支持原型开发方法的计算机环境与工具

王晓峰　（中国科学院计算中心）
陈志坚　（机电部经济信息中心）

原型化开发方法是一种新型的软件开发手段，但是需要一定的开发环境和工具来支持。本文在分析、比较传统软件开发方法（即软件生存期法）与原型法的特点的基础上，着重阐述了利用计算机辅助原型开发的支撑环境的结构及其工具的作用。同时，还分析和描述了在整个系统开发进程中，原型系统的演变情况。

七十年代初出现"软件危机"后，人们愈来愈重视软件开发方法的研究。为编出高效软件提出过各种各样的设计技巧和方法，如结构化分析法、Jackson方法等；为了使软件开发方法工程化，又提出了软件工程的概念及应运而生的各种开发工具。这些情况促使软件开发方法走向两个方面：一方面是着重研究与机器本身相关的软件开发工具，即高级软件语言及软件开发环境；另一方面是研究软件的设计和规格说明等，即软件的生存期。但是以上两方面没有一个能单独地对软件开发过程的效率和开销带来重大的改善。然而，如果综合运用以上两方面内容，在获取一组基本的需求定义后，利用高级软件工具和开发环境，快速地建立起一个目标系统的最初版本，并把它提交给用户试用、补充和修改，再进行新的版本开发；反复进行这个过程直到得出系统的"精确解"，即用户满意为止。经过这样一个反复补充和修改的过程，应用系统的"最初版本"就逐步演变为系统的"最终版本"，这就是软件原型法的基本思想。本文在比较传统生存期法和原型法的基础上，着重阐述计算机辅助原型开发的支撑环境及其工具的结构和作用，同时分析和描述了系统开发过程中的原型演变。

一、传统生存期法的缺陷与原型法的优势

现今大多数大型软件均采用软件生存期法来开发，这是一种按阶段逐步提炼的软件设计模式。生存期法把系统开发过程分为六个阶段：需求分析、规格说明、系统设计、编码、测试、运行及维护。这种方法强调阶段完整性和开发的顺序性。它要求系统的开发者和用户在系统开发的最初两个阶段中就确定系统的完整需求和全部功能，因为生存期法总是反复提醒用户和开发者注意，每项事后的修改或补充都是对系统范围的一次扩充，因而可能要增加开发费用或推迟开发进度。然而，由于没有足够好的方法来检验每一份规格说明是否完整，所以产生遗漏或不一致是难免的。

阶段完整性基于这样两个假设：在需求分析和规格说明阶段中，系统的使用者能够清楚地、完整地提供有关系统的需求；同时，系统的开发者能完整地、严格地理解和定义这些需求。在这些假设的支持下，生存期法自然地形成了严格的开发顺序。每一后继阶段的开始均依赖于前一阶段产生的完整定义

和描述等结果。在需求分析阶段，开发者与用户之间以问答方式进行工作，以期能在系统开发的初期完成对新系统的完整需求定义。

可惜，大多数系统的需求事先难以说清，更谈不上完整定义。一方面，用户心目中的"需求"只是一个模糊概念；即使在开发者的再三启发下，用户也只能把有关系统的功能等一些模糊概念加以重新阐述，使之成为具体细节。然而，问题是我们所面临的系统本身总处在不断的变化之中。强迫一个只有初步想法的人对需求做出完整的、精确的说明是不现实的。另一方面，系统开发者只起着询问者、顾问及问题的解决者的作用，他们不可能很清楚地了解现行系统的具体业务。因此在大量与用户交换意见的过程中，传错信息和发生误解的可能性极大。这样，按生存期法的要求，在最终产生的需求分析报告和系统规格说明书中的许多定义和描述只能是被迫或勉强做出的。尽管这些报告和说明书要经历反复的评审与综合，但因为它们只是一叠厚厚的"纸系统"，所以要从中发现隐患是很困难的。这样在设计初期埋下的陷井，只有到了测试和运行阶段才有可能被发现，这无疑加重了维护阶段的负担。在生存期法中就存在这样的统计数字：开发阶段的代价约为30%，而维护阶段的代价却为70%。最坏的情况下，不是对新系统进行一般的维护和修改，而是干脆推倒重来。尽管生存期法过去也一直强调需求分析是一个不断地启发、揭示和判断过程，然而问题是我们应如何进行这个过程。要知道人们不善于描述不存在的东西，却善于评价已存在的事物。

