基于分形原理的软件维护性评估

郝学良 朱小冬 刘 丽
(军械工程学院装备指挥与管理系 石家庄 050003)

摘 要 针对软件维护性评估难题,从软件结构复杂度和软件开发维护过程两方面着手,分别对软件维护性进行了定性和定量评估。分析了软件模块结构的空间域自相似性及软件生命周期内的时间域自相似性,提出了基于分形维数的软件维护性定性评估,实现软件维护性评估的定性要求和控制。以维护工作量作为软件维护性核心参数,结合自顶向下评估方法,提出了时间域内软件维护性定量评估。通过虚拟维修训练系统软件美腾2.0到3.0版本升级维护实例,对方法的有效性进行了验证。

关键词 自相似性,分形维数,维护性定性评估,维护性定量评估
中图法分类号 TP311.5 **文献标识码** A

Software Maintainability Evaluation Based on Fractal Theory

HAO Xue-liang ZHU Xiao-dong LIU Li
(Department of Management Engineering, Ordnance Engineering College, Shijiazhuang 050003, China)

Abstract Aimed at the software maintainability problem, qualitative and quantitative evaluation were studied, considering two aspects of structure complexity and software process. Spatial domain self-similarity among software module and temporal self-similarity between software development and software maintenance were analyzed. Based on fractal dimension, software maintainability qualitatively evaluation method was presented to ascertain software qualitative maintainability requirement and management. Combined with top-down maintenance method, temporal software maintainability quantitative evaluation method was put forward and maintenance workload was validated by a real example of software system.

Keywords Self-similarity, Fractal dimension, Software maintainability qualitative evaluation, Software maintainability quantitative evaluation

调查表明:软件维护工作量占软件整个生命周期工作量的70%以上。Dean Morton在一项研究报告中指出Hewlett-Packard公司一年有4千万到5千万代码行处于维护当中,公司60%~80%的人员都投身于软件维护活动当中。据中航工业统计,2005年至2008年,在29项任务的217款软件,累计327万余行源代码测评中,共发现1万5千7百多个缺陷,为了保障装备的完备率,就必须对这些缺陷进行维护。软件维护性问题是影响软件密集型装备完好率的关键要素。Spinellis在《代码质量》中提出21世纪的程序员首先应学会的不是编写代码,而是理解代码。这也说明,在当前软件不断泛化的背景与软件工业化生产的趋势下,传统的从零开始的软件开发活动已很少存在,软件维护活动占有了举足轻重的地位[1]。软件维护性定性评估主要以软件复杂度为参数,比较成熟的理论有Halstead度量法[2]和McCabe[3]度量法,前者注重的是规模复杂度,后者注重的是结构复杂度,这两种方法均取得了良好的维护性评估效果。由于软件规模和软件结构本身并不能完全割裂开来,因此这两种复杂度度量方法各有利弊,对两种度量法的评价也一直存有争议。软件维护性定量评估对于合理安排软件维护资源、制订维护计划具有重要作用。目前,关于软件开发工作量评估的研究方法比较多:以Boehm提出的COCOMO(Constructive cost model)模型为代表的结构型成本估算法对软件开发的结构进行分解,从而估算出软件开发工作量;Putnam提出的动态多变量回归模型,对影响软件开发工作量的影响因素进行回归分析,可以估算出软件开发所需的时间。此外,在软件工程实际中,经常采用的历史项目类比法、专家估算法等也对软件开发工作量的评估进行了有意义的尝试。但是,由于当前对软件维护的历史数据收集不够充分,缺乏可利用的有效历史维护信息,对于软件维护性定量评估方法的研究目前相对欠缺。软件维护性定量估算是一门科学,其实质是根据历史数据、人员经验、行业惯例,以及项目要求来预测软件资源、费用及工期[4]。

目前利用分形理论对软件维护性评估的相关研究较少。季方[5]利用软件的自相似性对软件缺陷数进行估计,通过分析小模块中的缺陷数来预测整个软件系统的缺陷数,从分形的角度试图对软件的缺陷产生机理进行分析,实现了软件质量的定量化预计。姜林[9]根据分形理论中的标度不变性描述

本文受总装备部预研项目资助。
郝学良(1984-),男,博士生,主要研究方向为软件保障,E-mail:hxl056700163.com;朱小冬(1965-),男,硕士,教授,主要研究方向为软件保障、装备保障;刘 丽(1980-),女,博士生,主要研究方向为软件保障,E-mail:wenhuaxiang@163.com。

