计算机科学 1999Vol. 26№. 5

# 再工程——概念及框架

Reengineering: Concepts and Framework

郭 耀 袁望洪 陈向葵 周 欣
(北京大学计算机科学技术系 北京 100871)

**Abstract** How to evolve legacy systems into new system is becoming one of the focuses of current software engineering researches. It's under this situation that reengineering becomes more and more important. Reengineering offers a practical and feasible approach to transform legacy systems into evolvable systems. The application of the disciplined reengineering methods and practices will facilitate to effectively reuse legacy systems, enhance their evolvability, and meet the requirement of the new software life cycle model. This paper summarizes the theory and methodology of reengineering, and then probes into the main problems which reengineering is facing and the solutions to them.

**Keywords** Reengineering, Legacy system, Reverse engineering, Program understanding, Software reengineering

进入九十年代以来,软件系统的规模变得越来越大,结构也越来越复杂,与此同时,软件的生存期也越来越长。过去的软件生存周期模型认为软件在开发完成之后,经过若干年的维护过程,会慢慢退出历史舞台,被新的软件系统所替代。然而,当前的生存周期模型趋于把系统看作能够随时间而进行连续演化的模型[16]。

同时,我们还可以看到,从头开始建造的大系统数量在急剧减少,很多遗产系统正在被逐步地利用起来,但利用遗产系统的同时,会遇到许多困难。由于时间的流逝,这些生存期已有10—25年的系统具有许多这样或那样的缺点。而其中最重要的一个问题就是:现有的遗产系统没有好的可演化性,这样就使得我们无法利用简单的方法把遗产系统转化为一个新的系统。

正是在这种情形下,再工程变得越来越重要,因为它提供了一条把遗产系统转化为可演化系统的现实可行的途径。通过再工程提供的一整套严格定义的方法和活动,可以有效地使遗产系统得到再利用,提高其可演化性,适应新的软件生存周期模型的需求。

由于我国计算机发展水平和计算机应用范围的限制,国内现有的可供利用的大型遗产系统并不很多。但是我们相信,随着我国信息产业的飞速发展,随着计算机应用的全面普及,在不久的将来,必定会有大量的遗产系统出现。而到时候,我们也一定会面临如何利用遗产系统开发新的软件系统的问题。因此,有必要研究再工程的理论和方法,为在将来能够更好地利用遗产系统提供理论和方法上的支持。

本文将在总结国际上现有的再工程理论和方法的基础上,探讨再工程面临的主要问题和解决方法。

## 一、基本概念

### 1.1 再工程的定义

在不同的文献中,对于再工程的概念有着许多不同的定义,主要有以下几种:

- 通过对现有系统的审查和改造将其重组为一种新的形式,以及这种新形式的实现过程[8];
- 不改变一个系统的功能,只改变它的环境或技术而对其进行改造的过程;
- 对现有系统的修改或可能的进一步开发;
- 通过逆向工程(和重构)以及之后的正向工程来改进一个系统的过程;
- 修改一个系统或程序的内部机制或数据结构而不改变其功能的过程[10]。

这些概念事实上都是从不同的侧面对再工程的

理解。它们都没有能够涵盖再工程的所有范围，给出一个全面的定义。下面将通过对有关再工程的一系列概念之间的关系的介绍来给出一个较为完整的再工程的定义。

E. Chikofsky 曾经对和再工程有关的术语之间的关系做过一番论述，图 1 就表明了这种关系，在这个广泛为人们所接受的分类方法中，软件系统的抽象(Abstraction)根据生存周期的不同阶段分为三个层次：需求、设计、实现。通过生成这些抽象来开发软件的传统方法就被称为正向工程。逆向工程则是指对现有系统进行分析，识别系统构件、抽象、及其相关关系，生成相应表示的过程。逆向工程主要可分为再文档和设计恢复两种形式。再文档是指同层抽象的表示和修正，而设计恢复是指使用包括领域知识在内的外部信息来获得关于现有系统的有意义的高层抽象。再工程的第三个过程是重构，它是指在保留系统的外部行为的前提下，对系统在同层抽象中的表示进行转换。从而可以得出再工程的定义：

