软件复用*

杨芙清 朱 冰 梅 宏

(北京大学计算机科学与技术系,北京 100871)

摘要 本文总结了软件复用技术的基本概念,介绍了有关软件复用的理论研究和实践活动,指出研究软件复用技术存在的关键问题,并给出了可选的解决方案. 最后,说明了软件复用与 CASE 环境的关系.

关键词 软件复用,可复用性,组装,生成,抽象,可复用构件.

60 年代的软件危机导致了有关软件复用的研究. 在 1968 年 NATO 软件工程会议上,会议的邀请论文"Mass Produced Software Components"首次提出可复用库的思想. 由于软件复用技术有助于提高软件开发的生产率,提高软件系统的可靠性,减少软件维护的负担,因而大专院校、研究所、企业界和政府部门都很重视软件复用的研究和实践.

80 年代中期,人们认识到复用是优秀的软件设计的关键因素之一,并且软件复用已在子程序库、报告生成器、编译器的编译器等方面取得进展,还认识到在软件复用实施中管理因素非常重要. 当时工业界复用软件的主要手段是复用整个软件.

到 90 年代初期,软件复用的实践有 3 个趋势,一是在软件界将软件复用的实践惯例化、用户化,不仅要考虑技术的因素,而且要考虑管理的因素;二是将复用技术集成到软件开发过程中,并且研究软件过程形式化的问题;三是将领域分析标准化,开发支持领域分析的方法和工具.

现在,有关软件复用的基本概念不断完善,软件复用的技术和方法也日渐成熟. 有些专有领域的可复用系统较为成功,通用领域的可复用系统则处于探索阶段. 展望未来,可以预测到:软件复用的技术和活动可能变得系统化和形式化;会有更多的组织使用复用程序;可能有支持复用和领域分析的软件开发环境和 CASE 工具产生;可能出现特定领域的可复用软件构件工厂;可能形成支持领域或领域间的复用软件的开发标准等.

1 基本概念

1.1 软件复用和可复用性

* 本文 1994－08－18 收到,1995－03－06 定稿

本项目由 863 高技术计划资助. 作者杨芙清,女,1932 年生,中国科学院院士,教授,博士导师,主要研究领域为软件工程,系统软件,面向对象技术. 朱冰,女,1967 年生,1995 年博士毕业于北京大学,主要研究领域为软件工程,面向对象技术,程序设计方法. 梅宏,1963 年生,副教授,1994 年 10 月博士后出站,主要研究领域为软件工程,新型程序设计语言.

本文通讯联系人:杨芙清,北京 100871,北京大学计算机科学与技术系

© 中国科学院软件研究所 http://www.jos.org.cn

软件复用是指在开发新的软件系统时,对已有软件的重新使用.该软件可能是已存在的软件,也可能是专门设计的可复用构件.

从软件工程角度看,软件复用发生在构造新的软件系统的过程中.例如,在一个程序的构造期间,对已存在的源代码的使用就是软件复用.但在程序执行期间重复调用某段源代码,则不属于软件复用;另外,程序的重复运行、为完成分布处理而进行的拷贝也不属于软件复用.

最为系统的、最为工程化的软件复用活动是基于复用库的软件复用.一般来说,基于复用库的软件复用过程可归纳为以下几步:(1)认知:即认识到可以有复用的机会;(2)分解及抽象;(3)分类并建立复用库;(4)检索与选择;(5)对复用库中的构件具体化;(6)重新组装.

软件的可复用性是指某软件产品在构造新的软件系统的过程中能再次被使用的能力.

1.2 一致性和不变性[1]

一致性和不变性是与工程有关的两个概念.一致性是指软件构件之间和软件系统之间的相似性.它有助于开发复用系统,因为如果构件不具有一致性,则很难分析和综合复用构件,也很难实现软件开发的自动化.

由于软件在开发和使用过程中不断演化,因而产生了不变性概念.不变性指软件在演化过程中具有的相似性.不变性对软件的维护极为重要.

例如,标准软件构件和标准软件构件体系结构的实例具有一致性和不变性.标准软件构件是能显示所有构件的公共特性的抽象构件.它的实例具有一致性,标准软件构件的概念有利于基于复用的大型软件开发.标准软件构件的思想已应用于细粒度可重用构件的开发中.而标准软件构件体系结构则是指如何组合可复用构件的规则.可以有不同的组合模式,一般为层次结构,如非循环有向图.构件本身也可以是构件的组合.