了软件程序的分形特征，通过计算分形复杂度来反映程序结构复杂度，提出了一种新的软件复杂度计算思想。

本文通过分析软件自身模块内部自相似性，用盒子维数来表征软件复杂度，实现了软件维护性定性评估；通过分析软件维护过程与软件开发过程的相似性，参照软件开发工作量估算方法，建立软件维护性定量评估的系统模型，对于指导软件维护工作的组织决策，以及对软件维护性实现定量评估与控制都具有重要意义。

1 分形理论基本原理

美国数学家曼德布罗特于1975年首先提出了分形的概念。曼德布罗特指出，自然界中存在一类复杂的几何对象，这些对象具有部分与整体以某种方式相似的形体称为分形，且难以用传统的欧氏几何进行描述[5]。自相似性和标度不变性是分形理论的两个重要性质。自相似性是指一个系统的某种结构或过程的特征从不同的空间和时间尺度来看都是相似的，或者某系统或结构的局域性质或局域结构与整体类似：如在空间域太阳系的构造与原子的结构之间在某些方面具有奇特的一致性，构成漫长海岸线的大小半岛及港口之间同样存在相似性；在时间域内，个体生命的生衰病死、一年四季的往复轮回，以及软件自身的生命周期等。

有些系统往往存在一种无限嵌套的自相似的几何结构或者周期分布，而且结构本身的某些定量性质与该系统的具体内容无关，著名的Logistic方程 $x_{n+1}=Yx_n(1-x_n)$ 就是其中一个典型[6]。同样的，在软件开发过程当中，系统与子系统、大模块与小模块、包与类、方法与函数，以及软件开发过程与软件维护过程之间均存在自相似性，因此，理论上可以通过借鉴开发过程中开发工作量的评估，对软件维护性进行定量评估。

2 软件维护与开发过程的相似性分析

2.1 软件维护过程

评估软件维护性可以从软件维护过程入手进行估算，即将过程分解为相对较小的活动或任务，再估算每个任务所需的工作量。软件维护的影响因素繁杂，软件维护过程难以用一种重复规范性的程式化过程进行管理。但是，同时对于大型软件项目，如果随意进行更改，不但难以完成维护任务，甚至会导致更加严重的后果。因此，软件维护过程必须在组织标准化和一致性要求与个人灵活性的要求之间进行权衡，根据软件维护人员及其任务的需要进行动态调整[7]。软件维护过程描述如图1所示。

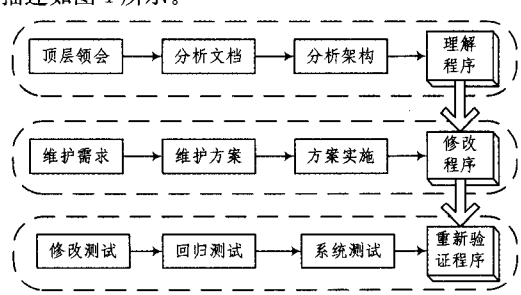

图1 软件维护过程模型

建立了软件维护过程活动模型之后，便可以估算完成每一个软件功能的过程所需的工作量，从而可以对软件维护性进行评估。

2.2 软件生命周期的自相似性分析

传统的软件工程理论一般将软件维护定义为软件交付后所进行的纠错、完善和扩充等软件更改活动。如果把软件维护作为一个独立的阶段来分析，其与软件开发过程具有较大的相似性。

根据软件工程实际，可以发现软件生命周期是一个持续开发的过程，极端一点说，软件从第一行代码就已经开始维护。软件维护与软件开发的关系如图2所示。

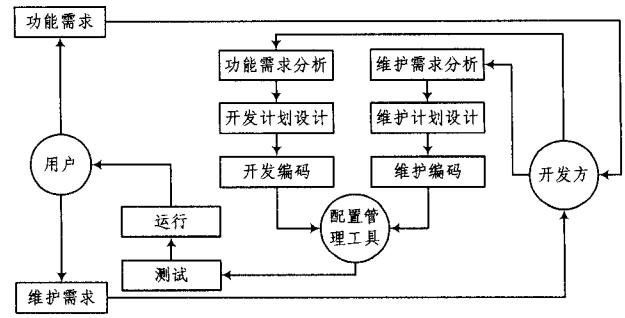

图2 软件维护与软件开发的关系图

实际的软件维护活动与软件开发十分类似，表现在以下几个方面。

(1)形式相似。软件开发过程是指软件开发和生产的过程，一般可分为需求分析、设计、编码、测试等阶段，同样软件维护过程也有相似的阶段，如软件维护活动中的维护需求稳定度统计也是一种需求分析，修复性维护中的设计软件修改计划与软件开发阶段的设计活动类似，修改代码、重新验证程序与软件开发阶段的编码测试也十分接近。

