计算机科学 1997Vol. 24№. 4

软件方法论研究中的若干问题*)

On Several Aspects of Research on Software Methodology

应 晶 何志均

(浙江大学人工智能研究所 杭州 310027)

摘 要 There exists urgent demand on software methodology in present software engineering domain. The authors put forward a new methodology for high-level software specification construction aiming at nowaday software development. This paper also addresses those existed problems in several aspects of methodology(include specification validation and verification, specification reuse, reverse engineering, re-engineering, and CASE), and presents author's viewpoints on them.

关键词 Software methodology, Software specification, Reverse engineering, Re-engineering, CASE

软件发展到今天,已出现了一大批较为实用的软件工具。这些工具以成熟的技术为基础,分别从不同的开发阶段、从不同的设计角度、依据不同的软件开发信息完成一个局部的开发功能。例如,从软件开发语言编码工具、编译工具、调试维护工具、测试工具、软件开发平台、多媒体开发支持工具、文档生成工具、项目管理工具、用户界面开发工具、面向图形的支持工具、需求分析工具、代码生成工具等计算机辅助软件工程开发工具。这些工具一般说来不能给软件生产过程带来根本性的改进。实现根本的改进需要采用"构造性方法",改变生产过程本身,而不是改善现有的实际做法。

"没有本事的工匠才责怪他的工具"。这句俗话在软件工程领域并不是很准确。浅层的工具使用得再好也不可能产生突破性的成效。基础是必要条件,而工具的组织及方法论指导可使过程较好地加以实施。可以说,"不珍视好工具的软件工匠是愚蠢的软件工匠"。

单靠设施与工具不足以使软件生产过程得到根本的改进。软件工程的未来方向必须强调能重用软件的设计和程序自动生成这类构造性方法。软件工程的主题是指构造软件产品实体所用的方法与手段,因此在软件工程领域,如果不对软件开发理论和软件开发方法学进行细致的研究,建立不起正确有效的软件开发方法和技术,其他方面(如工具、环境等的研究)都将无所依托。另一方面,这批从实践中产生的软件工具有着其广泛的应用领域与背景,并辅助软件开发人员成功地解决了一些实际问题。虽然说它们的研究对软件工程学科的本质性突破没有直接的关系,但把研究工作建立在这些实用工具上是合适的。

一、方法论的提出

我们提出一种支持软件定义高层构造的方法论MHSC:从软件开发的需求分析与定义层入手,提出一种能支持多重语义维描述(包括数据流,控制流,实体关系,模块组织,状态转换等)的软件定义语言,应用语言来构造实际领域软件系统的合一化功能模型,作为软件需求分析的中间结果,在此基础上通过进化方法(基于知识的变换方法、基于可视技术的求精、基于模拟机制的软件定义证实与系统功能验证),逐步过渡到设计阶段的软件实现,以支持软件的自动开发过程,从本质上改进现有的软件生产过程。MHSC所支持的是具有增生结构的软件增量式构造过程。

方法论强调一种多语义维的定义,统一地描述数据流、控制流、状态转换、实体关系及模块组织等语义信息,形成软件系统的一个可演化的单一模型,作为整个软件加工过程的核心。软件开发过程的描述可分为获得完整的需求、描述上下文系统的交互、

*)本文研究得到国家自然科学基金和863高科技项目支持

描绘子系统、组织成分(贯穿整个开发过程)、对象的动静态特性的定义、构造一个完整的设计模型、对此模型的加工过程等七个阶段,由此MHSC把软件开发过程简化为一个三元组:

$SD=(I_i,O_i,T_i)$

其中:I_i:功能,资源,性能需求;O_i:提供了实现系统所需的定义;T_i:变换、求精,概念具体化及合成方法。

MHSC提供三个方面的支持:

①定义语言的构造:提出一种描述不同粒度、不同层次的功能设计信息的定义语言。针对面向反应式领域的系统描述具有充分的表达能力,能形成统一地包括多语义维的目标系统的合一化功能模型作为开发的中间产品,通过对模型的进化方法逐渐生成实现层的系统构成(模块化,程序化及可执行性)。支持软件开发的多重视图表达与展现形式;

