计算机科学1992 Vol. 19 No.6

大型程序设计的过去、现在和未来*)

Walter F. Tichy 〔联邦德国〕 TP311
Tichy, WF 陈海荣

摘 要

本文综述了大型程序设计概念的发展历史，包括层次系统、信息隐藏、大型程序设计语言、面向对象语言和软件配置管理。大规模软件重用、更牢固的基础和更好的开发工具、软件结构的编目和改善对软件工程师的培养，被认为是未来的方向。

1. 引言

小型程序设计是实现各个数据结构和其上操作的活动。而大型程序设计是与整个软件系统的体系结构相联系的；它涉及模块范围以外的大型系统的构成、模块与其规格说明之间的接口、和成品软件结构的不断进化。

小型设序设计是已趋成熟的方法。算法和复杂的理论构成了它的基础。当用抽象的算法来解决一个应用问题时，程序设计语言、编译程序、调试程序和无数的支持工具可供使用。甚至在实际使用中，如风格和程序设计习惯问题已被全面地讨论。

大型程序设计没有被广泛研究，但不是说不重要。六十年代后期，早期的层次系统概念出现了，操作系统的研究起了推动作用。这些概念在七十年代早期，随着信息隐藏和面向对象概念的引入而得到改进，到七十年代末，这些概念同程序设计语言和其它工具有效地结合起来。八十年代，这个领域的研究重点转移到如何驾驭软件改变和增长方面，引出了三个新的领域：软件配置管理、重用和过程模型化。软件配置管理直接涉及到软件改变的控制。而重用旨在建立大型软件设计库和组件库，供许多系统使用，减少大量的程序设计工作。最后，过程模型化是研究整体的软件开发活动，希望通过更深刻地理解有关的相互作用的开发过程，减

八、若干体会

(1) 本次会议讨论的主题"可信赖的计算系统"十分重要，它表明当代计算机软件已成为各种行业中十分关键的部件，它关系到人的生命、安全、重大事故，诸如医疗系统、工业安全、军事控制及空间交通等领域。应该从软件的开发技术、管理、标准化、容错技术、形式方法及测试等方面给予极大注意。

(2) 从本次会议讨论内容可看出软件工程并无重大突破，目前研究动向仍然在软件可视化、O-O、度量技术、环境集成、软件过程、重用技术、形式方法及逆向工程等方面，但目前更加重视大型应用项目的软件工程实践。

(3) Bjoner教授以联合国大学澳门国际软件研究所所长名义发表了主题演讲，并欢迎各国学者、尤其是我国年青学者前去参加研究。他所领导的项目及美国V. Basili教授领导的SEL均是长期坚持大型软件开发项目的典型，只有结合各国应用实践，长期坚持才能取得有效成果。各国对与我国软件合作均表示出极大兴趣。

*)曾获《第十四届国际软件工程会议》优秀论文奖。

少软件的开发费用。

下面将更详细地讨论大型程序设计的主要概念，并提出一些未来研究的领域。

2．层次系统结构

一个层次系统由"层"或"级"组成。每层管理着一个对象的集合，并规定了能对这些对象施加的一系列操作，这种管理遵循如下两个基本规则：

层次 每层规定新的操作，并隐藏了在低层选中的有关操作。在给定层中，可见的操作形成一台抽象机的指令集。因此，在给定层中所写的程序，能引用低层中一系列可见的操作，但不能引用在高层中的操作。

封装 如何表达和贮存一个对象的细节被隐藏在代表它类型的层中。因此，一个对象的各部分不能改变，除非对它应用一个授权的操作。

层次模型中体现的数据抽象原理可追溯到1966年Dennis和Van Horn的论文，它强调了用户和内核间的一个简单接口。Dijkstra在1968年报告了第一个可供使用的、内核分为几层的操作系统。这个思想已经传播到有关机器的操作系统家族和网络操作系统。今天，不仅仅是操作系统，而且有大量系统的设计应用了这个思想，如：数据库管理系统或窗口系统。