(2)对象相似。软件开发的对象是软件，软件的功能来自用户的需求，所以软件开发的过程其实就是将用户需求落实到软件产品中的过程。软件维护的对象同样也是软件，如扩充性维护是想要对一个已有的软件进行功能扩充，即想将其开发为另一个状态的软件，而在软件的开发过程中这种扩充活动也比比皆是。这种需求的出发点同样也来自用户，所以软件维护的过程也是将用户需求落实到软件产品中的过程。

(3)主体相似。软件开发的主体是软件开发人员，或软件项目组、软件公司。软件维护的主体理论上应该不仅仅是开发人员，应该还包括使用人员，或者专门的第三方维护机构，但是当前的实际情况是除了较大的软件公司外，几乎都未建立或保持一个正式的软件维护机构，大多数的软件仍然是谁开发谁维护的模式，而用户对软件维护活动只负责提出需求和测试验收。

(4)方法相似。软件危机的爆发让软件开发的工程化原则已经日益深得人心，而软件维护成本负担的沉重同样也让软件维护工程化管理显得尤为重要。配置管理工具正是贯穿软件开发与软件维护的一个重要的桥梁。配置管理作为软件过程管理的重要工具，不仅可以保证软件开发活动高效、准确地完成，更起着保证软件具有良好维护性的作用。

为适应软件的不同特点和使用阶段，必须根据软件的具体情况在软件开发的整个生命周期中对软件维护性进行评估[5]。

3 基于自相似性的维护性评估技术

软件系统是由若干个结构化的模块构成的，只有理解每个模块后才能够理解整个软件系统。复杂度作为软件最重要的质量属性之一已被广泛认同，软件自身的复杂性与功能属性成为不可分割的两个部分[8]。对于每个模块来说，又可以将其分解为若干个更小的模块，从而又需要理解这些更小的模块[9]。在软件开发过程中，需要理解需求、分解需求，将功能设计、功能实现等最终落实到程序模块上。在软件维护过程中，对于扩充性维护和适应性维护，同样需要完成功能模块的实现。从模块复杂度的角度来看，一个模块的细节越多，在开发过程当中就需要花费越多的时间，同样的在维护阶段理解就越困难。

3.1 基于软件结构空间自相似性的维护性定性评估

程序复杂度指程序模块内部的复杂性，它直接关联到软件维护性的度量。关于程序复杂度最有代表性的是 Halstead 度量法和 McCabe 度量法，前者注重的是规模复杂度，后者注重的是结构复杂度。利用程序内部自相似性，将循环体中的内容看成一个盒子[8]。

程序中盒子总数可以反映程序结构上的复杂程度，它类似于 Mccabe 度量中的圈复杂度，并克服了 Mccabe 没有考虑语句个数对复杂性影响的不足。取循环体内的代码行数作为盒子的连长大小 δ_i，数出程序模块内盒子的数目 $N(\delta_i)$，求出相应的 $\ln\delta_i^{-1}$ 和 $\ln N(\delta_i)$，根据数据对 $(\ln\delta_i^{-1}, \ln N(\delta_i))(i=1, 2, \cdots, n)$ 用最小二乘法拟合直线，并在双对数坐标系 $\ln N(\delta)-\ln\delta^{-1}$ 中画出该直线，直线的斜率即为功能模块的分形维数为 d[10,11]。

可以统计出某类软件中子模块占程序的比重，为其确定一个权值 x_i，每个子模块相似的模块给出的分形维数 d_{ij}，子系统模块的复杂度为 r，从而得到该软件子系统复杂度的分形维数 D[9]：

$$D=\begin{bmatrix}A_1\\A_2\\\cdots\\A_n\end{bmatrix}=r\begin{bmatrix}d_{11} & \cdots & d_{1n}\\d_{21} & \cdots & d_{2n}\\\cdots & \cdots & \cdots\\d_{n1} & \cdots & d_{nn}\end{bmatrix}\begin{bmatrix}x_1\\x_2\\\cdots\\x_n\end{bmatrix} \tag{1}$$

