老师：张桂戌
学生：徐国庆(YS02241014)
莫杰众(YS02241011)

Coding Analysis

——对编码过程进行分析的尝试

引言

软件过程和产品的度量是软件过程走向成熟的关键之一。通过度量我们可以了解、评估开发过程和产品的品质，发现风险，并协助计划和过程改进。但是数据收集是要有代价的，尤其是在没有形成相应的文化和共识的情况下，数据的收集很可能是不准确和主观的。另外个人的开发过程（比如 psp【1】）的数据收集需要开发人员认识倒合理的个体开发过程和数据收集是重要的。但大多数情况下这需要个体感受倒它们对开发的帮助。而一般情况下，开发过程执行之初整体的效率不会增加相反会因为引入辅助活动而降低开发效率。这样会阻碍开发人员采用它们。如果能够有自动统计的工具相信对克服起步障碍是有帮助的。

本文试图提出一种自动或半自动的过程数据收集机制和实现设想，其间想讨论一下从结果程序回溯的程序结构时序分析。因为是期末考试的题目所以在最后将概要总结软件度量的现状并例举几个度量工具。

概念及实现设想

概述

使用现在比较通用的程序开发语言来实现一定的逻辑主要是依靠程序员通过键盘输入。而编码阶段的很大一部分时间是用来实现业务逻辑（信息管理系统）或者是数据处理等非可视的部分。常用的 IDE 对界面的可视化开发的支持极大的提高了相关部分的开发效率。而对这些非可视的部分则相应的帮助较少。这样，编码的主要时间一般是用在敲键式的输入上（对此观点没有相应的数据支持，只是开发经历的感觉）。

通过一个人的外在行为可以一定程度的了解他的心理活动情况。尤其是自然的行为，比如口误或梦等（精神分析【2】）。敲键的时序属性可能可以反映程序员和程序的某些特征。比如某个模块的消耗时间、编码效率（可以包括时段效率）、编码方式（比如好多粘贴）。对编码时序特征的进一步分析可能对程序的其它特征，比如修改率、修改消耗时间、各个阶段（编码、编译和 debug）的耗费时间、模块的稳定性。结合程序进一步的静态分析可以得出设计的某些特征，比如设计缺陷对实现的影响程度、代码的复杂度对编码的影响、代码复杂度对测试和除错的影响、重构耗费的时间和效果。最后，通过代码时序和静态特征的分析结

果可以对代码潜在的问题做一定的预测并指导以后的计划和改进。

在敲键过程中，不同的时间开发人员受到各种不同外界因素的影响，同时处于不同的心里状态。开发人员本身的属性对开发也有影响。这样基于时序特征的预测和断言是不精确的。对它的解释不能简单的映射倒手动过程记录的行为中（比如编码、重构或测试），也不是对手动过程统计的替代。

另一方面不同的语言和开发过程对编码风格也是有影响的，比如完整的设计之后的编码与部分设计之后的编码时序风格显然是有区别的。个人的思维模式也相应的会有影响。相应的调整对统计数据的解释是非常重要的。

需求描述

试图实现的系统通过对所记录的程序员敲键过程的分析来得出开发过程的属性和产品属性。系统将能够针对各种因素来调整解释。

实现设想

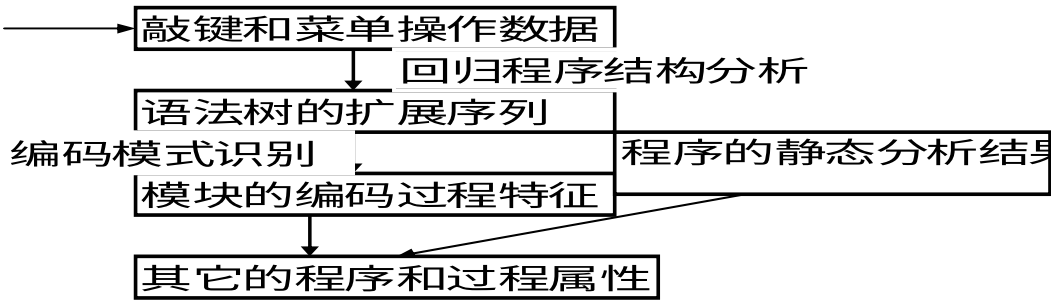
敲键过程带有随机性，而且相同的击键在不同的程序部分有完全不同的含义。所以程序逻辑结构的时序变化比简单的程序行数的变化蕴涵更多的信息。所以在动作记录的基础上应该先得出程序逻辑结构的时序关系。即语法树的时序变化情况。在此基础上就可以得出模块的耗费时间和编码模式（比如类重构编码模式、实现设计编码模式和除错编码模式）。相应的可以得出一些程序模块的特征，比如某个模块的消耗时间类型比例（比如开发/除错，开发/重构）。

根据敲键的顺序得出的语法树是极不稳定的，任何一个括号都可能改变整个语法树的结构，使分析无从下手。但是在某个稳定编辑阶段反向的分析击键过程会得到稳定扩展的语法树。后面将详细讨论这一想法。

