从软件工程的发展看软件自动化*

白光野 徐 崇 范植华 蒋东溟

(中国科学院软件研究所,北京 100080)

摘要 本文对软件工程方法学发展各个阶段主要的方法论做了简单的评述,并展望了软件工程今后发展的方向,同时举例说明了以此为指导思想进行软件自动构造的方法.

关键词 软件工程,面向对象,软件自动化.

随着计算机应用的逐步扩大,计算机软件的规模不断增大,进而也就引起了软件复杂度的增加.60年代末出现的软件危机呼唤人们寻求一种解决软件可靠性、质量、维护等诸多方面问题的有效途径."软件工程"的提出和应用就是在这一背景下产生的,其目的是要提高软件的生产率与可靠性,实现软件产品的优质高产.在软件工程发展的20多年时间里,为了实现这一目标,人们在软件开发技术,其中包括软件开发方法学、软件工具与软件工程环境,以及软件工程管理方法学上都取得了很大进展.本文将对软件工程方法学做一简要评述,并以管理信息系统的自动构造为例对软件自动化加以阐述.

1 软件生命周期模型

同其它工业产品一样,软件也具有生命周期.总体来讲,软件生命周期包括分析—设计—实现—维护这一系列过程.对生命周期的不同划分方法也就形成了不同的软件生存周期模型.这些模型大致可分为两类:传统的瀑布模型(waterfall model)和原型模型(prototype model).

1.1 瀑布模型

典型的瀑布模型可以用B. W. Boehm 的刻划来表示,他将软件生命周期划分为7个阶段,每个阶段的任务分别为:系统需求,软件需求,概要设计,详细设计,编码纠错,测试和预运行,运行维护.每一阶段工作的完成需要确认.

瀑布模型的主要特点:阶段间的顺序性和依赖性.开发过程是一严格的下导式过程,前一阶段工作的输出是后一阶段工作的输入;而确认过程则是严格的追溯式过程,后一阶段出现了问题要先从前一阶段的重新确认入手解决.因此,只有在给出正确的需求分析以后,才

---

* 本文1993-11-11收到,1994-04-19定稿

作者白光野,1946年生,研究员,主要研究领域为人工智能,专家系统.徐崇,1969年生,硕士,主要研究领域为面向对象技术,软件工程,人工智能.范植华,1942年生,研究员,主要研究领域为并行处理,实时处理,软件工程集成化平台.蒋东溟,女,1969年生,硕士,主要研究领域为软件工程,人工智能.

本文通讯联系人:白光野,北京100080,中国科学院软件研究所

© 中国科学院软件研究所 http://www.jos.org.cn

能得出预期的正确结果.然而,这里存在着一个弱点.在初始阶段给出正确的需求分析几乎是不可能的.一方面,用户对计算机缺乏必要的认识与了解,因此在定义需求过程中往往不够完全和不够准确;另一方面,软件人员对用户的专业知识也了解不深,因此在软件系统的定义上也不够完全、不够准确.同时,实际应用中许多方面的规定与约束并不是一成不变的,随之而来的是需求分析的不断变化,这种不确定性对瀑布模型是致命的.因为需求分析处于这种模型的最顶层,在开发过程中属最早和最原始的输入,完全由人来控制.需求分析中潜伏下来的问题只有在软件产品形成以后才暴露出来.而在解决问题的过程中却要求各个阶段逐一追溯分别寻求前一阶段的结果重新确认,从而造成瀑布流的不断反复,造成大量人力、物力、时间上的消耗.软件系统实现后的维护阶段,对软件系统的修改与完善一般在代码级上进行,而非从需求出发的再实现过程,从而使软件系统失去了其对于规格说明的完整性、正确性与一致性,即代码级维护可能给应用软件系统带来意想不到的严重后果.

1.2　原型模型

基于对传统瀑布模型的线性结构的缺点进行改造的考虑,人们又提出了原型模型.它的主旨在于首先建立一个能够反映用户主要需求的原型,为用户展示未来系统的概貌,使用户可以从最终产品的角度出发对原型提出修改意见,软件人员反复改进,最终建立完全符合用户要求的新系统.这实际是一个软件人员不断向用户提供样品,而用户对其做出迅速反馈以进一步改善样品的过程,大大简化了瀑布模型冗长、反复的过程.