②基于知识的变换方法:在领域分析的基础上形成大量的定义变换知识(规则)。结合作者在专家系统方面的研究基础,实现一个支持定义变换进化过程的开发框架,体现基于知识的表达方法与问题求解能力;通过应用自动变换与手工变换的过程,反映出领域变换方法对软件实现的支持特点;

③可执行原型的形成:利用定义语言的可执行能力,在变换与求精过程后形成一个可执行的软件原型,从外部行为反映系统的整体功能。在此基础上,提供定义仿真环境,支持定义的证实与验证,把实现层的系统组成与定义层的用户需求建立一个反馈,使目标系统能满足开发初期的用户实际要求,另一个方面调整用户对系统的认识,清楚地把握系统开发状态。

MHSC方法论跨越了软件开发的生命周期。作者认为在软件定义开发的研究中,其证实与验证、定义重用等若干方面均是方法论必须加以支持的不可分割部分。因此本文着重对这些问题进行探讨,以明确进一步的研究方向。

二、定义的证实与验证

定义开发不是单一和线性的。相反,在整个需求分析过程中用户思想不仅不断地具体化,也会遇到一些修改。随着开发的深入,甚至在系统功能基本实现之后,会发现与实际情况并不是一回事。在这种情况下,不可避免地要修改系统的初始定义。可执行的定义方法支持了这类遗漏、模糊或冗余的需求,同时提供了定义证实与验证功能的基础。

可执行定义能够有效地支持定义的证实与验证。其中定义的证实指用户需求描述与生成的定义在功能方面是否一致,定义的验证指可执行定义的能力与预想系统的功能是否符合。在这两方面我们从实现角度作了简化处理,依靠在交互构造定义和反馈过程中来完成定义证实;通过在原型定义的执行过程中进行观察、在执行过程中的跟踪、记录、动态维护、执行过程中的不一致性检查、通过定义生成过程(求精与变换)的重现机制等来完成可执行定义的验证。从技术角度出发,验证和确认技术可以是手工的或自动的,简单的或用数学形式的。使用数学的形式验证技术是以归纳断言、功能语义和显式语义为基础的。另一些数学形式化的技术是以静态分析、符号执行等为基础的。自动验证系统的例子有Boyer-Moore定理证明器开发的程序验证系统,标准验证程序,Gypsy验证环境,以及爱丁堡的LCF系统。

软件设计的验证和确认技术需考虑:

- 完整性:如果需求规格说明的每部分都完全阐述,则设计是完整的。
- 一致性:如果在设计的各部之间不存在冲突,则设计是一致的。
- 正确性:如果设计的输入和输出关系可以被证明其真假,则设计是正确的。
- 可追踪性:如果设计中的术语在早期规格说明中有先例,则设计是可追踪的。
- 可行性:如果系统可以实现和维护,使得它在生存周期中预期效益会超过预期开销,那么该系统的设计是可行的。
- 等价性:如果两个设计具有相同的行为,则它们是等价的。
- 终止:如果设计充分详细到可以实现,那么设计终止。

可以提供视图编辑器来自动保证复杂关系的一致性。进一步,通过浏览工具、交叉引用工具能够容易地发现和操作对象。同时对于定义中的不同约束、变换规则等亦可采用一致性维护方法。

证实支持软件定义与最初始的用户需求之间的一致性测试,可以采用文档化方法、执行定义、白盒子测试及说明性调试来支持。

①定义文档化　产生定义的合适文档是定义的证实与展现的基本度量。这一任务由视图编辑器来支持,允许在定义的组成部分上产生特定的报告。当最终定义展示给用户(或在软件开发后期给其它人员),这类报告的混合便可用于定义的建档。这种混

合由支持定义开发的工具产生:定义的整个行为将以 STD(状态转移图),Petri 网及其它视图来展现,文档通过合适的语义数据模型的说明来完成,包括对象和关系,分类,聚类和特定问题关系。对于过程性程序或逻辑程序则很容易将其转化成自然语言正文。针对 MHSC 构造中的转移描述,通过名词替换易于产生其正文描述成分(通过句法变换)。这种思想使初始需求与定义之间可以进行直接比较,是最简单的一种方法。