*再工程是一个工程过程，它将逆向工程、重构和正向工程组合起来，将现存系统重新构造为新的形式。*[9]

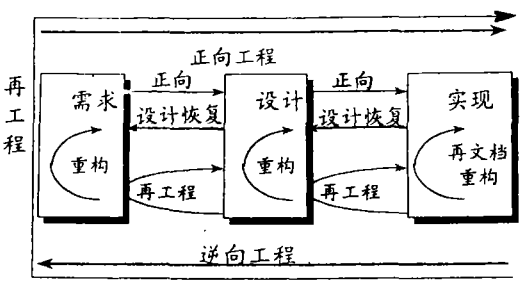


图 1　再工程的定义

另外，再工程和维护之间也存在着密切的关系，根据 ANSI/IEEE Std 729-1983，软件维护是指“在交货之后对软件产品的修改，以修正错误、提高性能和其它属性，以及使产品适应变化了的环境”。下面讨论再工程与维护及新开发之间的关系。

### 1.2　再工程与维护及新开发的关系

“如何区分再工程和开发?”，“如何区分再工程和维护?”这是人们在认识再工程的过程中最经常提出的问题，而正确地理解再工程的范围也是开发再工程的活动框架的基础。

再工程事实上可以以更少的开销、更短的时间、更低的风险把遗产系统改造为一个新的形式，从而在操作、系统能力、功能、性能、或可维护性和可支持性上得到提高。再工程的重点是以更高的投资收益率，来提高遗产系统的性能。如果再工程没有更少的开销，不能在更短的时间内完成，没有更低的风险，或者不能为顾客提供更好的质量，那么就应当考虑进行一次新的开发。

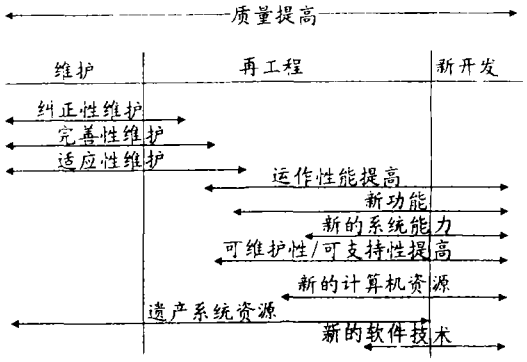


图 2　再工程与维护及新开发的关系

如果把再工程放在维护和新开发的连续统(Continuum)上(图 2)，就可以清晰地看出再工程所处的位置了[5]。维护承担对软件进行纠正性、完善性、适应性的改进，而开发则侧重于实现新的能力，添加新的功能，或使用新的计算机资源和新的软件技术来进行实质性的改进。而正如图中所示，再工程填充了二者之间的缝隙，又同时表现出二者的特性。但是由于这些活动事实上不可能有明确定义的界限，所以它们之间也存在着相当程度的重叠。

### 1.3　再工程与软件再工程

再工程经常会被人们错误地理解为软件再工程的同义词，然而，再工程的真正范围却比软件再工程要宽得多。在再工程中，运行中的系统只是整个遗产系统的一部分。虽然(遗产系统的)软件是再工程过程中要考虑的最重要的一部分，但它本身仍然只是我们要考虑的完整的系统的一个元素而已。系统的任何元素都有可能成为影响再工程的一个重要因素。

在这里，引用 R. S. Arnold 给出的软件再工程的定义：软件再工程是任何可以改进人们对软件的理解和/或改进软件本身的活动[2]。在这个定义中，对“软件”的解释是很广的：它包括源代码、设计记录、和其它文档资源。但是再工程所涉及的范围仍然不止这些，下一节将讨论再工程涉及到的其它因素。

