软件重用技术

金淳兆　余　江　全炳哲

（吉林大学　计算机科学系）

摘　要

软件重用技术是软件工程领域中一个目前比较活跃的分支，它以提高软件生产率，增加软件的可靠性为主要目标。近年来在国内外都有较快的发展。本文较全面地综述了软件重用的各种途径，现有的一些主要方法，以及可能产生的某些问题。

为了缓解今天的软件危机，许多人寄希望于软件重用技术。软件重用不仅能够提高软件生产的速度，提高生产效率，更重要的是它使软件的可靠性得以进一步的保证，因为组成软件的构件都是经过反复测试过的，是高质量的。

软件重用和重新工作的思想可以追溯到五十年代早期，那时计算机工业刚刚诞生。软件重用是指一个实体应用于不包括最初使用它的任何上下文中，被称为"黑箱重用"。一个实体必须经过修改后，才能应用于一个新的上下文中被称作"软件重新工作"，亦即"白箱重用"。随着子程序的出现及后来子程序库的建立，这些思想得到了发展。

尽管目前软件重用还多集中于编码阶段的部分工作，但现在正在开展对于更大范围的软件重用的研究工作。

一　软件重用层次

软件可以在四个层次上重用：重用数据，重用代码，重用设计，重用规范说明。

1.1　重用数据　重用数据是指程序不做任何修改（甚至输入输出数据的格式也无需改动）就可以从一个环境移到另一个环境中使用，其运行将得到同样的结果。这种软件是完全可以重用的。例如，某些程序在不同的机器上或不同的操作系统环境下运行可以获得相同的结果。

1.2　重用代码　程序的代码基本上不用改动，程序段中基本控制结构和数据结构也不需要改变，但必要时可能对某些部分做些小的变化，这种变化发生在以下情况：

（1）程序运行环境的变化，使得运行结果发生变化。例如：编译程序不同，操作系统环境不同，配置发生变化等等，都可能使程序运行后产生不同的结果。

（2）非本质的需求变化，如提示信息的修改，重复次数的增加，计算速度的改变，输入、输出格式的微小变化，等等。这里对于源程序的修改多数是针对程序中作为控制使用的数据进行的。也许是对于源程序中功能的增删。

1.3　重用设计　软件的设计与实现是两个不同的阶段，对于同一个软件的设计，可以采用不同的实现方法，这里的设计是可重用的。例如，用不同的程序设计语言来书写实现部分；在具体的软件实现中对原来软件的控制结构和数据结构的局部修改等。这可能是由于环境的改变，要求程序移植，语

言改变，或者需求变化，从而引起程序的修改。

1.4 重用规范说明 这是指基本需求不发生改变，或者是某一新问题与过去的某一软件在某种抽象层上属于同一类，因此可以使用（或参照）原规范说明。在需求发生非本质变化的情况下以规范说明为基础重新设计，或在与原软件同构的情况下，开始新的设计。

需要说明两点：

(1) 重用概念是模糊的，重用程度在各具体问题中也不一样，重用绝不是指软件百分之百是重复使用，这样的重用软件几乎不存在，或者说研究这种重用性意义不大。

(2) 上述四个层次界限是模糊的，有时很难分清某种重用性是属于哪一层次，在两个层次的临界区部分的情况可能要占多数。

二 软件重用途径

根据被重用软件成份的形式，人们把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(building block)，最理想的情况是在应用时不需改变，但这种理想很难实现，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。这方面的例子有：代码框架〔LANE84，CAVA83〕，子程序〔LANE84，CAVA83，RICE83〕，函数，程序和Smalltalk〔GOLD83〕风格的对象。

合成技术的另一种是从构造块衍生出新的程序，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（构件）。UNIX〔KERE84〕的管道机制是一个好的例子。Smalltalk中构件合成的两个原则是消息传递和继承。