层次结构是一个功能规格说明的分级制度，其设计是为了强化高度模块化，并能够以增量的形式进行软件的验证、安装和测试。在功能分层结构中，每层中的程序可以直接调用比它低的层中的任何可见的操作，而不需在中间层中有信息流。

重要的是应区别我们现在讨论的层次结构同国际标准化组织(ISO)的一长串网络协议模型。在ISO的模型中，在发送机器上，信息的传递要通过所有的层，并且返回到接收机时，也要通过所有的层。由于在数据传送时，每一层都要增加延迟时间，而不管这些延迟是否需要。在局域网工作时，这一长串网络协议可能是低效率的。功能分层结构胜过信息传递层次的一个显著优点就是高效率，如果一个程序不使用一个特定的功能，那么就不会有这个功能在系统中存在时所引起的开销。

用分层的一般原理进行增量软件开发时，它提出了接口之间哪些应该可见和哪些应该隐藏。为了回答这个问题，Parnas阐述了一个分解 (descomposition) 的观点，叫做"信息隐藏"。根据这个原理，在系统的进化过程中，容易改变的部分应该放进不同的模块，而这些模块之间的接口应该对这些细节的改变是不敏感的。

信息隐藏原理的动力是使所有有用的软件进行进化。通过分离容易改变的系统细节进入独立的模块并且各模块间用不变的接口进行隔离，那么可能改变的每个模块只需对各自的模块进行修改，只有极少可能的改变需要接口的修改和非局部的匹配。为了使信息隐藏原理能够生效，有必要要求通过它的接口例行程序对模块进行所有的访问，而不允许直接访问模块的内部数据结构。信息隐藏原理可以通过增强编译器的功能来实现，为了提高效率，编译器可插入过程调用或扩充一些功能。

信息隐藏的首要目标是允许独立地设计、实现和改变模块，降低软件开发和维护的费用。这个原理同分层系统概念间的关系是每层应该分成一个或更多的模块来实现。为了进行增量软件开发，必须避免各层之间的循环依赖。非循环依赖型分层的另一个优点是便于系统的重构，如：扩展、压缩或嵌入其它系统。Parnas给出了在若干模块中如何设计非循环依赖模块的指南，并展示了它在一个复杂、实时系统中的应用。

3．用于大型程序设计的语言

1976年，DeRemer和Kron写道：

我们表明，将大量的模块构成一个"系统"与构造个别模块相比，这是从根本上有别的和不同的智能活动。就是说，我们把小型程序设计和大型程序设计区分开。相应地

我们认为对这两种活动应使用在本质上有差别的或不同的语言。我们把描述系统结构的语言叫做"模块互连语言"(MIL)，它对支持大型程序设计是必要的。

然后作者引出了"模块互连语言"的基本准则，并提出了这个语言的第一个版本，叫做MIL75。MIL试图定义软件系统的总体结构。因此，它也是一个设计工具，一种编制文档和通讯的手段，和增强系统体系结构的手段。MIL75规格说明为系统和子系统的模块命名，并将它们安排在一个嵌套（树形）结构中以展示构成。此外，树结构中的所有节点都指定它们从其它结点所需要的资源，及它们提供给其它结点的资源。资源是能用程序设计语言定义的任何命名的实体，如：类型、变量、过程，等等。仔细定义的规则支配提供给其它结点的资源分配。对于小型程序设计来说，MIL75也是独立的语言，并能组合用不同语言编写的模块。

Tichy 将MIL75改进成INTERCOL语言。INTERCOL首先提供了二个新的设施：接口控制和版本控制。接口控制是在软件模块之间建立一致的接口，并当它们改变时，能维护这种一致性。它基本上完成了精化形式的模块间类型检查，它的类型信息全部是从INTERCOL语言的规格说明中得到的。版本控制是通过许多不同的版本和配置来获取和控制系统的进化。在这里，INTERCOL语言通过定义接口、实现、修改、派生版本、配置和系统家族的重要概念，为日后软件配置的管理提供了很大的余地。

