软件故障密度度量研究

蔡玲如[1]，陈 利[1]，郑斌峰[2]

(1. 华中师范大学 计算机科学系，湖北 武汉 430079；2. 华中科技大学 自控系，湖北 武汉 430070)

摘 要：主要介绍了软件质量的度量现状，同时对软件产品质量度量中的故障密度度量的重要性以及特点进行了分析，并针对影响度量结果的一些原因提出了解决的建议。

关键词：软件质量度量；故障密度；代码行；汇编语言

中图法分类号：TP311.53 **文献标识码**：A **文章编号**：1001-3695(2005)07-0066-02

Research of Defect Density Metric in Software Quality Metric

CAI Ling-ru[1], CHEN Li[1], ZHENG Bin-feng[2]

(1. Dept. of Computer Science, Central China Normal University, Wuhan Hubei 430079, China; 2. Dept. of Auto Control, Huazhong University of Science & Technology, Wuhan Hubei 430070, China)

Abstract: This paper discusses the current statement of software quality metrics, and analyses the importance and characters of the defect density metrics in software product quality metrics. Finally, aimed at some reasons that affect the measure results the author puts forward some suggests.

Key words: Software Quality Metrics; Defect Density; Lines of Code; Assemble Language

目前，软件的开发规模越来越大，复杂度也越来越高。软件开发正在经受一场危机：费用超支、生产率低下以及质量不高等问题。简而言之，软件开发处于失控的状态。软件工程就是在这样的背景下诞生的。软件工程的目的就是在合理开支的条件下准时交付质量满意的可靠的软件。可以说，软件工程的目标就是“质量、费用和时间”。软件工程通过管理对软件开发进行控制，而控制就离不开度量，所以软件质量的度量是软件工程的一个重要组成部分。

1 软件产品质量度量

软件的度量可以分为三类：软件产品度量、软件过程度量和项目度量。软件产品度量主要是对软件的大小规模、复杂度及功能等软件本身所具有的特征进行度量；软件过程度量是指在软件开发和维护过程中用来提高软件质量的一系列手段；项目度量则是描述了整个项目的特点和执行过程。一般来说，软件质量的度量更多的是软件产品度量和软件过程度量。但是，软件项目度量的参数，如开发者的人数、水平、计划和开发组的规模等都会对软件产品的质量产生一定的影响。上述三者之间存在一定的依存关系。表 1 分别从度量目的、度量对象、度量内容、度量结果、度量方法以及度量所解决的问题等方面对三者进行了分析。

软件产品的好坏从开发者与用户的不同角度可以分为两个方面，即软件产品本身的质量品质和用户的满意度。其中衡量软件产品本身的质量可以从两个方面进行衡量：故障密度(Defect Rate)和软件的失效间隔(Mean Time to Failure)。这两者之间既有区别又有联系。从应用的角度看，由于软件的失效间隔的数据收集较故障密度难以获取，因此，故障密度的应用较为广泛。故障密度在软件产品中较失效间隔更显得举足轻重。

表 1 软件产品质量度量、过程度量和项目度量

	软件产品度量	软件过程度量	项目度量
度量目的	获得高质量的软件	提高软件开发与维护的效率，降低成本	协调整个项目，促进项目与其他项目的高效运作
度量对象	软件产品本身	软件开发的各个阶段过程	项目各个阶段的人员、资金的投入与资源利用
度量内容	软件大小、复杂度、设计特点、性能、功能等	不用开发阶段故障的出现，故障移除对改善软件质量的有效性；维护的效率和质量；软件问题报告等	整个生命周期不同阶段软件的人力、物力投入；每个阶段的计划提交时间，软件开发的市场价值等
度量结果	平均运行时间、软件故障密度、用户问题和用户满意度等	基于开发阶段的故障移除模式和故障移除的有效性，测试阶段故障密度和故障出现模式等	对软件的整个生命周期有一个完整的计划
度量方法	LOC, PUM, CUPRIMDSO, FURPS, NSI, FC 等	DRE, BMI 等	专家评估、市场调查等
解决的问题	质量	质量、费用	质量、费用、时间

2 故障密度度量

2.1 故障密度

讨论故障密度，必然要对故障和基数范围进行定义。我们可以这样笼统地定义故障：故障就是在软件发布以后出现的只要不是引起系统崩溃的错误。在质量范畴里，故障密度(故障率)定义为

$$\text{故障密度} = (\text{故障的数量}/\text{错误出现的可能}) \times K$$

其中，K 表示从实验或调查中获得的一个常数。值得注意的是，故障密度是一个大概的数值。因为错误出现的可能性是不可知的；其次，我们观察到的故障不一定就是产品的所有故障。

2.2 软件工程中故障密度的特点

在软件工程中，软件的故障密度通常定义为

收稿日期：2004-05-22；修返日期：2004-09-08

故障密度 = 特定时间段里出现的故障数/每千行代码

故障密度的分子——特定时间段里出现的故障数的度量问题。软件类型不同,这个特定时间段可以是软件产品投入市场后一年或几年甚至是软件的整个生命周期。不同的软件开发组织定义的软件产品周期是不一样的。实践经验证明,大部分软件其 95% 的故障是在投放到市场四年后出现的,如 IBM 定义软件产品生命周期就是四年。除此之外,软件产品由于其本身的特点,如同一个错误可能出现多次而引起不同的故障现象。所以,故障密度的度量结果不是也不可能是一个精确的数字。我们只能是把软件产品的故障数近似为我们观察得到的数据。