### 1.4　再工程所涉及的因素

由于再工程涉及的话题很宽，许多有关再工程的问题在这里无法谈到。本文主要侧重于讨论再工

程的技术因素，然而，经济的和管理的因素都将对能否成功而有效地对遗产系统进行再工程起到很重要的作用。

通过再工程，一定要能够做到大幅度地降低利用遗产系统开发新系统的开销。前面提到，再工程的重点是以更高的投资收益率，来提高遗产系统的性能。如果再工程不能做到这一点，那么就没有进行再工程的必要了。关于进行再工程决策时可以使用的开销模型可以参见文[11]，[4]。这些开销模型，再加上对不同工程技术的有效性的更好理解，就可以帮助软件工程师们在选择再工程的演化策略时，做出合理的权衡。

再工程的有效性很大程度上取决于一个组织是否有能力管理好它的工程活动及提高其工程能力。卡内基·梅隆大学(CMU)的软件工程研究所(SEI)在这方面进行了大量的研究工作，而且一直处于领先的地位。他们提出的 CMM 模型及相关的工具都对如何组织软件过程，提高软件质量做出了很好的尝试，并在相当大的范围内得到了应用。关于如何能更加有效地改进再工程实践这方面的内容可以参见 SEI 以及其它的相关研究机构的工作[14]，[12]。

关于遗产系统，不可避免地还会引起其它问题，比如如何获得遗产系统，此外还有一些相关的法律问题，等等。这些虽然都不在本文讨论范围之列，但是当要真正进行一次再工程的实践时，它们都是要认真考虑的问题。

## 二、一个框架

一个再工程的框架给出了进行一次再工程实践的全过程中所应当考虑的活动，它对人们如何对遗产系统进行再工程开发具有指导性意义。当然，一个这样的框架所给出的并不是一个需要严格遵守的活动规范，它所起的作用只是提醒人们在进行再工程的过程中需要注意哪些问题，而且针对不同的具体系统，可能需要进行相应的修改才能适应具体问题的需求。

图 3 给出了一个再工程的基本框架。一个再工程的过程主要由决策分析、系统理解、和系统演化三部分组成。决策分析通过对遗产系统和系统目标的分析，决定是否要进行再工程，同时确定再工程的基本策略；系统理解是再工程的关键活动，通过对遗产系统的理解，得到有关它的详尽信息，对遗产系统到未来系统的演化来说是至关重要的一步；再工程的最后阶段是进行系统演化，得到未来系统，完成再工程。

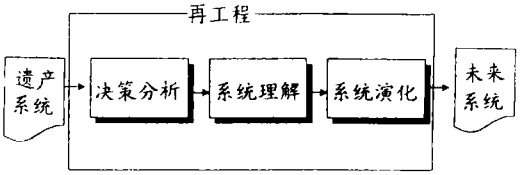


图 3　再工程的基本框架

### 2.1　决策分析

决策分析是进行再工程活动的第一个步骤。其中要进行的活动包括：确定再工程活动的范围和方向；建立反映未来系统的质量提高的初始系统需求；还包括对遗产系统的详细审查和分析；等等。在进行了这些初始的准备活动之后，就可以初步得出再工程活动的候选策略。

接着，需对再工程候选策略进行风险分析，在风险分析的基础上进行再工程的决策，决策的目的是决定是否要进行再工程。如果发现系统只需进行简单的质量提高即可，那么也许进行一次维护就足够了；如果发现有许多新的系统需求出现，那么可能就