原型法正是一个不断地运用系统的"原型"来进行启发、揭示和判断的系统开发方法。所谓"原型"是一个具体的可执行模型，它实现了目标系统的若干功能。原型开发是一种强调用户参与的系统开发的过程。在此过程中，按初步的需求调查和分析结果，迅速地开发一个缩小了规模的系统原型并使之以增量方式演变为全规模的系统模型，最终产生一个使用户满意的系统产品。

下图给出一个原型法的开发周期：

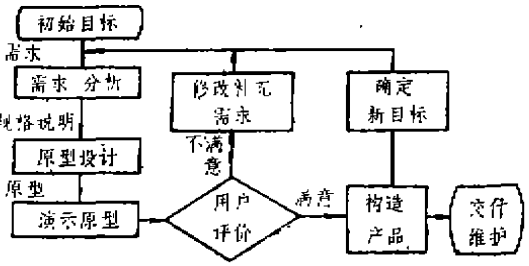

图1

在这个周期的前两个阶段中，开发者和用户一起为想象中的系统的某些主要部分定义需求和规格说明，并由开发者在规格说明级用原型描述语言构造一个系统原型，它代表了部分系统，包括那些为满足用户需求的必要属性。该原型可用来帮助作分析和设计工作而不是一个软件产品。

在演示原型期间，用户可以根据他所期望的系统行为来评价原型的实际行为。如果原型不能满意地运行，用户能立刻找出问题和不可接受的地方并与开发者一起重新定义需求。该过程一直持续到用户认为该原型能成功地体现想象中的系统的主要部分功能为止。在这期间用户和开发者都不要为程序算法的错误或设计技巧等枝节问题分心，而是要确定开发者是否理解了用户的意思，同时试验实现它们的若干方法。

开发者把修改后的需求作为设计产品系统的基础，而要完成一个产品的最后版本还要作一些附加和配套工作。因为原型并不完善，而且运行中使用的资源可能并不适合实际的运行环境，另外可能还存在系统性能等方面问题。

有了满意的系统原型同时也积累了使用原型的经验，用户常会提出新目标从而进一步重复原型周期。新目标的范围要比修改或补充不满意的原型大得多。我们的目的就是要在这种增量式的系统开发进程中，在每交付一个版本之前尽量减少需求错误。

原型法的增量式交付方式使用户能尽早地获得在实际产品环境中使用该系统的经验，也使得开发者能调整需求分析以使最初版本的系统能反映用户对自身问题的理解。这样增量式的交付方式就把原型法的优势带到了产品系统的环境中。

二、 原型法的支撑环境和工具

实际上，生存期法早就有"原型"的思想了，只是缺乏可以实现的工具和有效的开发环境而作罢。例如，生存期法中系统需求分析结束时，要求有一份初步的用户手册。这种手册就是一个系统"原型"。当然，"手册"只是一个极不直观的、不可执行的"死"原型。使用手工方式或传统的生产方式来构造原型，对于简单的、小规模的以及有明确需求定义的系统是可能的。但对于复杂而庞大的、需求不确定的系统则很难实现，而且也体现不了原型策略的优越性。因为没有一个能给用户展示未来系统的一些活动的可执行模型是很难确定系统的可行性及需求的完整性的。但是，花费过多的时间去生成大量的原型编码而又无法保证它们的可重用性是任何开发者（包括用户）所不能接受的，而这个问题正是传统的设计和编码环境中不可避免的。

由此看来，用原型法开发系统的问题实际上就转变为能否提供一个合适的软件开发环境以充分体现原型法的优越性的问题。这个软件开发环境必需具有以下一些特点：

- 高度集成化的设计语言和工具，而且其运行效率不应随集成化的提高而下降；
- 高效率的数据与软件的管理设施，以极大的可能性提供数据的共享和软件的可重用性；
- 高效率的调试、演示和运行环境。

