# 第5章 学科前沿★★

本章将介绍演化计算、软件自动编程、软件产品线、网构软件和软件工具酶。通过本章的学习，学生需要了解软件开发环境和工具的前沿理论和技术。此章理论性较强，有一定难度，为选讲内容。985或211院校计算机或软件专业的本科学生可以适度选讲，也可以作为计算机及软件或电子技术专业硕士的选讲内容。

## 5.1 演化计算与自动编程

### 5.1.1 演化计算概述

**1. 演化计算**

1）定义

演化计算是模拟自然界中的生物演化过程产生的一种群体导向的随机搜索技术和方法。生物在延续生存的过程中，逐渐适应其生存环境，使得其品质不断得到改良，这种现象称为演化。达尔文的自然选择学说认为，由于不同个体间的交配以及其他一些原因，生物的基因可能发生变异形成新的基因，这部分变异将遗传到下一代。将生物界所提供的解决问题的方法应用到求解实际问题已被证明是个成功的方法，并由此产生了仿生学。它遵循自然界中生物进化的优胜劣汰原则。演化计算用简单的编码表示各种复杂的结构，通过对编码进行简单的遗传操作和优胜劣汰的竞争机制来对问题的解空间进行搜索。演化算法不用明确地了解问题的全部特征，就可以通过体现生物进化机制的演化过程来完成问题的求解。

2）发展现状

演化算法在20世纪60年代被提出时并未受到普遍重视，主要原因有三点：一是这些方法当时并不成熟；二是这些方法运行需要较大的计算量，当时计算机速度跟不上要求；三是当时解决类似难题人工智能方法可以得出很好的结果，人们难以过多地关注其他算法。

传统人工智能解决问题的局限性在20世纪80年代初被凸显出来，当时计算机速度已经明显提高并且普及，制约演化计算的一大瓶颈已经不复存在。演化计算在机器学习、工程优化、过程控制等领域取得了极大成功，这引起了包括数学、物理学在内的各个学科及工程应用领域专家的兴趣。

自20世纪80年代中期以来，世界上许多国家都掀起了演化计算的研究热潮。目前，以演化计算为主题的国际会议在世界各地定期召开，如IEEE。随着演化计算的广泛应用，一些杂志都设置了专栏介绍这方面的文章，现在还出版了两种关于演化计算的影响力较大的新杂志 *Evolutionary Computation* 和 *IEEE Transactions on Evolutionary Computation*。

演化计算是一种通用的问题求解方法，具有自组织、自适应、自学习性和本质并行性等

特点，不受搜索空间限制性条件的约束，也不需要其他辅助信息。因此，演化算法简单、通用、易操作、能获得较高的效率，越来越受到人们的青睐。演化计算在大型优化问题求解、机器学习、自适应控制、人工生命、神经网络、经济预测等领域取得的成功，引起了包括数学、物理学、化学、生物学、计算机科学、社会科学、经济学及工程应用等领域科学家的极大兴趣。

现在，演化计算的研究内容十分广泛，例如，演化计算的设计与分析、演化计算的理论基础以及在各个领域中的应用等。随着演化计算的理论研究的不断深入和应用领域的不断扩展，演化计算将会取得很大的成功，必将在当今社会占据更重要的位置。

3）学科分支

自计算机出现以来，生物模拟便成为计算机科学领域的一个组成部分。其目的之一是试图建立一种人工模拟环境，在这个环境中使用计算机进行仿真，以便能够更好地了解人类自己以及人类的生存空间；另一个目的则是从研究生物系统出发，探索产生基本认知行为的微观机理，然后设计具有生物智能的机器或模拟系统，来解决复杂问题。例如，神经网络、细胞自动机和演化计算都是从不同角度对生物系统进行模拟而发展起来的研究方向。演化计算最初具有三大分支：遗传算法（Genetic Algorithm，GA）、演化规划（Evolutionary Programming，EP）和演化策略（Evolution Strategy，ES）。20 世纪 90 年代初，在遗传算法的基础上又形成了一个新的分支：遗传程序设计（Genetic Programming，GP）。虽然这几个分支在算法实现上有一些细微差别，但是它们都有一个共同的特点，即都是借助生物演化的思想及原理来解决实际问题的。

**2. 遗传程序设计**

遗传程序设计是学习和借鉴大自然的演化规律，特别是生物的演化规律来解决各种计算问题的自动程序设计的方法学。

1992 年，美国 Stanford 大学的 J. Koza 出版了专著《遗传程序设计》（*Genetic Programming*：*On the Programming of Computers by Means of Natural Selection*），介绍用自然选择的方法进行计算机程序设计。1994 年，他又出版了《遗传程序设计（二）：可重用程序的自动发现》（*Genetic Programming* Ⅱ：*Automatic Discovery of Reusable Programs*），开创了用遗传算法实现程序设计自动化的新局面，为程序设计自动化带来了一线曙光。遗传程序设计已引起计算机科学与技术界的关注，并有许多应用。

1985 年，Cramer 首次提出 GP，1992 年，由 Koza 教授将其完善发展。GP 是一种全局性概率搜索算法，它的目标是根据问题的概括性描述自动产生解决该问题的计算机程序。GP 吸取了 GA 的思想和达尔文自然选择法则，将 GA 的线性定长染色体结构改变为递归的非定长结构。这使得 GP 比 GA 更加强大，应用领域更广。

Koza 选择 Lisp 作为 GP 的程序设计语言。有了程序结构的概念，在前面介绍的演化算法的基础上，即可讨论自动程序设计（Automatic Programming，AP）。可以用下面的公式来概括自动程序设计的思想：EA＋PS＝AP（演化算法＋程序结构＝自动程序设计）。

Holland 遗传算法中的遗传群体是由一些二进制字符串组成的；而 GP 或 AP 的遗传群体是由一些计算机程序组成的，即由 PS 的元素形成的程序树组成。AP 从以程序结构的元素随机地构成计算机程序的原生软体开始，应用畜牧学原理繁殖一个新的（常常是改进了的）计算机程序群体。这种繁殖应用达尔文“适者生存、不适者淘汰”的原理，以一种领域无关的方式（演化算法）进行，即模拟大自然中的遗传操作：复制、杂交与变异。杂交运算用来创造有效的子代程序（由程序结构的元素组成）；变异运算用来创造新的程序，并防止过早

收敛。所以,AP就是把程序结构的高级语言符号表示与智能算法(一种以适应性驱动的、具有自适应、自组织、自学习与自优化特征的高效随机搜索算法)结合起来,即AP=EA+PS。一个求解(或近似求解)给定问题的计算机程序往往就从这个过程中产生。

## 5.1.2 自动编程

本书的作者在《软件演化过程与进化论》(清华大学出版社,2009年)中就如何进行软件基因编程进行了阐述。

**1. 软件基因/组的定义**

软件基因(Software Gene),也称为软件遗传因子,是指携带有软件遗传信息的一条序列串,由0和1组成,是遗传物质的最小功能单位。0和1的不同排列组合决定了软件基因的功能。每一个软件基因是一个指令集合,用以编码软件的程序。基因中的指令可以明确地告诉软件开发工具和程序员如何设计程序。

软件基因组就是由所有的软件基因构成的一个长长的序列串,也是由0和1组成。软件基因组由三个部分组成,它们是基因组头、基因组体和基因组尾。

**2. 软件中心法则**

软件开发的过程就是需求分析到设计(概要设计和详细设计),再到编码的过程。也相当于把软件基因组转换成为软件程序代码的过程。这一过程与生物的中心法则相似,即把DNA转换成RNA,再转换成蛋白质。见图5-1。"软件中心法则"(Software Central Dogma)是指将软件需求转换为软件设计的模块,再将其转换为执行程序的过程。

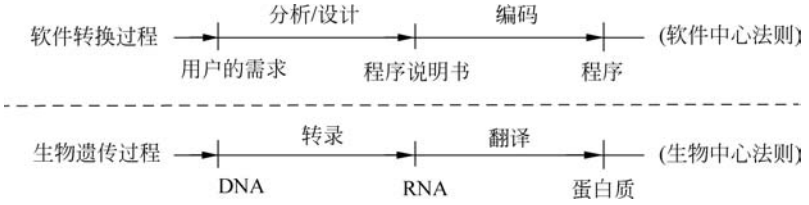


图5-1 软件中心法则与生物中心法则

**3. 转换的步骤**

软件开发的过程就是需求分析到软件设计,再到编码的过程,见图5-2。

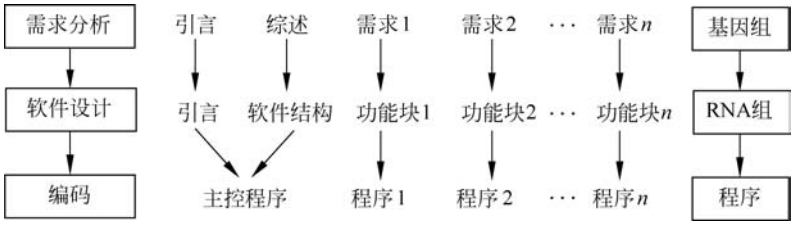


图5-2 软件开发过程

**4. 具体转换过程**

1) 需求分析与基因提取

设用户需求可以表示为集合的形式,见图5-3(a)。实际上,需求分析的过程,就是软件基因提取的过程。根据软件需求规格说明书标准,在软件基因提取的过程中,即需求分析过

程中，将用户需求逐一划分，得到用户的各种需求及彼此的关系。见图 5-3(b)，由此可以得到软件基因组和软件基因。软件基因就是 $X_i(i=1,2,\cdots,n)$，它是用户的 $n$ 个功能需求。$X_0$ 是基因之间的关系，也是基因组的头，$X_i(i=1,2,\cdots,n)$和 $X_0$ 共同构成了基因组。

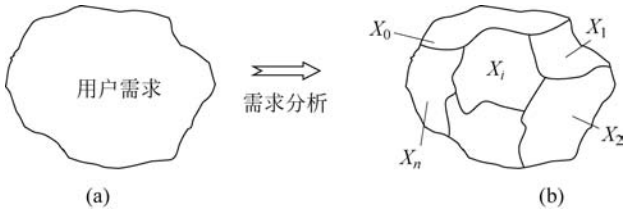


图 5-3　用户需求分析及基因提取

2）软件设计

软件设计的任务有两个，一个是概要设计，一个是详细设计。

(1) 概要设计，就是要将 $X_0$ 转换为软件的总体结构，还要进行内外接口设计和运行组合/控制设计，见图 5-4。

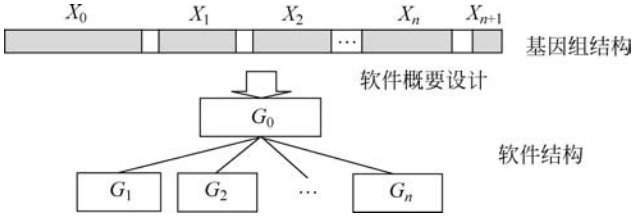


图 5-4　基因组转换成软件结构

(2) 详细设计，就是要将每一个 $X_i(i=1,2,\cdots,n)$转换为每一个程序的具体设计 $G_i(i=1,2,\cdots,n)$。它包括每一个程序的输入/输出、算法、存储等设计，见图 5-5。