非MIL方法将系统体系结构的信息嵌入各自程序模块，而不是将其分解成精炼的描述。在提出INTERCOL语言的同一次会议上，Lauer和 Satterthwaite 报告了他们使用Mesa 编程语言的经验。Mesa允许接口分离并用独立的文件实现。每当一个程序需要访问另一个模块时，编程者只要简单地在源代码中写入一条命令，编译器就读入相应的接口文件。这种方法保证了系统范围内的类型一致性，但还不能处理多版本和配置。然而，分离接口和实现的思想，和简单地输入所需接口信息到它的代码的方法，在Modula、Ada和其它语言，都是这样实现的。

Mesa和MIL这两种方法对于有效地进行软件配置管理来说，都不是令人满意的。在Mesa和类似的语言中，嵌套的结构信息使我们很难得到系统结构的清晰描述。这也使得在实践上，不可能增强给定的系统结构。任何程序员都能通过简单地修改源程序来改变系统的结构。许多时候，由于程序员加入了许多不必要的接口，明显地导致了许多不必要的交叉结构。为了克服这些缺陷，在Mesa中加入了一个特殊的配置语言，叫做C-Mesa（Ada或Modula语言中没有）。然而，这不足以处理每一模块的多次修改。

INTERCOL语言遇到的是不同的问题。INTERCOL源程序必须通过专门的编译器来处理，这个编译器准备了一个数据库来保留各种接口、模块等信息。改变这个描述需要相应地重新编译和转换数据库，这是一个相当大的难题。在INTERCOL描述下，数据库转换的问题只能够通过消除冗余表达来解决。而数据库本身需要通过交互式命令去创立和操纵。一个MIL的描述可以根据需要通过数据库生成。这个方法在 Gandalf 项目中获得了成功的尝试。面向Gandalf 语法的编辑器使得数据库和相应的 MIL 描述很难区分。此外，整个Gandalf 系统从语法及语义描述方面为大型程序设计提供了精制的环境。还有许多想法有待于在Gandalf 项目的试验中，产生成高质量的软件开发环境做出结论。

面向对象的编程是另一种适合大型程序设计的技术。归纳其精华，这种技术允许组织数据类型到一个分类的层次中，并且子类型从超类型继承性质。这种层次结构不是象在层次系统中由使用-关系规定，而是通过类型的特化所确定的。这两种技术是互不相关的。

类型分类层次结构结合以继承性，在许多环境下是有用的。首先，分类是一个强有力的组织规则。数据类型集的共同方面可以分理出来，并一次性地在一个超类型中统一描述。第二，超类型使扩展简化。它们定义的不完善开放系统可以扩展到目前还不能确定的未来系统范围。相反，传统的类属或参数化类型（如带有一个元素类型做参数的队列）定义封闭的类型，设计时，这类型的性状变化范围是固定的。第三，继承简化了受控增量的生长。以分类层次结构为代表的系统，通过定义新的子类型进行扩展。置换不需要的过程和增加新的过程不会影响现存的软件。第四，由于不需要对每一个子类型重复改变，维护工作简化了。这些优点从技术上带来了许多积极因素。改为面向对象的一个有益副作用是采纳了一个较老的抽象数据类型思想，这在软件产业中同样是重要的。

值得注意的是，面向对象语言和MIL的并肩发展。六十年代末，在分层系统概念问世的同时，类和继承的思想在Simula-67编程语言中首次得到了试验。在冷漠了许多年之后，到七十年代末类和继承的思想才在Smalltalk语言中实现，并命名为面向对象。以后几年里，面向对象的思想渗入到许多其它语言之中。

4. 软件配置管理

八十年代中期，随着软件开发观点从产品为中心转向过程为中心的同时，形成了软件配置管理领域。软件配置管理是已成熟的一般配置管理学科的一个特例。一般配置管理的目的是控制大型复杂系统的改变。配置管理旨在防止任何大型系统因生存期内大量修改、扩充和适配所易于引起的混乱。配置管理必须把握系统性的、可追踪的开发过程，使系统在任何时候都处于规格说明准确和质量属性被检验的良好定义状态。