本节重点讨论支持快速开发系统原型的环境以及有关的工具。一个支持快速原型构造及其演变过程的环境至少应具有如下的结构：

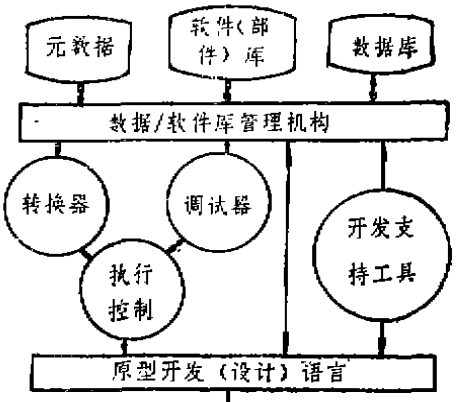

图2 原型开发环境

下面先简单谈谈原型设计语言。

原型设计语言是开发者开发原型的主要工具。这种语言是非常高级的，而且是基于表达数据结构的，如集合、表等。用该语言写出的程序比完成同样功能的、用普通高级语言写的程序短得多。同时该语言应是高度参数化的。一方面只要调整某些参数就可以构造出一个应用模型；另一方面语句本身也具有可截剪性和缺省性。诚然，这种高度集成化的设计语言对目标程序的运行效率会有一定的影响，但决不致于降低使其作为原型开发工具的价值。现把原型设计语言应具有的特点归纳如下：

·支持高度模块化：原型设计语言能方便地产生带有高度模块化的、独立的原型部件，并且在经过多次修改后其模块化结构不会被破坏。这种模块化结构既有利于开发和修改，又利于装配和联接。原型设计语言的主要目标就是把信息局部化、把功能局部化；使模块间的一切相互活动都明确化并统一在数据流和控制流中加以考虑。任何两个不同的操作模块间只能通过明确的、直接的或间接的联接它们的数据流来相互通讯并影响对方的行为。

·支持软件的可重用性：原型语言为检索软件库中的模块制定了统一的规范以保证开发者能方便地从软件库中检索可用的原型部件，从而支持部件的可重用性。一个用原型语言写的部件规格说明包含描述该部件的接口和行为特征。这些特征有助于用自动化的方式来产生统一的可重用部件的规格说明，这些统一的规格说明即可用来检索可重用部

件，又可用来组织软件库。

·支持对需求的跟踪：

该语言通过一个阐明与原型各部分有关的需求结构来支持要求跟踪。这点很重要，因为原型必须适合由于演示原型而带来的需求概念的修改。每项需求与实现该需求的原型的某些部分间的链接决定了当变动需求或取消需求时应该修改原型的那些部分，而且要不断地把需求和对应的编码记录下来，这样可为维护和使用跟踪信息提供自动化的帮助。

现在再来看一下处在这样一个开发环境中开发者与用户是如何工作的。

首先，开发者利用原型设计语言按用户提出的初步需求，设计和定义系统的各原型部件，如用户接口屏幕、初步的数据输入和输出格式、执行逻辑等。这时主要使用的各种辅助和支持工具有：编辑器、语法分析器、调试器、浏览器等。

其次，系统将所有原型部件（屏幕定义、数据集定义、表格定义及处理逻辑模块等）全部存入数据库和软件库中，以备进一步开发和调试用。建立数据库和软件库的主要目的是支持部件的可重用性，即为某项应用而建立的各种定义（如数据定义、程序定义等）均可被其他开发者（包括自身在内）在以后的开发阶段或其它应用中重复引用。

第三，按定义及有关约束来测试原型。测试工具应由执行控制机构加以管理，因为测试应包括两个方面内容：一是原型的静态特征，如每个屏面在实际终端上的显示样子、数据格式与样本数据匹配与否、程序部件的编译和调释等；二是原型的动态特征，如在模拟实际运行环境时，程序部件的执行时间特征，多部件间的转换、多任务间的协作等等。本工作是用户与开发者共同参与进行的，往往要不断地分析调试结果并取得进一步修改或扩充。

最后，将经过反复修改和补充的、满意的原型部件通过转换器转换并装配成可以高效执行的原型产品。通过执行控制机构就可以进行整个原型产品的演示从而获得对整个原型的评价。

这里要强调的是，快速原型开发既是指各系统成份的原型（即原型部件）的开发，也是指整个系统的原型（即原型产品）的开发。由转换器产生的原型产品完全可以独立于该环境而直接在某个操作系统支持下运行。我们把提供以上这种开发环境的软件系统称为计算机辅助原型开发系统。下面将从该系统的内部结构（见图3）来进一步阐述它的各功能子系统的作用。