原型也就是可执行的模型或称为目标系统的初始版本.原型法在各阶段用户反馈活动的基础上突出了快速的改造过程,它改变了传统瀑布模型的线性结构,采用逐步求精方法使原型逐步完善,以满足用户要求,是一种不断在新的高层次上反复推进的过程.

1.3　基于传统生命周期的方法学

在软件生命周期模型的基础上,出现了一些软件分析和设计的方法学.其中典型的有:

1.3.1　结构化方法学

早期的结构化方法学主要是面向处理(process－oriented),而对实体与数据的模块化不太重视.其间的代表人物是 DeMerco[1],他将结构化的方法引入到分析之中,通过描述一系列的步骤来实施结构化分析,把现存系统的模块化转变为对要开发系统的模块化.类似的还有 Gane－Sarson 结构化分析方法学.

继承结构化的传统,Ward 和 Mellor 建议对结构化分析做更多的扩展来更好地支持实时系统(real－time systems)的模块化.这种方法学在结构化分析工具集中加入了实体关系图(entity－relationship diagram)和状态转换图(state transition diagram).

在认识到系统、语言以及工具在过去 20 多年中的发展以后,Yourdon[2]发展了现代结构化分析(modern structure analysis)方法,对结构化分析进行改进.这种方法不再要求对当前实现的系统进行结构化,而加入预先分析来发展系统的基础模块(essential model);它采用自顶向下功能分解使用的事件分解(event partitioning)技术来代替数据流图(dataflow diagram)的构造;它更强调通过 E－R 图进行信息的模块化,通过状态转换图进行处理的模块化,并且采用原型.

1.3.2　信息工程(Information engineering)

在 70 年代末 80 年代初,数据的计划与模块化开始在系统开发中占重要的地位,尤以由

© 中国科学院软件研究所 http://www.jos.org.cn

信息工程为代表的面向数据(data－oriented)的方法学的发展而盛极一时. 其间的代表人物是 Martin.

Martin[3]的信息工程是一种意义更广泛的方法学,他把面向数据的方法扩展至整个开发生命周期. 与结构化方法在生命周期中的编码逆向(backward)发展不同,信息工程在生命周期中从计划和分析开始前向(forward)发展. 与结构化方法相比,信息工程提供了更多的分析技术与结构化工具,包括企业模块化,关键成功因素分析,数据模块化,处理模块化,联合需求计划,联合应用设计,时间盒方法学和原型.

在面向数据(data－oriented)的软件开发方法学中,JSD 方法学首先定义了数据结构的输入输出,而后通过把所有程序数据结构结合起来定义程序结构. Warnier－Orr 设计方法学与 Jackson 方法不同,它是面向输出的方法学. 程序开发者首先定义系统输出,而后逆向从基础系统模型定义设计阶段的处理和输入部分.

2 人工智能(AI)的引入

从软件工程方法学的角度看,其处理对象是千姿百态的客观世界,因而其行为必然是大量知识密集型的. 软件系统时空的变更性与管理的不确定性使得引入 AI 做为支持软件工程活动的手段成为必然的选择. 我们可以把 AI 范畴的问题同传统软件的范畴做一个对比:

AI	传统软件
1. 不完全的行为模式的定义	完全的抽象定义
2. 充分/不充分的解决方法	可测试正确/不正确的解决方法
3. 粗略的静态近似	非常好的静态近似
4. 反模块近似	非常好的模块近似
5. 模糊可限制的	精确可限制的

由此可见 AI 与软件工程方法相结合的意义所在. 在过去的若干年中,AI 研究的焦点一直是知识的表示与使用技术,而这正是知识密集型的软件工程活动需要依靠的技术. AI 能够对知识进行智能化的表示与管理. 把软件系统中适应于变化的部分提取出来封装在各自的规则集中,并将整个规则库与处理机(推理机)完全分离开来. 这样,系统可以从最基本的非启发式规则入手以简化将要引入启发式规则和启发式推理机的系统的建立.