1.3 软件复用技术

软件复用技术分为组装技术和生成技术.

组装技术是利用组装方式来复用软件构件.采用组装技术时,对已有软件构件不作修改或仅作很少修改,就将软件构件插装在一起,从而构造出新的目标系统.其思想类似于将集成电路芯片插装成硬件系统.如较为成功的子程序库技术和正在探讨之中的基于面向对象思想的软件IC技术.

生成技术由程序生成器来获得复用.它是对模式的复用.生成器导出模式的专有的或定制的版本,来实现模式的复用.模式相当于"种子",从中可以生长出新的专有的软件构件.

1.4 抽象,选择,实例化和集成[2]

抽象是软件复用的关键.抽象是指对复用对象的概括提炼,即将基本的操作和处理对象从语言、机器和其他环境细节中提炼出来.每一个抽象可以描述相关的可复用对象集,而每一相关的可复用对象集决定了一个抽象.软件抽象有抽象规范和抽象实现两个层次.能否在软件工程中成功地应用某一种复用技术,取决于复用技术所具有的抽象层次.抽象层次越高,复用的可能性越大.

选择是存放、比较和检索可复用对象的功能.

实例化指对类属复用对象进行参量提供、转换、约束或某种形式细化的技术.

集成是根据复用技术提供的框架,将选定的已实例化的复用对象组合成完整的软件系

© 中国科学院软件研究所 http://www.jos.org.cn

统的过程.

1.5 软件复用方法

根据抽象、选择、实例化和集成,可将软件复用方法分成 8 类[2]:高级语言形式,设计和代码整理形式,源代码构件形式,软件模式形式,应用生成器形式,甚高级语言形式,转换系统形式,以及软件体系结构形式.

高级语言中可复用的对象是汇编语言模式,高级语言是抽象规范,其对应的汇编语言部分是抽象实现.

设计和代码整理形式:可复用对象是源代码片断,代码整理是从已存在的软件代码系统中整块地进行拷贝,设计整理是拷贝大块代码后,仅保留设计的全局样本,而删除许多内部细节.

源代码构件形式通过构造可复用库来实现该形式的软件复用.可复用库要提供描述复用构件的行为的抽象规范说明,还要提供分类和检索可复用构件的模式.可复用构件方法在专有领域,如数学分析中的 IMSL 数学库,卓有成效,但在通用领域收效甚微.原因在于数学库能采用一字检索,即一字抽象规范形式,如 SIN 来描述正弦函数.

软件模式是可复用软件构件的形式化扩充.软件模式复用对象是算法和数据结构,软件模式的抽象规范是算法或数据结构的形式化描述,而抽象实现对应于模式初始化时产生的源代码.

应用程序生成器复用完整的系统设计,如专家系统生成器,编译器的编译器,面向结构的编辑器的生成器等.

甚高级语言即为可执行的规范语言.它适合于通用软件开发,而应用程序生成器适合于特定领域.

转换系统复用方法的关键在于软件开发者只需维护和提供系统的规范说明,而不用维护和提供实现;对规范进行修改时,常可复用以前的大部分历史开发信息.

可复用的软件体系结构是指大粒度的软件框架,以及软件系统全局的结构设计.如在不同的领域实例化并且复用数据库子系统;将不同的词法分析器、语法分析器和代码生成插入到一个编译器框架中;专家系统中的基于规则和黑板的结构;示波器的设计框架等都是可复用的软件体系结构的实例.

1.6 宽域语言和窄域语言

语言机制是软件复用的重要问题.复用所涉及的语言可分为宽域语言和窄域语言等.

宽域语言是为软件生命周期各个阶段,即从规范说明到可执行代码提供丰富表达能力的语言.

窄域语言是为程序开发过程某一阶段使用的语言(如有规范说明语言,设计语言,可执行语言等),而程序开发过程就是从一种语言的描述转化成另一种语言的描述的过程.