1.开发支持子系统 开发支持子系统应提供以下功能：输入有关需求和设计信息、为开发者提供执行原型的结果、指导产生和选择显示原型的指定部件、帮助开发者分析由修改和补充带来的影响。该子系统的组成有：带图形功能的语法指导编辑器、联机帮助生成器、浏览器和调试器。

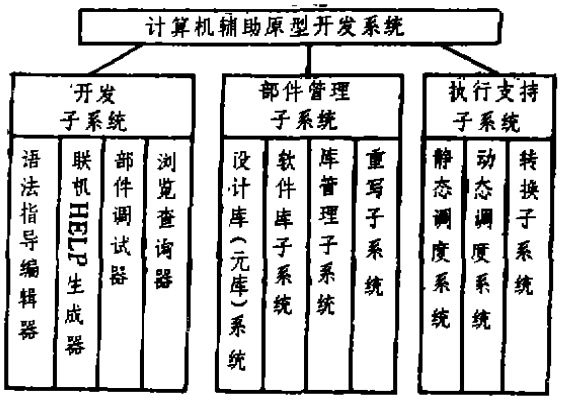

图3

编辑器是开发者创建原型部件的基本工具。为防止出错，该编辑器提供了原型设计语言的语法或句法指导功能。它支持以图形方式显示原型的概貌，并且能按需求踪迹来提供相关原型部件，即自动提供对正在编辑的部件的前一或后一部件（如果存在）的编辑。这样能让开发者在开发或编辑原型的某个部件时能容易地获得其它相关部件的信息（描述及定义）。

联机帮助(HELP)生成器的目的是在开发某个原型部件时，辅助开发者同时产生有关这个原型部件的HELP信息。它往往作为这个部件的附件而存在。最理想情况是一旦原型设计结束，开发者就

能直接将所有的附件汇总而形成一本使用手册。

浏览器帮助开发者与数据库及软件库打交道。帮助他们检索和检验存在于软件库中的可重用部件的功能，这样不必要求每个开发者都了解库结构，只要提出浏览要求即可。

调试器允许开发者与执行支持系统打交道，对原型部件进行调试、显示结果或跟踪信息并收集有关原型行为和性能的统计参数。其目的是帮助开发者尽快地检验正在进行的工作。

开发子系统还可以帮助原型设计小组确定修改原型所要作的工作。它保持需求与原型部件间的通讯。修改原型时伴随有一系列未解决的新需求和修改过的需求表。每当一个小组成员准备接受新任务时，系统就出示这些表，让他解决其中某项问题。如果他选择一个被修改过的需求，那么系统就把支持以前的需求的一系列模块交给他，让他审查这些模块是否仍然可用或是否需要修改。总之，为加速开发和修改原型过程，该子系统帮助协调开发者执行各项任务，并使其最大可能地将注意力集中到与总体任务有关的信息上。

2.部件管理子系统 数据库是现代管理软件的核心，也是系统开发环境的核心。数据库在整个原型开发期内保存与该项原型有关的所有信息和逻辑部件。为了区别传统概念中的数据库，在这里使用部件库。部件库包括系统开发时涉及的元数据库、基本数据库（模拟或实验数据等）及软件库。该子系统的目的是让各类部件库能同时支持多个项目和任务的开发。它由设计库、软件库、库管理以及支持重写功能等组成。

设计库中包括有关软件项目的原型语言描述、各种文档和原型部件的定义等。它还提供并行控制功能以允许多个设计者修改同一原型的某些部件而又不互相干扰，这可以用有关部件的状态信息来控制完成。软件库中包含有所有可重用的软件部件的原型描述和编码，以便为给定的原型语言描述提供可重用软部件。同时，也为加速原型演变提供大量不同版本的公用部件以用于试验所选择的设计。