在生成技术中，构件是程序的模式（patterns），通过一个生成程序产生新的程序或程序段。产生的程序可以看作是模式的示例。可重用的模式也可取两种不同形式：代码模式和规则模式。前者的例子是应用产生器，可重用的代码模式存在于产生器自身；后者的例子是变换系统，它利用变换规则集合。与构造块的重用相比，代码模式和规则模式的重用更趋于某些具体的应用领域。但它们具有共同特点：都难于刻划它们的性质，难以恰当地管理和应用。Draco〔NEIG83〕系统是应用产生器和变换系统的一个典型实例。

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超高级软件规范说明语言，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），把用超高级规范说明语言写的程序转化成为某种可执行语言程序。这种超高级语言抽象能力高，逻辑性强，形式化好，便于软件人员维护，也便于程序的修改。

目前，这两种方法各有优缺点，但都还不成熟。对于构件方法，构件的描述、组织、管理、使用等都存在一定问题。一般说来，构件的描述语言应选择一种不依赖于具体机器，具体语言的较抽象语言，否则由于对机器和语言的依赖程度高，不但会降低软件的可重用性，而且必将受到机器和语言的限制。例如用某种高级语言写的构件将会受到该语言中数据结构、控制结构的约束，不易在其他程序设计语言中找到相应的概念或结构。另外，大批构件必须用一个数据库来管理，对于这种专用数据库来说，库的管理、库中内容的组织、分类、检索等都是新的研究课题。总之，构件应具有独立性，可读性，易理解性，正确性，可靠性，易修改性等等。

对于变换方法，由于形式化程度很高，抽象性强，不易被普通程序设计人员掌握。另外，用形式化方法描述某些实际问题时，

也有一定的困难，而且经过多次变换产生的可执行程序的效率较低，在时间、存贮空间的需求方面要求很高。这些问题有待进一步研究。

总之，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，以知识库作为辅助工具，可以促进软件重用的研究。

三　代表性的方法

3.1　子程序库法

一个应用于专门领域的、精心定义的子程序库，如数学子程序库，对软件的重用来说是非常希望的。库中的每一个子程序应属同一问题领域，每个子程序都是精心设计的，并具有良好的接口说明。子程序的参数及供选择的项目都尽可能地少，库中的每一个项目都是静态的，为的是子程序库能够重用，子程序之间的关系是松散的，使得每个单个子程序能够被独立地使用，也应该能够被独立地编译。

子程序库法是成功的，因为它紧密地匹配了在真实世界中已被彻底理解了的对象。它需要较高的技术支持，包括存贮、分类，根据功能检索等。该方法的主要不足在于它高度地依赖于书写子程序的程序设计语言和具体机器的特性。想要创造有效的、可靠的、便于所有系统使用的子程序是非常困难的。此外规范说明与实现的分离，一致性的文档管理等都是需要注意的问题。

3.2　程序生成器

程序生成器是这样一个特殊软件系统，它接收一些特殊的描述信息作为规范说明，产生符合要求的具体程序。程序生成器可分为两种形式：

（1）表驱动方式　软件的设计采用了一种独特的方式，把程序中的控制信息从程序中分离出来，形成一个抽象程序与控制信息表两部分，该抽象程序是可重用的，在具体的应用中不必改变，而控制信息表是可变的。由于可采用不同的控制信息表，就会产生不同的运行结果。抽象程序和控制信息表是建立在一个统一的标准之上的。控制信息表的结构通常比较繁琐，用手工编制是难以完成的，通常利用一种工具——表生成程序（称为程序生成器）来产生。该程序的输入是比较简单的、用户容易使用的描述语言，也可以采用问答方式及菜单选择等简单方式。编译程序的编译就是一个很好的例子。

（2）目标程序方式　程序生成器在一个专用的程序代码段或程序代码模式库的支持下工作，依据输入的需求信息，程序产生器自动生成符合要求的具体程序或程序段，并按照目标程序描述语言语法的要求，生成必要的说明或语句，使之与库中的程序段一起组成可执行的目标程序（可以是高级语言的，也可以是其他语言的）。这种工作方式与编译程序中代码生成部分的工作类似。不同语言之间的转换多采用这种方式。