通过对两类软件的分形维数进行比较，便可以确定维护难度系数，从而可以对软件维护性评估提出定性要求，指导维护性改进和软件维护工作。

3.2 基于软件开发过程时间域自相似性的维护性定量评估

维护工作量是表征软件维护性的核心参数。正确地对软件维护工作量进行评估是一个软件企业最有价值的财富[12]。尽管对于维护工作量的研究持续了几十年，无论从理论上还是实践上都取得了丰硕的成果，但是仍然不能实现准确的评估[13]。基于合同式的传统软件生产模式当中，不当的变更甚至会导致软件整个项目的失败[14]。实践证明，传统的维护性定量评估方法已不适用于当今软件产业，对于维护工作量的评估需要采用新的办法[15]。对软件系统结构图进行分析，由顶向下可以发现软件在结构上总是以树形结构为处理构成，对其中某个单元再进行放大，仍然表现出这种结构，直到分解至基本模块为止。从软件的顶端至基本模块之间都会表现出这种相似性，这意味着可以满足分形的标度不变性。自顶向下的评估思想是从项目的整体出发，进行类推。即评估人员根据以前已完成项目的总工作量，来推算将要开发项目的总工作量，然后按比例将它分配到各开发任务单元中，再来检验是否满足要求。结合 Boehm 给出的软件开发工作量评估方法，利用软件自相似性，对“虚拟维修训练系统(美腾)2.0”软件维护工作量进行评估，结果如表 1 和表 2 所列。

表 1 软件开发各阶段工作量的评估

软件：美腾 2.0	更新情况	开发人员：美腾开发小组	日期：2010.7.1
阶段	任务	工作量分布(小时)	小计(小时)
开发计划与需求	软件需求定义	3	4
	开发计划	1	
产品设计	产品设计	4	8
	初步的用户手册	3	
	测试计划	1	
详细设计	详细功能描述	4	12
	数据定义	4	
	测试数据及过程设计	2	
	正式的用户手册	2	
编码与单元测试	编码	6	10
	单元测试结果	4	
组装与联合测试	编写文档	4	9
	与联合测试	5	
总计			43

表 2 软件维护各阶段工作量的评估

软件：美腾 2.0	更新情况	开发人员：美腾开发小组	日期：2013.7.1
阶段	任务	工作量分布(小时)	小计(小时)
维护需求	确认维护需求	6	6
分析设计	分析软件问题报告、修改建议	5 8	13
详细设计	设计及分类鉴别	5	5
代码更改	安排修改程序计划	6	6
回归测试	清除数据及文档的副作用	10	10
组装与系统调试	系统确认和测试	10	10
总计			50

表 3 美腾 3.0 维护工作量的评估

软件：美腾 3.0	更新情况	开发人员：美腾开发小组	日期：2014.6.2
阶段	任务	工作量分布(小时)	小计(小时)
维护需求	确认维护需求	5	5
分析设计	分析软件问题报告、修改建议	7 4	11
详细设计	设计及分类鉴别	6	6
代码更改	安排修改程序计划	8	8
回归测试	清除数据及文档的副作用	9	9
组装与系统调试	系统确认和测试	9	9
总计			48

在美腾 2.0 到美腾 3.0 的升级维护过程中，对表 3 中的数据进行了验证，结果证明其具有较好的一致性。从表中数据可以看出，维护各阶段的工作量要略大于开发阶段的工作量，这是因为在初始阶段个别维护需求不是很明确，同时也说明当前软件维护所面临的严峻形势。特别是在前期需求阶段设计不合理，后期也存在需求变更的情况下，会导致实际的维护工作量增大。

结束语 当前，软件质量成为学术和产业界共同关注的热点之一。软件维护性作为一个重要的质量度量指标具有较

(下转第 503 页)

实例中 $e7$ 需要在特定事件序列执行后才能执行，而在维护后的映射表 CM 中，最后一行已经成功记录下了 $e7$ 和执行它所需要的事件序列。之后在生成测试用例时这一信息就可以利用，而不再会根据最短路径的算法生成无法执行的测试用例。因此 Mapping with Nodes 算法已经达到了预期的效果。

最后将 Construct Mapping 算法和 Mapping with Nodes 算法做对比，通过 OME* 模式对实例进行完整的测试，两个算法对映射表的维护效果分析如表 6 所列。

表 6　算法对比表

算法名	Construct Mapping	Mapping with Nodes
测试标准	边	点
调用次数	9	3
平均用时(MS)	15	3
映射表行数	27	7
事件序列平均长度	2.22	2

从表 6 中可以发现，由于 Construct Mapping 算法以边为标准，它对映射表的维护次数和运行时间远大于 Mapping with Nodes 算法，而映射表的体积也更庞大，表现在映射项目更多，而且映射的事件序列更长，但这也说明了之后会生成更多的测试用例。因此，在实际测试时如果希望能在更短的时间和消耗更少资源的条件下完成测试，应该以点作为测试标准，使用 Mapping with Nodes 算法。如果对测试的完整性要求更高，则应该以边作为标准使用 Construct Mapping 算法。

