软件开发 软件重用

计算机科学 1994 Vol. 21 №. 4

68-71

# 关于软件复用*)

TP311.52

郑明春 张家重 王岩冰
(山东师范大学计算机科学系，济南250014)

**摘要** In this paper, we discuss the basic concepts, reusable software components and essential factors of software reuse. We also analyse its main techniques by software schema reuse, and propose a method of acquiring reusable software schema automatiocally.

## 一、引言

1968年，D. McIlroy 在 NATO 软件工程会议上，首次提出了软件复用的问题，他倡导生产软件的标准构件用于软件系统的开发。二十多年过去了，虽然软件复用在其某些方面发挥了较大潜力，的确能够在一定程度上缩短软件开发周期，提高软件可靠性，减少投资风险，但并不象人们所希望的那样，能摆脱软件危机，相反，软件复用本身又受到了种种限制，其意义却不尽人意。有鉴于此，我们有必要对软件复用进行再认识，弄清何为软件复用，可复用成分有哪些，切实可行的复用方式为何，软件复用的难点在哪里等问题。基于上述目的，本文首先讨论了软件复用的有关概念和基本技术，阐述了软构件复用所要考虑的基本问题；其次以自学习软件自动化系统为例介绍了机器学习用于复用成分的自动获取；最后指出了软件复用的主要难点和进一步研究的内容。

## 二、有关概念

### 2.1 软件复用

所谓软件复用是指利用现有软件成分(资源)来构造新的软件系统。可利用的现有软件成分即复用成分是软件复用的核心。复用成分的获取、管理和施用构成了软件复用的三个要素。

复用成分的获取有两层含义：一是为了构造软件系统之便，将现行软件成分设计(抽象)成可以复用的；二是在复用成分库中选取用于某具体问题的可用成分。现有工作多集中在提供软件抽象的描述手段方面，基本上无机器支撑。

在无机器支撑的情况下，靠手工方式来管理、组织复用成分是进行软件复用的一大障碍。Prieto-Diaz R. 在文[2]中提供了一种复用成分库的组织方法，能够有效地组织和扩充复用成分。

复用成分的施用是获取和管理的目的。施用过程包括根据要求选择抽象的可复用成分，将其进行适应性修改，并把它集成到现行开发的软件系统等几个处理步骤。

### 2.2 复用成分

复用成分不应狭义地理解成软件的源程序代码，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、测试和维护信息等。

上述复用成分的定义限定了软件复用的范围。D. McIlroy 提出的软件复用，其复用成分属程序代码复用的范围，而 R. Balzer 于1983年提出"代码是不可复用的"。可见，目前对复用成分的认识尚不统一，对复用成分不同的理解形成了不同的复用途径与技

*)南京大学软件新技术国家实验室和山东自然科学基金资助。

ter February 1991

[2] W. B Frakes et al., Representing Reusable Software. Information and Software Technology DEC 1990

[3] James W. Hooper et al., Software Reuse-Guideline & Methods

[4] Prieto-Diaz, Status Report: Software Reusability, IEEE Software 3. 1993

[5] Prieto-Diaz, Domain Analysis For Reusability

[6] Mark B. Ratcliffe et al., System Development through the Reuse of Existing Components, Software Engineering Journal November 1991

术。对此，我们的看法是：由于代码往往针对特定问题，如数学库函数，较难用于另一场合，要实现代码的复用，必须研究刻画其功能的设施。从构造软件系统的过程来看，软件开发方法等的复用不是实现软件复用的本质。在软件复用中，我们认为构成系统的软部件的抽象结构的复用，才是切实可行的途径。因为软件的抽象结构适合用于一类问题。事实上，为实现源代码的复用，在继承其原来某些结构的基础上而作的适应性修改，本质上就是复用了抽象结构。

