软件维护中程序理解的应用与研究

丁剑洁,鱼 滨,侯 红
(西北大学 信息科学与技术学院,陕西 西安 710069)

摘 要:程序理解是软件维护中的一项重要活动。从软件维护和认知学的角度分析了程序理解的任务,描述了几种主流的程序理解的模型。最后针对影响程序理解的因素提出了相应的对策,为提高维护效率提供帮助。

关键词:程序理解;软件维护;理解策略;Pennington模型;Soloway模型
中图分类号:TP311.53 **文献标识码**:A **文章编号**:1673-629X(2007)04-0218-04

Research and Application of Program Understanding in the Software Maintenance

DING Jian-Jie, YU Bin, HOU Hong
(School of Info. Sci. & Techn., Northwest University, Xi'an 710069, China)

Abstract: Program understanding is an essential part of all software maintenance and enhancement activities. From the view of cognizance and software maintenance, the task of program understanding is described. Moreover a set of models of program understanding is explained in this paper. At last, give some suggestion to improve program understanding.

Key words: program understanding; software maintenance; understanding tactical; Pennington model; Soloway model

0 引 言

有关调查表明,软件系统交付之后对系统实施维护的成本约占软件系统整个生命周期总成本的50%~70%,而且随着软件规模和复杂性的增大,这个比例还有上升的趋势[1]。软件维护已经成为许多组织面临的最大任务。在IEEE2004中明确规定程序理解是软件维护活动中的一项任务,程序理解是一项困难费时的工作,是一个从计算机程序中获取知识信息的过程,这些知识信息可以用于知识排错、增强程序、重用程序以及整理文档等方面的工作上。程序理解活动包含的三步活动是:阅读有关的文档;阅读源代码;运行程序[2],如图1所示[3]。

第一步:阅读有关程序的文档。在这个阶段,理解者要浏览,仔细阅读不同来源的信息,例如系统文档,包括规格说明书和设计文档,以建立对系统的总体理解。对于很多复杂老系统,如果系统文档不准确、过时或不存在,可以省略这个阶段。

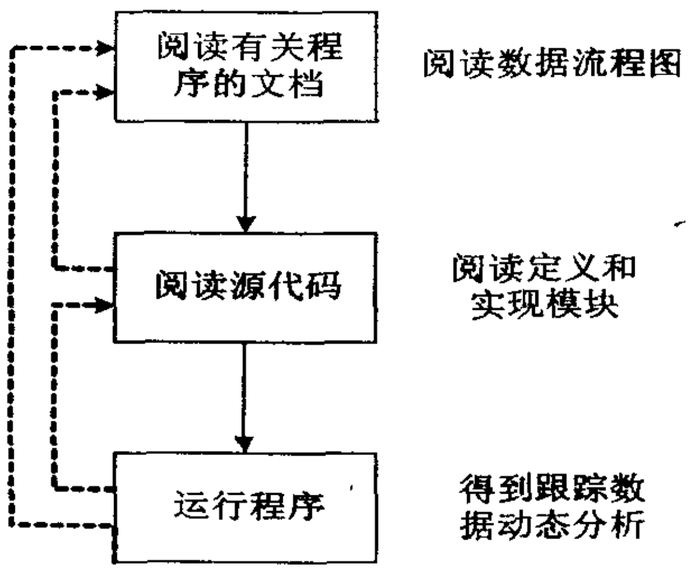

图1 理解过程模型

第二步:阅读源代码。可以实现对程序的全局和局部了解。全局了解可以从顶层理解系统,确定更改可能对系统其它部分带来的副作用的范围。局部了解使程序员能够集中精力到特定的系统部件上,通过局部了解,可获得有关系统的结构、数据类型和算法模式。