事实上,在软件生命周期各个阶段的工程技术都需要 AI 技术. 应用领域的知识存储和反馈需要知识表示技术;系统需求分析与规格说明需要知识获取技术;系统的维护与发展则需要知识解释技术. 启发式搜索范型、形式化推理和基于规则的系统等技术都在系统设计与实现的模块化及目标系统特征分析和推导等过程中不可缺少. 因此,人工智能的理论与技术可以帮助软件工程在软件自适应、自动化和软件智能化管理等方面取得突破.

3 计算机软件工具与环境

早期的软件工具是人们为了支持软件的编制和调试而构建的一些孤立的实用程序. 随着围绕软件生命周期全过程的研究和各种分析与设计方法的提出,软件工具开始走向集成化. 这种支持一定的软件开发方法或按照一定的软件开发模型组织而成的软件工具集合就

© 中国科学院软件研究所 http://www.jos.org.cn

称为软件开发环境.软件工具与环境体现的是早已为人们熟知的计算机辅助软件工程(CASE)的概念.

软件工程环境的基本要素一般包括:软件信息数据库,交互式的人机界面,语言工具,质量保证工具,需求分析及设计工具和配置管理工具.针对各个要素和软件生命周期各个阶段的软件工具与环境的例子很多,在此不再赘述.

CASE 技术的发展起于 80 年代初,从计算机辅助文档和计算机辅助图表分析与设计工具开始.80 年代中期,自动设计、分析和检测以及处理信息存储自动化的 CASE 出现了.到 80 年代末,CASE 的重点集中在两个方面,其一是从规格设计得到自动代码生成,其二是将设计自动化与程序自动化链接起来.而到 90 年代初,CASE 的研究越发关注智能化方法论和作为开发方法学的可重用性.

最近几年,人们开始普遍讨论和开发集成化软件工程环境 ISEE(Integrated Software Engineering Environment)和 ICASE(Integrated CASE).Wasserman 等人[4]提出了 5 种类型的集成:平台(platform),主要提供构架(framework)服务;描述(presentation),侧重于用户的交互活动;数据(data),侧重于工具对数据的使用;控制(control),侧重于工具间的通讯和交叉操作;处理(process),侧重于工具在软件处理过程中所担当的角色.当集成环境中的所有部件都作为一个统一、协调、一致的整体中的部分运作时,集成环境提供的支持比非集成环境更为有效.要做到这一点,同样需要软件工具的支持.

4　软件复用与再造工程

4.1　再造工程(re－engineering)

任何软件系统均有一定的生命期,任何对系统结构的修改都使以后的改变更为昂贵.时间的推移会造成用于改造系统的费用过高,而不予改造的系统又不能满足要求.当一个系统很难改变而又具有很高的商业价值时,就需要对其进行再造.再造工程是对一个现有系统进行抽象描述,对于在更高抽象级上的改造进行合理推断,进而重新实现该系统的过程.

再造工程＝逆向工程＋系统改造＋正向工程[5]

逆向工程是确定更抽象、更易理解的系统表示的活动,系统改造包括系统功能与实现技术的变化.正向工程构造可执行系统的活动.

这种方法的目的在于通过分析现有的代码,构造一个可用软件部件库,该部件库包括可作为 CASE 软资源库(repository)与代码生成器输入的概念数据模型、过程性伪码、屏幕操作、记录,而后通过复用对系统进行重新构造.再造工程有许多策略,其中包括代码转换技术或根据现有程序的信息(设计或代码)的重新开发.逆向工程可将"纯"的代码转换为高层的规格说明,而经过修改后,再通过正向工程重新构造成新的应用系统[6].但如果对该应用领域的知识无较深入的了解,就无法对程序进行较好地分析,从而为逆向工程带来一定困难,另一方面软件部件的选取(标准)以及软件部件库管理与正向工程中软件部件的应用的智能化仍十分棘手.因而,再造工程方法对于软件的自动生成而言仍不够成熟.