应该提供程序员在敲键之外向系统表述自己的手段，比如说明自己是在写程序还是喝咖啡，今天是否有些头痛等等。这些表述将是系统的和可处理的。

对于结果的表示还没有明确的想法，数据的利用可能根据不同的度量计划和需要有不同方式。表示也是和需要相关的。

数据收集和处理的过程设想如图 1 所示。



回归的程序结构分析

程序的语法树在编码过程中不是稳定的扩展的。而这种不稳定不是起源于思路的混乱或者设计的变动，而来自敲键时的失误。这些失误在整个编码过程中随机的出现。它不能反映程序特征或者程序员的思维属性（可以认为失误较多的人比较马虎，但是这一点意义不大）。可以认为这是分析中应该去除的噪音。对于基于语法树的时序变化需要过滤调这些噪音。

识别某一次击键在语法树生成过程中的位置，或者识别为失误非常重要。这是进一步分析的基础。可以发现最终，或者是阶段性的通过编译的代码是稳定的。而回溯代码中的每个元素的产生时间是完全有可能的。这样我们就可以得到一个从一棵完整的最终语法树到初始状况的，递减的语法树序列。由于好像很显然，这里就不举例子说明了。

从某个稳定的语法树向后的推导是另一个问题。猜测从一个正确的程序为基础而生成的不正确的程序的意图是困难和有风险的。当树的叶节点的抽象层次越高，这种风险就越小。当然如果叶就是根节点就没有风险了。也就是说，我们可能不一定要生成一课完整的语法树，而止于某个抽象层次，比如方法的定义。在更低层次的代码处理时只了解它的代码行数属性就满足了。不同的层次对数据的处理可能是不同的。

另一个在数据的初步处理中可能遇到的问题是粘贴。粘贴的来源可能表达了程序的相似性，这种相似性在程序设计中应该避免，而且如果有错误，那很可能预示着错误是双份的。对于粘贴来源的追踪可能会有帮助，但这跟实现的技术有关，就不再多讨论了。

与 IDE 和程序员的接口

原始数据的收集有很多方法，比如监控敲键，在 window 下可以用 hook 技术来捕获消息。

另一种方法是真对不同的 IDE 开发相应的程序，有些 IDE 是提供扩展接口的，比如 JBuilder。这样使用它们提供的接口可以得到编码过程中的绝大数活动。这种方法相较监控敲键有很多优势，它可以得到有语义的动作。因为某些击键通过 IDE 解释，动作语义是未知的。而直接知道动作本身当然比了解敲键要好很多。

但是对键盘和鼠标输入的监控可以猜测当前程序员是不是在写程序。这样也是有帮助的。

提供程序员系统的和可理解表达自己状态的方法非常重要。定义相应的属性和域值及在认识上达成一致对正确的解释数据至关重要。

表示方式

数据的表示方式对后期的处理有很大的影响。简单的敲键序列非常难以处理。带有过程信息的语法树是一个好方法。这些过程信息不只是节点本身的过程信息，还包括节点见的时序关系信息。比如 A 节点的开发是在一个完成很久的 X 下据 Y 的需要而开发的可能意味着 A 的设计由于 Y 而不完整。这样对设计中的问题关联分析，错误的关联分析都是有帮助的。

从带有过程信息的语法树我们进一步可以得到编码过程更加抽象的表示。比如某段时间内的动作是重构。

活动类型识别

活动类型的识别是困难和带有猜测性的。但是这种识别对于向程序员提供进一步的帮助很重要。

各种不同的因素将很大程度的影响这种识别的有效性。所以程序须对这些因素做适应性调整。可以把这种调整看作是一个学习的过程。程序员在开发过程中或是事后通过对活动的说明来指导程序的学习。

从带有过程属性的语法树到活动类型的识别的方式可以有很多，但现在还不清楚，也就不多讨论。

工具的使用

工具收集有用的数据基于程序的理解和程序员的理解的一致性。工具不能够对任意的编码过程提供有效的分析。它要求程序员在一定程度上学习和理解工具对数据的理解。同时工具也要学习和理解程序员的活动含义。前一种学习就是对程序员的培训。并不试图完全让程序员在不知不觉之中完成统计，而仅仅是最大限度的减少程序员在数据收集时的消耗，从而降低过程规范的门槛。

第二种学习也就是机器的学习，需要程序员的帮助。这和语音识辨工具的学习很相似。目的是使工具适应程序员的风格。对此，现在的了解非常有限，也不深入讨论了。

软件度量和工具概述

下面对软件度量做简单的介绍，并着重介绍功能点的度量方法和几个软件度量的工具。自下的部分决大部分是引用或翻译的，由于太多，所以不再注明出处。

软件度量的介绍

软件度量（Software Measurement）通过各种不同的量度(metric)对软件生命周期中的各个元素进行度量(Measure)，它能够为项目管理者提供有关项目的各种重要信息，同时也是进行大多评估活动的基础。