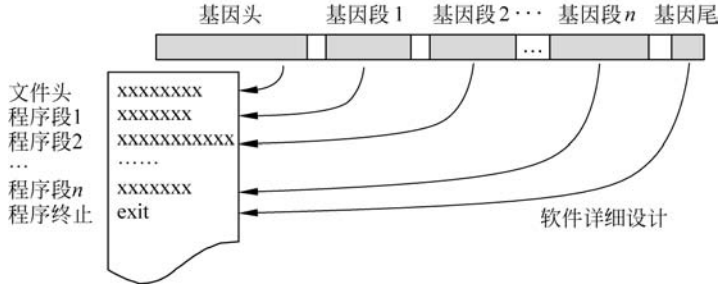


图 5-5　转换成程序

3）编码实现

软件编码实现的任务有两个，一个是每一个程序的编码，一个是整个软件的测试组装。

(1) 每一个程序的编码，将每一个程序设计结果 $G_i(i=1,2,\cdots,n)$转换为执行文件 $Y_i(i=1,2,\cdots,n)$，见图 5-6。

(2) 整个软件的测试组装，就是将所有的程序 $Y_i(i=1,2,\cdots,n)$，根据软件的总体结构，内外接口设计和运行组合/控制等组装成最后的软件产品结果，见图 5-17。

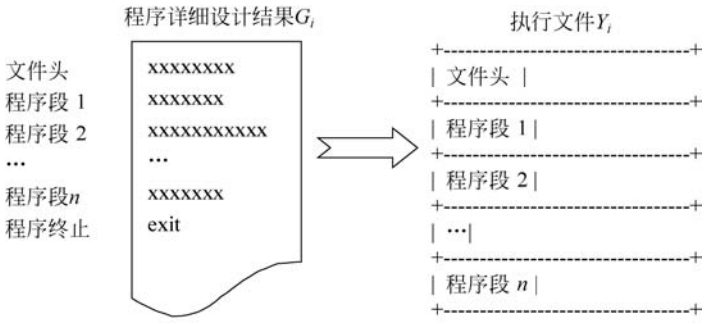


图 5-6　程序详细设计结果转换成执行文件

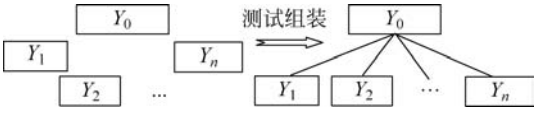


图 5-7　软件结构转换为软件产品

## 5.2　软件产品线与网构软件

### 5.2.1　软件产品线的历史

**1. 软件工程方法发展**

20 世纪 80 年代中期到 20 世纪 90 年代,出现了面向对象语言和方法,并成为主流的软件开发技术;开展软件过程及软件过程改善的研究;注重软件复用和软件构件技术的研究与实践。构件技术是影响整个软件产业的关键技术之一。1998 年在日本召开的国际软件工程会议上,基于构件的软件开发模式成为当时会议研讨的一个热点。美国总统信息顾问委员会也在 1998 年美国国家白皮书上,提出了解决美国软件产业脆弱问题的五大技术,其中之一就是建立国家级的软件构件库。

构件技术的出现是对传统软件开发过程的一次变革。构筑在“构件组装”模式之上的构件技术,使软件技术人员摆脱了“一行行写代码”的低效编程方式,直接进入“集成组装构件”的更高阶段。基于构件的软件开发,不仅使软件产品在客户需求吻合度、上市时间、软件质量上领先于同类产品,提高了项目的成功率,而且使软件的开发和维护变得简单易行,用户可以随时随地应对商业环境变化和 IT 技术变化,实现“敏捷定制”。从最终用户的角度来看,采用基于构件技术开发的系统,在遇到业务流程变化或系统升级等问题时,不再需要对系统进行大规模改造或推倒重来,只要通过增加新的构件或改造原来的构件来实现即可。

**2. 软件产品线**

软件产品线是一组具有共同体系构架和可复用组件的软件系统,它们共同构建支持特定领域内产品开发的软件平台。软件产品线的产品则是根据基本用户需求对产品线架构进行定制,将可复用部分和系统独特部分集成而得到。软件产品线方法集中体现一种大规模、大粒度软件复用实践,是软件工程领域中软件体系结构和软件重用技术发展的结果。

软件产品线的起源可以追溯到 1976 年 Parnas 对程序族的研究。软件产品线的实践早在 20 世纪 80 年代中期就出现了。最著名的例子是瑞士 Celsius Tech 公司的舰艇防御系统

的开发，该公司从 1986 年开始使用软件产品线开发方法，使得整个系统中软件和硬件在总成本中所占比例从使用软件产品线方法之前的 65∶35 下降到使用后的 20∶80，系统开发时间从近九年下降到不到三年。据 HP 公司 1996 年对 HP、IBM、NEC、AT&T 等几个大型公司分析研究，他们在采用了软件产品线开发方法后，使产品的开发时间减少 30%～50%，维护成本降低 20%～50%，软件质量提升 5～10 倍，软件重用达 50%～80%，开发成本降低 12%～15%。

虽然软件工业界已经在大量使用软件产品线开发方法，但是正式的对软件产品线的理论研究到 20 世纪 90 年代中期才出现，并且早期的研究主要以实例分析为主。到了 20 世纪 90 年代后期，软件产品线的研究已经成为软件工程领域最热门的研究领域。2000 年，Gartner Group 预测到 2003 年至少 70%的新应用将主要建立在软件构件之上。随着 Web Services 等技术的发展，将会进一步地推动构件技术的发展，而基于构件的软件开发方式也成为软件开发的主流技术。得益于丰富的实践和软件工程、软件体系结构、软件重用技术等坚实的理论基础，对软件产品线的研究发展十分迅速，目前软件产品线的发展已经趋向成熟。很多大学已经锁定了软件产品线作为一个研究领域，并有大学已经开设软件产品线相关的课程。

**3. 软件产业**

1997 年，由北京大学主持的国家重大科技攻关项目“青鸟工程”中，采用软件构件技术开发的“青鸟Ⅲ型系统”通过了技术鉴定。至今，“青鸟工程”一直在研究开发软件构件库体系，继续推进基于构件的软件开发技术。

2004 年 3 月，由北京软件产业促进中心、软件工程国家工程研究中心启动了“软件构件库系统应用示范”项目。同年 5 月，北京软件行业协会、北京软件产业促进中心、软件工程国家工程研究中心和北京软件产品质量检测检验中心，共同组织开展了“北京第一届优秀软件构件评选活动”，进一步推行基于构件的软件开发方法，丰富了公共构件库系统的资源，并取得了显著的成效。构件化已成为软件企业的需求，软件构件市场已现端倪，软件工业化生产模式正在推进软件产业的规模化发展。

当前我国软件企业面临着日益激烈的国际市场竞争，如果仅依靠软件技术人员，采用手工作坊式的生产模式，当需求稍有变动，就得重新开发系统。基于构件的软件开发技术是当前软件生产的世界潮流，只有掌握这样的技术，才能造就具有竞争力的国际软件企业。

杨芙清院士认为，未来的软件产业将划分为三种业态：一是构件业，类似于传统产业的零部件业，这些构件是商品，有专门的构件库储存和管理；二是集成组装业，它犹如汽车业的汽车工厂，根据市场的需求先设计汽车的款型，然后到市场上采购通用零部件，对特别需要，还可委托专门生产零部件的企业去设计生产，最后把这些零部件在组装车间按设计框架集成组装成汽车；三是服务业，基于互联网平台上的软件服务，已经是当前正在推行的一种软件应用模式，未来这种应用将更加广泛。以上是软件产业发展需求，而且并不遥远，也许几年之内就可能逐步实现。

## 5.2.2 软件产品线的结构与框架

**1. 软件产品线的基本概念**

目前，软件产品线没有一个统一的定义，常见的定义有以下几种。

定义1：将利用了产品间公共方面，预期考虑了可变性等设计的产品族称为产品线(Weiss和Lai)。

定义2：产品线就是由在系统的组成元素和功能方面具有共性和个性的相似的多个系统组成的一个系统族。

定义3：软件产品线就是在一个公共的软件资源集合基础上建立起来的，共享同一个特性集合的系统集合(Bass，Clements和Kazman)。

定义4：一个软件产品线由一个产品线体系结构、一个可重用构件集合和一个源自共享资源的产品集合组成，是组织一组相关软件产品开发的方式(Jan Bosch)。

相对而言，卡耐基·梅隆大学软件工程研究所(CMU/SEI)对产品线和软件产品线的定义，更能体现软件产品线的特征："产品线是一个产品集合，这些产品共享一个公共的、可管理的特征集，这个特征集能满足选定的市场或任务领域的特定需求。这些系统遵循一个预描述的方式，在公共的核心资源基础上开发。"

根据CMU/SEI的定义，软件产品线主要由两部分组成：核心资源，产品集合。核心资源是领域工程的所有结果的集合，是产品线中产品构造的基础。也有组织将核心资源库称为"平台"。核心资源必定包含产品线中所有产品共享的产品线体系结构，新设计开发的或者通过对现有系统的再工程得到的、需要在整个产品线中系统化重用的软件构件；与软件构件相关的测试计划、测试实例以及所有设计文档，需求说明书，领域模型和领域范围的定义也是核心资源；采用COTS(Commercial Off-The-Shelf，商用现成品或技术，或商用货架产品)的构件也属于核心资源。产品线体系结构和构件是用于软件产品线中的产品的构建和核心资源最重要的部分。

**2. 软件产品线的结构**

软件产品线的开发有4个技术特点：过程驱动、特定领域、技术支持和架构为中心。与其他软件开发方法相比，选择软件产品线的原因有：对产品线及其实现所需的专家知识领域的清楚界定，对产品线的长期远景进行了策略性规划。软件生产线的概念和思想，将软件的生产过程细分到三类不同的生产车间进行，即应用体系结构生产车间、构件生产车间和基于构件、体系结构复用的应用集成(组装)车间，从而形成软件产业内部的合理分厂，实现软件的产业化生产。软件生产线如图5-8所示。

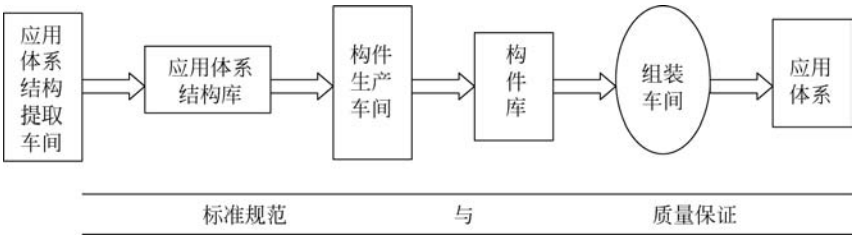


图5-8　软件生产线

1) 软件产品线工程

软件产品线是一种基于架构的软件复用技术，它的理论基础是：特定领域(产品线)内的相似产品具有大量的公共部分和特征，通过识别和描述这些公共部分和特征，可以开发需求规范、测试用例、软件组件等产品线的公共资源。而这些公共资产可以直接应用或适当调

整后应用于产品线内产品的开发，从而不再从草图开始开发产品。因此典型的产品线开发过程包括两个关键过程：领域工程和应用工程。

2）软件产品线的组织结构

软件产品线开发过程分为领域工程和应用工程，相应的软件开发的组织结构也有两个部分：负责核心资源的小组和负责产品的小组。在产品线方法中，主要有三个关键小组：平台组、配置管理组和产品组。

3）软件产品线构件

产品线构件是用于支持产品线中产品开发的可复用资源的统称。这些构件远不是一般意义上的软件构件，它们包括：领域模型、领域知识、产品线构件、测试计划及过程、通信协议描述、需求描述、用户界面描述、配置管理计划及工具、代码构件、性能模型与度量、工作流结构、预算与调度、应用程序生成器、原型系统、过程构件（方法、工具）、产品说明、设计标准、设计决策、测试脚本等。在产品线系统的每个开发周期都可以对这些构件进行精化。