4.2　软件复用(reuse)

软件复用包括设计的复用(播种复用"sowing" reuse)与现有模型,领域知识和部件(收获复用"harvesting"reuse)的复用.

© 中国科学院软件研究所 http://www.jos.org.cn

在复用方面，虽然有许多人在做工作，但即使使用面向对象的环境与方法，高层复用自动实现仍未解决. 实际上，所有的面向对象方法均强调重用应从开始就得以体现，而对于收获复用的研究至今未受到足够的重视. 在播种复用方面 Coad 与 Yourdon 提供了将现有软件设计和程序部件并入新系统的方法[7]. Booch 强调在设计过程中就从现有的类库中寻求可复用的软件部件，但二者均未能提供如何提取与评估现有部件的方法. 在收获复用方面 Champeaux 和 Faure[8]提出一种基于软资源库(repository－based)的方法，他们认为，软件开发过程应是创建，修改含有分析、设计、实现部件的交叉引用的 3 个软资源库的过程. 他们建议使用一种智能库转换机制，用以辅助可重用部件的确定. Caldiera 和 Basili[9]在确认与评估软件部件方面做了许多工作，他们建议软件复用专家(reuse specialist)从应用系统开发者中独立出来，专门从事于部件工厂(component factory)及可重用部件的软资源库的建造与维护. 我们所追求的目标与方向正是软件构造及管理的智能化，而在这方面上述诸多方法均显得无力.

软件开发过程中方法与工具对软件产品的质量有重要的影响作用. 软件工程发展至今，随着系统复杂性的增加，传统结构化方法已无法保证软件开发的效率与质量. 传统的结构化方法存在两方面的问题：(1)由于应用领域与计算机领域(问题描述空间与求解空间)对所面对问题的描述不一致，应用领域与计算机领域的鸿沟始终存在，造成软件人员必须参与需求分析，系统的设计与实现只有通过软件开发人员而得以统一，用户无法真正参与. (2)软件系统的动态性导致传统的瀑布流模型的不断循环. 这两方面的问题正是造成软件危机无法克服的根本原因.

5 面向对象的方法

面向对象技术的出现与发展，为我们提供了一种更“新”的思维方式. 面向对象的开发方法在软件系统开发过程中体现出一致性的知识表示的方法.

面向对象的软件开发的目标是通过面向对象的分解，将用户的要求变为可执行语言的结构. 面向对象的开发过程可分为分析、设计、实现 3 个阶段. 在面向对象分析阶段，软件工程师以一种可理解的形式对用户的要求进行精确的确认与理解. 这一阶段直接涉及用户的问题，或称问题域. 在设计阶段，软件工程师通过描述如何实现用户需求修正并扩展分析模型. 实现阶段则将设计模型转化为具体的语言实现.

面向对象开发的一个重要特征是分析、设计、实现阶段采用相似的模型，这使各阶段转换十分便利. 面向对象的软件开发过程中内在的思维方式比传统结构化方法更接近于自然，即在构造现实的抽象模型中使用对象进行思维较之使用功能思维更为自然；另外，从问题空间的模型到求解空间的映射过程采用面向对象的开发方法更为直接，因为在面向对象的开发中二者在知识的表示(问题的描述)方式上取得了一致，即这种映射是同态变换，而结构化方法却不然.

面向对象的分析与设计方法：

(1) Coad 与 Yourdon 的面向对象分析(OOA)方法[10]

面向对象分析方法导致问题域的基层模型、主体、类层、对象、结构、属性、服务，每层构筑在前一层上. 主体是有简单语义的子系统. 结构是继承与部分关系. 属性标识对象特征. 服

© 中国科学院软件研究所 http://www.jos.org.cn

务表示一对象提供的操作集.

(2) Bailin 的需求规格说明(OOS)方法[11]

OOS 方法包括 7 步:①确定关键问题域的实体,②区分主动与被动实体,③在主动实体间建立数据流,④将实体(或功能)分解为子实体与(或)功能,⑤检查新实体,⑥将新实体功能归类,⑦将实体划入适当域. 这种方法的目的在于消除传统结构化分析与面向对象设计的不一致性,其结果为实体—关系图及实体—数据流层次图.