度量可能包括对工作量，软件规模，软件复杂度，错误，效率，可用性，变化，一致性等产品和过程属性进行度量。其中有些度量结果直接来自收集的数据而另外一些则要经过处理。

软件规模的度量单位常分为功能点和代码行数。这两种单位各有特点，而且有很多方法在两种度量结果之间转换，详情可参见【10】。这里也不进一步粘贴和翻译了。

复杂度的度量是通过对程序的静态分析，得出程序的环，耦合，分支，行数等特征的值。并以之来表达程序的复杂度。详细情况可参见【11】。这里就不进一步粘贴了。

由于软件的规模，对软件进行人工的统计分析一般比较困难，所以出现了很多软件自动分析和数据收集的工具。有非常简单的行数统计，也有对耦合度、环数等的复杂统计。后面会就使用过的和瞧见过的举例说明。

对软件计划的执行支持现在也有很多工具，象 microsoft 的 project。在对执行过程中收集的数据进行处理和表示也可以对软件的开发量化有所帮助。

下面就功能点的度量做较详细的介绍。

功能点的软件度量

1. 介绍

在软件度量领域存在不同的方法度量软件产品、软件开发过程和相关资源的特征。包括最近几年提出来的几种不同的点度量如功能点、特征点、对象点、全面功能点。这些方法使得及早地估计成本、工作量成为可能并能为开发过程的管理活动提供基础。因为这个主题越来越重要和点度量方法的多样性，我们将在第二章讨论下面方法的基本模型和特性：