第三步:运行程序,这个步骤的目的是研究实际程序的动态行为,例如执行程序并获得跟踪数据。运行程序的好处是可以发现仅仅通过阅读源代码很难发现的系统的某些特征。

在实践中,理解程序的过程并不是以这种有序的方式展开,往往要通过各种活动的迭代和回朔澄清疑问而获得更多的信息[3]。

收稿日期:2006-05-30

作者简介:丁剑洁(1979-),女,陕西韩城人,硕士研究生,助教,研究方向为软件工程;鱼 滨,副教授,研究方向为软件工程;侯 红,高工,博士,研究方向为软件工程、软件度量。

1　程序理解的任务

软件理解的任务是在不同的抽象层次上建立目标程序的概念模型,实际上是建立从问题领域到程序设计领域的映射集。不同的理解者的理解层次不同。软件维护是一个多人参加分工合作的活动,如果不受维护团队规模和组织的影响,在理论上可以把维护人员分类,分别在不同的层次上进行软件理解。但目前很多维护组织中这种划分并不是那么严格,维护人员的任务取决于诸如维护工作的组织和维护团队的规模因素。但是,在规模较大的公司中,往往要根据所使用的组织模式对维护人员的角色进行严格规定,并不要求每个成员都理解被维护系统的所有方面,维护团队的成员包括经理、分析员、设计员和程序员,都会根据自己所起的作用,有不同的理解要求或信息需要[4]。这也是软件工程发展、软件维护工作被重视的必然趋势。下面就按照角色划分来分析程序理解的任务。

1.1　经　理

经理的责任之一是做出决策,他需要决策支持知识,以便做出有远见的决策,不必知道系统体系结构设计或下层程序实现细节。例如,经理需要根据所理解系统的规模,决定系统新版本的发行时间。另外,经理还要保证维护工作按计划进行,在有人员对所理解程序出现歧义的时候,要做出正确的决策,协调全体维护人员的工作。

1.2　分析员

分析员也是要在较高层次上理解系统,在更改之前,要对系统有一个全局的认识,理解主要功能单元之间交互的整体情况,还需要确定系统性能变化的内部原因。同时,在软件维护期间,分析员需要知道软件外部环境的变化,如新的政府规定或新的操作系统这些变化会怎样影响系统。分析员不需要局部认识,即对系统局部部件,以及实现这些部件的方式的认识。通常可以运用诸如背景图这样的物理模型,表示系统的主要组件及其如何与系统环境关联,从而帮助分析员在不抽象下层设计或编码细节的情况下很好地理解系统。

1.3　设计员

设计员在软件维护期间,应深入了解软件系统的设计。理解软件系统的体系结构设计和详细设计。试图了解功能组件划分,概念数据结构和不同组件之间的相互关联等信息。并且要提取详细设计产生的详细算法、数据表示、数据结构以及程序与例程之间的接口。通过提取这些信息,确定通过现有系统的体系结构、数据结构、数据流和控制流如何适应新系统的增强,通过现有系统的源代码,大致了解工作的规模,会受影响的系统的部分以及程序设计团队完成这种工作所需的知识和技能。

1.4　程序员

维护程序员需要理解不同抽象层次上的系统执行效果,在较高抽象层次上,程序员需要了解系统的单个组件的功能及其因果关系。在较低的抽象层次上,程序员需要理解程序的功能、执行顺序、数据对象的转换效果和程序语句集合的用途等等细节信息[3,5]。

虽然将维护人员进行了分类,并分析了各自的理解任务,但是它们之间的工作并不是独立的,而是相互依赖、相互影响的,要完全正确理解目标系统,他们之间需要协同完成理解任务。

2　程序理解策略和模型

程序理解策略是用来形成目标程序概念模型的手段,它包含了理解者在程序理解过程中的思维方式和行为动作。在软件心理学和认知学的基础上,通过对设计员和程序员理解过程的研究,总结出三种理解策略:自顶向下、向底向上和机会主义策略,并且提出若干相应模型[5~7]。