2 支持复用的理论研究和实践活动

为了提高软件开发和生产的竞争力,开发和研制了许多支持复用的软件系统.从 1984 年开始,HP 公司开展几个软件复用项目的研究.一些使用 DBMS 和库来存贮和分布复用构件;另一些采用 C++或 Objective-C 来开发类库.开发 Ada 语言的目的之一就是为了复

© 中国科学院软件研究所 http://www.jos.org.cn

用.日本软件工厂通过多年的努力,已获得许多复用的经验.目前,软件工厂主要以非面向对象技术实现系统复用,面向对象技术还处于研究阶段.系统复用的效果可以由生产率和产品质量来衡量.下面具体介绍几个支持复用的理论研究和实践活动情况.

2.1 Toshiba 软件工厂[3]

日本 Toshiba 软件工厂中,最为关心的问题是质量控制和提高软件生产率.重点在于复用已存在的软件模块.对每个软件模块,定义了需求分析级、设计级和程序级等三级抽象,并对各抽象级别进行规范说明.另外,给出模块的"表示"(Present).表示是指程序模块的规范说明,以及在其它的应用中复用程序模块时要对它做的修改范围说明."表示"概念的引入,清晰地介绍了程序的行为,大大地提高了生产率.而且,还建立软件模块的抽象级别间的映射,使得表示与可复用模块间具有可跟踪性.为提高软件工厂中每个程序模块重用的频率,采用下列规范和策略:(1)针对复用而重写已存在模块;(2)对已存在模块书写"表示";(3)将"表示"存入 CASD(计算机辅助软件设计支持系统);(4)设计新软件时,要求设计者遵循以下过程:浏览已存在模块的复用;找到匹配,则做相应文档工作,并复用之.

2.2 商用软件的复用[4]

多年的商用软件开发经验表明:60%的商用应用程序是冗余的、可标准化和复用.针对已存在软件,研究和开发可复用模块.可复用模块的设计包括功能模块的设计和 Cobal 程序逻辑结构的设计.功能模块是为特定目的而设计和编码的模块.能在标准拷贝库上浏览、测试、归档和存贮.已有近 3 200 个模块,支持 3 个工厂的 50 个系统应用程序.Cobal 程序逻辑结构则是从许多程序中提炼出来的程序框架.其好处:(1)有助于程序员阐明他的编程思想;(2)尽早地浏览设计和程序;(3)有助于分析员和程序员讨论系统需求;(4)方便测试;(5)由于已建立的逻辑结构是经测试过的,故能消除某些如文件的结束引起的易错情况;(6)能减少程序准备时间;(7)有助于理解和修改他人的程序.

2.3 Draco 系统[5]

Draco 系统提供利用复用技术构造软件系统的途径.除代码重用外,重点考虑了分析和设计信息的复用.这样,构造相似的系统时,可以提高软件专家的生产率.具体而言,根据问题域来组织复用软件构件;这些特定领域的程序语句被源代码到源代码的转换器优化,再细化至其他领域.

2.4 Genesis 系统[6]

Genesis 系统是一个支持大型软件项目的基于软件工程的程序设计环境.Genesis 系统采用组装技术来实现复用.它具有可检索性、可组装性、可理解性等特点.Genesis 系统采用实体关系模型,它使用的 ESL 实体规范说明语言允许插入、修改和删除系统中的有关的多个实体的信息;Genesis 系统使用了一个数据库来存储所有的这些信息.Genesis 采用一种比 UNIX 管道更通用的方法,即功能组装法来作为其组装机制.Genesis 系统提供 C 摘要器和 Infoview 工具来加强模块的理解性.C 摘要器从 C 程序中抽取有关对象的位置,对象之间关系等信息存入程序数据库中,而 Infoview 工具可用查询命令从数据库中查询这些信息.如果程序设计时不考虑应遵循的规则,则所有关于可复用的概念都是无用的.Genesis 有以下规则:①复用性选择;②复用级别;③增加其通用性以便复用;④可复用性评估标准.

Genesis 系统驱动复用的算法如下:分解给定部件;是否在库中?若在,复用该部件,否

© 中国科学院软件研究所 http://www.jos.org.cn

则,再分解,直至不能再分;则解决一个更为一般的问题,并且加入文档;它能否通过可复用性测试? 若能,把它加入库中,否则,解决一个更为一般化的问题.