(3) Shlaer 与 Mellor 的面向对象分析[12]

该方法步骤:①建立信息模型,②确定对象生命周期,③建立动态关系,④建立动态系统,⑤建立过程模型,⑥确定域与子系统. 本方法通过子类建立对象的关系网.

(4) Wasserman 等的面向对象结构化设计[13]

Wasserman 等人提供的面向对象结构化设计旨在提供一种标准的设计方法,包括面向对象方法与结构化方法用以支持所有的软件设计,该方法是用于体系结构描述的详细表示法,即通过确定个体模块而定义高层设计.

(5) Booch 的面向对象设计[14]

Booch 认为对使用针对问题的适当技术、分析、设计工作应是反复与增式过程,主要可分为 4 个步骤:①确定类与对象,②确定类与对象的语义,③确定类与对象的关系,④实现类与对象.

(6) Wirfs—Brocd 等的责任驱动设计[15](responsibility—driven design)

责任驱动设计是基于计算的客户服务器(client—server)模型. 系统被认为是具有各自责任的服务器. 服务器以定义了的客户服务器交互的本质与有效范围的合同为依据向客户提供服务. 客户与服务器被认为是对象,而责任与服务是操作.

(7) Rumbaugh 等的面向对象建模与设计(Object—Oriented Modeling and Design, OMT)[16]

OMT 方法引入了 3 种模型来描述系统:对象模型、动态模型与功能模型. 对象模型描述了系统中对象的静态结构及对象间的关系. 动态模型描述与时间和变化有关的系统的性质及对象的交互行为. 功能模型描述了系统中数据的变化过程. OMT 方法包括以下几个阶段:分析、系统设计、对象设计.

6　软件系统的自动化

基于上述分析,软件工程环境(软件工具)应以面向对象技术为指导,然而这其中存在着所面向的应用对象的问题. 软件工程环境的目标在于为有效的软件过程提供有效的支持,而这种支持的对象是软件工程师还是(最终)用户是决定方法学出发点的决定因素. 当今多数 CASE 环境均以软件工程师为其应用对象,并以传统的软件开发方法为指导思想,因而最终无法从根本上解决软件危机. 我们认为今后软件开发的方向在于软件的自动化,即软件工程师向用户提供软件自动构造的工具,该工具根据用户对软件系统的定义自动生成软件系统,作为系统最终使用者的用户也同样应该是系统的设计者,这样才能始终保持系统设计与需求的一致性. 软件开发人员也应该彻底从具体的应用问题领域的细节中脱离出来,而为用户提供可行的方法学和可靠的设计工具,把系统的设计与实现分离开来,提供从设计到系统实

© 中国科学院软件研究所 http://www.jos.org.cn

现的统一机制以满足用户动态设计的要求. 这将导致一种新的软件人员产生,即领域分析员(domain analyst). 其责任是与领域专家(domain expert)合作,为应用领域中的系统建模,作为软件工具所构造的软件系统的概念模型. 软件自动化的作法之一是希望构造一种普遍适用的软件自动生成工具,即提供一种抽象的描述语言,用以对数据及控制进行描述,而软件工程环境本身的自动生成器检验描述语言的合法性,并根据合法的描述语言转换为软件系统. 这种做法有较广的运用范围,但现实生活中的任何工具一般只适用于一定的领域,许多工具均为专用工具,因而我们采用另一种做法,即软件工具的构造应针对某一领域. 下面以管理信息系统为例具体讲述.