程序生成器是行之有效的，特别是对于那些已经被人们完全了解了、全部掌握了、并且标准化较好的应用领域。但是它也有一些不足，如应用面较窄，已经建立的并且当前正在使用的标准今后难以修改，生成的程序难以与其他程序联合使用等。

3.3　面向对象方法

面向对象的程序设计技术被认为是软件工程发展的另一个主要方向，它使问题空间与计算机世界对应，问题的解空间与解此问题的程序对应，在此基础上建立起一套全新的软件开发方法，提高了程序设计的效率和可靠性。

在现实世界中的每个客体，对应于程序中的一个对象，具有共同性质的对象的抽象称为类。在类的定义中，定义了这一类对象的属性名称，及对于这些属性允许的操作等。类与类之间可以有继承关系，子类的对象可以使用父类（及其祖先）中定义的操作，也具有它的祖先中的属性。对象之间通

过消息传递完成某些特定的任务。这种方法支持了数据抽象和隐藏等原则。按着这种方式定义的类是可以重用的，通过继承重用其他的类。

面向对象的软件开发方法是直观的，但目前的面向对象的程序设计语言还不是很成功的，比如在描述问题能力方面，所需要的处理时间和空间方面等，有待进一步研究。另外，对象的分类，类与类之间的关系，类的组织等都是面向对象技术要解决的问题，它们是程序设计的基础，也是与软件重用相关的。应该建立一些基本原则，为进一步软件重用打好基础。

3.4 Ada与软件重用

Ada语言是面向过程的高级程序设计语言，它支持现代软件工程的基本原则，支持面向对象的软件开发方法，提供了许多支持软件重用的机制。Ada在代码、规范说明、参数化类属模型方面支持了软件的可重用性。另外，Ada有丰富的程序单位，如子程序、包、任务，它们都是可重用的。

Ada的程序单位采用规范说明与实现部分分离的的方式，使得规范说明是可重用的。规范说明部分描述了与外界的关系。包括对其他程序单位的引用，及提供给外界使用的对象或操作的接口说明。实现部分给出数据表示方式及操作的具体描述。当实现部分变化时对规范说明部分没有任何影响。

Ada的类属程序单位是参数化程序设计的体现，类属程序单位可以是类属子程序和类属程序包。类属程序单位是在一般程序单位上增加了类属参数。通过对类属参数的示例化，生成可以使用的程序单位。类属参数可以是常量、变量、类型名、子程序名，其中通过对于类型名和子程序名参数的示例可使类属程序单位生成各种具有很强功能的程序单位。示例化过程是极其简单的，它们相当于对于某程序单位中类型和操作的替换，因而它的可重用性是极强的。

Ada的分离编译功能，支持了对于程序库的使用，程序库中可以有一般的程序单位，也可以有类属的程序单位，而且规范说明与实现部分也可以分别编译和保存。此外，Ada的程序设计环境为Ada语言的使用提供了有力的支持，环境中提供了许多有助于Ada程序的设计、编译、修改、维护等有力工具。

但是Ada语言中没有提供直接支持面向对象程序设计中继承等机制。尽管有人在探讨用语言的其他机制来模拟这些机制，但仍有很多人认为Ada语言不是真正的面向对象的程序设计语言。

四 矛盾与问题

在建立一个实用的支持重用技术的软件系统之前，在确定该系统设计目标时，首先会遇到许多矛盾和问题，需要在某种标准的基础上折衷考虑。

4.1 两难问题

(1) **通用性问题** 各种软件技术都存在通用性与处理能力的矛盾。通用性强则对具体问题（特别是某些特殊问题）的处理能力就会差些。反之，若对于某些具体问题的处理能力强，对于其他问题则未必适用。在[Austin 87]中给出了通用性和处理能力的二维图示，图中对现有的十余种软件技术，包括代码框架，面向对象的知识库，子程序库，数据流语言，变换技术，形式化方法等作出了评价。作为两个极端的形式，处理能力最强的是应用产生器。而通用性最强的是汇编语言，尽管它对于问题的描述能力极弱，但却可用于解决任何问题。