配置管理是五十年代首次在航空工业中形成的。那时宇宙飞船的制造遇到了困难，工程变化量大且文档贫乏。软件配置管理拓广了传统配置管理的概念，使之适用于以软件为主所构成的系统或系统的一些部分。软件配置管理的一个新特点是软件的改变比硬件快，所以需要自动支持。幸好软件配置管理能自动化，因为软件通常是联机的并可自动地操纵。

4.1 软件配置管理的概念

软件配置管理的主要内容是软件的配置项、配置、基线（baseline）和派生项。软件配置项是软件项目过程中产生的任何可分别标识的可机读文档。它由纯信息组织，例如：需求文档、设计文档、规格说明、接口描述、源程序模块、机器码模块、数据库文件、测试程序、测试数据、测试输出、用户梗概文档、用户使用手册、超大规模集成电路设计、数字化绘图和画片、录音带等等。配置项是单独改变的最小单位，其中不包含更小的独立改变的单元。

相反，配置是几个部件的组合，部件是配置项或其它配置。改变配置可以通过取代、增加或删除部件实现。例如硬件、软件和文档的配置组成了一个完整的计算机系统。这三个主要部件又分别是一个配置，最终分别由集成电路、程序模块或手册的段、节组成。

基线是配置的描述，基本上由部件表（partslist）构成，准确无二义性地说明哪些部件组成一给定配置。由于基线也可改变，理应是配置项。在系统的开发中，基线被认为是重要的参照点。基线一经建立，所有与之相对应的后续改变都需描述，一直保持到记录下一个基线为止。

派生项由其它对象自动生成。例如：编译代码、连接好的系统、格式化正文和测试输出。由于派生项能根据需要随时再生成，所以可以进行删除。派生项的存贮与再生成之间的时空平衡问题由软件配置管理的系统建立功能进行处理。

4.2 自动的软件配置管理功能

前面已经提到，软件配置管理比传统的配置管理有更高的自动化，其自动化的主要功能是：标识、版本控制、配置选择和系统建立。对于传统的配置管理，其改变控制、配置审计和配置统计的功能大多仍是手工完成的。

标识是指给每个对象分配一个唯一标识符。对于有效的配置管理来说，可靠的标识很重要。如果一个标识符分配给两个不同的对象，将会导致不少混乱。所以凡改变一个对象时，就要给出一个新的唯一标识符。唯一标识符一般由描述名和几个子域组成，这些子域含有版本指定者、系列号或日期的信息。

如果每次改变都分配一个新的标识符，有可能使对象间的关系不明朗。例如：欲记录一个给定配置项是另一个配置项的修订，软件配置管理的版本控制功能将记录这样的事实：它将收集的配置项分入版本组并管理这些组的变化。版本组中的项通过一些关系相联系。例如：记录历史开发线索的修订关系；联系项的变化关系（这些项在某些方面，如功能、设计、实现方面有所不同，但在其它方面是相同的）。

版本控制完成许多版本管理的功能。重要的一点就是，它对重叠改变提出警告：当两个设计者试图同时改变同一项时，版本控制建议其中的一个设计者等待另一个设计者完成修改后再做修改，或者系统支持合并他们的修改内容。登记/检查(Checkin/Checkout）协议避免了重叠改变：在任何实际执行修改之前，为了得到所期望的版本，设计者必须执行一个检查操作，获得自己专有的修改许可权，完成以后，登记操作将新版本回记到版本组中。在确认其没有在任何地方使用过之前，对老版本一般不删除。

版本控制也维护一个记录改变原因的日誌，其中的项一般记录了改变时间、开发者标识符，外加描述改变内容的简要说明等。日誌便于观察系统经历的变化情况。版本控制也须节省被多版本占用的空间，如使用delta 存贮的压缩技术，只存贮版本的不同部分，而不是其全部副本，与不压缩相比，压缩存贮可以减少存贮空间到10%以下。当然，一旦需要一个版本时，必须有一个专门程序能将其从delta 存贮中恢复。SCCS 和RCS是通过不同的 delta 机制实现的版本控制系统。