**3. 青鸟的结构**

“七五”期间，青鸟工程就提出了软件生产线的概念和思想，其中将软件的生产过程分成3类不同的生产车间，即应用构架生产车间、构件生产车间和基于构件、构架复用的应用集成组装车间。

由上述软件生产线概念模式图中可以看出，在软件生产线中，软件开发人员被划分为3类：构件生产者、构件库管理者和构件复用者。这3种角色所需完成的任务是不同的，构件复用者负责进行基于构件的软件开发，包括构件查询、构件理解、适应性修改、构件组装以及系统演化等。图5-9给出了与上述概念图相对应的软件生产线——生产过程模型。

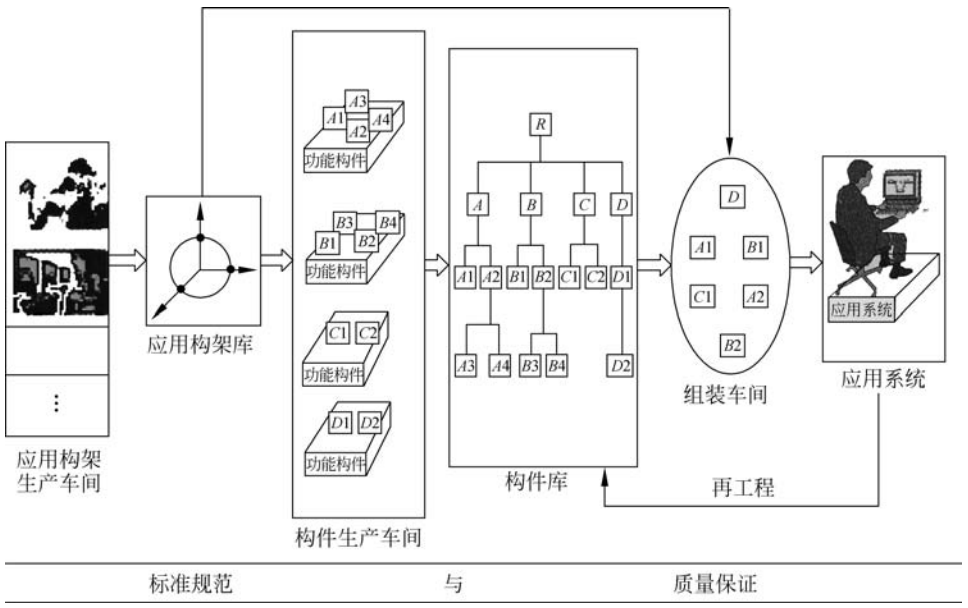


图5-9　青鸟软件生产线系统

从图5-9中可以看出,软件生产线以软件构件/构架技术为核心,其中的主要活动体现在传统的领域工程和应用工程中,但赋予了它们新的内容,并且通过构件管理、再工程等环节将它们有机地衔接起来。另外,软件生产线中的每个活动皆有相应的方法和工具与之对应,并结合项目管理、组织管理等管理问题,形成完整的软件生产流程。

### 5.2.3 网构软件

**1. 背景**

2002—2007年,在国家重点基础研究发展规划(973)的支持下,北京大学、南京大学、清华大学、中国科学院软件研究所、中国科学院数学研究所、华东师范大学、东南大学、大连理工大学、上海交通大学等单位的研究人员以我国软件产业需支持信息化建设和现代服务业为主要应用目标,提出了"Internet环境下基于Agent的软件中间件理论和方法研究",并形成了一套以体系结构为中心的网构软件技术体系。主要包括三个方面的成果:一种基本实体主体化和按需协同结构化的网构软件模型,一个实现网构软件模型的自治式网构软件中间件,以及一种以全生命周期体系结构为中心的网构软件开发方法。

在网构软件实体模型中,剥离了对开放环境以及其他实体的固化假设,以解除实体之间以及实体与环境之间的紧密耦合,进而引入自主决策机制来增强实体的主体化特性;在网构软件实体协同方面,针对面向对象方法调用受体固定、过程同步、实现单一等缺点,对其在开放网络环境下予以按需重新解释,即采用基于软件体系结构的显式的协同程序设计,为软件实体之间灵活、松耦合的交互提供可能;在网构软件运行平台(中间件)方面,通过容器和运行时软件体系结构分别具体化网构软件基本实体和按需协同,并通过构件化平台、全反射框架、自治回路等关键技术实现网构软件系统化的自治管理;在网构软件开发方法方面,提出了全生命周期软件体系结构以适应网构软件开发重心从软件交付前转移到交付后的重大变化,通过以体系结构为中心的组装方法支持网构软件基本实体和按需协同的开发,采用领域建模技术对无序的网构软件实体进行有序组织。

**2. 网构软件**

进入21世纪,以Internet为代表的网络逐渐融入人类社会的方方面面,极大地促进了全球化的广度和深度,为信息技术与应用扩展了发展空间。另一方面,Internet正在成长为一台由数量巨大且日益增多的计算设备所组成的"统一的计算机",与传统计算机系统相比,Internet为应用领域问题求解所能提供的支持在量与质上均有飞跃。为了适应这些应用领域及信息技术方面的重大变革,软件系统开始呈现出一种柔性可演化、连续反应式、多目标自适应的新系统形态。

从技术的角度看,在面向对象、软件构件等技术支持下的软件实体以主体化的软件服务形式存在于Internet的各个结点之上,各个软件实体相互间通过协同机制进行跨网络的互连、互通、协作和联盟,从而形成一种与WWW相类似的软件Web(Software Web)。将这样一种Internet环境下的新的软件形态称为网构软件(Internetware)。传统软件技术体系由于其本质上是一种静态和封闭的框架体系,难以适应Internet开放、动态和多变的特点。一种新的软件形态——网构软件适应Internet的基本特征,呈现出柔性、多目标和连续反应式的系统形态,将导致现有软件理论、方法、技术和平台的革命性进展。

网构软件包括一组分布于Internet环境下各个结点的、具有主体化特征的软件实体,以

及一组用于支撑这些软件实体以各种交互方式进行协同的连接子。这些实体能够感知外部环境的变化，通过体系结构演化的方法（主要包括软件实体与连接子的增加、减少与演化，以及系统拓扑结构的变化等）来适应外部环境的变化，展示上下文适应的行为，从而使系统能够以足够的满意度来满足用户的多样性目标。网构软件这种与传统软件迥异的形态，在微观上表现为实体之间按需协同的行为模式，在宏观上表现为实体自发形成应用领域的组织模式。相应地，网构软件的开发活动呈现为通过将原本“无序”的基础软件资源组合为“有序”的基本系统，随着时间推移，这些系统和资源在功能、质量、数量上的变化导致它们再次呈现出“无序”的状态，这种由“无序”到“有序”的过程往复循环，基本上是一种自底向上、由内向外的螺旋方式。

网构软件理论、方法、技术和平台的主要突破点在于实现如下转变，即从传统软件结构到网构软件结构的转变，从系统目标的确定性到多重不确定性的转变，从实体单元的被动性到主动自主性的转变，从协同方式的单一性到灵活多变性的转变，从系统演化的静态性到系统演化的动态性的转变，从基于实体的结构分解到基于协同的实体聚合的转变，从经验驱动的软件手工开发模式到知识驱动的软件自动生成模式的转变。建立这样一种新型的理论、方法、技术和平台体系具有两个方面的重要性，一方面，从计算机软件技术发展的角度，这种新型的理论、方法和技术将成为面向 Internet 计算环境的一套先进的软件工程方法学体系，为 21 世纪计算机软件的发展构造理论基础；另一方面，这种基于 Internet 计算环境上软件的核心理论、方法和技术，必将为我国在未来若干年建立面向 Internet 的软件产业打下坚实的基础，为我国软件产业的跨越式发展提供核心技术的支持。

**3. 网构软件模型**

基于面向对象模型，提出了一种基于 Agent、以软件体系结构为中心的网构软件模型，如图 5-10 所示。图 5-11 为网构软件中间件模型。图 5-12 为网构软件开发方法体系。

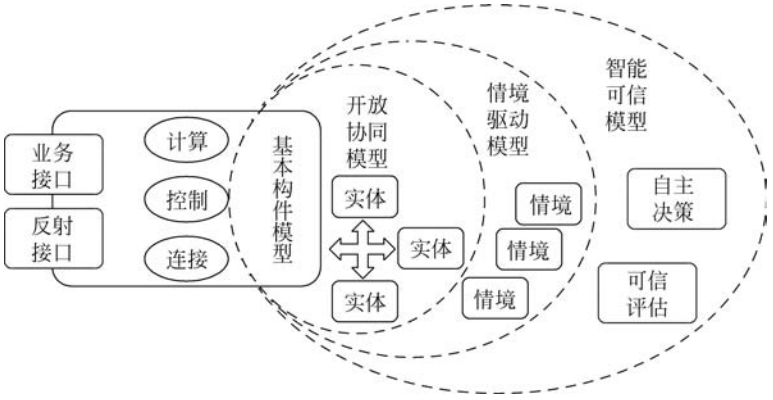


图 5-10　网构软件模型

进一步的工作主要是加强现有成果的深度和广度。在深度方面，完善以软件体系结构为中心的网构软件技术体系，重点突破网构软件智能可信模型、网构中间件自治管理技术，以及网构软件开发方法的自动化程度。在广度方面，多网融合的大趋势使得软件将运行在一个包含 Internet、无线网、电信网等多种异构网络的复杂网络环境，网构软件是否需要以及能否从 Internet 延伸到这种复杂网络环境，成为我们下一步的主要目标。

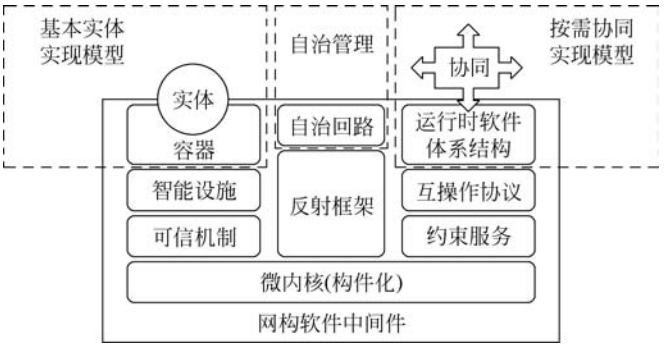


图 5-11　网构软件中间件模型

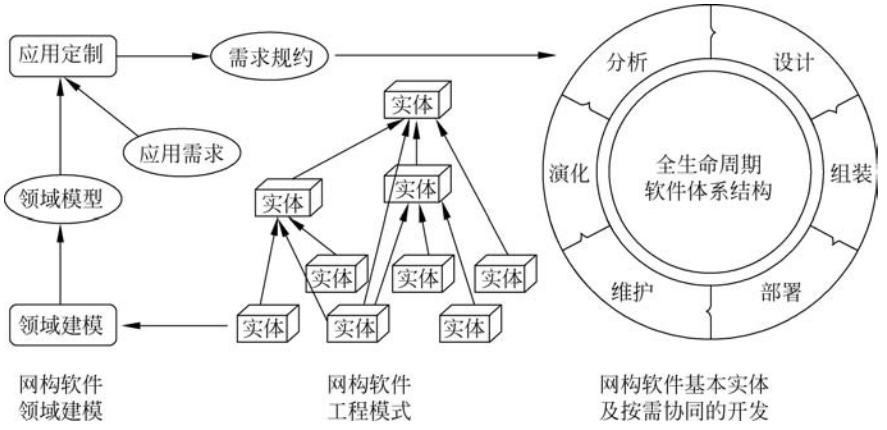


图 5-12　网构软件开发方法体系

## 5.3　软件工具酶

### 5.3.1　软件工具酶的作用

**1. 生物酶**

酶(Enzyme)是由细胞产生的具有催化能力的蛋白质(Protein),这些酶大部分位于细胞体内,部分分泌到体外。新陈代谢是生命活动的重要特征,它维持生命的正常运转,然而生物体代谢中的各种化学反应都是在酶的作用下进行的。没有酶,生命将停止。因此,研究酶的理化性质及其作用机理,对于阐明生命现象的本质具有十分重要的意义。