②执行定义。证实的另一种更具吸引力的方法是执行定义。由于支持可执行定义的描述能力,任何时刻系统定义都能加以执行,通过一些环境设施(界面、资源分配)及预处理工作,可以完成一个快速的执行-修改周期。亦可把定义安装到环境中的一个菜单中。根据功能性与用户界面,软件定义可以建模成非常接近最终系统的实际行为。(这一点需要增加类似事件/进程操作,如事件处理,失败处理,当前状态和状态转移等设施)。最后通过动画设施来可视化整个处理行为,过程编辑窗口被激活并显示当前过程,通过过程图中相应状态的视图增强(颜色变化,动态感)来反映下一状态的变迁。两种设施均不影响定义的正常行为。

③测试方法。通过控制流分析来导出执行的期望路径,这一点可以在定义的整体运行中帮助开发人员,亦称白盒子测试。

④调试方法。在定义执行过程中进行跟踪,发现实际运行中发生的错误。在实际工作中 MHSC/E 提供了建立初始运行路径功能:在系统合一化模型上交互构造期望路径。在实际运行中若发生冲突,则进入测试分析状态。

三、定义重用

软件重用直接利用现有软件成分(资源)来构造新的软件系统。作者认为,软件的重用成分不应狭义地指软件的源程序代码,而应广义地理解为一切可用来构造软件系统的成分,包括软件开发方法、软件需求、设计规格说明、源程序代码与模块或其抽象结构、系统文档、开发工具与支撑环境、测试和开发与设计动态信息、维护信息等,甚至是 MHSC 所反映的变换过程重现。

软件重用技术分为合成技术和生成技术。在合成技术中构件是软件构造块,理想的情况是在应用时不需改变,但很难实现。构件必须修改或增删,使之符合具体问题的要求,通过一个外部机构对构件处理,合成为实用的程序或程序段。合成的另一种是从构造块衍生出新程序,把一些经过巧妙定义的构件合成原则应用于构件,合成出具有新功能的程序。

在生成技术中,构件是程序的模式,通过一个生成程序产生新的程序或程序段。合成技术和生成技术也称为构件方法和变换方法。构件方法源于子程序库思想,以抽象数据类型为理论基础,借用了硬件中集成电路的思想,构造出软件的集成块,通过软件集成块的组装形成更大的软件模块,经过联接,调试,最终形成一个可用的软件。变换方法中通常采用甚高级软件定义语言,形式地给出需求说明,利用一个程序变换系统(有时要经过一系列的变换),把用甚高级定义语言写的程序转化成为某种可执行语言程序。

构件方法侧重于构件的描述及组织管理,支持自底向上的软件开发方法,提供基本的软件模块。变换方法则侧重于程序的推导方式,推理规则,支持自顶向下的软件开发技术。这两种方法的结合使用,可以取长补短。通过吸取人工智能的研究成果,以知识库为辅助工具,可以促进软件重用的研究。

MHSC 支持变换方法,其整个开发过程的出发点是软件定义构造,这种包含多种语义信息的软件定义具有较强的领域适应性,因此只考虑构造同类软件过程中的软件重用,事实上几乎没有一个通用的软件重用策略或技术得到成功的实际应用。在此基础上,同类软件的构造便成为合一化模型中所包含的对象重构问题。由于支持环境中提供了丰富的对象库(如机构、服务、信息库、事件表等),以及一部分可以利用的模型部件库,通过环境中的可视求精算子,很容易重新建立一个新系统的合一化模型,这是第一层次重用;其次,在构造软件系统时,开发人员可以应用各种变换方法,另外通过领域具体分析,亦能形成一部分新的变换知识,结合变换的作用策略,这些知识形成一个可以重复作用的变换库。因此知识库形成了第二层次的重用;再者,一个软件系统的定义模型形成之后,利用自动的变换作用策略,能自动地把软件定义转变成包含更多功能性描述信息的可执行定义,即可执行定义的形成是建立在一个变换作用序列之上。对这一序列的作用过程加以合适的记录并提供重现机制,可以辅助在领域其它软件定义开发中的变换过程,这是第三层次的重用,即前所述的动态开发信息重用过程。

通过实际领域的开发,对软件重用问题的考虑是:人们能重用软件的唯一现实想法是要求软件的

设计者一开始就考虑软件的重用性。我们认为应该把重用性作为初始设计的一个目标。在实际工作中，我们提供了能描述软件系统的定义模型、变换方法（包括变换序列）及各类构造成分的重用库，通过对系统标准库的访问、扩展来支持部分重用功能。通过软件定义高层构造的方法论来支持软件开发为以后的设计工作建立部分重用基础。