DeMarco 的 Bang 度量（DeMarco's Bang Metric），

数据点（Data Points），

对象点（Object Points），

特征点（ Feature Points），

3-D 功能点（3-D Function Points），

IFPUG 功能点（IFPUG Function Points），

Mark II 功能点（Mark II Function Points），

全面功能点（Full Function Points）。

在第三章将从以下几个方面评价上面的功能规模度量方法：

对不同功能业务领域的适用性，

渗透程度和实践背景，

工具支持情况，

试验和验证，

标准化情况，

确认。

接着，本文会给出在某种特定情况采用哪种方法的建议。

因为讨论功能规模度量的共性问题非常重要，本文将在第四章讨论以下问题：

自动操作，

客观性／可靠性，

可兑换性，

加权因子值的意义，

重用的包含，

新技术问题

可能的度量工作产品

既然功能规模度量方面不断地连续变化，我们在第五章说明一些可供选择地最近发布的方法：

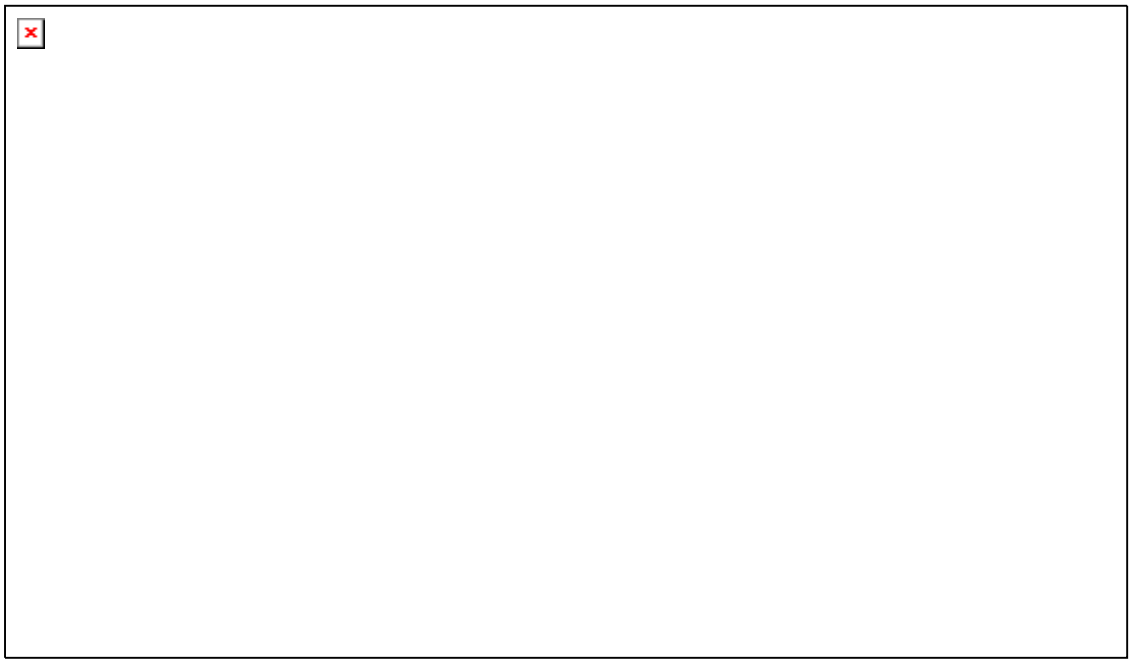
预言性的对象点（Predictive Object Points），

组件重用方法（Component Reuse Method），

构造点（Construction Points）。

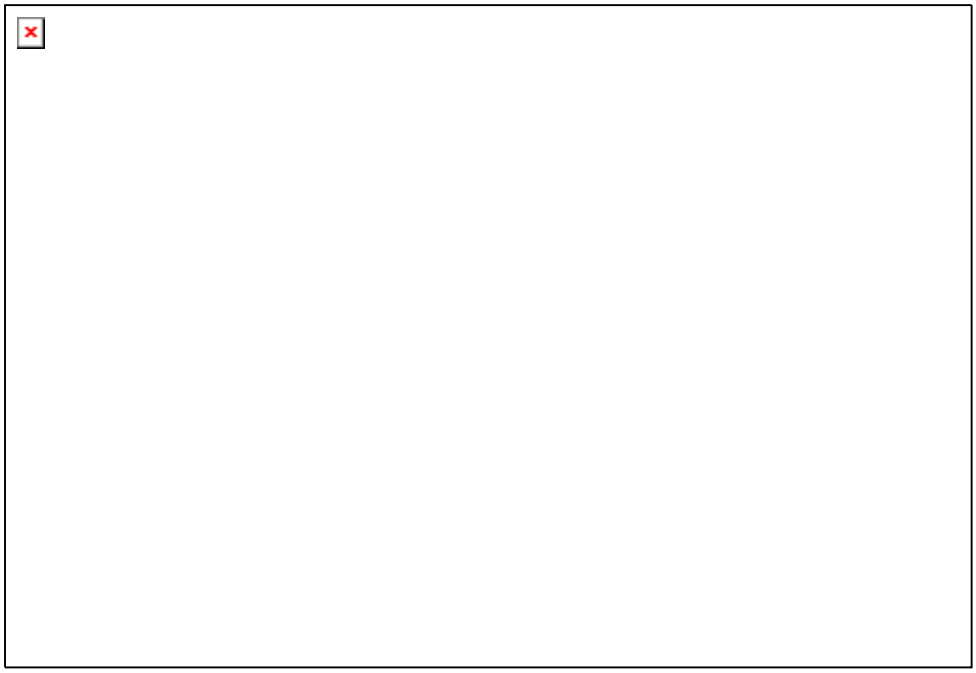
2.功能规模度量方法概述

自从工程方法和原理应用到软件开发过程，功能规模度量的重要性不断提高。图 1 表示软件规模度量的一般过程。如图所示，软件规模度量有两个基本阶段。映射阶段是应用概念和定义来代表软件，评价阶段是根据特定的规则和过程计算/度量萃取的元素。



Fetcke 做了功能规模度量一般结构提议结果的更进一步的调查研究[21]。

自从 1979 年第一个世界范围的功能点方法发表以来，许多针对原著变革、扩展的可供选择的方法被提出来。图 2 以时间顺序显示了包括那些在下面要详细描述的方法的演变重要阶段。方法之间的箭头表示各个扩展版本的相互影响。图中显示最近的方法是 COSMIC 全面功能点。既然早先的方法对这种方法存在大量影响，我们的研究会特别关注这种方法。



功能规模度量方法发展过程[08]

2.1 DeMarco 的 Bang 度量

开发者／机构 日期

Tom DeMarco, 1982

开发原因和隶属领域

DeMarco's 的咨询活动常常使得他面临比 MIS 系统更加复杂的软件系统。由于这个原因，他致力于系统和科

学软件领域[04]。

基本输入／模型

基本度量元素 [04]：

功能基元（functional primitives ），

修正功能基元（modified functional primitives），

数据元素 （data elements），

输入数据元素 （input data elements），

输出数据元素 （output data elements），

存储数据元素（stored data elements），

对象（实体） （objects (entities ），

关系（relationships ），

在状态转变模型中的状态（ states in a state transition model），

在状态转变模型中的转变（transitions in a state transition model），

数据符号（data tokens），

介入被保留的数据模型中的关系（relationships involving retained data models）。

根据在功能复杂的自然区别的分级计算，被度量的元素被加权衡量。

特性

DeMarco 的 Bang 制是 Albrecht 方法的扩展。他考虑了数据符号和状态转变，数据符号和状态转变常常和更复杂的软件如操作系统和电信系统相关联。 方法中权重非常主观 [01]。

现今意义

尽管是一个非常有趣的技术主意，但由于随后的 IFPUG 有更好的市场和更大的团体支持，这种方法在紧随 Albrecht 功能点方法后倒下，在今天的功能规模度量这种方法没有扮演一个重要角色并且只有少数使用者 [03] 。

推荐读物: 文献[04]