2.5 C 源代码清理系统[7]

C 源代码清理系统提供从已存在软件系统中清理出可复用构件的机制. 系统从已存在软件中探测出具有较高可复用性的构件,而不是给出所有有可能复用的构件. 系统由一组工具集组成,采用专家系统技术. 工具集使用的知识库主要来源于原始软件系统开发人员采用的知识,该知识局限于与复用有关的具有复用特性的所有抽象级(高级、低级和实现)的设计知识,该知识有助于增强复用工具的查询能力(从原始软件系统中查询);其次来源于应用领域的知识,该知识有助于增强系统的选择功能(选择进生成的复用库).

工具系统包括 C 分析器,Prolog 解释器和交互式前端. C 分析器输入 C 源码,在主存中生成整个程序的抽象语法树;查找此树进行模式匹配,将匹配的模式信息转换成 Prolog 事实,并输出到一个文件中. Prolog 解释器对 C 分析器产生的供候选的可复用构件进行推理,推理器是由用户的需求动态启动的,它将满足规则的源代码部分报告给用户. 前端允许选择要分析的程序,启动 Prolog 分析器,给用户显示分析的结果.

2.6 支持复用的分布式应用系统[8]

文献[8]中的分布式系统将应用领域模型化,形成可用消息方式通讯的并发对象系统,在模型化过程中,开发可复用规范说明、可复用结构、可复用构件类型,并将它们放入可复用库中. 再通过裁剪可复用规范说明和结构来生成目标系统. 本分布式系统采用可演化的领域生命周期模型,该模型是一种软件生命周期模型,不同于传统的瀑布模型的是它没有软件开发和软件维护的区别,系统是通过几次迭代过程来演化的. 每次迭代,都能适应需求分析的变化. 该模型适用于具有软件升级特性的系列软件的开发. 模型包括 3 个主要活动和一个复用库. 3 个活动指领域模型化(输入领域需求信息和变化的需求信息,输出可复用规则说明、结构、构件类型到复用库中),目标系统生成(输入目标系统需求分析和已存在的可复用构件类型,输出新的可复用构件类型、要变化的需求信息、以及目标系统的规范说明、结构和构件类型.),目标系统配制(输入目标系统的规范说明、结构和构件类型,以及目标系统配制数据,输出目标系统实例).

3 软件复用的关键问题与解决方案

3.1 领域分析

要实现复用,必须对要复用的领域有足够的了解. 要在一个崭新的快速变化的领域里实现复用,则支持复用的系统要能适应不断变化的需求要求. 面向对象的基于类库的软件系统是解决这一问题的较优选择.

但是,在粗粒度继承的类库中,由于设计者不能预见将来复用的具体环境,类定义的适应性不强,不利于对该类库的复用. 可以用细粒度继承[9]的方法来设计类库以利于复用. 细粒度继承是用以增加设计代码的适应性的一种设计继承图的方法. 它设计的库具有如下特性:(1)每个类或者从其它类继承特性,或者声明它自己的特性. (2)声明特性的类很小,常仅包含一个特性;仅非常相关的特性能在类中声明. 细粒度继承与功能分解无关. 前者回答是什么,后者回答怎样做. 细粒度继承使得"设计继承"非常繁琐,但有利于"实现继承"的设计.

© 中国科学院软件研究所 http://www.jos.org.cn

其中,“设计继承”的目标是子类,“实现继承”的目标是复用.

3.2 抽象

复用具有很多优点;然而,除库程序外,要真正实现复用,几乎都要做许多的额外工作,一般程序员开发新系统时,往往情愿从头开始,而不愿过分依附已有程序.其原因在于:在任何高级语言中,都是非常具体的数据表示和算法,而不是某些概念或抽象的规范说明.简单的抽象已被有效的具体实现所代替.如何将基本的操作和对象从语言、机器和其他环境细节中抽象出来是关键问题.焦点在于语言机制的研究上.解决的方案是抽象程序和规范说明.抽象程序采用对问题域极为自然的表示法来编写.抽象程序关键在于:(1)抽象程序是机器可表示的;(2)存在对抽象程序进行语法分析和语义分析的工具.另外表示抽象结构时,应选择对问题域最为自然的表示法.例如,使用一个基语言中提供的符号,再加上语法扩充,即允许程序员指定一些符号作为操作符,说明其优先级和结合性,以此来为不同高层结构提供新的表示法.复用抽象程序的方法之一是从单一的抽象程序中产生一系列具体程序,而每一个具体程序对应于某一特定的目标环境.规范说明使软件从它的实现细节中分离出来并给出了有关概念的清晰说明,并且使它们在其它软件的规范说明中有可能重新被使用.