(2) **构件规模与可重用潜力** 构件的规模与它的易重用性成反比，构件规模大、功能强，则适应性差，很难与所需要的吻合，因而为了满足应用的要求在对构件作修改时需要花费较多的精力。

(3) **经济方面** 软件重用技术的主要目标之一是要降低软件的开发成本，取得较大的经济效益；但是，在把该技术用于实际的软件系统开发之前，需要大量的投资来建立可重用的软件构件库。

4.2 技术问题

目前许多人认为，软件重用技术领域中必须要解决的技术问题有：

(1) **构件的分类与查找** 经过特殊方式设计好的构件，首先遇到的问题是怎样分类和组织。从不同的观点出发，对于构件的分类和组织方法是不同的，这不仅与构件的应用领域有关，更主要的是它与对它进行分类和组织的操作者有关，但总的原则应当是尽量为它的使用和检索提供方便。

要在一个复杂的构件库中找出与使用者的要求适合程度最高的构件决非易事。在多数情况下，库中的构件不能完全符合使用者的要求，但是使用者却非常希望找出与要求最接近的构件（或一组构件）。这里有必要建立一套完善的衡量标准，综合地衡量构件的符合程度。

为了把可重用的构件加入到构件库中，有必要提供一种构件说明语言，指示对该构件的分类和管理。还应该为构件库的使用者提供一种构件查询语言，用来表达他们所需求的构件。

(2) **构件的理解** 已经查找到的构件，如果完全符合要求，可以直接使用。但是更多的情况是必须经过修改之后才能使用。在修改之前，首先要对构件进行理解，掌握构件的功能，数据结构的使用等。分清哪些部分满足要求，哪些部分与要求无关，还需需要增添或修改哪些部分等。因此，需要提供一组构件的辅助理解工具，为使用者理解构件，参考构件的文档等提供方便。

(3) **构件的修改和调整** 经常需要对查找到的构件进行修改或调整，以满足使用者的要求。有时只需要使某些参数示例化，如Ada中类属程序单元的示例，代码框架的填充等。有时也需要对构件进行修改，做些必要的增删，这是件不容易的事，尤其是复杂一点的构件，因为必须对构件保持原上下文一致，还要尽可能地保持原设计风格。

(4) **构件的组装** 把调整好的构件组装起来，可能还需要增添一些必要的东西，以满足各构件之间接口的要求，多个构件可以组合成一个更大的构件，或成为可用的程序段，构件的组装也需要有效的工具来支持。

以上各方面对于工具的依赖都很大，如果能够提供一组智能化程度较高的工具支持构件的重用将会是更有效的。

4.3 潜在问题

在软件的生产过程中，大量地使用可重用的构件将会使程序设计（以致软件开发的整个过程）产生一次重大变革，它将打破现有的程序设计（软件开发）方式。

程序员培训问题 程序设计人员将受到一种特殊的训练，否则他们将不但不会提高程序的生产率，甚至适得其反，因为那些可重用的软件构件的存在，干扰了程序员原有的思维方式，他们要花费一定的时间去理解那些最后才知道适不适合他们使用的构件，他们的主要工作从设计程序或程序段转到修改和组装软件的构件。一个良好的支持软件重用的开发环境将对他们有巨大的帮助。培训程序员的重点将是怎样去理解构件，怎样组装构件，以及怎样有效地利用支持软件重用的开发环境等。

效率问题 这里至少有两种效率需要考虑，即程序员的工作效率以及他们开发的程序的执行效率。程序员之间的工作效率的差距将会比现有的差距更大，主要与程序员的素质及经验密切相关。最终程序质量的好坏与被选取的构件、构件的组装等都直接相关。在组装构件时，为了符合构件的接口要求，可能要设计一些辅助代码，这种代码若是大量出现，将使程序的有效执行时间相对地减少，最终也许不能达到需求说明书中规定的指标。

环境的改变 与现在相比，将来的程序员对于软件开发的支撑环境的依赖程度会更大，当他们的工作环境发生变化时，他需要花费大量时间来熟悉新的环境。

还有一些其他问题，也许是目前没有预料到的，我们应当有足够的精神准备，以便采取相应的对策。

重用软件技术的研究现在只是刚刚开始，只有很少通用性的实验系统，也都只是处于原型阶段，更多的是提出了设计方案。已从不同角度开展了很多研究，在可重用技术的各方面都取得了一些成果。