1) 酶的作用机制

酶通过其活性中心先与底物形成一个中间复合物,随后再分解成产物,并放出酶。酶的活性部位是它结合底物和将底物转换为产物的区域,通常是整个酶分子相当小的一部分。活性部位通常在酶的表面空隙或裂缝处,形成促进底物结合的优越的非极性环境。在活性部位,底物被多重的、弱的作用力结合,在某些情况下被可逆的共价键结合。酶结合底物(Substrate)分子,形成酶-底物复合物。酶活性部位的活性残基与底物分子结合,先将它转

变为过渡态，然后生成产物，释放到溶液中。这时游离的酶与另一分子底物结合，开始它的又一次循环。底物是接受酶的作用引起化学反应的物质。已经有两种模型解释了酶如何与它的底物结合。1894 年，Emil Fischer 提出锁和钥匙模型，底物的形状和酶的活性部位被认为彼此相适合，像钥匙插入它的锁中（见图 5-13(a)），两种形状被认为是刚性的和固定的，当正确组合在一起时，正好互相补充。诱导契合模型是 1958 年由 Daniel E. Koshland Jr. 提出的，底物的结合在酶的活性部位诱导出构象变化，见图 5-13(b)。此外，酶可以使底物变形，迫使其构象近似于它的过渡态。


图 5-13　底物与酶的结合

2）酶的催化特点

（1）催化能力。酶加快反应速度可高达 $10^{17}$ 倍。

（2）专一性。大多数酶对所作用的底物和催化的反应都是高度专一的。不同的酶专一性程度不同。有些酶专一性很低（键专一性），可以作用很多底物，只要求化学键相同。具有中等程度专一性为基团专一性。大多数酶呈绝对或几乎绝对的专一性，它们只催化一种底物进行快速反应。

（3）调节性。生命现象表现了它内部反应历程的有序性。这种有序性受多方面因素的调节和控制，而酶活性的控制又是代谢调节作用的主要方式。酶活性的调节控制主要有下列几种方式：酶浓度的调节、激素调节、共价修饰调节、限制性蛋白水解作用与酶活力调控、抑制剂的调节、反馈调节等。

**2. 软件工具酶**

为了用生物的方法讨论软件的演化过程，现从生物的角度看软件过程。在软件开发过程中，软件工具相当于生物学中“酶”的角色。也就是说，软件工具在软件开发过程中起“酶”的作用。那么，什么是软件工具酶（Software Tool Enzyme，STE）呢？软件工具酶是在软件开发过程中辅助开发人员开发软件的工具。

1）软件工具酶的作用

（1）软件开发工具作为酶，它是催化剂（Catalyst）。可使用户需求转换为程序的过程加快。这一点很多做过软件开发的人都有体会。与生物酶一样，软件工具酶作为催化剂时，它只辅助需求到程序的转换，而且参与其活动，但是，它不会变成为被开发软件的一部分，而且软件“酶”可以被反复使用。

（2）软件开发工具作为酶，也是黏合剂（Adhesive）。它可以把底物切碎，也可以将碎片连接起来。这就是所说的酶切（Enzyme Digestion）和酶连接（Enzyme Ligation）。例如，在软件开发过程中，需求分析常常将用户的需求分类、系统化，然后再合并成需求分析说明书，需求分析工具就有这种所谓的“黏合剂”作用；在第四代语言中，例如，VB、PB 等都有将分散模块集成在一起的作用，这一点与生物酶的功能一样。目前，很多基于组件的“软件工厂”

平台,都具有组装软件的功能,比如北京大学的青鸟系统。在开发过程中,项目管理工具也具有连接每一个软件开发过程(需求分析阶段,设计阶段,编程阶段,测试阶段,运行维护阶段)的能力。

(3) 软件底物是软件工具酶作用的对象。在软件开发阶段,软件工具酶作用的对象或底物不一定相同。在需求分析阶段,软件工具酶作用的对象是用户需求;在设计阶段,软件工具酶作用的对象是用户需求说明书,它要将用户需求说明书转换成软件概要设计说明书和详细设计说明书;在编程阶段,软件工具酶作用的对象是详细设计说明书,它要将详细设计说明书转换成软件程序;在测试阶段,软件工具酶作用的对象是程序单元和整个软件系统;在运行维护阶段,软件工具酶作用的对象是整个软件系统;而对于项目管理来说,软件工具酶作用的对象是整个软件开发过程的活动。

2) 软件工具酶的作用机理

实际上,软件工具酶是通过其活性中心先与底物形成一个中间复合物(Compound),随后再分解成产物,酶被分解出来。酶的活性部位在其与底物结合的边界区域。软件工具酶结合底物,形成酶-底物复合物。酶活性部位与底物结合,转变为过渡态,生成产物,然后释放。随后,软件工具酶与另一底物结合,开始它的又一次循环。软件工具酶与底物结合的两种模型如下。

(1) 锁和钥匙模型认为:底物的形状和酶的活性部位被认为彼此相适合,像钥匙插入它的锁中,刚好组合在一起时,互相补充。实际上,这是一种静态的模型,它可以解释软件工具酶与底物的配套关系。有些软件工具酶和底物都是不可变的,彼此非常专一,像锁和钥匙一样,配合得"天衣无缝"。例如,为某一个单元专门设计编制的测试台/床(一种专用的单元测试工具酶),或功能单一的程序编辑器(一种专用的工具酶),或功能单一的数据流图绘制工具(也是一种专用的工具酶),见图 5-14。专用测试台/床的接口数与底物的接口数相同,且其他方面也像锁和钥匙一样,配合得天衣无缝。

(2) 诱导契合模型认为:底物的结合在酶的活性部位诱导出构象变化。酶可以使底物变形,迫使其构象近似于它的过渡态。这样一种动态的模型,也可以解释软件工具酶与底物的适应关系。有些软件工具酶和底物都是"可变的",它们在催化结合时,或软件工具酶适应底物,或底物适应软件工具酶,或彼此可以适应,相互被诱导契合,同样配合得天衣无缝。例如,为某一个单元通用的测试台/床(一种"通用"的单元测试工具酶),见图 5-15。通用测试台/床的接口数是动态的,它通过侦测底物的接口数,然后与之适应,被诱导与被测单元契合,形成天衣无缝的组合。对于其他的"通用"工具酶,其核心结合原理也是侦测底物的属性,然后与之适应,形成酶-底物复合物,进行催化作用。

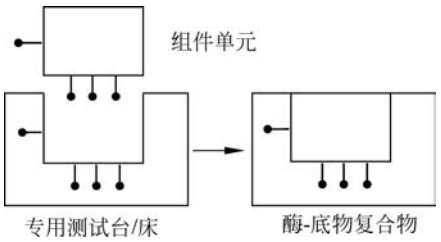


图 5-14 锁和钥匙静态的模型

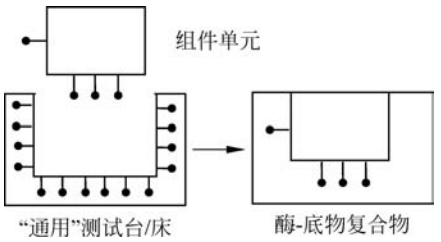


图 5-15 诱导契合动态的模型

关于软件工具酶与底物的界面讨论见 5.3.2 节。

3) 软件工具酶的催化特点

(1) 催化(Catalysis)能力。本书作者曾做过一个实验,对比软件工具酶加快反应速度。使用课件自动生成酶与没有使用软件工具酶编制课件,所用的时间比是 480。当时,用 PowerPoint 编制“系统分析与设计”课程的课件时,耗时约 40h,而使用我们开发的“课件自动生成系统(湖北省教育厅 2003 重点教学科研项目,编号:2003X182)”,自动生成课件耗时约 5min,所用的时间比是 480。这说明,软件工具酶的催化作用是非常大的。该实验只是从一个侧面反映了软件工具酶的加速催化能力。

(2) 专一性(Specifity)。大多数软件工具酶对所作用的底物的催化反应也是高度专一的。当然,与生物酶一样,不同的酶专一性程度不同。例如,软件开发的通用工具 Word 编辑软件,它可以作为开发文档的开发工具使用,但专用性很低,其针对软件开发的效率和专业性自然非常差,其功能和性能绝对无法超越 IBM 的 Rational Rose。大多数软件工具酶(需求分析工具酶,设计工具酶,程序生成酶,测试工具酶,项目管理工具酶)呈绝对或几乎绝对的专一性,它们只催化一种底物进行快速反应。例如,需求分析工具酶只针对需求分析过程的活动,结构化概要设计工具酶只针对结构化概要设计,C 语言程序生成器只生成某一功能的 C 语言程序,单元测试工具酶只针对单元测试,项目管理工具只针对项目管理。

(3) 调节性(Adjustment)。软件开发是一个有序性的工作,其中,软件项目管理工具的调节和控制功不可没,它在其中担当起了较强的控制调节作用。软件工具酶活性的调节控制方式有一些:增加软件工具酶的品种和数量(浓度)的调节,利用管理软件的反馈调节等。

4) 影响软件工具酶活力的因素

(1) 酶的速度。也就是酶催化反应的速度。由上面的例子可知,软件工具酶可以“催化”底物反应的速度上百倍,甚至上千倍。

(2) 底物的浓度(数量)。当底物的数量较大时,因为软件工具酶的用户数限制,软件工具酶因为数量较大,忙不过来,而实际导致能力“下降”。

(3) 软件工具酶的浓度(数量)。当软件工具酶的数量较多(比如局域网上的所有机器都安装了软件工具酶,每个开发人员都能用到一套软件工具酶;另外,各种功能的软件工具酶产品也比较多)时,整体开发团队的软件工具酶的应用能力相对提高。

(4) 开发人员。人数和人的素质都是影响软件工具酶活力的重要因素。人的素质高,使用软件工具酶的效率也高。人数和软件工具酶的数量越多,整体软件工具酶的能力就比较高。

(5) 环境。好的软件工具酶运行环境,对提高其性能会有比较大的帮助。反之,会限制其能力的发挥。

5) 软件工具酶的任务

软件工具酶的任务就是辅助开发人员完成软件开发的某一个过程。专用工具酶,只完成一个具体任务;集成工具酶,可以辅助完成整个开发过程的“所有”任务。

软件工具酶的具体任务是:①把用户需求转换为需求说明书。首先对用户需求细分,再把需求的细分部分(如数据字典的数据项、子功能要求、数据的结构等要求)重新连接起来,形成需求分析说明书;②将需求分析说明书转换成概要设计说明书和详细设计说明书;

③将详细设计说明书转换成一个个模块,最后将模块连接起来转换成软件。作为酶的软件开发工具起到了需求转换和设计集成合并的作用。

这一过程相当于生物学中的遗传信息传递的过程,即 DNA→RNA→Protein(蛋白质)。软件工具酶的作用就是实现从用户需求到软件程序的转换,即需求(Requirement)→设计说明书(Specification)→程序(Program)。

**3. 软件工具酶的任务**

1) 生物中心法则与酶

DNA 核苷酸序列是遗传信息的储存者,它通过自主复制得以永存,通过转录生成信使 RNA,进而翻译成蛋白质的过程来控制生命现象,即储存在核酸中的遗传信息通过转录,翻译成为蛋白质,这就是生物学中的中心法则。该法则表明信息流的方向是 DNA→RNA→Protein(蛋白质)。