软件重用概念虽然简单，但系统的软件重用期望从中收到巨大的效益，仍存在不少困难和问题。九十年代，软件重用的研究和发展必将同其它计算机学科和技术紧密联系在一起（专家系统，人工智能，程序设计语言，软件工程，库科学，知识表达方法，软件度量技术，形式化方法，超级文本技术，面向对象方法，CASE 技术，图示）。管理问题和标准化问题的解决将日益受到各国政府、机构和标准化组织的重视；在技术问题方面，过程重用将成为主要的研究对象，领域分析和领域工程将构成软件重用研究的一个重要分支。如 Bohem 等人提出的元程序设计，将得到进一步发展，它可能支持某些专用领域的重用，并具有建立新的、过程驱动的开发范例的潜力。

四、逆向工程与重设计工程

逆向工程是软件重设计工程的第一步。定义为“一个分析已有系统并且标识出系统的组成以及之间的关系，同时用一种更高层的抽象形式来创建系统的表达的过程”。许多种类的程序信息可以加以抽象和检查。对于一个以往的软件系统而言，对其设计加以抽象和记录尤其有用。一个设计通过把从源代码、已有文档、个人的开发经验及应用领域知识中得到的信息加以聚类来完成恢复过程。因此设计恢复与逆向工程的区别在于强调了帮助实现信息高层抽象的应用领域知识的恢复。Biggerstaff 曾对设计恢复加以论述：“恢复的设计抽象必须包括传统的软件工程表达，如形式化定义、模块分解、数据抽象、数据流和程序描述语言…。设计恢复必须重新生成一个人能完全了解一个程序的行为所需的全部信息”。现代的软件设计者和开发者从他们多年的经验和观察中形成了众多行之有效的技术和方法来指导具体的开发工作。自动化工具亦集成到环境中支持软件的开发，这些工具体现了软件工程技术，即能辅助开发出结构化程度高、可理解、易维护、错误少及良好的建档。通过恢复已有系统的设计，这些现代技术可以施加于设计的变换，并生成一个更为理想的实现结果。这就是软件重设计工程。

软件系统功能更新的一种有效方法是重设计和重设计系统。这种方法从当前系统的需求出发，用目标语言构造一个全新的系统。这个新系统尽管不是从旧系统中导出的，但要求在功能上是等价的。这种方法在较大可能上可生成合适的新系统。用一种新的语言来重设计和重实现系统在建造最终产品上提供了很强的功能和很大的灵活性。形成的系统与其它方法产生的系统相比大大减低了维护费用。然而，这种方法存在缺陷：因为要模仿已有系统界面的需求，使得这种方法比初始设计更加困难；这种方法在初始构造上费用很大，其等价于建造一个新系统。最严重的问题是对许多系统而言，几乎不可能从系统的需求入手进行重设计，因为有些需求根本不存在。老的系统中对系统能力和功能的描述也就是源代码本身了。逆向工程为此提供了一种新的方法。如果系统不存在需求规格说明书，对系统进行逆向工程开发能产生一个包容了系统功能的重新构造的设计。此设计应在一种抽象层次上加以表达以消去实现语言的相关性。这一点使得用一种新语言来重新实现系统变得可能。另外，重新构造的设计可以施加变换以调整，重结构化，并加入新的需求等等。这就是重（设计）工程。因此基于一个逆向工程过程的再工程提供了重设计和重实现方法的许多优点。

通过上述对逆向工程的方法与过程的阐述，作者体会到如下几点：

—实现的倾向。也许在恢复一个设计时最重要的问题是从实现信息中分离出设计信息。在实现层上要想识别哪些信息不应记录在恢复设计中是比较困难的。在抽象过程中很难“舍弃”信息尤其是在目标要重新实现系统的时候。程序建档反映了源码的实现细节，设计建档则反映了程序如何工作的高层视图。需要关注的实现细节是数据项名字，基本数据类型，数据结构，系统界面，高效的代码段，偏于所在计算机体系结构的代码，以及其表达偏于初始实现语言的代码。

—可跟踪能力。一个恢复的设计应记录恢复信息和初始源码之间的关联信息。这一点支持了恢复设计和初始实现的跟踪能力。一个合适的软件开发环境可用以记录一个恢复的设计，而这样的工具必须提供记录哪些工具建立时没有预见到的信息的灵活性。