一个可复用的软件抽象结构需从规模、适用范围和易复用性三个方面来刻画，如图1所示。

按照我们的观点，可复用的软件抽象结构应为：

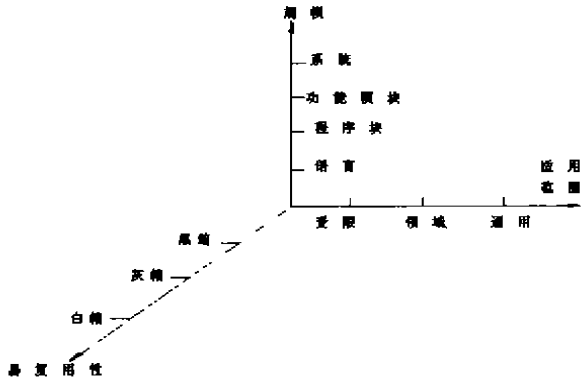


图1　软件抽象结构的刻画

黑箱的、面向领域的功能模块级的软构架。后文中，软部件或复用成分均指上述含义。

### 2.3　基于复用的软件开发过程

由于传统的软件开发方法没有考虑复用问题，因而借助这种方法开发的软件系统，其软部件难于复用，因此应研究基于复用的软件开发方法。

"基于复用"应含两方面含义：一是利用可复用的软部件开发新的目标软件系统；二是在目标系统分析、设计过程中，构造新的可复用部件，供现行或未来软件开发使用。结合传统方法，基于复用的软件开发过程包括两个阶段：自上而下地分析设计和自下而上地构造系统，其过程示意可如图2所示。

## 三、软件复用技术

目前，尽管人们对软件复用及复用成分的理解色彩纷呈，所采用的复用途径也各有千秋，但其技术无外乎以下四种：抽象技术、选择技术、例化技术和集成技术[4]，这四个方面在图2所示的软件开发过程中有所体现。下面，我们从上述四个方面讨论软构架的复用所要考虑的基本问题。

### 3.1　抽象技术

使软部件成为可复用的前提是对其进行抽象，抽象的结果为软构架。抽象有规范抽象和实现抽象两个级别。

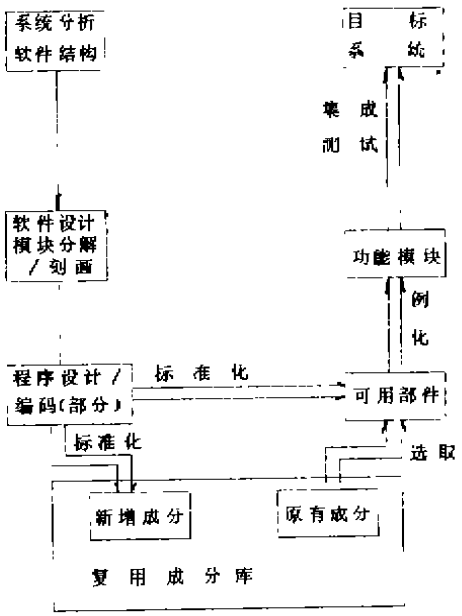


图2　基于复用的软件开发过程

相应地，软构架由两部分构成：一是抽象的形式规范；二是软部件实现体的描述。抽象规范应包括的具体内容有：

·软构架的形式语义描述——刻画它能干什么；

·软构架的例化约束条件——可改变的约束条件；

·软构架的适用条件——实现体的前后置断言。

软构架虽与软部件内容相似，但它侧重于算法和数据结构的描述，而非实现体的程序代码。对施用者而言，一个软构架可分为三部分，即隐藏部分、固定部分和可变部分。

可见，抽象技术主要研究如何由一个软部件得到一个可复用的软构架，关键是提供一种能够形式刻画软构架各部分的描述设施，以便于复用。

### 3.2　选择技术

选择是根据问题要求，在复用成分库中进行检索和匹配以得到所需软构架。选择技术涉及三方面问题：分类、检索和说明。

(1)分类问题　为了对软构架进行有效的组织，就需对其进行分类。Prieto-Diaz 提出了一种基于软构架的功用与环境的分类模式。其中，功用描述了软构

架的功能，包括：系统类型，功能范围和应用领域。

(2)检索问题　要求为一给定问题检索出与之匹配的软构架，必须给出刻画计算需求的问题规范。其形式与软件规范相似，主要包括问题要求的前置条件和后置条件。一般地，借助一阶谓词逻辑的形式，或某种功能规格说明语言来刻画。