配置选择涉及同基线有关的更新选择问题。例如：除了已经被检验的内容外，大多数设计者加入了他们各自的修改内容。另一个关键点是挑选开发历史作检查的一给定点上的所有当前改变和被检验过的改变。选择完成以后，就过渡到系统建立，这个功能也称为系统生成，生成所期望的派生对象集。它完成这样一些任务：编译、连接、装配、预处理、后处理、文档生成、安装等等。一个重要的问题就是避免冗余的处理步骤，如编译已经编译过的对象。

MAKE 系统将上面提到的几个功能组合在一个工具中。它能用于配置描述，挑选最新版本的所有改变，自动地运行特定的生成过程，甚至起动回归测试。然而，它不能区分同一配置项的多版本。

总之，软件配置管理的基本概念和技术已发展完善。然而，实践工作是在能力有限的旧工具（如 MAKE、SCCS 和 RCS）上进行的。

5. 大型程序设计应考虑的内容

改善软件质量和提高软件生产率有三个途径：良好的教育、更好的开发工具和避免编程。避免编程是指在一个新系统中重用已有的软件。尽管标准库的重用已被广泛实践，也有大量未整理好的重用库，但还没有在任何软件工厂中均可使用的软件，有效的软件重用的主要障碍是智能的编目、检索和匹配。

更好的开发工具可以提高生产率，减少程序员的重荷，可以改善软件质量，防止出

44-49 需求工程 形式化 软件工程 (8)

计算机科学1992 Vol.19 №.6

需求工程的形式化途径

钱家骅 吕建国 洪 梅 徐智晨 TP311.5

陈 锋 田 忠 郭孝洪 刘 畅

（复旦大学计算机科学系，上海200433）

摘 要

Requirement Engineering is the first step in software engineering life cycle. The computer support for requirement engineering is an urgent and difficult subject in the research for software production automation. In this paper, the definitions for requirement engineering and the related concepts are given, main problems in the research for formal approach to requirement engineering are presented, and some typical works in this area are introduced and compared according to these problems.

1. 引言

需求工程是软件工程的初始阶段，其总的目标是从用户的模糊而又不完整的要求生成准确的、完整的规格说明。需求工程的研究主要有形式化途径和非形式化途径，后者的出发点是认为需求主要是用作系统开发人员、客户和用户之间的通讯，因而强调需求的可理解性。这方面的工作主要是提出基于自然语言或图形的需求描述方法，使得需求说明书有一定的结构和准确性，且能为非专业人员所掌握。对于形式化途径，形式机制的缺乏使得需求说明书难以验证和确认，难以提供工具以支持需求的分析和需求到设计规格说明的转换，也难以有一个一致的方法来支持需求工程。因此该途径从一严格的形式基础出发，力求寻找需求工程的形式方

错。“工具”这个词有很广的含义，从“思维工具”，如概念、模型和关系，到“工作工具”，如表示、正式模型、验证、程序生成器和编程环境。对一个特定的应用领域或质量特性来说，开发新的或先进的工具是很容易的。只要仔细地研究一个具体单位的软件开发过程，就可以找到其它的改进。

最后，必须改进对软件工程师的教育。目前，软件工程仍被看做是一种技巧，是主程序员或设计者从工作中学到的，而没有看做是已建立起的原理和事实的应用。除了现有教科书中的经验材料，第一步，就是收集和讲授重要系统类型的体系结构。Shaw在这方面开了一个头，但还需要更多的系统类型。就象学生查阅，而不是重新发明算法一样，他们能够回忆或查阅软件体系结构，而不是每次重起炉灶。从预备更好的人才出发，应大大地不断改善软件工程教育。（参考文献略）

〔陈海东 译自“第十四届软件工程国际会议论文集”，1992 ACM 0-89791-504-6，董士海校〕