2.1　自顶向下

自顶向下理解策略的原则是理解者从程序的顶层细节开始,逐渐以从上到下的方式转向理解下层细节。主要的自顶向下模型有 Soloway 模型和 Brooks 模型。

Soloway 在他的模型中采用自顶向下的方法,在代码或代码类型知道的情况下可采用这种模型。采用所拥有的知识公式化假设,以便把系统分解成能够在代码中实现的已经知道的子系统,然后轮流分解每个子系统直到实现既定功能的一个个代码块。用这种方法构造的智力模型由目标和设计的层次构成,利用论述规则把目标分解成设计,继而分解成更加底层的设计。

该模型使用 3 种不同的设计类型:

①战略(Strategic)性设计,它描述程序中的整体策略;

②策略(Tactical)性设计,它与局部的问题解决策略有关;

③实现性(Implementation)设计,它考虑实现策略格局的语言的特征[8]。

图 2 说明模型的三个主要部件。其中三角形代表知识,常见的如编成规则或论述假设等。菱形代表理解过程。矩形代表内部或外部的程序表示(外部表示包括了文档、代码、用户手册、维护手册等等;内部表示包括设计和模式)。理解过程是进行外部表示到编程设计的匹配过程。一旦匹配完成,就更新内部表示,反映新获取的知识。这些新获取的思维表示保存为新的

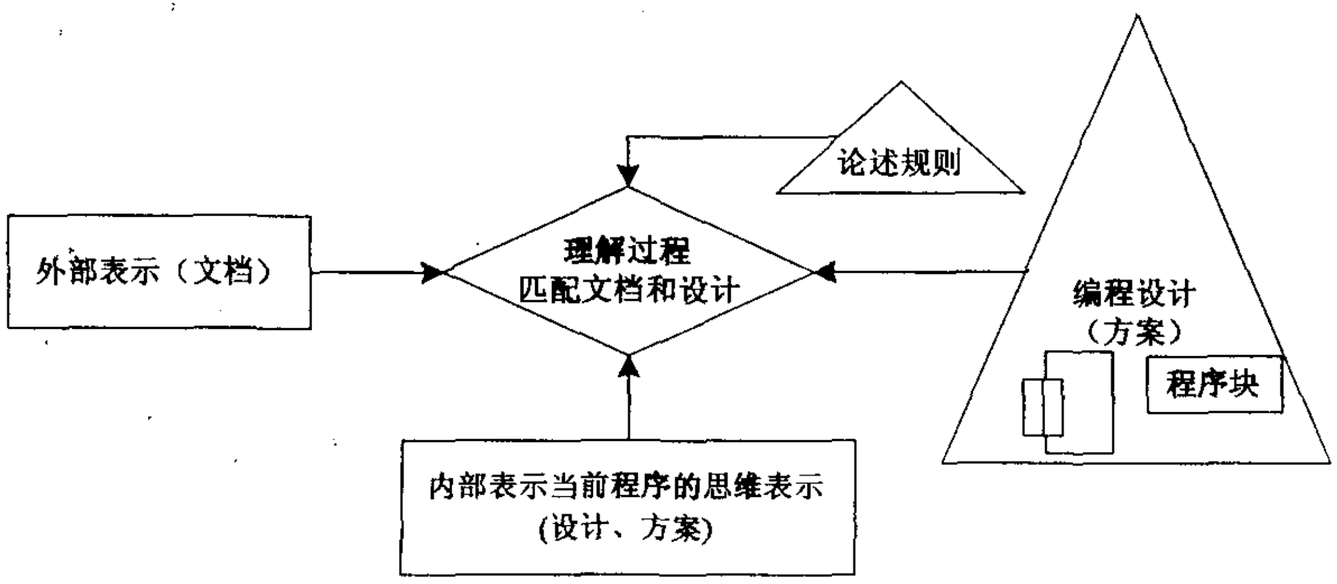

图 2 Soloway 理解模型

设计。理解从高层目标开始,然后产生出更详细的子目标[3,8]。

Brooks 提出的理解模型假设在设计阶段设计者做出的许多决策都会反映在代码中。理解包含了自顶向下恢复这些决策并通过中间领域的重新构造再把它们映射到程序设计领域[6,8]。

