ILF 就是 EIF。

要根据数据元素类型(DET)和记录元素类型(RET)的数量确定 ILF 和 EIF 的复杂度。如前面描述的，既然程序中的类对应于数据功能点，那么 DET 就是类中简单变量(如整型、字符型和布尔型等的变量)的总数，而 RET 就是类变量的总数，即类变量就是有意义的数据组合。

3.2　事务功能点类型的计算

在 IFPUG FPA 中，事务功能定义为输入/输出的处理，这些处理会从边界外更新/引用数据功能。如果把类看作数据功能，那么这些更新/引用该类数据的方法就可用于抽取事务功能[4]。

基于上述思想，这里在方法调用序列中定义基本元素。基本元素就是子方法调用序列，如图 2 所示。

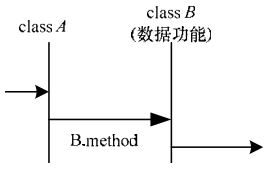

图 2　基本元素

在子方法调用序列中，一个数据功能类中定义的方法会被其他类调用。

(1)识别事务功能

事务功能的识别过程如图 3 所示。具体步骤如下：

1)为所有的测试用例收集方法调用序列，因为这些序列就是候选的事务功能。

2)为识别出事务功能必须要识别出应用程序边界。当然，这里要识别的是类，因为类的方法在用户给程序输入数据的时候肯定要被调用，如边界类、GUI 类或 Java Servlet 类都是边界类。

3)如果在一个方法调用序列中有基本元素出现，那么这个序列就可以看作是一个事务功能。当然，定义在边界类中的一个方法被调用时意味着一个方法调用序列的开始；该方法调用结束时意味着一个方法调用序列的终止。

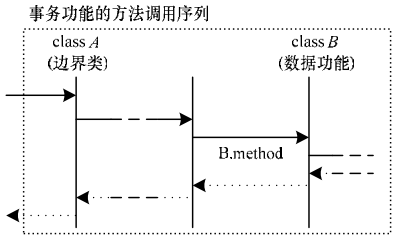

图 3　事务功能的识别

如果一些事务功能的方法调用名称相同，参数类型和参数个数也相同，方法调用次序也相同，那么这些事务功能只能看作是一个事务功能。

(2)事务功能的分类规则

对于每个被识别出的事务功能，应该把这些事务功能划分成 3 种类型 :外部输入(EI),外部查询(EQ)和外部输出(EO)，划分规则 1 如图 4 所示，其中，x 为用户自定义的类，规则 2 如图 5 所示。

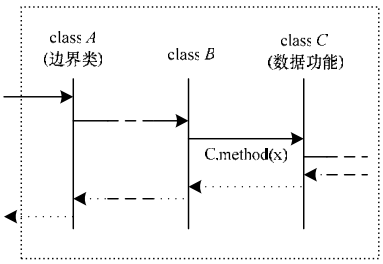

图 4　事务功能划分为 EI 的规则 1

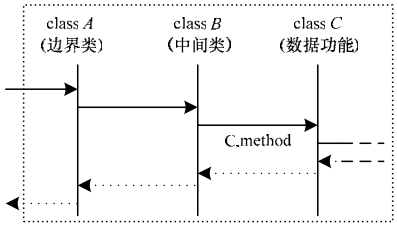

图 5　事务功能划分为 EO 的规则 2

在规则 1 中，由于用户定义的类可作为一个参数传送给数据功能类，因此这种序列可看成是个 EI。

在规则 2 中，在数据功能把返回值传送给边界类之前，一些类将会修改其返回值并把结果传送给边界类，当然，这些类并没有被识别为数据功能类。在 IFPUG FPA 中，数据功能维护数据以及把数据输出到应用程序边界的这样一个处理过程应该被看作是 EO。因此，这种维护数据以及输出数据的序列应该看作是 EO。

对于一些不满足规则 1 和规则 2 的事务功能，笔者认为它们不包含数据功能的更新，它们把处理以后的数据输出到应用程序边界外。因此，这些事务功能应该看作是 EQ。

(3)复杂度的确定

在识别出事务功能并划分好具体的功能要素后，人们需要确定 EI, EO, EQ 的复杂度。当然事务功能复杂度的确定是根据数据元素类型(DET)和参考文件类型(FTR)的数量。