2.2 数据点

开发者／机构 日期

Harry Sneed, 1989

开发原因和隶属领域

数据点是为使得功能点方法适应现在软件开发需要而改变功能点方法产生的。它打算转移测量依据从功能到功能对象，分别到他们的数据表示法[22]。

基本输入／模型

通过数据模型和图形用户界面获得软件规模。数据点来自对下面几个方面的加权数量：

消息对象 （information objects ），

属性（attributes），

通信对象（communication objects），

输入数据（input data），

输入数据 （output data），

视图 （views）。

被度量的元素根据 8 个质量因素和 10 个项目条件进行加权衡量[05]。

特性

数据点方法是功能点方法一个变种。

现今意义

没有发现相关陈述文献

2.3 对象点

开发者／机构 日期

Sneed, 1994

开发原因和隶属领域

对象点是为适用于面向对象系统开发而出现的，根据 Sneed 的观点，传统的方法不能够适用于面向对象系统开发。

基本输入／模型

对象点是根据以下几个方面的加权量进行计算：

相应类的对象类型（object types respectively classes），

对象属性（object attributes），

对象关系（object relations ），

对象方法（object methods ），

消息（messages），

消息参数（parameters in messages），

消息源（message sources），

消息的目的地（message destinations），

重用百分比（ percentage of reuse）。

计算值根据 10 个影响因素进行加权[22]。

现今意义

没有发现相关陈述文献

特性

还有几个学者开发和介绍了适用于面向对象系统规模的方法。这些作为对象点或和对象点有关的方法已经常常被关注。如

对象点分析（Object Points Analysis (Banker, 1991)），

面向对象的功能点（Function Points with OO (Below, 1995)），

对象点分析（Object Points Analysis (Gupta, 1996) ），

用例和面向对象（ Usecases and OO (Fetcke, 1997)），

面向对象功能点（Object Oriented Function Points (Caldiera,1998) ），

增强对象点（Enhanced Object Points (Stensrud, 1998)）。

本文没有考虑这些方法。更详细的信息可以看 Abran 和 Desharnais 的文献综述[46]。

2.4 特征点

开发者／机构 日期

Capers Jones/Software Productivity Research, 1986

开发原因和隶属领域

这种方法的主要目的是为系统和实时软件提供更好的度量，因为 IFPUG 功能点方法最初是为 MIS 系统发明的[01]。

基本输入／模型

与 IFPUG 功能点方法比较，这种方法在下面几个元素外还添加了一个新的参数，运算法则。

输入（inputs），

输出（outputs），

查询（inquiries），

外部接口文件（external interface files），

内部逻辑文件（internal logical files ）。

权重被修改了，如根据 Jones 观点，逻辑文件的重要性减少了[01]。

特性

特征点是 IFPUG 功能点 4.0 版的扩展。对于 MIS 应用程序，特征点和 IFPUG 功能点的度量结果几乎一样[01] 。

现今意义

特征点方法被实验了很长时间。但没有充分的数据显示它能够稳定地进行使用。但是研究者们在他们的例证环境下说明了成功使用这种方法稳定性[07]。他的优点也是最主要的问题是运算法则的定义和权重[08][17]。这种方法今天不被 SPR 支持。

2.5 3-D 功能点

开发者／机构 日期

Boeing Computer Services, 1991

开发原因和隶属领域

3-D 功能点是一种覆盖系统软件（包括科学和实时领域）而得到与具体业务技术无关的度量方法[07]。

基本输入／模型

为了确定 3-D 功能点，需要度量以下几个方面：

数据(data (according to IFPUG 4.0))，

功能复杂度数字的(number of complexity of functions)，

控制状态的数字（系统状态和状态转变）(number of control statements (system states and state transitions) [05])。

特性

3-D 功能点识别了反映应用程序难题的 3 维（数据，功能，控制）。数据难题多是 MIS／商业软件的典型，而科学／工程软件是功能难题多，实时软件是控制难题多[07]。因而说，3-D 功能点适用于提及的软件方面。

现今意义

依照 Symons, 该方法一直被波音公司成功使用，但是不幸的是除了波音公司外没有别的详细信息发布[08] 。

推荐读物: 文献[09]