3.3 复用库的组织

复用库中,如何组织复用构件的方案之一是使用层次分类模式.便于检索,设置许多复杂的索引.这些索引详细程度不同,形式化程度不同,范围很广,包括关键字到形式化公理.

3.4 可复用构件的分类

当讨论的可复用构件来自各种领域、采用不同的设计方法并用不同语言编写时,为了跟踪可提供的所有构件的特性,构件管理系统是非常必要的.当构件的数目很大,如大于1 000时,人们就很难同时处理有关的信息,构件管理系统就要提供相关构件的跟踪信息,这样,就要存贮许多细节,并能比较这些细节以选择最合适的构件.复用时对可复用构件作一些小的修改是很必要的,因而构件管理系统要能检索到仅需最小修改的构件.由此,对分类模式有如下要求:分类信息要包括从复用者角度反映构件之间相关联系的信息;分类模式要能应用于不同粒度的可复用构件,并能应用于软件开发生命周期的各个阶段;分类模式不能太复杂,否则会增加理解的难度,阻碍复用.

基于关键词的分类模式可分为两类:枚举模式和刻面模式.枚举模式提供一维的信息分类形式,其构件是线性分类的.刻面模式是由S. R. Ranghanathan在1957年提出的.它从不同的方面对构件进行分类,这样可以同时考虑构件的几种不同的方面或特性.这些不同的方面称为刻面.

3.5 检索(定位和理解)

使用大型复用库时,用户可能检索到许多复用构件,而只有一部分对当前任务有帮助.因而,如何定位当前有用的复用构件是很重要的问题.检索与可复用构件的分类是相关联的.下面介绍几种对复用库的检索方法.

(1)以关键字为索引,提供不同的分类方法,针对不同的领域采用不同的分类方法来索引存在不精确的问题.但针对小型复用库,这种方法很有用.因为,在复用库中可复用构件不多的情况下,用户更喜欢将分类目录打印出来,而不是求助查询工具,因为前者更快.分类目录打印要求分类目录必须有许多索引,它能帮助软件工程师从不同的方面来检索复用库.另

© 中国科学院软件研究所 http://www.jos.org.cn

外,常常利用超文本的技术来进行基于关键字的检索.

(2)相关检索是在关系数据库中基于属性值的相关性搜索来检索构件,这种检索需要搜索关键词和属性的完全匹配.

(3)正文检索是根据某一正文模式来检索.这些模式通常局限于正则表达式.进一步的研究是利用自然语言理解的成果来分析词义或语义通过询问方式来检索.

(4)行为样品化技术,即针对软件构件具有给定输入后能执行产生输出的功能,系统产生随机输入向量,用户给出期望的输出,系统再用选择的输入运行构件,将所得输出与期望输出比较,将满足的所有构件提交给用户,这种方法精确地采用了构件的语义.

在对可复用构件定位时,要注意相似构件的区别.因为,为了增强构件的可复用性,可在可复用库中对同一个构件概念提供不同的实现选择,也即孪生构件.用户复用时,可根据需要选择合适的构件实现.为了在库中实现孪生构件,必须要考虑库的空间开销、维护及孪生构件之间相互变换的问题.一般基于软件构件的复用库中,文档用于记录开发信息和说明构件信息[10].文档有利于理解可复用构件.提供复用文档的机制也有利于对可复用构件的进一步理解和检索.复用文档的方法包括面向文档的方法和面向超文本的方法.面向文档的方法重点放在组成文档的字和提要上,面向超文本的方法重点放在同义词和调查表上.文档复用涉及文档的组织、文档的检索、文档的再组织(即通过重用产生新文档).字有利于检索,但不便于理解;提要既方便检索又方便理解,还助于文档的再组织.同义词方便检索,调查表便于理解.通过同义词和文档提要,可以从文档间产生新的文档,以实现复用的目的.