从上面的过程可以看出,DNA→RNA→蛋白质,经过了复制→转录→翻译三个过程。在这三个过程中,DNA 解链酶,RNA 聚合酶和肽基转移酶分别参与了其转换活动。图 5-16 给出了生物中心法则与酶的关系。

2) 软件转换法则

软件工具酶的中心任务就是辅助开发人员,将用户需求转换为计算机可以运行的程序。众所周知,软件开发就是将用户需求正确地转换为软件程序。一般地说,软件开发需要经过三次转换过程,一是用户需求的获取,二是从用户需求到程序说明书的信息转换,三是从程序说明书到程序的信息转换,见图 5-17。

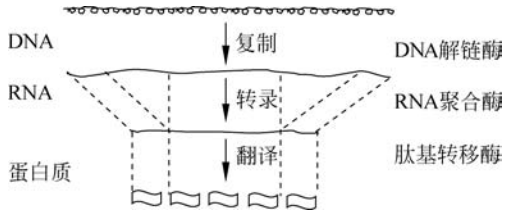


图 5-16 生物转换过程

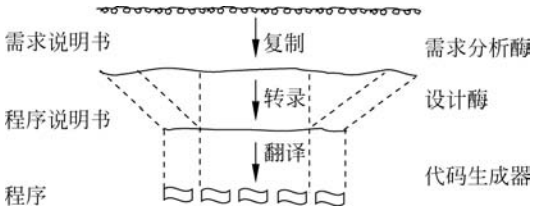


图 5-17 需求到程序的转换

(1) 用户需求的获取。用户需求的获取相当于设计人员“复制”用户的需求,经过需求分析后得到需求分析说明书,为下一阶段“转录”做准备。

(2) 从用户需求到程序说明书的信息转换。这一过程首先是根据需求分析说明书为模板,将其“转录”为概要设计说明书,并进一步产出详细设计说明书,也就是程序说明书。这一过程相当于将用户的要求转录成 mRNA,为最后的程序“翻译”做准备。

(3) 从程序说明书到程序的转换。这一过程就要将程序说明书,也就是将详细设计说明书“翻译”为程序。

3) 生物与软件转换的比较

从需求到程序的转换,首先是将用户的需求“复制”给系统分析员,产出需求分析说明书,然后将其“转录”为概要设计说明书和详细设计说明书,最后将其“翻译”为程序,见图 5-18。

这一过程相当于遗传信息传递的过程,即 DNA→RNA→Protein 的转换,见图 5-19。

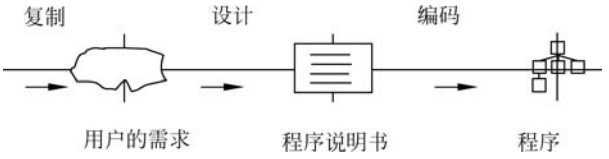


图 5-18　软件需求到程序的转化过程

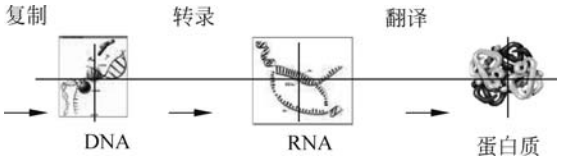


图 5-19　生物遗传过程

尽管生物和软件的转换法则之间有一些相同之处，例如，它的转换法则都是三步，且三步的性质和任务也非常相似，但是两者之间还是有不少的差异，例如，整个过程的工作内容不同，结果也不同，也必然导致一些细节不同。

(1) 第一个过程同异。软件转换开始的用户需求开始往往是模糊的、不清楚的、不准确的。在从用户到系统分析员的“复制”过程中，也时常要反反复复，所以，其需求获取过程是艰难的，而且，用户需求想法也未数字化。与之相比，生物的中心法则的 DNA 复制则是非常准确的，毫不含糊，清清楚楚。不过，经过了反反复复的需求分析后得到的需求分析说明书应该是数字化的，清楚而准确的，它是 DNA 转录的模板。相同之处是它们的任务都是“复制”，而且任务相对其他几个过程算是比较“简单”的。

(2) 第二个过程同异。在软件开发过程中，用户需求“复制”给系统分析员后，系统分析员开始将其“转录”为需求分析说明书，然后对用户的需求进行分析加工，再将其“转录”为概要设计说明书，并进一步“转录”为详细设计说明书，即程序说明书。与生物过程基本相同，这一过程也分为两步：需求分析说明书→概要设计说明书→详细设计说明书。第一步，需求分析说明书→概要设计说明书相当于生物中的 DNA→hnRNA；第二步，概要设计说明书→详细设计说明书相当于生物中的 hnRNA→mRNA。这一阶段的任务相对于第一个过程来说是比较“复杂”的。

(3) 第三个过程同异。在软件开发过程中，第三个过程是详细设计说明书，即程序说明书到程序的“翻译”，一个个程序是没有什么用的，必须组装成软件后才能发挥作用。与生物过程基本相同，这一过程也分为两步：详细设计说明书→程序→软件。第一步，将详细设计说明书“翻译”成程序，相当于生物中的 mRNA→肽链；第二步是程序组装，即程序→软件，相当于生物中将肽链“剪接”成为有功能的蛋白质的过程。这一阶段的任务比第一个过程“复杂”。

## 5.3.2　软件工具酶的结构

软件工具酶的功能与软件的结构有密切关系。软件功能越多，其结构也越复杂。反之亦然。本节将讨论软件工具酶的功能与其结构有密切关系。

**1. 软件工具酶的一般结构**

专用工具酶是针对某一个过程的，而集成工具酶是针对整个过程的，它们在功能方面有

很大的区别,因此,其结构也有很大的区别。

1) 专用工具酶的结构

软件开发阶段划分成需求分析阶段、设计阶段、编程阶段、测试阶段、运行维护阶段几个大的开发阶段。对应的专用软件工具酶包括:需求分析工具酶、设计工具酶、程序生成酶、程序测试酶、维护工具酶和过程管理工具酶。

在软件开发的早期,软件工具酶常常以单独或专用工具酶的方式存在,使用的过程中也是彼此独立的,当然,功能也是彼此独立的,而且,它们所带的信息库也是彼此独立的,甚至使用它们的人也不相同。例如,系统分析员通常使用需求分析工具酶和设计工具酶,编程人员则使用程序生成酶和程序编辑工具酶,测试工程师则使用测试工具酶,维护人员则使用维护工具酶,项目经理则使用软件过程管理工具酶,见图 5-20。

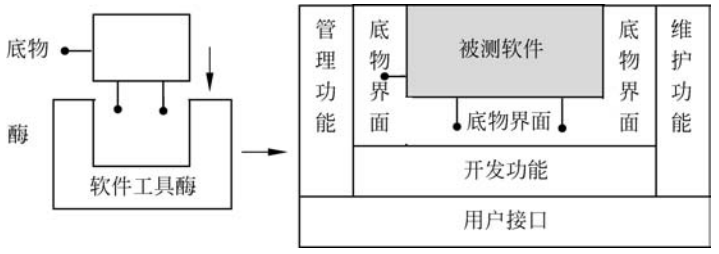


图 5-20 专用工具酶结构

从图 5-20 中可以看出,软件集成工具酶与用户交流,有用户接口部分,与被开发软件接触,则有底物界面。其最重要的核心功能是项目管理(左)、开发功能(中)和维护功能(右)。其中,开发功能包括需求分析的功能,软件概要设计和详细设计的功能,程序编码的功能和软件测试的功能。

如图 5-21 所示,虚线内的部分为针对某一开发过程或专项工具酶,用虚线区别不同的专用工具酶之间彼此的独立关系。

2) 集成工具酶的结构

随着技术的不断完善,软件工具酶集成变成为一种趋势。一般地,集成工具酶是针对整个过程的工具。尽管其内部是由多个部分组成的,但是它们彼此之间必须通力合作,完成整个软件开发过程。实际上,集成工具酶是由多个单项工具酶或专用工具酶组成的。在不同的时期,这种组合或融合的程度是有差别的。至少可以分为松散型和紧密型两种。实际上,软件工具酶集成的历史经过较长一段时间,它从一个侧面反映了软件工具酶的进化程度。在本章的最后,将讨论软件工具酶的进化。

集成工具酶有别于专用工具酶,其结构和功能有比较大的区别。由于专用工具酶功能比较单一,因此其结构也比较简单。而集成工具酶则有时集成了整个软件开发过程,其功能不仅复杂得多,而且结构也复杂得多,见图 5-22。

图 5-23 仅给出了集成工具酶的结构框架,下面将仔细讨论其细部的结构和功能。从集成的角度看,集成工具酶至少包括四大部分:软件工具酶与底物界面、专用工具酶集、用户界面与总控台。其中,专用工具酶集包括需求分析工具酶、设计工具酶、程序生成工具酶、测试工具酶、维护工具酶、项目管理工具酶和信息库,见图 5-23。