2.6 IFPUG 功能点

开发者／机构 日期

首先由 IBM 的 Albrech 在 1979 年发表 ，随后被 IFPUG 继承，现行版本是 4.1, 1999

开发原因和隶属领域

主要是为了克服代码行规模度量对语言的依赖性。目标领域是 MIS 系统。

基本输入／模型

如图 3 所示，依照 IFPUG 4.1，下列实体需要度量和分别加权：

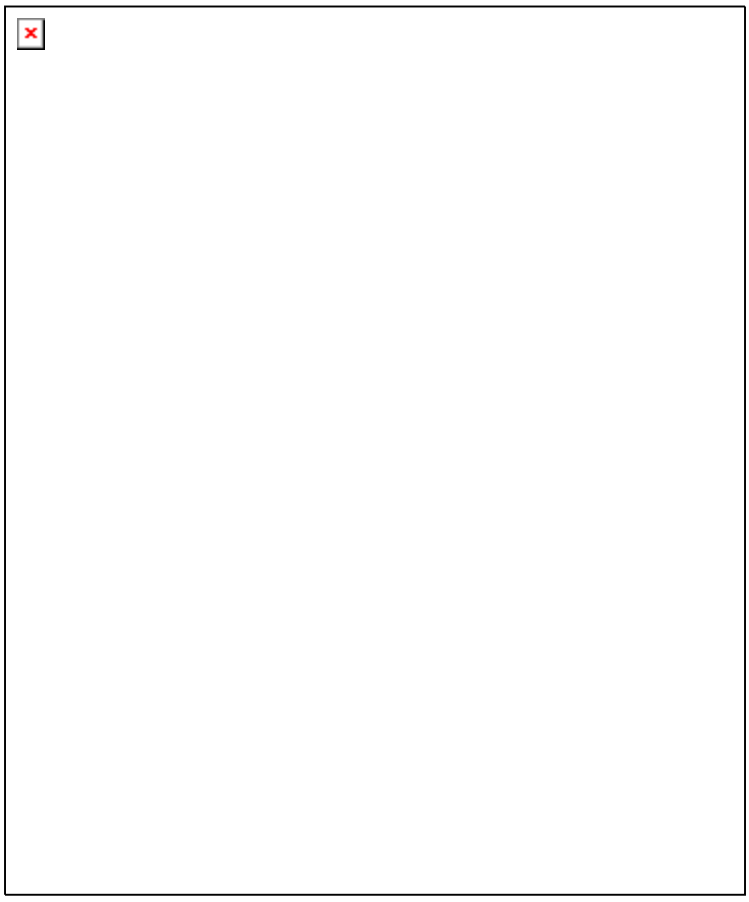
外部输入（external input），

外部输出（external output），

内部逻辑文件（internal logical file），

外部逻辑文件（external logical file ），

外部查询（external inquiries ）。



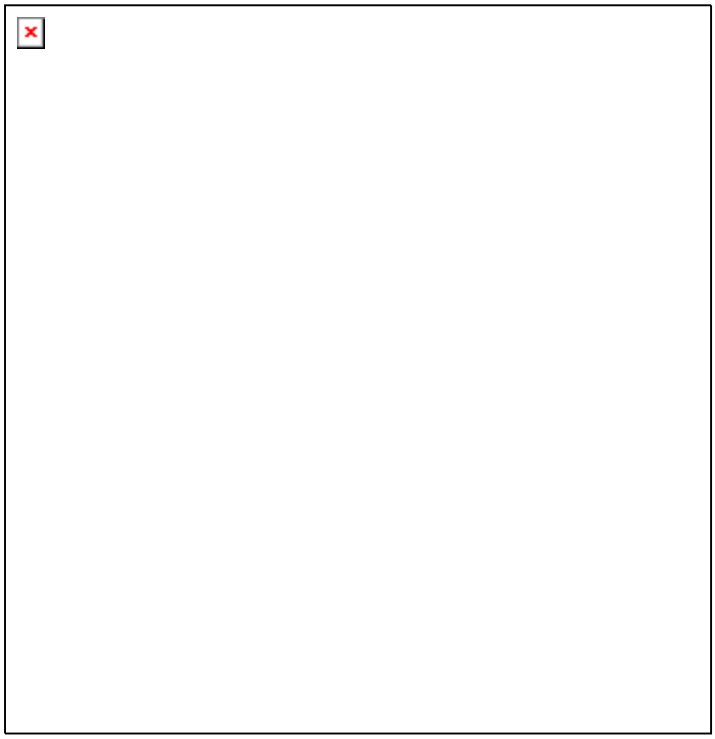
以 14 个影响因素为基础计算的加权因子值对上面计算值进行调整[11]。

特性

功能点方法是针对 MIS 系统开发的。虽然有些对其他软件领域（实时，面向对象）的案例研究，但始终存在这种方法是否满足这些软件领域的要求的讨论。本文有关这种软件规模度量方法的评论在下面的评价和共性问题讨论进行。

现今意义

功能点分析是一种应用最广的功能规模度量技术。IFPUG 功能点已经变为准标准。



2.7 Mark II 功能点

开发者／机构 日期

Charles Symons, 1988 现行版本: 1.3.1, 1998

基本输入／模型

依照 Symons ，Mark II 功能点目标是：

与 IFPUG 比较，Mark II 功能点减少对文件处理的主观性；

确保整个系统整体度量和部分计算的总和的结果一致，

比交给用户的功能，更多关注需要生产的功能的工作量 [14]。

基本输入／模型

如图 4 所示，Mark II 方法度量以下事物类别，他们要被一些因素加权：

输入（ input），

处理过程（processing），

输出（output）。

计算值通过可选的 19 个影响因素（其中 14 个和 IFPUG 一样，另外再加 5 个）加权[13]。

现今意义

Mark II 方法被英国专门使用

推荐读物: 文献[13], [14]