2.2 自底向上

自底向上理解策略的原则是理解者不断认知程序中的模式,把这些模式迭代地组成更有意义的高层结构,然后再按照自底向上的方式,把高层结构组合在一起,构成更大的结构,直到程序被完全理解。Pennington 模型是典型的自底向上的理解模型。

在 Pennington 模型中,理解过程分为两个不同的思维表示:程序模型和状况模型。程序员遇到新代码的时候,首先会创建控制流程序抽象,这种思维表示被称之为程序模型。这种自底向上通过标志构建的表示是程序中代码控制的基本块。状况模型是建立在程序模型之后的,它也是按照自底向上方式进行构造的,是数据流/功能的抽象。模型需要现实世界的领域知识,如操作系统领域的一般系统结构和功能,一旦达到程序的目标就完成了状况模型[5,9]。

图 3 是 Pennington 模型的描述[4,10]。图中左半部分构建了程序模型,右半部分构建状况模型。文本结构知识和外部表示(代码、设计文档等等)都是理解过程的输入部分,其中文本结构知识包括控制主流、程序结构、语法、编程习俗、控制顺序知识(如顺序、反复或条件),标志可以触发一些特殊模式(例如,交换操作能引起程序员联想到排序函数),代码语句相互之间的关系形成了微观结构。微观结构精简成宏观结构。这样自底向上以便构造出更大的程序块。

交叉引用表示允许从过程级、语句级表示直接映射到功能化抽象程序角度。更高层次的设计可以引起程序员重新考虑程序模型并且做出必要变化和改进。程序模型可以直接修改文本库或者使用这些设计作为程序模型理解过程的输入。