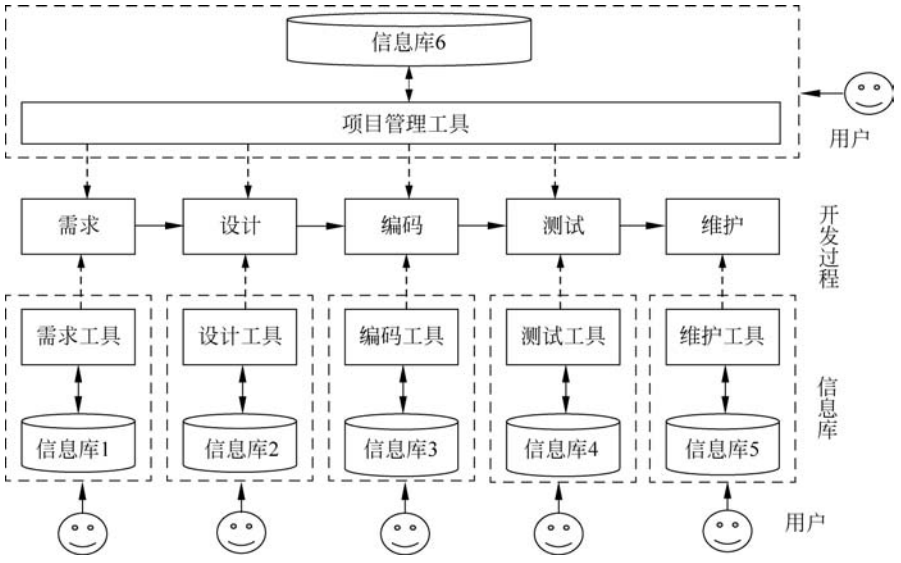


图 5-21　专用工具与开发过程的关系

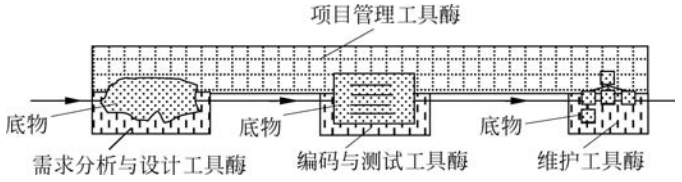


图 5-22　集成工具酶与底物

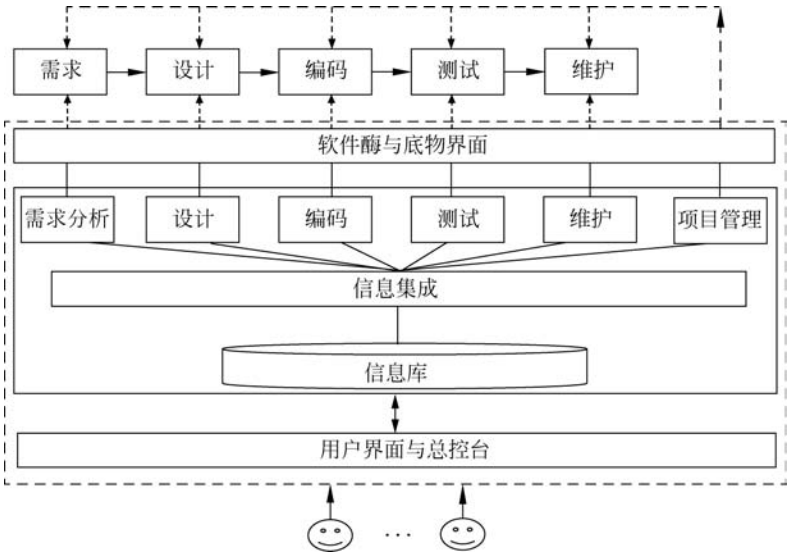


图 5-23　集成工具与开发过程的关系

**2. 软件工具酶与底物界面**

1) 生物酶与底物的关系及工作原理

酶通过其活性中心先与底物形成一个中间复合物,随后再分解成产物,并放出酶。酶的活性部位是它结合底物和将底物转换为产物的区域,它只占整个酶分子相当小的一部分。活性部位通常在酶的表面空隙或裂缝处,形成促进底物结合的优越的非极性环境。在活性部位,底物被多重的、弱的作用力结合,在某些情况下被可逆的共价键结合。酶结合底物分子,形成酶-底物复合物。酶活性部位的活性残基与底物分子结合,先将它转变为过渡态,然后生成产物,释放到溶液中。这时游离的酶与另一分子底物结合,开始它的又一次循环,见图 5-24。

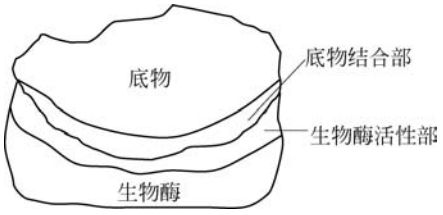


图 5-24 生物酶与底物关系

2) 软件工具酶与底物的关系

"底物界面"(Enzyme-Substrate Interface,或 Substrate Interface)是本章提出的一个新概念,它是从生物中借鉴过来的,实际上,"底物界面"就是软件接口。软件接口有两种含义:一是指软件本身的狭义"接口",指各种应用软件接口 API;二是指人与软件之间的交互界面,即人与软件之间的接口,称作"用户界面",也就是 UI。

人机界面是用户进入软件的门面,它是人机交互的纽带,以及人机交流的桥梁。早年的人机界面设计,功能和性能比较单一,软件设计什么样的人机界面,用户就必须使用或适应什么样的人机界面。近年来,随着人机界面技术的飞速发展,有关"自适应人机界面(Self-adaptive User Interface)"和"智能人机界面(Intelligent User Interface)"的文章和系统开始逐渐问世,这是历史和技术的进步。人机界面必须具有适应不同用户的功能特性。

与人机界面的概念类似,"底物界面"是底物结合软件工具酶的边界,是底物与软件工具酶交互和交流的纽带和桥梁。

早年的软件工具酶让用户普遍感觉不好用,原因是人们在设计软件工具酶时从来没有考虑过软件工具酶与底物的关系、适应性和专一性。如果我们能像现在考虑"自适应人机界面"和"智能人机界面"那样,考虑软件工具酶中的"底物界面"设计,特别是智能"底物界面"设计,那么,软件工具酶与底物的配合将让用户感到它们非常配套,非常合适。

3) "底物界面"功能

"底物界面(Substrate Interface)"是软件工具酶的一部分,是面向底物的一部分。尽管这一部分在软件工具酶中占的比例较小,但是,其功能就是要吸住底物。当然,"底物界面"吸引和结合的方式和手段很多,例如,与底物呈锁和钥匙关系,弱外部力和功能度等。"与底物呈锁和钥匙关系"方法主要是利用软件工具酶与底物的结构匹配性。"弱外部力"方法主要是利用开发人员的选择权利。由于市面上的产品很多,开发人员可以选择这种产品也可选择那种产品,最后的使用权还是在开发人员。"功能度"手段也是一种,因为如果软件工具酶的整体功能度比较强,即使匹配性弱一点儿,有时也能战胜匹配性强的软件工具酶。总之,"底物界面"的强弱决定软件工具酶整体的功能和性能,即综合指标。

那么,"底物界面"究竟要包括一些什么样的能力呢?根据软件工具酶的性质可知,"底物界面"至少应该具备这样一些能力:吸引力,匹配能力和结合力。吸引力指吸住底物的能

力。如果底物可以与这种软件工具酶结合，也可以与另外一种结合，那么，凭什么会选择 A 而不是 B？关键是开始的吸引力。一旦被吸引住，匹配能力就开始起作用。倘若能马上匹配上，底物与软件工具酶的结合即告成功。否则，即使被吸住也将因无法匹配而分开。待底物与软件工具酶匹配成功后，结合力的强弱将是导致软件工具酶能否催化底物反应成功的关键。

能力是抽象的指标，能力只有转换为功能指标才能具体实现。本书作者认为软件工具酶应该具备以下几个功能：结构功能匹配，适应功能，抓附功能，通信功能。

(1) 结构功能匹配指某一种底物催化反应需要的结构和功能。例如，需求分析肯定需要能做需求分析的工具酶，这种酶在功能和结构方面一般可以匹配。

(2) 适应功能指一旦软件工具酶的基本结构和功能达到后的局部微调能力。有时这种能力也非常重要，它能驱使软件工具酶与底物结合得更贴切，催化效果更好。关于它的实现，读者可以从“自适应人机界面”和“智能人机界面”角度理解。

(3) 抓附功能指一旦匹配到达要求后，软件工具酶对底物的吸附与结合的强度。如果软件工具酶对底物的抓附功能不强，即使彼此匹配上，也会因为环境影响而彼此分开而不能完成对底物的催化反应任务。

(4) 通信功能指一旦软件工具酶与底物结合后彼此的信息双向交流能力。

4) 举例解释

无论怎么解释，总会感觉“底物界面”的概念和功能比较抽象。下面举例说明“底物界面”的概念和功能。例如，软件单元测试的工具内设置测试台或测试床，见图 5-25。由于每一个被测试的单元不同，其单元测试酶的活性部位，即“底物界面”担当起“诱导契合”的作用。当一个被测试单元进入测试台/床时，测试台/床与被测单元被合而一体，形成“软件工具酶-底物”复合物，单元测试酶开始测试，一旦这项测试完成，软件工具酶与底物(被测试单元)分离，单元测试酶又进行下一次循环。

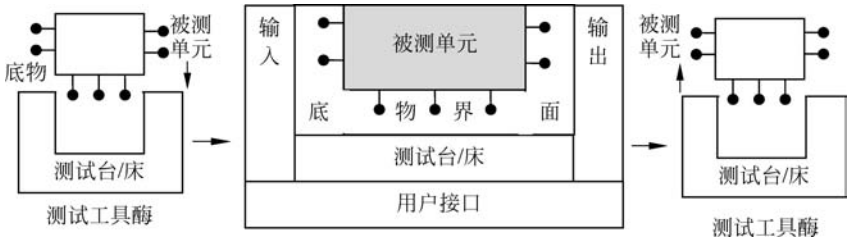


图 5-25　底物与酶契合状态

实际上，很多软件都具有匹配能力。例如 Windows 操作系统，它就具有适应不同硬件环境的匹配能力。Windows 通过扫描，在安装库中寻找可以匹配的硬件驱动程序，如果没有这样的硬件驱动程序，则找类似的替代硬件驱动程序。另外，Windows 也允许从外部安装硬件驱动程序。这从一个侧面反映出酶与底物的关系，即酶与底物相互适应和调整的特点。在此，不能过度强调软件工具酶对底物的匹配和适应能力，催化与被催化是双方的事。正如上面的单元测试的例子。如果程序单元“千变万化”，任何一个软件测试工具酶也无法工作。因此，程序单元的标准化也是非常重要的。

相信“底物界面”这个概念，或者叫功能，如果它在以后的软件体系结构设计中不仅要考

虑到,而且被应用,其功能将会对未来软件工具酶产生巨大的影响。

### 5.3.3 软件工具酶与底物界面



从生物的角度讲,底物是接受酶的作用引起化学反应的物质。从软件工具酶的角度讲,软件底物是软件工具酶的作用对象。软件"底物界面"实际上就是软件接口。众所周知,软件接口有两种:一是各种应用软件接口 API,二是人与软件之间的"用户界面"。上面已经讨论了"用户界面",因此,本节仅讨论应用软件接口。

**1. 软件接口**

根据《英语计算机技术词典》(英语计算机技术词典编委会,电子工业出版社,1992)中"接口"的定义,它指两个功能部件之间的共同边界。

接口(Interface)用来定义一种程序的协定。实现接口的类或者结构要与接口的定义严格一致。有了这个协定,就可以抛开编程语言的限制(理论上)。接口可以从多个基接口继承,而类或结构可以实现多个接口。接口可以包含方法、属性、事件和索引器。接口本身不提供它所定义的成员的实现。接口只指定实现该接口的类或接口必须提供的成员。

软件接口实际上是不同功能部件之间的交互部分。通常就是所谓的 API 应用程序编程接口,其表现的形式是源代码。下面以 COM 组件为例,简单谈谈软件的接口技术。

1) COM 组件接口

在 COM 组件模型中,接口是最为重要的概念,在整个应用系统中起决定性作用,外界和组件方所有的交互都通过接口实现,因此接口设计的优劣直接影响组件的质量。良好的接口的设计有利于提高组件的可用性、可理解性,有利于软件的维护、扩展和重用。不合理的设计则会导致组件难以理解、难以选择,从而影响整个软件的可靠性。

接口是一组逻辑上相关的函数集合,客户程序利用这些函数获得组件对象的服务。COM 组件的位置对客户来说是透明的,因为客户并不直接去访问 COM 组件,客户程序通过一个全局标识符进行对象的创建和初始化工作。COM 规范采用了 128 位全局唯一标识符 GUID,这是一个随机数,并不需要专门机构进行分配和管理。虽然 GUID 是个随机数,但它有一套算法用以产生该数,发生重复的可能性非常小,理论上完全可以忽略。

2) COM 接口的设计

一个接口可以用来完成某个功能或者说明某个行为,它包含一组相关操作,这些操作相互协作实现接口功能。因此接口设计的第一步是将需求阶段获得的功能需求转换为接口。在面向对象的需求分析中,功能需求通常由用例图描述,包括活动者、用例以及用例之间的关系,它能够在不考虑细节的原则下清晰地描述系统的边界和行为等,从而表达系统的主要功能,开发人员获取的这些用例图和说明又称为用例模型。所以首先对用例模型进行分析,将用例映射为组件的接口,一个用例可以映射为一个接口,也可以映射为若干个接口。然后从用例出发,找出参与该用例的对象,分析该功能的执行流程,确定对象之间的交互过程,这个过程通常可以用顺序图描述,顺序图中的操作最终要映射成接口的操作,因此为提高组件的可靠性着想,这个过程需要不断地求精,待找出全部合理的操作后,再将这些操作映射成接口的成员函数,并用 IDL 描述。IDL 是专门用于描述接口的脚本语言,不依赖于任何开发环境,是组件程序和客户之间的共同语言。