2.8 全面功能点

开发者／机构 日期

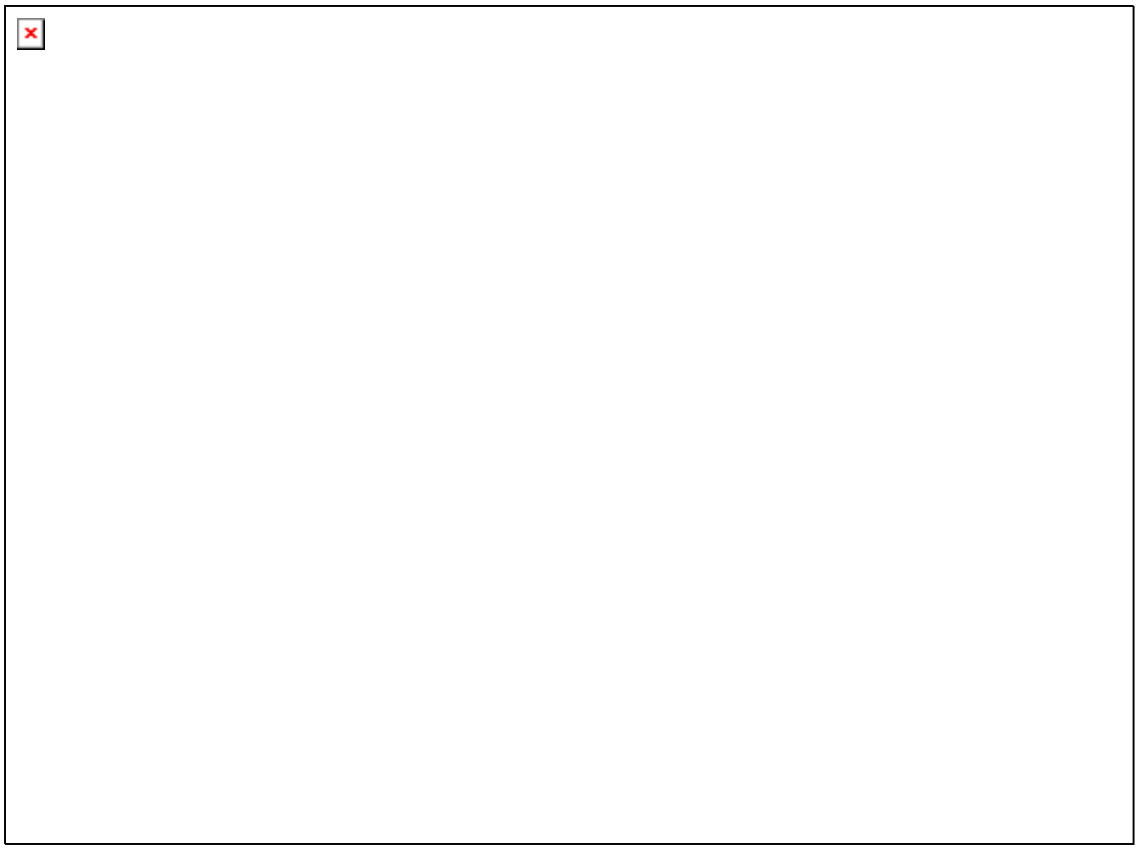
St.-Pierre at al., 1997 [06] 后来被 COSMIC 继承。

现行版本： 2.1, 2001

开发原因和隶属领域

全面功能点分析的目的是为了象满足 MIS 系统一样满足实时，技术和系统软件，因而企图克服如 IFPUG 功能点分析等早先方法应用类别的边界。优先权给予了实时系统。

.基于对数据流程表现系统规模的近似假设，这种方法适用于大部分 MIS，实时和操作系统软件[15][16]。



基本输入／模型

全面功能点方法通过分析用户功能需求决定软件规模。

一个非常重要的手段是综合层理念。可以分成几个层面表现软件的不同视域。图 5 显示了可能的变量，因而分布和复杂系统的同等度量成为可能。如图 7 所示，通过度量以下数据流类别决定功能规模：

进口（entries)，

出口（ exits)，

读取（reads)，

写入（ writes)。

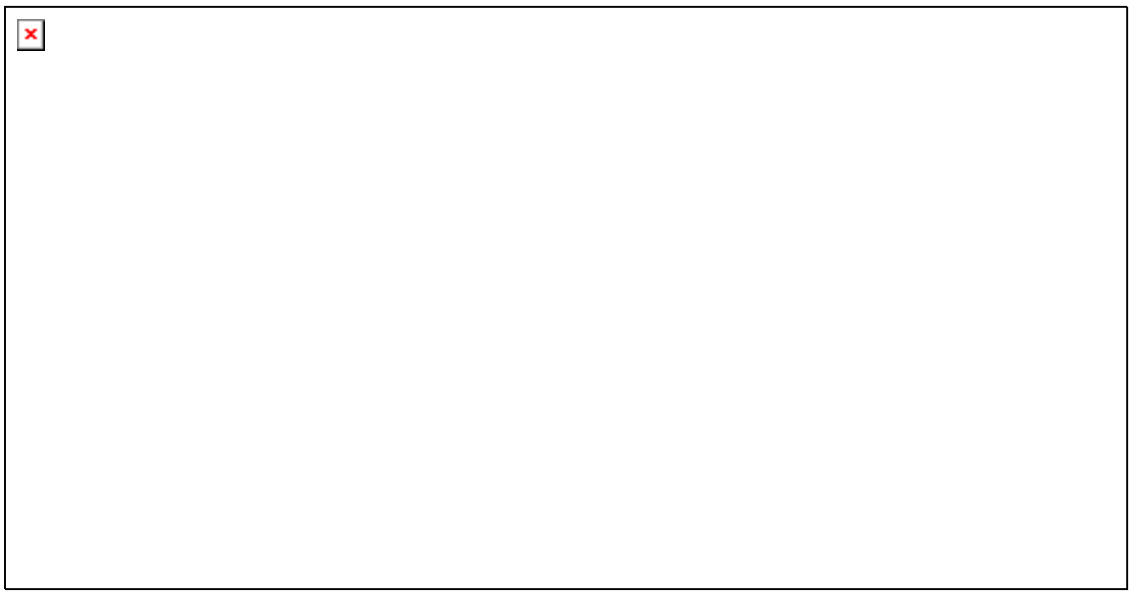
标准单位是 1 Cfsu(COSMIC functional size unit) ，它等同于一个数据流。系统规模是所有数据流的总和。