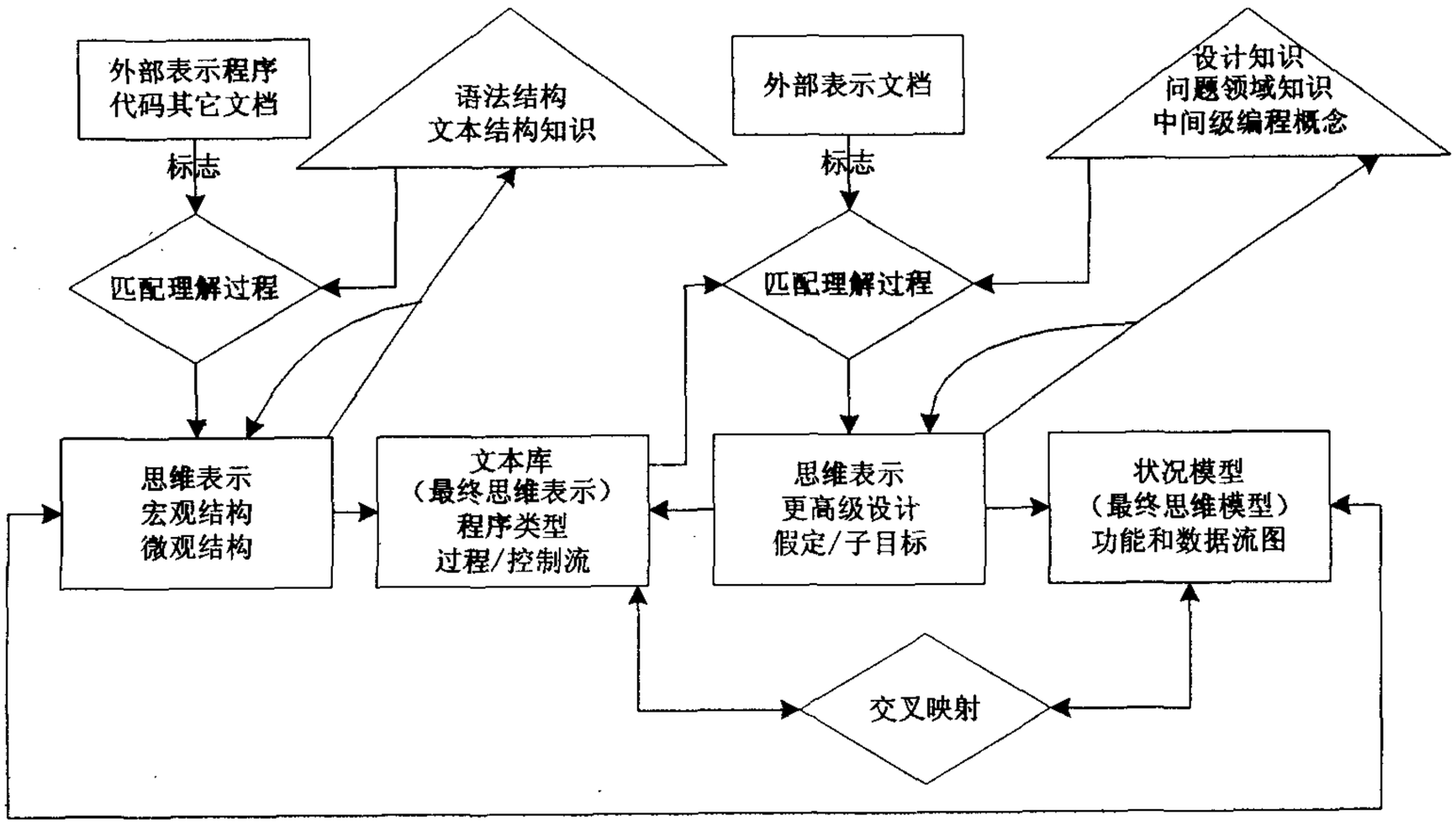

图 3 Pennington 理解模型

2.3 机会主义

虽然在前面分析了自顶向下和自底向上的理解策略,但是在实际中程序理解过程很少像这些模型所描述的那样定义完备[8]。所以,von-Mayrhauser 和 Vans 提出了使用一种集成模型,应用机会主义策略。他们的研究发现,理解过程是自顶向下或是自底向上两种方法的结合过程。集成模型包括 4 个主要部分,即程序模型、状况模型、自顶向下模型和知识库。当代码熟悉时,使用自顶向下模型,而对代码完全不熟悉时,使用自底向上模型,通过这种灵活的方式推进理解过程。知识库帮助其他 3 个部分模型的构造,每个模型均由代码的中间表示以及建立这种中间表示的策略构成,知识库融合了以前需要的相关信息和知识,在理解过程中,新的信息被开发出来放在知识库中以便将来使用。

3 影响程序理解的因素及对策

正确、完整、快速地理解程序意味着程序理解的效率高。在程序理解中,理解者要尽可能多地搜集信息(程序设计文档、源代码、程序的组织与表示等等),而这些信息的完整性、易读性、可靠性都直接影响理解的效率。另外,理解者自身的专业知识和应用领域知识也很重要,这些都是影响程序理解的因素,针对这些因

素,可采用下列对策。

3.1　提高理解者的素质

理解者是软件理解过程的主体,所以理解者自身的素质直接影响理解的效率。程序员在应用领域或程序设计语言上的经验越多,越能够更容易和更迅速地理解程序以及整个软件系统。因此,应该多给维护人员培训的机会,提高他们的专业水平,使维护团队的整体素质得到提高。另外,还应该拓宽维护人员的知识领域,例如,若要理解的是某个应用在金融方面的系统,在理解的开始,就应该邀请此方面的专家对理解者做相应的培训工作,拓宽他们的知识面,能够帮助他们较快地进入到软件理解的环境中去。