3.6 接口

不利于软件复用的重要阻碍之一就是接口问题.

(1)接口问题也是软件设计方法要考虑的内容之一.在结构化设计中,要求模块高内聚、低耦合,尽可能减少模块间的接口.信息隐藏技术使得接口尽可能明确而稳定.明确是指可见的接口信息是"公有的",并便于理解;稳定是指被隐藏信息的改变尽量不影响接口.面向对象技术通过消息通讯来完成对象的接口,其多态性提出了一种不固定的接口方式.

(2)现在兴起了接口定义语言的研究.接口定义语言简化了软件系统设计,也方便了不同实现方式的构件的互联和组装;另外为解决由系统的演化和修改导致的接口问题提供了一个依托的基础.

接口和实现之间应有较明显的界限.在系统开发的最初,对系统的认识应是一组接口集,而不是具体的实现集.在一个简单的系统中,针对一个给定的接口,应有多种不同的实现共存的可能.

系统的演化时要求保存原有的功能,新的系统可以增加新的接口而不影响已有的实现.IDL是OMG(Object Management Group)的接口定义语言,它是面向对象范型的,并支持多继承.IDL仅关心接口特性(方法和其参量),而不提供任何实现信息.

(3)一般大型应用系统的数据可能使用不同的记录格式[11].和子程序要求的接口相比,可能包含多余的域或缺少一些域,或使用不同的域名,或使用了不同的度量单位.如果接口是固定的表示形式,如子程序通常的接口表示形式,则对于稍微大一点的应用系统,它们使用的复杂数据类型就很难和子程序的接口相匹配.为此,介绍两种方法来解决数据转换的问题.其一是先由用户编写一个数据转换程序,以把应用系统的数据格式转换成子程序所需要

© 中国科学院软件研究所 http://www.jos.org.cn

的数据格式;其二是通过一过滤程序产生接口程序以适应应用系统的数据结构形式.

(4)接口问题还涉及将来复用时语言的实现效率问题.因为复用库中构件一般都具有通用性,若该构件的参量是一个空间很大的数据结构,则复用该构件要拷贝大量数据,开销很大.为此,可用一种交换(swapping)操作原语来代替拷贝(copying)操作的方法[12],以提高实现的效率.拷贝存在的问题有:①拷贝一个大型数据结构的物理开销很大;②尽管通过拷贝指针减少了工作量,但对指针的拷贝不能取代对数据结构的拷贝;因为指针拷贝引入了别名,程序设计时易产生错误,并且削弱了抽象,导致对程序行为的推理更为复杂;③基于拷贝的可复用代码设计注定效率很低.而交换原语的优点是:①无论参量类型复杂与否,类属模块都能有效执行;并且有可能复用类属模块的目标代码,因为对任何类型,交换都以一致的方式统一实现.②虽然变量表示可能涉及指针,但从未引入别名;有助于形式或非形式推理可复用模块的实现行为,以及客户程序对它们的使用行为.

3.7 组织

即使解决了上述问题,还存在复用软件的有效组织和管理的问题.如果管理不好,不便于检查软件构件的规范说明,也就不能确定该构件是否可复用;如果维护的责任不明确,就不能很好地复用软件产品.因而具体要抓以下问题:(1)谁对复用构件的维护负责;(2)识别并确认复用构件的规则是什么;(3)是谁决定来使用复用构件,并准备接收复用构件.

4 软件复用与CASE环境

缺乏复用的支撑环境是阻碍软件复用的极大因素.要真正做到软件复用,必须要有支持复用的软件工具.将支持复用的工具集成到CASE环境中,形成支持复用的CASE环境,是开发软件平台、形成软件产业、从手工编程到软件工厂过渡的台阶.软件复用的各个阶段和各个环节都需要CASE工具的支持.从复用收益看,软件开发的早期复用,即需求分析和设计时的复用是很重要的.通过CASE工具来支持早期软件复用.软件复用技术包括生成技术和组装技术,这两种技术的具体实现都需要有CASE工具的支持.软件复用方法包括高级语言形式,设计和编码清理形式,源代码构件形式,软件模式形式,应用生成器形式,甚高级语言形式,转换系统形式,以及软件体系结构形式等8类方法,这些方法的推广使用必须要辅以相应的支持工具.至于可复用库的管理,可复用库的建立和维护,可复用构件的检索、定位和理解,可复用构件的组装等都需要工具的支持.也需要支持文档管理的工具.