3）COM 组件接口编码

COM 组件是一种基于二进制对象协议的概念。一个 COM 组件对外是一组接口。从 COM 的意义上讲，接口是一种和 vtbl 机制相容的二进制协议，并且 vtbl 的前三项与 IUnknown 接口相容（从继承角度上来讲，可以理解为要求从 IUnknown 继承，但只是这样理解而已）。例如，可以定义如下接口：

```
interface IFoo : IUnknown
{
virtual void __stdcall fooA() = 0;
virtual int __stdcall fooB(int arg1,int arg2) = 0;
};
```

**2. 软件工具酶连接器**

1）软件工具酶连接器及其作用

软件工具酶连接器，也是底物界面，是软件工具酶与软件底物之间联系的特殊机制或特殊部件。它们之间的联系包括：消息和信号的传递、功能和方法的请求或调用、数据的转换和传送、之间特定关系的协调和维持等所有涉及它们之间信息、行为、特性的联系和依赖。软件工具酶连接器承担了实现它们之间信息和行为关联的作用。软件工具酶只有通过连接器才能与底物发生关系，也只有连接器才能对被操作的软件产生作用。最简单的连接器从结构上退化为彼此之间的直接连接方法。较复杂的连接器就需要专门的结构来完成。

2）连接器的类别

根据连接的用途，连接器有标准、通用、专用之分。

根据连接的状态，可分为静态连接和动态连接。

根据连接的复杂性，可分为简单的连接器和复杂的连接器。

3）连接器的特性

连接器的特性反映了其对连接关系的处理性质，体现了对连接器设计的宏观性能要求。它包括连接的关系、角色和方向、交互方式、可扩展性、互操作性、动态连接性、请求响应时间、请求的处理策略、连接代价、连接处理能力、概念等级。

（1）连接的关系。连接的关系分为 1∶1、1∶$n$、$n$∶1、$m$∶$n$，分别指一对一、一对多、多对一、多对多的连接关系。

（2）连接的角色和方向。连接的角色是指参与连接一方的作用或地位，有主动和被动或请求和响应之分。连接的方向是指任何一端口是否可进行双向或仅可进行单向请求传递。连接中的角色也体现连接的方向性。

（3）连接的交互方式。连接的交互方式指请求信息传递的形式，包括信号式、语言式。对于信号式，请求和响应之间按照约定的信号实施处理；对于语言式，则需要建立较复杂的语言（协议）的解释、翻译、转换机制，以完成信息的处理。

（4）连接的可扩展性。连接的可扩展性指操作接口、功能、连接关系的动态可扩展性。所谓“动态”是相对于设计实现时的“静态”而言的。操作接口扩展为动态扩大或改变连接器的处理功能提供了可能。连接关系的扩展允许动态地改变被关联的部件集合和关联性质。

（5）连接的互操作性。连接的互操作性指连接的部件双方通过连接器所建立的关系，直接或间接操作对方信息的能力。例如，被共享的部分是允许双方互操作的。在 DCOM 实

现中,客户可以直接请求服务组件的操作,但服务组件不能直接操作客户。

(6) 连接的动态连接性。连接的动态连接性,指接口所提供的操作,允许根据请求者或接收者或传送数据对象的不同,实施动态确定处理方法的性能。也就是连接行为的动态约束特性。

(7) 连接请求响应特性。连接请求响应特性包括响应的顺序性、同时性、并发性(同一、不同请求的多激发)。简单情况下,请求是根据发生的顺序一个一个地被处理的。在并行/并发环境下,连接器无法知道请求发生的时间,请求的到来可能是近乎同时的,而对每个请求的处理所需的时间和资源会有很大差别。为此,在不破坏处理逻辑的前提下,需要对多个请求具有并行处理的能力。在某些情况下,具体的处理次序是希望可以选择的。

(8) 连接请求的处理策略。连接请求的处理策略,指对请求处理的条件转移,包括对请求的传递、扩展、撤销。传递是当被连接器关联的某个部件无能力完成处理时把请求传递给其他部件。扩展是根据请求的特性将其分解成多个新的请求并发送给其他部件处理。撤销是根据请求的特性有条件地抑制掉。例如,设计模式中的责任链,可视界面对象对消息的处理等,都存在对请求或事件的传递、扩展、撤销处理。

(9) 连接的代价,处理速度或能力。连接的代价,如同一切软件系统一样,需要考察其对资源和时间消耗情况处理计算的复杂性。包括建立连接的代价、处理请求的代价。连接处理速度或能力,是反映连接代价的一方面,以处理请求的单位时间个数或处理信息的单位时间数量来衡量。

(10) 连接器的概念等级或层次。连接器的概念等级或层次是根据建立连接所处理问题的概念层次确定的。

(11) 共享数据的连接器。共享数据很早就被用作程序间的数据传递和交换机制。有共享数据区和共享文件两种形式。共享数据是指那些不具备主动行为的被动数据。具有主动性的共享连接数据是有主动行为的对象,可以作为一般部件处理。被动的共享连接数据作为连接器需要解决共享访问的互斥。

**3. 软件工具酶与底物的连接**

连接,是软件工具酶与底物间建立和维持行为关联和信息传递的途径。实现连接需要两个方面的支持:一是连接得以发生和维持的机制,一是连接能够正确、无二义、无冲突进行的保证。前者是连接实现的基础,后者是连接正确有效进行的信息交换规则,称作连接的协议。因此,连接的本质是连接的两个方面:实现机制和信息交换协议,简称机制和协议。

最简单的连接只有机制的作用,这种连接的信息传送能力是非常低的,因此使用上受到很大限制。复杂的连接是由于实现机制或协议的复杂化而产生的。这种连接的信息传递协议的复杂性却很高。简单的连接可以通过部件直接联系得到完成,复杂的连接则需要专门复杂连接的连接器得以实现。

1) 连接的实现机制

(1) 在硬件层。在硬件层可直接用于连接的机制有过程调用、中断、存储、栈、串行输入/输出、并行输入/输出、DMA。其中,过程调用是实现功能服务和抽象连接的基础;中断是实现硬件和复杂连接不可缺少的机制;存储是实现共享的主要方面和基础;栈是实现高层过程调用的参数传递的形式;DMA 是实现大体积快速共享传输的机制;串行和并行是一切高层输入/输出连接的基础,包括文件和网络连接。该层面还应包括 I/O 端口的硬件

和软件机制。

(2) 在基础控制描述层。在基础控制描述层建立了高一层面的连接机制,它们是过程调用、动态约束、中断/事件、流、文件、网络、责任链。其中,过程调用是包含各类高层参数传递的;在低层次的过程调用上建立的高层抽象(包括同步调用、异步调用);动态约束是实现动态连接和扩充的基础;流和文件概念是形成复杂连接关系的机制;责任链是建立在代码块链接之上的另一种功能动态连接关系;网络则是建立在串行输入/输出之上的远程连接的主要机制。

(3) 在资源及管理调度层。在资源及管理调度层建立了更高层面的连接机制,它们有进程、线程、共享、同步、并行、分时并发、事件、消息、异常、远程调用,以及实现更复杂逻辑连接的注册表、剪贴板、动态连接、应用程序接口。

(4) 在系统结构模式层。随着应用技术的发展,在系统结构模式层建立了面向应用的最高层次的连接机制,它们有管道、解释器、编译器、转换器、浏览器、组件、客户/服务、浏览/服务、OLE、ActiveX、ODBC 等。

2) 连接的协议

协议是连接的规约,是实现有意义连接的保证。协议通常也是按照层次构成的,例如网络的 7 层协议结构。

即使在简单的过程调用中,也有协议在起作用。基础控制描述层的过程调用是对软化硬件层的过程调用的扩充,表现为有类型参数的传送。

在消息连接机制中,不同类型的消息都产生于同一个基础消息类,并具有不同的消息属性。对于消息系统来说,不同类型消息的属性和取值就是消息机制的连接协议,当然,还应该包括消息的传送和处理规则。

3) 连接的特性

连接的特性包括连接的方向性、角色、激发方式、响应特性。

(1) 连接的方向性。连接作为信息传送和控制的渠道具有方向性。控制有主控方和被控方,信息传送有信息的发送方和接收方。然而,复杂的连接都伴随有双向的通信联系。虽然连接是有方向性的,但在一次连接的实现中,通常都伴随有双向的通信。

(2) 连接的角色。角色是对连接的双方所处地位不同的表达。在过程调用中,角色有调用方和被调用方;在客户端/服务器连接中,角色有客户方和服务器方。角色和地位的不同在连接的实施中表现为所进行的操作不同、期望获得的信息不同。

(3) 连接的激发。激发是指引起连接行为的方式,分为主动方和从动方两个方面。主动方的行为激发方式有操作调用和事件触发,从动方的行为激发方式有状态查询、事件触发。连接的发出方式有一对一和一对多,其中,一对多又分为定点和不定点的连接广播方式。

(4) 连接的响应特性。响应特性包括连接的从动方对连接请求的处理实时性、时间、方式、并发处理能力。在基于中断或并行、并发的连接中,多个连接请求同时发生的情况是存在的。连接是否具有处理这个复杂性的能力在许多应用中是十分重要的。

4) 连接的不匹配及其解决方法

连接是使软件工具酶与底物实现互连和协同工作的机制。如果连接发生冲突或不匹配,则会造成它们连接的失败。产生连接冲突或不匹配的原因有多方面,包括连接的实现机

制、协议、特性。一个方面有问题就会造成连接的不匹配。可以通过以下方法解决连接的不匹配问题。

由于在实现机制、数据表示、通信、包装、同步、语法、控制等方面的差异,软件工具酶与底物不能协调工作。下面只考虑软件机制和谐方面的问题,以及解决不匹配的方法。

(1) 全面改变软件工具酶的结构和功能,使其符合底物的要求。为了与底物协调,彻底重新设计实现软件工具酶,这是可行但代价高的处理办法。

(2) 把数据在从软件工具酶传输到底物的过程中,将其形式转换为底物可以接受的形式。在发生协议不匹配时,这是最常采用的解决方法。

(3) 通过协议的"握手"机制,使双方在开始正式传输前,识别对方的协议,以确定双方可以接受的连接行为,达成统一可接收的信息交换形式。

(4) 使底物的连接成为可支持多种实现机制和协议的形式。

(5) 在复杂的情况下,建立专门的标记语言处理协议的不匹配性。标记可以建立在传输的开始,也可以建立在传输的过程中,甚至是任何协议的转换过程中。数据库表的结构信息往往通过标记语言加以描述,这样可以在不同的环境中实现顺利的信息转换。该方法的不足是占用了额外的存储空间,降低了传输效率。

(6) 为底物提供信息输入/输出的转换器。可以使用外界独立工作的部件,或内嵌的转换部件达到功能扩充,以完成内外部数据格式之间的相互转换。

(7) 引入信息交换的中间形式。有两种形式:在它们之外建立信息交换表示,或发布信息交换的通用标准。

(8) 在软件工具酶上添加转换器,使其与底物连接,达到两部分之间的正常交互。这其实是在连接的两方之间增加一个中介部件(连接器),负责完成数据和协议等实现机制的转换,以协调连接双方的行为,消除连接的不匹配。

(9) 把软件工具酶通过代理而包装起来,使其最终的部件模拟要求的连接。包装而形成的新的连接软件工具酶的行为消除了不匹配。

(10) 保持软件工具酶和底物的工作版本始终一致,这样可有效避免产生非预期的不匹配。

5) 与底物的连接方式

软件工具酶与底物连接有四种方式:过程调用、远程过程调用、事件触发、服务连接。过程调用表是实现更复杂过程调用关系的方法。

(1) 过程调用方式。即部件与部件之间通过对方的过程、函数或方法的显式调用实现连接的方法。这是最普通和常用的方法,它是通过硬件 CPU 提供的 CALL 和堆栈机制实现的。为了实现调用,必须知道对方部件的标识和对方部件所对外提供的操作过程标识及其参数设置。