由于边界类的集合就是应用程序的边界，因此这里规定 DET 的计算规则如下：

1)EI：DET 是 EI 处理过程中由边界类调用的方法的参数的个数之和。

2)EO 和 EQ：DET 是 EO 或 EQ 处理过程中由边界类调用的方法的返回值的数量总和。

由于 FTR 是出现在事务功能中的数据功能的总和，因此在某些场合若同一个数据功能类重复出现，则只需计算一次。

4 Java 源代码度量举例

以 Java 伪代码为例，说明基于源代码的功能规模度量的应用。下面的程序描述了“职员查询服务”子系统。程序代码为

```
public class Organization {
        private String no;
        private String name;
    public void add();
        public void delete();
        public void browse();
}
public class Worker {
    private String name;
    private String SSN;
    private String phone;
        public boolean vaildSSN();
        public void add();
        public void delete();
        public void update();
}
public class Position {
        private String no;
        private String name;
        public void add();
        public void delete();
        public void browse();
}
public class Place {
        private String number;
private String road;
private String city;
private String country;
        public void add();
        public void delete();
        public void browse();
}
public class Photo {
        private String SSN;
        private Date date;
        private byte[] photo;
        public void display();
        public void add();
        public void delete();
}
public class App {
        public boolean login();
        public Place[] queryPosition1();
        public Place[] queryPosition2();
        public void passinSafeFile(SafeFile file);
}
public interface SafeFile {
...
}
```

以上程序描述的“职员查询服务”子系统是用于从职员信息数据库中请求查询信息。程序包含 5 个类：Organization(机构)，Worker(职员)，Position(职位)，Place(工作地点)和 Photo(照片)，还包含 1 个接口，即 SafeFile(安全文件)。对职员查询服务涉及的方法都定义在类 App 中。该子系统的简明框图如图 6 所示。

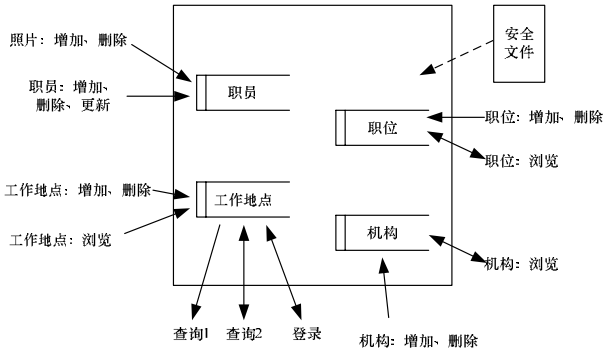

图 6　职员查询服务的系统框图

根据上面的基本度量原理，该系统的功能点计算过程为

(1)职员类是 ILF，包含 5 个 DET 和 2 个 RET。5 个 DET 分别为 name，SSN(保险号)，phone, photo 和 date；2 个 RET 分别为职员信息和照片。同样，工作地点类是 ILF，包含 4 个 DET 和 1 个 RET；职位类和机构类也均是 ILF，且均包含 2 个 DET 和 1 个 RET。具体的 DET 和 RET 参照源代码，不再一一列出。

(2)安全文件接口是 EIF，包含不多于 20 个 DET 和 1 个 FTR，而属性在程序中省略。

(3)职员类包含的 add 和 update 方法均属于 EI，均包含 3 个 DET 和 4 个 FTR；其包含的 delete 方法也属于 EI，包含 1 个 DET 和 1 个 FTR。职位类包含的 add 方法属于 EI，包含 2 个 DET 和 1 个 FTR；其包含的 delete 方法也属于 EI，包含 1 个 DET 和 2 个 FTR。工作地点类包含的 add 方法属于 EI，包含 4 个 DET 和 1 个 FTR；其包含的 delete 方法也属于 EI，包含 1 个 DET 和 2 个 FTR。机构类包含的 add 方法属于 EI，包含 2 个 DET 和 1 个 FTR；其包含的 delete 方法也属于 EI，包含 1 个 DET 和 2 个 FTR。照片类包含的 add 方法属于 EI，包含 3 个 DET 和 1 个 FTR；其包含的 delete 方法也属于 EI，包含 2 个 DET 和 1 个 FTR。

(4)类 App 包含的 queryPosition1 方法用于导出数据，故属于 EO，包含 5 个 DET 和 1 个 FTR。类似地，其包含的 PassinSafeFile(登录)方法属于 EQ，包含 2 个 DET 和 1 个 FTR；其包含的 queryPosition2 方法属于 EQ，包含 10 个 DET 和 4 个 FTR。

(下转第 34 页)