5 结束语

对软件复用的概念和技术已获得许多共识.1983年Freeman指出,软件复用是指在构造新的软件系统的过程中,对已存在的软件人工制品的使用技术.软件人工制品的种类包括源代码片断、设计结构、模块级实现结构、规范说明、文档和转换程序等.1989年Biggerstaff和Richter指出,尽管涉及不同复用技术的软件工程技术差异很大,但存在共同点.例如有软件构件库、应用生成器、源代码编译器、以及类属软件模块等.

但现实世界对软件复用有许多的限制.如软件系统最初并不是为将来的复用设计的;很难将多个部件集成到一个部件中,经验表明,接口不当会产生许多隐患;不容易理解他人的

© 中国科学院软件研究所 http://www.jos.org.cn

程序；即使软件设计时考虑到复用，也还存在一个“限期效应”问题；缺乏支持复用的工具和环境等.

通过对软件复用的研究，有如下体会：(1)当前工业界复用软件的主要手段还是复用整个软件. 提供给软件开发人员的“可复用库”主要是仅可列出函数原型索引的、带功能介绍文档的子程序库. (2)复用的观念应成为软件过程模型的思想之一. (3)好的形式化规范说明语言对于软件复用非常重要. (4)在大规模的开发活动中，软件复用构件的规范说明应该比当前正使用的规范说明更为形式化. 复用软件构件规模越大，对它进行形式化描述的要求越高. (5)通过面向对象技术来研究软件复用，有待于进一步形式化和实用化.

参考文献

1 Weber H. Uniformity and invariance in support of reuse. IEEE Trans. on SE, July 1993, **18(7)**: 2—7.

2 Krueger C W. Software reuse. ACM Computing Surveys, June 1992.

3 Matsumoto Yoshihiro. Some experiences in promoting reusable software: presentation in higher abstract levels. IEEE Trans. on SE, Sept. 1984, **SE—10(5)**: 502—512.

4 Lanergan R B, Grasso C A. Software engineering with reusable designs and code. IEEE Trans. on SE, Sept. 1984, **SE—10(5)**: 498—501.

5 Neighbors J M. The draco approach to constructing software from reusable components. IEEE Trans. on SE, Sept. 1984, **SE—10(5)**: 564—572.

6 Ramamoorthy C V, Garg V, Prakash A. Support for reusability in genesis. IEEE Trans. on SE, August 1988, **14(8)**: 1145—1153.

7 Dunn M F, Knight J C. Automating the detection of reusable parts in existing software. IEEE Trans. on SE, 1993, **20(6)**: 381—390.

8 Gomaa H. A reuse—oriented approach for structuring and configuring distributed applications. Software Engineering Journal, March 1993. 61—71.

9 Johnson P, Rees C. Reusability through fine—grain inheritance. Software practice and experience, December 1992, **22(12)**: 1049—1067.

10 Rada R, Wang W, Mill H *et al*. Software reuse: from text to hypertext. Software Engineering Journal, September 1992. 311—321.

11 Novak G S, Hill F N, Wan M *et al*. Negotiated interfaces for software reuse. IEEE Trans. on SE, July 1992, **18(7)**: 646—653.

12 Harms D E, Weide B W. Copying and swapping: influences on the design of reusable software components. IEEE Trans. on SE, May 1991, **17(5)**: 424—434.

SOFTWARE REUSE

Yang Fuqing　Zhu Bing　Mei Hong

(*Department of Computer Science and Technology, Beijing University, Beijing* 100871)

Abstract　This paper discusses basic concepts of software reuse techniques, introduces theoretical and practical activities on software reuse, points out the crucial problems and some resolutions on software reuse techniques. Finally, the relation between reuse and CASE environment is discussed.

Key words　Software reuse, reusability, compose, generate, abstract, reusable component.

© 中国科学院软件研究所 http://www.jos.org.cn