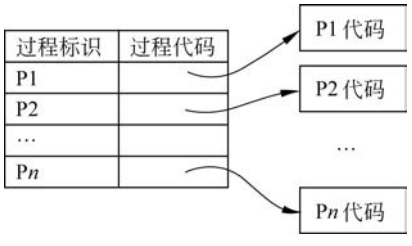


图 5-26　过程调用表方式

(2) 过程调用表方式。过程调用表(见图 5-26)是通过过程表格系统地进行管理过程调用的方法。各过程按照标识排列在一个过程表中,建立起标识和具体执行代码之间的对应。具体过程的调用可以按照过程标识或过程标识在表中位置的号码通过转换进行。按照过程标识调用时,经过标识的搜索确定执

行代码后转入过程的运行。按照过程标识在表中位置的过程号码调用时，直接根据号码确定执行代码后转入过程的运行。

（3）中断部件触发方式。中断部件触发方式（见图 5-27）是通过硬件提供的中断及其控制机制实现部件连接的方法。在该方法中，部件操作的调用是通过中断设置和中断触发实现的。中断设置是将特定的中断号码的中断指针指向待操作的代码入口，并允许接受所关心号码的中断请求。当相应号码的中断发生后，随即在参数参与下执行对应操作。用特定名称标识中断号码，就形成事件触发的部件连接方式。消息传递是建立在此方式之上的更加系统和复杂、更易于用户控制的事件触发机制。

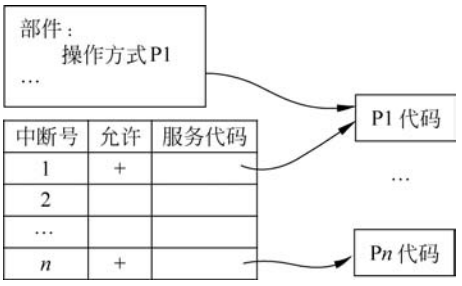

图 5-27　中断部件触发方式

（4）过程链接方式。过程链接（见图 5-28）也称作操作链接或责任链传递结构。这是对具有相同标识，但操作代码各不相同的多个过程或事件操作的链接方法。相同的过程或事件标识标明控制要转到相同的代码中。为了实现不同过程或事件的操作，需要增加额外的控制标识。在过程或事件标识被识别后，控制标识作为参数起到进一步完成控制转向的作用。为了实现链接，通常在过程或事件的主代码中采用条件语句，或开关语句，或过程调用表控制结构，并要求在代码结尾处用特定代码将不同控制标识的操作代码块链接起来，而形成过程链接方式。

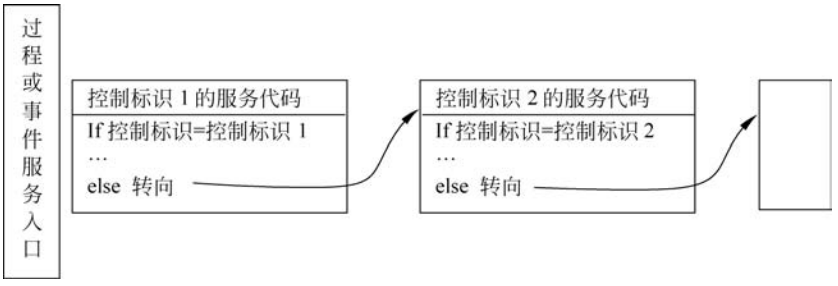

图 5-28　过程链接方式

（5）服务连接方式。服务连接方式（见图 5-29）的服务部件由接口、分析器、执行器构成。其他部件与服务部件进行交互是通过接口进行的。分析器按照指定的句法形式对接口收到的服务请求信息进行分析，确认正确后交执行器完成操作，并将结果返回给请求部件。

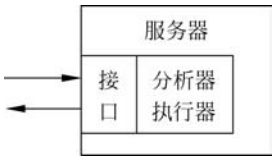

图 5-29　服务连接方式

（6）远程过程调用。远程过程调用（见图 5-30）即 RPC（Remote Procedural Call）。这是网络分布运行状态下的过程调用。为了实现远程过程调用，需要在操作请求端建立一个被请求部件 2 的代理部件。部件 1 的所有请求都发送给代理部件。代理部件通过连接网络建立与部件 2 的联系，把操作请求发送给部件 2。部件 2 处理完操作后，又通过连接网络将结果返回给代理部件，并最终传送给部件 1。

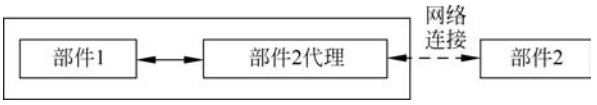


图 5-30　远程过程调用方式

### 5.3.4　专用工具酶,集成及演化

**1. 需求分析工具酶的结构**

需求分析工具酶应用于软件生命周期的第一个阶段,即软件开发的需求分析阶段。它是能够辅助系统分析人员对用户的需求进行提取、整理、分析并最终得到完整而正确的软件需求分析式样,从而满足用户对所构建系统的各种功能、性能需求的辅助手段。

根据以上提出的需求分析工具酶的功能,现抽象简化后归纳为以下几点:用户界面,信息仓库,辅助需求的描述,需求分析说明书生成,见图 5-31。

**2. 设计工具酶的结构**

(1) 结构化的设计工具酶。根据以上提出的需求分析工具酶的功能,现抽象简化后归纳结构化的设计工具酶的功能点为:用户界面,信息仓库,多种方法设计工具,结构图编辑功能,一致性检查功能,设计文档说明书生成,见图 5-32。

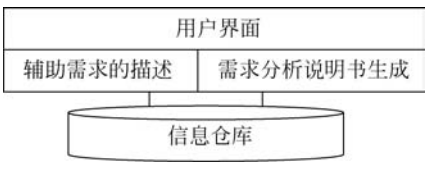


图 5-31　需求分析工具酶结构

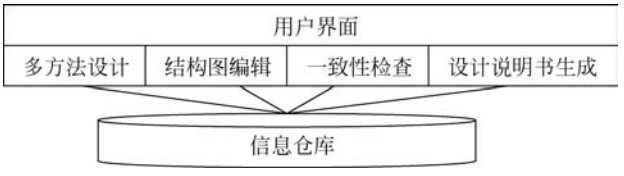


图 5-32　结构化设计工具酶结构

(2) 面向对象的软件工程方法设计工具酶。根据以上提出的需求分析工具的功能,现抽象简化后归纳面向对象的软件工程方法设计工具酶功能点为:用户界面,信息仓库,多种设计方法工具,视图的生成和编辑功能,对象和类定义描述,框架代码生成,设计文档说明书生成,见图 5-33。

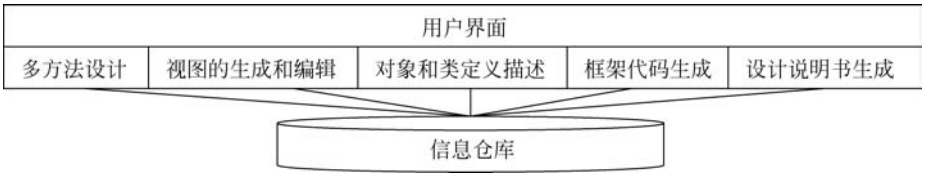


图 5-33　面向对象的设计工具酶结构

**3. 代码生成器**

代码生成器(Code Generator)的基本任务是根据设计要求,自动或者半自动地产生相应的某种语言的程序。图 5-34 是代码生成器工作的基本轮廓。它的输出是程序代码,而输入有三个方面:信息库中存储的有关信息、使用者通过人机界面输入的命令、参数及其他要求和用于生成代码的程序框架及组件。

输出程序代码是这个模块的目标。输出的代码有两种情况:某种高级程序设计语言的

代码或某种机器(包括硬件和操作系统)环境下可运行的机器指令。

生成代码时依据的是三个方面的材料：首先是信息库里已有的资料；其次，代码生成器还要利用各种标准模块的框架和构件；第三是使用者临时通过屏幕前操作送入的信息。

**4. 测试工具酶**

作为软件测试工具酶，一般认为其构成最少包括五方面的要素：用户接口，系统配置管理子系统，软件评价方法编辑子系统，软件评测子系统和评测报告生成子系统，见图 5-35。

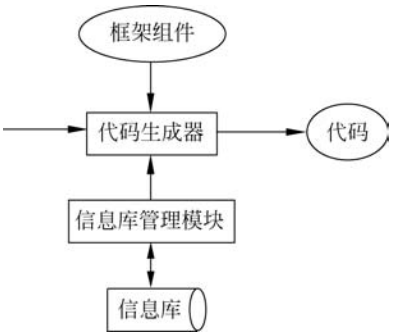


图 5-34　代码生成器工作图

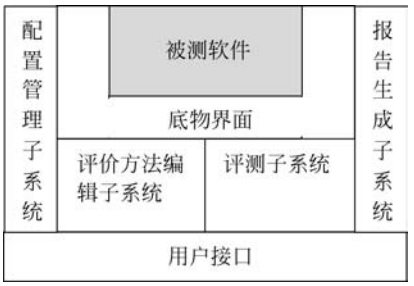


图 5-35　测试酶结构

(1) 用户接口。最好采用基于 GUI 的界面方式，通过统一的交互式用户接口以协调其他各子系统的工作。

(2) 系统配置管理子系统。对评测系统的系统参数进行管理，通过修改这些参数可以重新配置评测系统和对软件进行不同验收标准的测试。

(3) 软件评价方法编辑子系统。可以编辑对语言描述进行评价的方法。该系统可进行词法分析、语法分析和错误检查，从而形成软件评价方法的内部表现形式，并将其存放于软件评测方法库中。

(4) 软件评测子系统。对被评测软件进行分析和测试，通过调用软件评测方法库中的方法和评测标准对被评测软件进行评测。软件评测子系统应该既可采用人机交互的方式进行评测，又可采用自动方式进行评测并将软件评测数据放入评测数据库中。

(5) 评测报告生成子系统。创建、编辑评测报告的模板，放入评测报告模板库中，用户可以根据具体模板生成符合特定需要的测试报告。

**5. 项目管理工具酶**

20 世纪 90 年起，软件工具酶理论中，项目管理占有一定的地位。根据前面的讨论，项目管理的目标有六点：进度控制，费用控制，质量控制，合同管理，信息管理和协调沟通，见图 5-36。

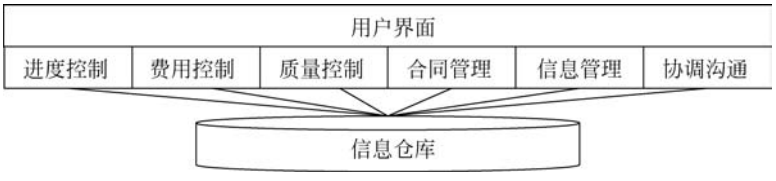


图 5-36　项目管理工具酶

# 思 考 题

**一、名词解释**

1. 软件产品线
2. 网构软件
3. 生物酶
4. 软件工具酶
5. 演化计算
6. 软件基因
7. 软件基因组

**二、简答题**

1. 简单介绍软件工程的发展历程。
2. 简单介绍软件产品线的结构。
3. 软件工具酶有什么作用？其作用机制是什么？
4. 软件工具酶有哪些催化特点？
5. 简单介绍软件转换法则。

**三、分析题**

1. 详细分析软件工具酶与底物结合的两种模式。
2. 对未来软件的开发模式做简单的分析。
3. 分析国内最新网构软件的发展与研究。