相应地，检索机制还要求软构架的规范包括下述信息：替换断言、前置条件、不变部分、后置条件和例化验证信息等。

根据问题规范，候选软构架的匹配条件为：①问题规范的前置条件是软构架规范的前置条件的超集。②问题规范的后置条件是软构架规范的后置条件的子集。

(3)说明问题　问题规范和软构架规范的形式描述便于机器自动地进行检索匹配。然而，对于非系统支持的复用方式，还应具有一些易于理解的说明信息，这类信息一般有三方面①规范信息：软构架功能；②使用信息：如何正确使用软构架；③实现信息：软构架是如何实现的。

这些信息对正确地例化和集成都是需要的，可采用形式、非形式、图形化和文体等形式描述。

### 3.3　例化技术

例化是选择过程的继续，通过例化，把选定的满足问题规范的软构架之实现体转换成可执行的程序代码。例化有两个层次：

(1)基本软构架的例化：按照常规例化方法，将其变化体转化成固定成分；

(2)嵌套软构架的例化：对上层软构架转化成两部分：一是固定成分；一是新软构架，新的软构架需根据前后置条件重新例化，直至全部为基本固定成分为止。

例化的内容主要有两部分：

(1)数据结构：根据问题规范中的数据类型及变量(输入量)，将软构架中对应部分例化；

(2)操作变元：根据操作变元(函数变元)的前/后置条件，将其例化成已知基本函数或新的软构架。

值得说明的是，一个软构架有时对应着多个实现体，不同的实现体对应着不同的实现算法。如，对一索引顺序表元素的访问，既可以采用分块查找的方式，又可采用顺序查找的方式进行。那么，若一软构架存在多实现体，对其例化时究竟使用哪一个实现体，就要根据具体情况，权衡性能和通用性。

### 3.4　集成技术

例化软构架的结果往往是一些较小规模的功能程序模块，如何把各程序模块组合成一完整的软件系统正是集成所要做的工作。也就是实现由软部件构造软件系统的过程。

集成的一种可行途径是采用模块连接语言，该语言必须能够描述模块的句法和语义说明，并方便地刻画模块间的接口信息。

集成分纵向方式和横向方式两种。前者类似于前述的嵌套软构架例化方式；后者类似于软构架的一般模式。如，Goguen 的 LTL[3]即如此。

### 3.5　NDSAIL 系统[4]的软构架复用

作为例子，下面介绍自学习软件自动化系统 NDSAIL 的软构架的复用过程。

系统中问题规范的描述形如 P(f)=(FD,SP)，其中 f 为所给定问题对应的函数，FD 是函数说明，其形式为(y=f(x),D,R)，D、R 分别指出输入量 x 和输出量 y 的数据类型。SP 为函数功能描述，它由前/后置断言构成(PRE(x),POST(x,y))，其中 PRE,POST 采用功能规格说明语言 FGSPEC 描述。如，问题"求实型序列的最大项"的规范描述如下：

```
specify max
spec:seq(real)→real                    ;数据类型
        S→Z                            ;输入/输出
pre:nonempty(s)                        ;前置条件
post:Z∈S∧(∀x∈S,Z≤x)                    ;后置条件
```

软构架是系统中用于刻画问题分解方法的软件结构描述，它包括体描述，I/O 类型描述和语义约束描述三部分。

系统的求解机制就是将合适的软构架作用于具有上述问题规范的待解问题，生成相应的设计规格说明，即抽象算法，再借助 NADUTO 系统自动转换可执行的程序代码(Pascal 程序)。其基本过程体现了软构架复用的几个方面，可概括成如下几步：