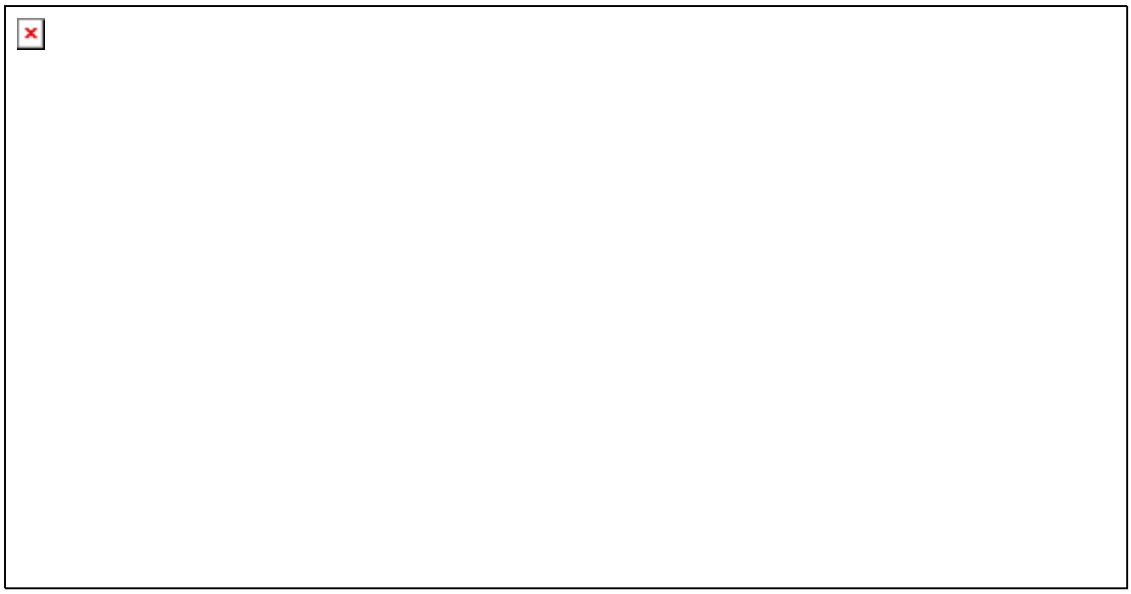
特性

虽然版本 1 是 IFPUG 功能点的一个扩展（图 6 显示了它的基本模型)。版本 2 挖出、捕获了以前的方法的基本理念 。这个版本的开发是由 COSMIC 的来自 8 个大家 40 个专家完成的。

现今意义

既然全面功能点版本 2 是一个新的的方法（所谓的现场试验在今年已经完成)，在不久的将来将会显示，要不这种方法将变得更加重要，要不 IFPUG 功能继续重要





。

工具介绍

很多工具支持自动或半自动的软件度量。包括产品度量、过程数据统计、数据分析和表格处理。

数据分析可以使用通用的统计软件比如 SPSS、NoSA 或 SAS。这里不作进一步讨论。

这里就静态代码分析，错误记录统计，过程支持记录的几个工具做一个介绍。

java 分析工具介绍

Java 源代码分析的工具很多如【3】【4】【5】。

其中 JStyle【4】不但对 java 的代码进行统计，而且对代码的风格提出建议，比如在构

造函数中不应该调用非静态函数。

JMetric【5】提供多种度量方法，包括：CK 度量方法，Aoki 度量方法，标准度量，基于 Java 语言的特定度量。

这些工具都提供多种数据表示的方法。包括表格和部分可定义的图形化表示方法。

通用的代码行数分析工具介绍

对多种语言的代码行数进行统计工具有一个是中国人开发的 CodeCounter【6】，还有 LineCounter 等等很多。

这些程序大都只对代码做简单的统计，包括代码行数，空格行数，注释行数和总行数。

错误跟踪和统计工具介绍

错误跟踪和统计的工具有很多，好像 rational 的 ClearQuest 是可以实现这一功能的。还有 ASTI 开发的 ITT 可以对错误进行统计并生成报表。其它还有一些比如 Bug/Defect Tracking Expert【7】

软件过程及产品统计工具介绍

ASTI 开发的 eprocess 提供对软件过程建模和执行的支持，同时支持多运行的统计和监控【8】。还提供一定的报表功能。

pspstudio 介绍

pspstudio 对 psp 的各种报表提供支持。同时对数据进行自动的处理【9】。

引用

【1】 psp 和 tsp 的相关资料可见 http://www.sei.cmu.edu/tsp/

【2】 关于精神分析法 http://www.apsa.org/

【3】 http://ivs.cs.uni-magdeburg.de/sw-eng/agruppe/forschung/tools/jmt.html

【4】 http://www.mmsindia.com/jstyle.html

【5】 http://www.asti.com.cn/products/JMetrics.htm

【6】 http://www.csdn.net/cnshare/soft/10/10789.shtm

【7】 http://www.bug-tracker-software.com/

【8】 http://www.asti.com.cn/products/E-Process.htm

【9】 http://www-cs.etsu-tn.edu/psp/

【10】 Roger S. Pressman ,Software Engineering A Practitioner’s Approach 5nd,WCB McGrawpHill

【11】 http://www.watertek.com/cekong-jishu2.htm