库管理子系统的功能是管理并检索版本信息、可供选择的原型以及软件库中的可重用部件等。另外它还帮助维护设计历史并确定相应的可重用部件。设计历史记录了需求的各版本与原型的某些部分的相应版本之间的关系，以备用户要求返回到原来的需求版本上。有时因为用户顾虑开销或原型测试做出的性能估计而放弃某项需求。这时，需求的某些部分会倒退到原来的版本上。该系统应帮助把原型的某些部分恢复到它们以前的版本上。

重写子系统的任务是把原型设计语言描述翻译成规范形式以便库管理子系统能利用它来检索软件库中的可重用部件。

在计算机辅助原型开发中根据原型设计语言的描述来产生部件模块的工作主要由以下三个步骤来完成：

1)．从软件库中检索适用的可重用部件。软件库中的部件大部分为带参数的原始模块，把这些参数作为检索的一部分，再加上相应的元数据库中还有一些相匹配的规格说明规则。这要借助于由操作体网络（类似于数据转换流程图）组成的一个集成操作体来实现。这些操作体中至少有一个是可用的可重用部件。这样，检索机制就能完成那些自下而上产生的子例程，从而保证开发者在不了解软件库结构的情况下也能使用可重用部件。

2)．把部件分解成由更简单的部件组成的低层网络。当不能直接从软件库中检索到所要求的部件或部件太复杂以致不得不把它分解为更简单的部件时，就由开发者自己来完成这项分解工作。

3)．直接用程序设计语言来产生。如果软件库中没有能按指定速度完成所要求功能的部件时，就由开发者按各类需求的规格说明使用能满足要求的程序语言来作这件事。

在这里必须考虑到软件库中会有大量的部件的事实。因此，检索这样的数据库一般要掺进有限程度的逻辑推理以确定匹配某个被查询的部件是否可以从有限数量的可重用部件中构造出来。之所以加上“有限”是为了保证检索不会无限地进行下去。根据与部件库有关的规则，最好能有一个规则库和推理机来完成这样的逻辑推理。当然，我们这里暂不讨论这类问题。

3．执行支持系统 原型执行支持系统包括转换子系统、静态调度子系统和动态调

度子系统。

转换子系统负责把从软件库中提取出来的可重用部件装配起来并生成代码以利于高效运行。它的一个主要功能是在这些可重用部件之间实现数据和控制的传输及约束。

静态调度器的功能是在执行开始前为具有实时约束的操作体分配时间片。如果分配成功，就要保证所有的操作体都满足给定的执行时间限制。随着执行的推进，动态调度器利用带实时约束的操作体不用的时间片来调用不带实时约束的操作体。如果静态调度器没能找到一个有效的时间分配方案，那么，它应该确定分配困难的原因并提供用于判断是否增加更多的处理器就能解决困难的诊断信息。这些功能是很重要的，因为系统中的时间约束可能已把交互作用复杂化了，很难用人工方法来分析它们。

目前还没有较完善的计算机辅助开发系统原型的环境和工具，但是却有一些具有相当规模的开发支持系统，如IBM公司80年代推出的CSP (Cross System Product) 就是其中颇有代表性的软件开发系统。CSP一改传统的软件开发方法，从较高层次上支持应用系统的定义、测试、生成及维护，并初步实现了软件可重用性以及与数据库的接口等问题。CSP直接支持数据库的管理设施，而且支持与多种高级程序设计语言的联接。从原型开发方法的需求看，CSP作为一种高效率的应用生成器完全可以成为原型法的有效工具用于大型应用系统的开发中。图4是CSP的逻辑结构，详细描述请见文献〔5〕。

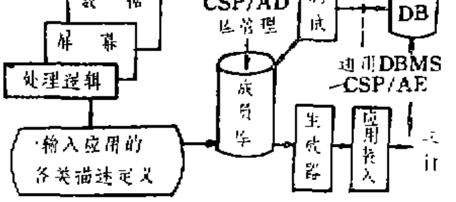

图4 CSRP的逻辑结构图

综上所述，由计算机辅助开发原型工具支持的原型法可以减少为软件系统的演变所付出的代价。原型法既能稳定需求又不冻结需求。这里的需求既是新系统的需求，也是对已有系统的改造所提出的需求。从演示系统的行为中反馈回来的信息对于完善和修改需求是基本的。同时，用户和开发者必须不断地检验对系统行为作出的一系列修改以及为了达到一种共同的理解而构想的全部需求。