(1)选择　分析待解决问题的特征，在软构架库中选择合适的软构架。

(2)匹配　根据所给问题，与所选软构架中的参量进行匹配，以便产生统一例化形式。

(3)验证　检查软构架是否满足作用条件，包括语法条件、语义约束条件，以保证其正确性。

(4)作用　对软构架中的 I/O 量及中间量按照问题的功能规格说明信息进行例化；把待解问题归约为子问题，调用推理机制，形成子问题的功能规范(待解)。至此，便完成了一步求解。对于待解子问题的求解可重复上述过程，或将子问题例化成基本算法，或选用新的软构架，直到所有子问题皆为基元问题为止。

软构架的复用表现在两个方面：一是使用嵌套软

构架，将待解问题分解为子问题及其分解；二是使用基本软构架作为子问题的直接解。正是这两方面的复用共同决定了系统的解题能力。

## 四、软构架的自动获取

如前所述，现有软件复用的研究大都集中在复用成分的管理和施用两方面，对复用成分获取的研究仅限于提供软件抽象的描述手段方面，尚未见到有关自动获取复用成分的研究报道。

NDSAIL 系统在将机器学习用于软构架的自动获取方面进行了成功尝试。其自动获取机制采用了解释学习(EBL)的途径，能够从用户给出的具体算法设计实例中抽象出适于求解一类问题的软构架。该软构架即可被系统复用作算法设计模板。

依照解释学习范型，软构架获取过程分为两个阶段：

(1)解释　验证训练实例满足规格说明，找出相应的函数构成方式，作数据类型与操作的相关性分析。

(2)推广　选择适当的操作符和数据类型在满足操作性准则的条件下进行推广，并导出语义约束条件，生成相应的软构架。

经过学习所得到的软构架能否有效地复用于未来算法设计过程中，取决于软构架的可操作性。关于操作性的处理方法是，对每一个被推广的操作变元，引进语义约束条件，就是一组限定该变元在问题求解时例化范围的谓词公式。采用约束条件解决软构架的可操作性问题，既有确定性又具灵活性。系统对每个操作变元都能推导出一组逻辑公式，明确地限定了例化范围。

系统运行中通过对算法设计实例的分析，获取了若干软构架，在以后的软件开发中发挥作用。其详细模型与过程可参阅文[7,8]。

## 五、结束语

从软件复用的若干成分来看，由于纯程序代码的复用仅限于标准库函数，难以适于变化的情况，而抽象的软件结构的复用才是行之有效的途径。

软构架的复用强调抽象的算法和数据结构，而非源代码，这样便于从非形式的功能规格说明，到形式的功能规格说明，再到可执行程序代码的转换，而且该过程易于借助机器支撑，自动或半自动地实现转换。

遗憾的是，定义一种抽象设施，要求其既自然易懂，又精确易表达是极为困难的。如果抽象级别太高则难以表达与转换，如果抽象级别过低，类似于一般的程序设计语言，即失去复用的意义。

总之，使软件复用走向实用化的主要难点体现在两个方面：一是复用成分的刻画与获取；二是如何实现施用。虽然面向对象技术便于实现软件复用，但其应用目前仍限于代码复用范围。因此，我们认为下述问题需要进一步研究：

(1)探索支持软件复用的软件开发方法；

(2)软件抽象结构的形式描述设施；

(3)支撑软件复用的自动化工具与环境；

(4)复用成分的自动获取机制。

### 参考文献


[1] D. Mcllroy. Mass Produced Software Components. Software Engineering. 1968

[2] Prieto-Diaz. R. et al. . Classifying Software for Reusability. IEEE Software. 1987

[3] Biggerstaff. T. J. . Proc. ITT Workshop on Reusability in Programming. 1983

[4] Charles W. K. . Software Reuse. Computing Surveys. 1992

[5] Goguen. J. A. . Reusing and Interconnecting Software Components. IEEE Computers. 1986

[6] 徐家福等；自学习软件自动化系统 NDSAIL 的设计与实现，计算机学报，92年第11期

[7] 费宗铭，张家重，徐家福；基于解释的算法构架的学习，软件学报，94年第3期

[8] 张家重，费宗铭；基于算法构架的软件自动产生，计算机研究与发展，92年第6期