—领域信息。在程序中记录的信息足以解释“做什么”而不是“为什么”。为了理解一个处理步骤的意义，往往需要深层的信息。这一信息往往是没有在源

代码或是存在的文档中加以记录。对应用领域的理解可以辅助对一个函数的目的信息及重要性的恢复。

—再工程。逆向工程主要作用在那些文档已过期或不存在的旧软件系统。这样的系统可选作再工程:重写系统以改进它的可理解性,易于维护,消去无效代码等等。改变一个设计要比修改源码容易得多。一个恢复过的高层设计加以更新并产生新的详细设计,从此设计中产生的新版系统能用最新技术和程序设计语言加以开发,从而导致一个更加结构化,建档的及易于维护的系统。作为再工程第一步的逆向工程在老的程序的扩展中被证明是较为有效的。

一已有的建档文件。最后一方面是注解和已有建档的使用。这些信息源有可能是过期或不正确的,在使用时要格外注意。它们提供了关于程序的有用信息,但有可能是错误的。源代码才是对程序实际行为的最终描述。

基于上述几个方面,我们面临的是迫切需要建立软件逆向工程的方法论及其相应工具和开发框架来辅助开发实际软件。MHSC 方法论支持了软件系统在定义层上的开发、证实和维护工作,实际上从一开始便充分考虑再设计工作的可能性与灵活性。作者的观点是:现有开发系统中,在实现层上要识别软件的重设计信息是相当困难的,无论从功能分析还是结构分析,一个倾向便是重新导出彼此缺乏语义性和依赖关系的软件设计信息,包括数据结构图、数据流、控制流、实体关系图等。若把逆向工程的目标(或现实的考虑)定为这些信息从已有软件系统中的自动抽取(暂不考虑其一致性与完整性),则把重设计工作建立在这些基础之上依然是难有突破性进展的。首先,现有软件形形色色,从开发方法、工具、技术、设计人员风格等千差万别,有些程序甚至"只能从其运行过程才能发现还是有用的",这一点给设计工作带来很大的阻碍;其次,开发人员依照逆向工程的结果进行再设计工作,仍然是采用人工解释,并面向实现层的设计方法,上述的抽取信息实质是软件的部分文档作用,这仍然是断层所在;第三,重设计工程强调的是以一种新的形式及其相应实现来重新构造软件系统,则必须把新的形式建立在原有形式的基础上,作一些灵活的变更、检查、扩充,借鉴原有的实现功能(或步骤)来构造新系统,即不能脱离原有的开发过程的中间产品(包括软件需求定义、设计文档等开发过程记录信息)。作者有理由相信,MHSC 所支持的软件高层定义构造,为软件的重设计工程建立了一个良好的基础(至少体现了软件的设计者必须一开始就考虑到软件的更新、再设计及重用能力,即作为一个重要的设计目标),软件的开发、更新、维护等工作均建立在定义层的中间产品上,此时再考虑逆向工程应该得到的结果便会发觉均是原有的定义开发过程包含的信息,软件系统的重设计工作便迎刃而解。可以说,MHSC 支持软件开发的双向能力,即从定义层出发经过变换、求精形成软件可执行原型(准实现层),再从系统功能验证与证实中发现的问题反馈到定义层描述,完成下一个再设计过程。

综上,今后十年的软件工程将极其强调软件开发中的重设计工程与逆向工程,这反映了软件产业界迫切需要利用多年来的软件产品与成果,迫切需要寻找一种合适的方法来进行合适的修复、更新工作;同时作者认为仅从重设计工程本身去探索一些技术、方法并不是关键所在,本质问题是在软件开发过程中就融入同类思想,引入一种新颖的方法论来指导整个正向、逆向工程的开发产业,最终将淡化逆向设计(工程)的概念,软件开发过程必须包括这方面的考虑。这也是 MHSC 的思想所在。