目前人们的研究重点是集成化的支持软件重用技术的环境，包括构件的检索，理解，修改，组装等各种工具。

未来的发展方向是：

1.进一步设计和完善集成化的开发环境。

2.开展软件构件及构件组织的标准化研究。

3.把软件工程与知识工程相结合，利用人工智能技术，实现构件的查找、理解、组

装等，实现程序的自动生成。

4.针对具体应用领域，设计有效的构件，带动通用软件重用的研究。

还有一些其他的相关问题，有待进一步研究，但是我们可以期望，在不久的将来，对于软件重用技术及其相关技术将有较快的发展，将会使软件生产方式产生一次革命。

参 考 文 献

〔CAVA83〕 Michael J. Cavaliere and philip J. cham beault, "Reusable Code at the Hartford Insurance Group" Proceedings of the Workshop on Reusability in programming (September 1983)

〔GOLD83〕 Adele Goldberg and David Robson, Smalltalk-80: The language and its implementation. Addison-wesley (1983).

〔HIRO88〕 Hiroyuki Tarumi, Kiyoshi Agusa, Yutaka Ohno, "A Programming Environment Supporting Reuse of Object-Oriented Software" Proc. Tenth Int'l Software Engineering Conf., (1988).

〔KERN84〕 Brain Kernighan, "The Unix System and Software Reusability." IEEE Transactions on Software Engineering, Vol.SE-10, No5, Sept. (1984).

〔LANE84〕 Robert G. Lanergan and Charles A. Grasso, "Software Engineering with Reusable Designs and Code." 同上.

〔RICE83〕 John R. Rice and Herbert D. Schwetman, "Interface Issues in a Software Ports Technology." Proceedings of the Workshop on Reusability in Programming (September 1983).

〔TED87〕 Ted Biggerstaff and Charles Richter, "Reusability Framework, Assessment, and Directions". Proceedings of the Twentieth Annual Hawaii International Conference on System Sciences, 1987.

〔NEIG83〕 J.M.Neighbors, "The Draco Approach to Constructing Software from Reusable Components." Proc. Workshop Reusability in Programming. ITT. Straitford, Conn. 1983.

（上接44页）

远，优越性越大。

三、小结

本文针对现有汉语词切分词典匹配法中检纠分错误方法的不足，特别是其独立于句法语义分析，难以作为汉语理解的有机组成部分的现状，提出了在汉语理解中，利用词法，句法和语义知识帮助检出切分错误的方法，初步解决了汉语理解与词切分结合起来的问题。同时，设计了一种后加词典，以提高纠错时判断重切点的效率。当然，这种结合句法语义知识检纠切分错误的方法尚待进一步研究，另外，对于遇到新词，计算机如何处理，也是颇有意义的研究课题。本文对汉语理解中的词切分方法，提出了一些初步想法和观点，有些已在计算机上实现。欠妥之处，尚请指正。

致谢：本文工作是南京大学计算机软件研究所研制的汉语会话系统NDTALK的一部分，这项工作始终得到徐家福教授的指导，NDTALK小组成员也提出了有益的建议，在此一并表示感谢！

参考文献

〔1〕 朱德熙，"语法讲义"，商务印书馆，1982.

〔2〕 梁南元，"书面汉语自动分词与一个自动分词系统——CDWS"，北京航空学院学报，1984.4.

〔3〕 梁南元，"书面汉语自动分词综述"，计算机应用与软件，1987.3.

〔4〕 杨春雨，刘源，梁南元，"论汉语自动分词方法"，中文信息学报，1989.1.Vol.3 No.1

〔5〕 曾纪文，谷新英，"结合上下文辅助分词的学习系统"，中文信息处理国际会议论文集，1983. 北京.

〔6〕 伊波，杨抒，"Hash树，——一种词典存放方法."1989.待发表.