6.1 角 色

由于 MIS 具有层次结构、可分解性、功能同一性、动态性的特点,面向对象的方法学成为我们的基本出发点. 我们把构成 MIS 的基本单元称为角色. 角色是在 MIS 逻辑结构中高度统一的实体,是可抽象的对象——系统、子系统、单个实体(人). 每个角色均可被理解为接收信息、处理信息、发送信息的对象. 同时,角色为 MIS 提供了分解机制,对 MIS 的逻辑结构进行相应的划分如图 1:

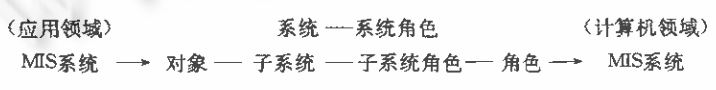

图1 角色机制

角色机制正是应用面向对象的基本观点,在问题描述与求解一致的基础上,使用户真正成为系统的设计者,而无需软件人员的干预. 同时,角色机制提供的系统自动生成器又把系统的设计与实现进行合理的区分,应用系统的设计就独立于软件开发人员而进行,而系统的实现仅依赖于用户的设计和软件开发人员提供的生成工具,成为由设计而导致的唯一结果,完全剔除了人的不稳定因素. 同时,系统的动态实现就直接表现为用户的动态设计.

6.2 MIS 自动构造工具的实现

6.2.1 MIS 定义文法描述

⟨1⟩⟨MIS 系统⟩::= $\{$⟨角色⟩$\}_0^n$

⟨2⟩⟨角色⟩::=[⟨角色说明部分⟩]
⟨角色定义部分⟩
[⟨信息获取部分⟩]
[⟨信息发送部分⟩]
[⟨规则部分⟩]

注:规则是对角色日常动作的描述.

6.2.2 MIS 自动构造工具的概念模型(见图 2)

6.2.2.1 角色编辑器

角色编辑器也就是用户的系统设计环境. 用户通过定义角色主体、数据、操作和消息从而定义了 MIS 中的职能个体. 这些个体角色通过相互间的消息传递与驱动所形成的子系统角色、系统角色以及它们之间的层次与结构关系,映射到 MIS 系统中. 用户的设计过程是一

© 中国科学院软件研究所 http://www.jos.org.cn

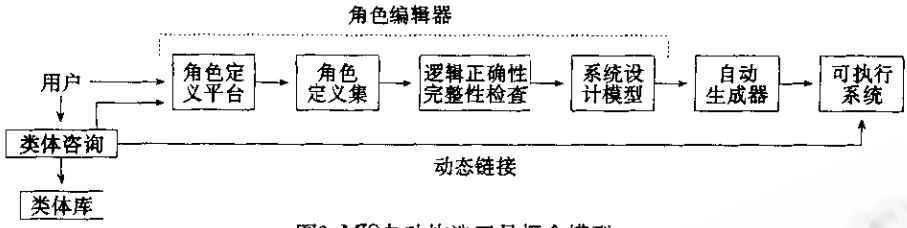

图2 MIS自动构造工具概念模型

个动态的过程,每个职能个体的定义相对独立.这就使得面向不同职能个体的用户可以彼此独立地设计自己的职能个体,而由角色系统环境通过各职能个体用户的消息——各角色间的消息关系描述自动形成系统的整体设计结构模型——子系统角色和系统角色.

6.2.2.2　类体库

我们把可以作为模板的个体角色、子系统角色、系统角色统称为类体,把类体的集合称为类体库.用户可从任意其需要的类体模板出发,改造或构造成新的个体角色、子系统角色或系统角色,以动态构造出新的个体、子系统或系统.任何已构造的职能个体、子系统和系统中的可重用的部分都可作为可修改的角色嵌入到其它系统、子系统中去.用户通过不断地修改、添加和拼装角色来构造新的系统.与此同时,用户也可成为这种动态机制的积极开发者.在不同应用系统的建立过程中,也在不断地丰富类体模板库.从一个更为广义的意义上看,这种开放式的类体模板库也就具有了自增长能力.从可执行系统的运行角度看类体库的作用,它起到了动态链接库的功能,使应用系统在职能个体、子系统甚至系统级等不同级别上实现了共享,最大限度地实现了系统局部和全部的复用.