故障密度的分母——代码行的度量问题。据度量源代码程序长度的这个属性,一个可选的度量就是把代码行属性定义(映射)为可执行代码的语句数;另外一个可选的度量就是把代码行属性定义为其在计算机中存储字节数。但是,这只是我们在定义问题非常清楚的情况下不严格的术语说法。如果只有两种可选的定义方法的话,它是可行的。但是事实上存在很多定义代码行的方法。早期大多数程序都是汇编语言编写的。汇编语言的语句和结构都比较简单,代码行数与计算机中存储的字节数基本是一对一的关系,因此,代码行度量的问题不大。随着高级语言的出现和发展,数据结构的发展,面向对象概念的提出,构件的研究,给编程人员提供了广阔而又自由的编程空间。源代码与计算机物理上存储字节数的一对一的模式被打破。物理行与程序结构之间的差异,高级语言之间的差异给代码行度量带来了很大的分歧。即使使用同一种语言的程序,由于计数工具的方法和算法不同也可能带来不同的度量结果。

3　故障密度度量存在的问题

度量的目的在于给度量对象一个定量的标准,使得不同的度量对象之间、度量对象与计划标准之间可以进行比较,从而对软件进行评价或控制、调整。在软件产品度量中,故障密度度量起着至关重要的作用。代码行概念提出的初衷是为了衡量软件规模的大小。由于当时软件和软件工程的发展还不是很快,代码行定义比较单一,代码行度量问题不大。但是随着软件危机的出现,软件的失控问题使得人们不得不采取一些度量手段来对软件质量进行控制。由此导致在故障密度定义中,一些概念由于时代的发展而产生了一些分歧,随之出现的相关问题也随着故障密度度量方法的广泛应用而日渐突出。

3.1　代码行度量的标准问题

代码行的度量问题首先是对代码行概念的定义,也就是说,代码行是对什么代码进行衡量。如果从前面提到的两种映射方法来看的话,就比较简单。但实际上,在代码行的度量上存在着许多不同的度量方法。

Jones(1986) 将代码行的度量分为六类:①计算可执行的代码行;②计算可执行的代码行 + 数据定义;③计算可执行的代码行 + 数据结构 + 注解;④计算可执行的代码行 + 数据结构 + 注解 + 程序控制语言;⑤计算在输入屏幕上显示的程序行;⑥计算所有的代码行直到某一个终止符。

从代码行与物理上存储空间的关系上,我们可以把源程序中的代码分为三种:①程序控制语句,一般来说它与物理上的存储空间是一对一的关系;②模块语句,如某一个子程序或构件,它与物理上的存储空间是多对一或一对多的关系;③注释语句,它是用来帮助程序员进行程序说明的语句,在程序执行过程中是不占用任何物理存储空间的。

度量的目的从某种意义上讲可以分为两种,即为了度量软件的大小和为了衡量软件开发的开销。因此,代码行的度量标准根据其目的也分为两种,即软件代码行度量和开销代码行度量。表 2 为代码行度量标准模型。其中公式中的 E 表示程序控制执行代码的度量结果; M_i 表示对第 i 个模块大小的度量结果; k_i 表示第 i 个模块的应用概率或者使用频率; C 表示注解语句的度量结果。

表 2　代码行度量标准模型

代码行度量标准	度量代码类型	度量计算公式	适用度量目的	相对应的 Jone(1986) 分类
软件代码行度量(Software LOC Metric)	程序控制语言、模块语言	$SLOC = E + \sum_{i=n}^{i=0} M_i \times k_i$	衡量软件大小,与软件运行时间相关	1, 2, 6
开销代码行度量(Cost LOC Metric)	程序控制语言、模块语言、注解	$CLOC = E + \sum_{i=n}^{i=0} M_i + C$	衡量软件开销代价,与软件开发的时间与人才资源相关	3, 4, 5

3.2　高级语言与汇编语言的转换比率问题

不同高级语言的源代码与物理存储字节大小之间存在不同的映射关系。根据 Jone 于 1992 年调查研究得出,大部分源代码度量大小与物理存储字节空间大小之比最大可以达到 500%,平均为 200%。对于面向商业的通用语言 COBOL,结果却正好相反,物理存储空间度量大小是源代码度量大小的 200%[2]。表 3 给出了对同一个项目不同语言版本运用 LOC 进行度量的结果。两种语言的平均转换率如表 4 所示。

表 3　同一个项目不同语言版本的 LOC 度量结果对比

语言	汇编语言	C 语言	CHILL	PASCAL	PL/I	Ada83	C ++	Ada95	Smalltalk
语言等级	1	3	3	4	4	5	6	7	15
LOC 度量结果	375000	190500	157500	136500	12000	106500	83500	73500	31500

表 4　某个项目高级语言与汇编语言的平均转换率

语言	汇编语言	C 语言	CHILL	PASCAL	PL/I	Ada83	C ++	Ada95	Smalltalk
语言等级	1	3	3	4	4	5	6	7	15
LOC 度量结果	375000	190500	157500	136500	120000	106500	83500	73500	31500
汇编语言平均转换率	1	1.97	2.38	2.75	3.13	3.52	4.49	5.10	11.9

对于不同语言版本的项目,我们如何对它们进行统一的度量比较呢?首先,应该有一种基准语言,而且在所有的高级语言与基准语言之间应存在一个平均转换率。找到这个平均转换率,我们就可以对不同语言的度量结果进行统一。由于汇编语言与物理上的存储空间的映射关系比较统一,因此,可把汇编语言作为一种基准语言。表 4 通过部分测试数据给出了相对表 3 高级语言与汇编语言之间的一个平均转换率。由于测试数据有限,这个结果仅仅是作为一个参考数据,平均转换率有待进一步的数据积累与研究。但是,从表 4 我们可以看出,语言的等级越高,与汇编语言的转换率就越大。(下转第 70 页)