结束语　本文以 GUI 测试下的 OME* 模式作为基础，研究了 EFG 模型元素与到达路径映射表的维护算法，提出了以 EFG 点作为测试标准时映射表的维护算法 Mapping with Nodes，实验证明它能记录下执行特定事件所需的事件序列，且消耗的时间和资源更少。未来打算在确保测试高效性的同时，进一步提高 OME* 模式的兼容性，使其适用于各个覆盖标准和测试平台。

参 考 文 献

[1] Shehady R K, Siewiorek D P. A method to automate user interface testing using variable finite state machines: Fault-Tolerant Computing[C]// Twenty-Seventh Annual International Symposium on Digest of Papers, 1997(FTCS-27). Seattle, WA, USA: IEEE, 1997: 80-88

[2] Reza, Hassan, Endapally S, et al. A model-based approach for testing gui using hierarchical predicate transition nets: Information Technology[C]// Fourth International Conference on Las Vegas, 2007(ITNG'07). NV: IEEE, 2007: 366-370

[3] Memon A M, Nagarajan A, Xie Q. Automating Regression Testing for Evolving GUI Software[J]. Software Maintenance, 2005, 17(1): 27-64

[4] Vieira, Marlon, Hasling B, et al. Automation of GUI testing using a model-driven approach[C]// Proceedings of the 2006 International Workshop on Automation of Software Test, 2006. New York: ACM, 2006: 9-14

[5] Memon A M, Banerjee I, Nagarajan A. GUI Ripping: Reverse Engineering of Graphical User Interfaces for Testing[C]// Proc. 10th Working Conf. Reverse Eng, 2003(WCRE' 03). Canada: IEEE, 2003: 260-269

[6] Memon A M. A Comprehensive Framework for Testing Graphical User Interfaces[D]. Pittsburgh: Department of Computer Science, University of Pittsburgh, 2001

[7] Nguyen, Bao, Memon A M. An Observe-Model-Exercise* Paradigm to Test Event-Driven Systems with Undetermined Input Spaces[J]. IEEE Transcations on Software Engineering, 2014, 40(3): 216-234

(上接第 499 页)

高的研究价值。同时，软件的复杂性本质和趋势也使得这一领域的研究困难重重。相似性原理可以对自然界中的很多复杂性问题进行建模，能够为软件维护性评估提供一种新的途径，在接下来的研究中，将进一步结合工程实际，对该方法进行完善。

参 考 文 献

[1] 褚文奎，张凤鸣，樊晓光. 综合模块化航空电子系统软件体系结构综述[J]. 航空学报，2009，30(10)：1912-1917

[2] Halstead M H. Elements of Software Science[M]. New York: Elsevier North Holland, 1977

[3] Mccabe T J. A complexity measurement[J]. IEEE Transaction on Software Engineering, 1976, 2(4): 302-308

[4] 王越，陈旭，曹长修. 应用软件系统维护过程成本的研究[J]. 计算机工程，2001，27(7)：65-66

[5] 季方. 基于分形理论的软件缺陷数的估计[J]. 现代商贸工业，2008，20(3)：277-278

[6] 张济忠. 分形[M]. 北京：清华大学出版社，2011：9-111

[7] 朱小冬，王小巍. 基于 CMM 的软件维护过程研究[J]. 计算机工程与应用，2005，29：66-69

[8] 石柱，郑重. 软件可靠性度量实例研究[J]. 系统工程与电子技术，2011，33(1)：233-236

[9] 姜林，艾波，漆涛. 分形理论在软件复杂度中的应用[J]. 计算机应用，2010，30(10)：2729-2734

[10] 孙洪良. 分形几何与分形插值[M]. 北京：科学出版社，2011：64-68

[11] Pandey, Poonam. Analysis of the Techniques for Software Cost Estimation[C]// 2013 Third International Conference on Advanced Computing and Communication Technologies(ACCT). 2013: 16-19

[12] Hihn J, Tregre G. Assuring software cost estimates: Is it an Oxymoron[C]// 2013 46th Hawaii International Conference on System Sciences. 2013: 4921-4929

[13] Susan A, Gabri B. Software development: why the traditional contract model is not fit for purpose[C]// 2013 46th Hasaii International Conference on System Sciences. 2013: 4842-4851

[14] Streiffert B A, Francis L K, Smith B D. Using Modern Mehtodologies with Maintenance Software[C]// Space Ops 2014 Conference. 2014: 124-128

[15] Trujillo A, Gregory I M. Wetware, Hardware, or Software Incapacitation: Observational Methods to Determine When Autonomy Should Assume Control[C]// 14th AIAA Avitation Technology, Integration, and Operations Conference. 2014: 322-324