6.2.2.3　推理机

规则为系统的动态性提供了一种描述手段,其定义形式如下:

〈规则〉::=外部条件:〈外部激发条件〉
　　内部条件:〈内部制约条件〉
　　动　　作:〈动作序列〉

其中外部条件是激发该规则的时序、外部状态、消息等条件;内部条件是与动作对象(数据)有关的制约条件;动作序列是一系列动作的集合,动作是个体角色由类体库中的抽象类继承而来.推理机是可执行系统的组成部分,是系统本身提供的.在MIS系统运行过程中,环境因素、时序因素、状态因素不断变化,从而可能使个体角色的某些规则条件成立,激发动作序列.而动作的结果可能造成系统状态的改变以及消息的发送,进而使其他(个体角色的)规则的条件得到满足.激发别的(个体角色)动作序列.这一过程正是推理机执行过程.

6.2.2.4　自动生成器

对MIS中个体的定义结束后,角色系统环境通过个体间消息传递的关系对整个MIS的逻辑完整性进行检查,就可以确定MIS的系统逻辑结构.作为角色编辑器输出的MIS系统定义就成为MIS自动生成器的输入,实现了角色实体、属性、操作与消息定义到可执行MIS系统中个体、子系统、系统动作的映射.由此可见,自动生成器仅仅提供一种转换机制,其方法具有很强的独立性和多样性.

© 中国科学院软件研究所 http://www.jos.org.cn

7 结束语

本文从软件工程发展的角度对软件工程方法学进行了简单的评述，并展望了软件工程今后发展的方向. 同时举例说明以此为指导思想进行软件自动构造的方法，希望能对各位同仁有所启发，并起到抛砖引玉的作用.

参考文献

1 Demarco T. Structured analysis and system specification. New York: Yourdon Inc., 1978.

2 Yourdon E. Modern structured analysis. New Jersey: Yourdon Press, 1989.

3 Martin J. Information engineering. Books I, II, and III, New Jersey: Prentice Hall, 1990.

4 Wasserman A I. Tool integration in software engineering environments. In: Long E ed. Proc. Int'l Workshop on Environments, Berlin: Springer—Verlag, 1990. 137—149.

5 Jacobson I, Lindstrom F. Re—engineering of old systems to an object—oriented architecture. In: OOPSLA'91 Conference Proceedings, Sigplan Notices, November 1991, **26(11)**: 340—350.

6 Jarzabek S. Domain model—driven software reengineering and maintenance. The Journal of Systems & Software, 1993, **20(1)**: 37—51.

7 Coad P, Yourdon E. Object—oriented design. New Jersey: Prentice Hall, 1991.

8 Chanpeaux D D, Faure P. A comparative study of object—oriented analysis methods. J. Object—Oriented Programming, 1992, **5(1)**: 21—33.

9 Caldiera G, Basili V. Identifying and qualifying reusable software components. Computer, February, 1991, **24(2)**: 61—70.

10 Coad P, Yourdon E. Object—oriented analysis. New Jersey: Yourdon Press, 1990.

11 Bailin S C. An object—oriented requirements specification method. Comm. ACM, May 1989, **32(5)**: 608—623.

12 Shlaer S, Mellor S J. Object—oriented systems analysis. New Jersey: Yourdon Press, 1988.

13 Wasserman A I, Pircher P A, Muller R J. An objected—oriented structured design method for code generation. Software Engineering Notes, Jan 1989, **14(1)**: 32—55.

14 Booch G. What is and what isn't objected—oriented design? Programmer, 1989, **2(28)**: 14—21.

15 Wirfs—Brocd R, Wilkerson B, Wiener L. Designing objected—oriented software. New Jersey: Prentice Hall, 1990.

16 Rumbaugh J *et al*. Object—oriented modeling and design. New Jersey: Prentice Hall, 1991.

SOFTWARE AUTOMATION FROM THE DEVELOPMENT OF SOFTWARE ENGINEERING

Bai Guangye Xu Chong Fan Zhihua Jiang Dongming

(*Institute of Software, The Chinese Academy of Sciences, Beijing* 100080)

Abstract In this paper, a survey on the software engineering methodology is provided and the trend of the software development is prospected.

Key words Software engineering, object—oriented, software automation.

© 中国科学院软件研究所 http://www.jos.org.cn