如果用人工方式实现原型，那么原型法的效率是有限的。一套系统化的原型方法加上一整套计算机辅助原型开发工具对于表现原型的潜在优势来说是很重要的。简便而迅速是原型法的主要目标。从长远来看，针对某一应用领域（特别是对同一问题）要求开发出多个软件系统的情况，如果我们能为它们建立一个全面的部件库，那么以后的开发工作会省力得多。任何实际的计算机辅助原型化环境的一个重要的基本成份就是设计部件库。该库保存有原型的层次结构及在原型化期间开发的多版本间的关系和实体。当然还需要更好的自动化方法和工具以协调开发者之间的工作，并且能把不同开发者设计出的有关部件结合起来，同时能找出潜在的冲突。这样既有助于解决问题又能快速裁剪原型以满足用户的需求。

最后强调一点，原型法并不是一种全新的思想，它是基于生存期法的基本思想并针对生存期法的弱点而产生的。原型法只有在计算机辅助开发工具和环境的支持下才能发挥出它的巨大优越性和高效率。我们不应该有了原型法后就完全抛弃生存期法的基本思想，而是要把它们融于原型法中，因为严格的需求定义和完整的规格说明对一个软件产品来讲仍然是很重要的。

参考文献

〔1〕Maria H.Penedo, "Prototyping a Project Master Database For Software Engineering Environments", ACM SIGPLAN NOTICE Vol. 22, No. 1, 1987.

〔2〕Ian A.Gilhooley, "Productive Systems Derelopment with Prototyping", Journal of Information System Management, 1987.

〔3〕M. Ketabchi and Luqi, "A Computer-Aided Prototyping System", IEEE Software, Vol. 5, No.2, 1988.

〔4〕Luqi, "Software Evolution Through Rapid Prototyping", Computer May 1988

〔5〕CSP/AD and CSP/AE General Information, IBM Corp.

〔6〕Patrick A. V. Hall, "Software Compon-

神经网络计算机系统及其现状

蔡义发 （浙江大学生物医学工程研究所）

摘要

In 1980's,studies of neural computer systems and related theories have caught world-wide attention. Scientists of computer science, cognitive science,psychology and some other disciplines are studying them from different aspects.This paper discusses the concepts of neural networks first. Then, fourteen neural network models are outlined. Two implementation strategies of neural computers and the main differences between neural computers and traditional ones have also been studied. The summarization of reported artificial neural systems and their problems end the paper.

人工神经网络(Artificial Neural Networks—ANS)的研究已超过四十年，其兴衰与人工智能的研究紧密相关。自从1982年Hopfield利用神经网络解决了TSP问题以来，神经网络再次成为一个研究热点。这方面的研究已取得了许多成果[1][2][8]。由于传统的数字计算机在模式识别、符号推理、组合优化问题等方面的严重缺陷，八十年代后期人们便又重新认识到了联结机制(connectionism)的价值，并在第五代计算机、神经科学、语言学、认知科学等学科的研究结果基础上，开始了以神经网络计算机为核心的第六代计算机的研究[4]。该研究将会在人类尖端科研方面产生重大突破，并使人类社会逐步过渡到超工业化。

1 神经网络的定义及模型

1.1 神经网络的定义

ANS是以大脑的神经网络结构为基础的。形式上说，ANS是一个以有向图为拓扑结构的动态系统。它通过对连续的和间断的输入（刺激）作出状态反馈而完成信息处理工作。网络中的节点称作处理单元(PE)。每个PE中的处理是由传递函数定义的，且依赖于输入信号的当前值和局部存储器中的值。网络中的PE或进行连续操作，或依赖一定序函数 (scheduling function) 来变更。典型的神经网络中，PE被划分成不同的层次。每一层上的PE具有相同的传递函数，虽然

ents and Reuse-getting more out of Your Code", IEEE Software Vol. 5, No. 2, 1988

[7] 罗晓沛，"信息系统开发的原型化方法"，研究生院，1987.11.

[8] D. C. INCE, "Software Prototyping-Progress and Prospects", Information and Software Technology Vol.29, No. 1, 1987.

[9] Rogers. Pressman, "Software Engineering —A Practitioner's Approach", McGraw-Hill Book Company 1982.