五、与 CASE 研究的接轨

CASE 是一种在软件开发活动中使用的方法步骤、工具、技术和工程的总和。其目的是设计出优质的软件系统,指导和管理开发活动,控制维护和有效地使用资源。CASE 为人们提供了一种理解软件生命周期概念的新方法,其本质是提供一套集成化的、良好的、能节省大量人力物力的工具,利用这些工具来连接并自动完成软件开发周期中所有阶段的工作,重点解决整个开发过程的生产效率问题,力求通过计算机技术自身自动地完成软件开发生命周期各个阶段的工作,提高软件生产率,改善软件质量,最终解决"软件危机"。

CASE 的进一步研究必将朝着使 CASE 环境能从很大程度上支持一种合适的软件开发方法论。CASE 环境亦将从语言中心环境、面向结构环境、工具群环境转变为基于方法论环境。

作者在CASE方向做了不少的研究工作。研究

计算机科学 1997Vol. 24№. 4

Z 和 VDM 规格说明的差异比较

A Comparison of the differences between Z and VDM Specification Languages

孙未未　白雪峰

（复旦大学计算机科学系　上海 200433）

摘　要　This paper attempts to provide an understanding of the interesting differences between well-known specification languages--Z and VDM.

关键词　Z,VDM,Differences

实现软件开发过程各阶段的自动化是软件工程的重要目标之一。软件自动化的前提是形式化，包括软件需求规格、软件设计规格和算法描述等的形式化。形式化软件规格说明不仅是对用户需求，也是对软件系统的严格定义，在软件开发中有着相当重要的作用。基于此，为发展我国民族软件产业，与国际标准相衔接，在我们的软件设计中，应坚持必要的规格说明，这样才能生产高质量、长生命期的优秀国产软件。现在已经有了比较流行的 Z 和 VDM 两种面向模型的规格说明语言，准确了解这两种语言的差别，对我们在具体工作中使用将有指导意义。因此，在参考了 I. J. Heyes 等的"Understanding the differences between VDM and Z"之后，形成了本文。希望我们的工作可以对我国软件设计规格化起一点积极作用。

一、起源

VDM 是由在维也纳的 IBM 实验室开发的。

表明，CASE 市场已有巨大增长，然而当前一代 CASE 工具却局限于浅层表达及浅层推理方法，因而 CASE 工具必须进化到或者替代以具有更深层表示、更成熟推理方法的工具，其可用技术来自 AI，形式化方法，程序设计语言理论及计算机科学的其它领域。

软件实践发展迅猛，覆盖了从软件系统的开发、技术的创新、工具的研制等各个方面。至今已形成众多的支持各种软件开发任务的 CASE 工具。然而据统计，在美国只有 2%的组织在软件开发过程中采用了 CASE 方法与工具，仅有 4%的软件厂家是 CASE 用户。这给软件研究软件人员提出一个新的问题。究其原因在于软件构造需要包容个体的知识、经验、技巧，形成一种可以共享知识的合作与协调开发过程，应用于软件开发任务，而现有 CASE 工具存在一些局限性，即没有强调开发群体（包括技术人员，用户，协调人员等）之间的合作式的开发，尤其是在需求分析阶段的计算机支持。MHSC 的研究从 CASE 工具的应用薄弱点分析出发，认为必须在软件需求分析阶段提供有力的支持，不仅可以促进开发人员与用户之间的沟通反馈，为软件的后端开发建立有效的基础，对软件质量进行保证，也为软件从需求定义层到实现层的自动转换提供一种新的途径，并为当前 CASE 工具的研究提供更加完善的支持和指导作用。

目前 CASE 工具仍没有发挥它的全部潜力，因为编码大部分仍是劳力密集型的手工过程。随着编码逐渐自动化，支持软件设计初始阶段的更多成熟工具将变得更为有用。以这些工具发展的快速原型系统将在人的最少指导下，由程序综合系统转换成具有产品级质量的软件。因而，CASE 表示将从软件设计结构和组织的部分表示迈进到形式定义。只要软件生产仍然还是一种劳动密集型活动，质量和生产率就不能期望有阶跃性的提高。软件产品必须通过自动化生产来确保其质量和生产率。CASE 研究将朝着工具、方法的集成，逐步形成一个跨越整个软件生命周期以及跨越多个应用领域的集成开发环境，R. J. Norman 提出的发展趋势引起了 CASE 研究人员的关注，将不断使 CASE 的研究达到应达的目标。（参考文献共 23 篇略）

孙未未　讲师，主要从事形式方法、数据库理论等方向研究。白雪峰　教授　博士生导师。