3.2　科学的管理开发过程

程序理解是在现有系统和保存信息的基础上进行的。所以程序理解活动中经常要咨询系统开发时候的参与者,但这存在一定的困难,原因之一可能是这些工作人员任务繁重,或是由于遗忘很难配合理解者的工作,也有可能已经离开了公司,根本无法咨询。所以,在系统开发时就应该采取科学的管理。它包含两层含意:一是保留所有有关的文档,并做到及时更新。例如从开始的需求规格说明书,系统设计时的各种文档,维护文档的更新等等,这些信息都直接影响到理解者搜集信息的质量;二是要注意系统的实现问题,例如命名风格、注释、嵌套层次,最好使用统一的规则,这些都直接影响理解人员理解的容易程度和深度。

3.3　有效地使用自动化辅助工具

阅读源代码是程序理解的一项重要活动,但是阅读别人的代码是枯燥乏味而且困难的工作,所以开发辅助工具是软件理解的一项重要研究内容,并且在这一领域已经有了很多成果。这些工具能以更清晰、更可读、更可理解的方式组织和表示源代码,把人们从烦躁的代码阅读中解放出来。常见的辅助工具有下列几种:程序切分器、静态分析器、动态分析器[3]等。程序切分器能够帮助程序员选择并只观察所提议更改影响的程序部件,不受无关部件的干扰,显示数据链和相关特征,使程序员能够跟踪更改影响。静态分析器能够帮助程序员快速提取模块、过程、变量、数据元素、对象与类、类层次结构等信息。在理解过程中,理解者应该使用这些工具,提高理解效率。

4　小　结

软件理解在软件维护中发挥着日益重要的作用,文中主要分析了几种常用的理解策略,并提出了提高理解效率的几点建议,对软件维护中的程序理解活动提供了一定程度的帮助和指导。虽然 Woods 和 Yang Qiang 已经形式地证明了软件理解问题是一个 NP - hard 问题[11],但是仍旧存在着值得进一步研究的工作。例如现有的理解模型需要进一步细化以符合人们的思维模式;已有的理解工具还存在通用性、精确性等限制等等。

随着软件新技术的出现,软件理解也将面临着越来越多的研究课题。

参考文献:

[1] Boehm B W. Software Engineering Economics[M]. [s. l.]: Prancticе Hall, 1981.

[2] Corbi T A. Program understanding: Challenge for the 1990s [J]. IBM Systems Journal, 1989, 28(2): 294 - 306.

[3] Grubb P, Takang A A. Software Maintenance: Concepts and Practice[M]. [s. l.]: Publishing House of Electronics Industry, 2004.

[4] von Mayrhauser A, Vans A M. From program comprehension to tool requirements for an industrial environment[C]// In Proceeding, 2nd Workshop on Program Comprehension. Los Alamitos, California: IEEE Computer Society Press, 1993: 78 - 86.

[5] Pennington N, Grabowski B. Psychology of programming [C]// In: Hoc J - M, Green T R G, Samurcay R, et al, ediors. Psychology of Programming. London: Academic Press, 1990: 45 - 62.

[6] Brooks R. Towards a theory of the comprehension of computer program[J]. International Journal of Man - Machine Studies, 1983, 18(6): 543 - 545.

[7] Hoc J - M, Green T R, Samurcay R, et al. Psychology of Programming[M]. London: Academic Press, 1990.

[8] Li Bi Xin, Zheng Guo Liang. The Research and Development of Software Comprehension[J]. Journal of Computer Research & Development, 1999(8): 897 - 907.

[9] Li Ying, Zhang Qin Yan. Program Understanding[J]. Journal of Computer Research and Application, 2001(6): 40 - 44.

[10] Pennington N. Stinulus Structures and Mental Representations in Expert Comprehension of Computer Programs[J]. Cognitive Psyhology, 1987, 19: 295 - 341.

[11] Woods S, Yang Qiang. The program understanding problem: Analysis of a heuristic[C]// In: The Proceedings of the 18th Int's Conf on Software Engineering (ICSE - 18). Berlin: [s. n.], 1996: 25 - 29.