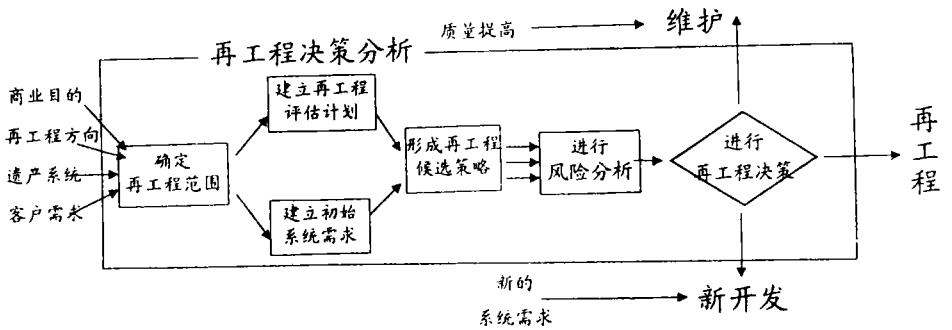


图 4　再工程决策分析

要考虑进行一次新的开发了。决策的过程其实是复杂的，有多方面的因素需要考虑，不仅对于具体的系统有着不同的方法，同时也依赖于决策者对系统的理解和个人的经验。最后，如果是决定进行再工程的话，那么决策还应当包括确定再工程的基本策略和方法。

这些活动之间的关系可以通过图 4 来表示出来[5]。

在完成了再工程决策过程之后，可得到如下结果：

- 对系统进行再工程的真正的范围和程度；
- 清晰的再工程方向和目标；
- 对遗产系统的量化数据描述；
- 对遗产系统进行再工程的可行性与现实性分析结果。
- 确定的再工程策略和实现技术。

### 2.2 系统理解

在决策分析得到结果之后，就要开始对遗产系统进行系统理解。系统理解的作用是通过对遗产系统的源代码、设计记录以及其它文档资源的分析，得到对系统的全面而详细的信息，为从遗产系统到未来系统的转化提供坚实的基础。

系统理解基于以下三个方面的内容：

- 现有系统本身；
- 可能存在于系统用户、维护人员、和最初开发人员头脑中的与系统有关的经验；
- 以错误报告和修改记录的形式而存在的系统历史。

图 5 给出了可以用来生成系统理解潜在的信息源[5]。系统本身包括源代码、手册和运行中的系统。有关系统的知识和经验包括对系统决策、基本原理、可以采用的其它替代手段的理解，同时也包括对没有文档记录的历史和非功能性的属性的理解，例如：系统性能、健壮性等等。系统历史通过错误报告和修改记录，能够提供对系统各个组成部分的健壮性的进一步的认识。

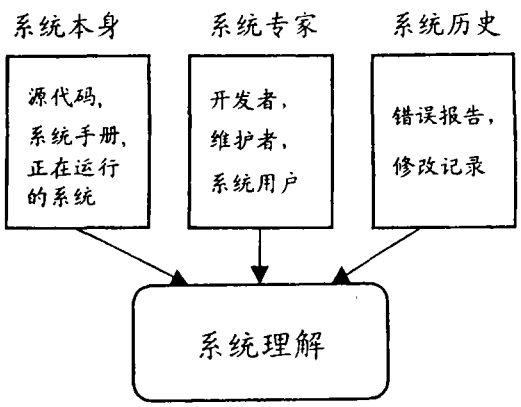


图 5 系统理解的生成

其中对系统本身的理解当然是最为重要的一部分。这部分通常可以以程序理解技术为手段来获取系统本身的详细信息。

程序理解可以大体上分为两种：白盒理解和黑盒理解。白盒理解即深度理解（Deep Understanding），主要是指通过对源代码进行详细的词法和语法分析，得到有关系统的详细信息。黑盒理解即浅度理解（Shallow Understanding），主要是指不需要对系统源代码进行分析，而只对系统的各个构件的接口进行分析，得到对系统各个组成部分之间的关系的理解。

程序理解可以采用的主要手段有以下几种：

- 人工浏览源代码；
- 对系统进行静态/动态的分析；
- 利用描述系统特定性能的度量手段。

在完成了系统理解阶段之后，得到的是对遗产系统的全面而详尽的信息。在此基础上，就可以开始对遗产系统改造了。

### 2.3 系统演化

在得到了对系统的全面理解之后，就可以进行再工程过程的最后一个阶段，即系统演化。系统演化的过程就是通过对遗产系统的改造和重组，使之转化为未来系统的过程。

对于一个大的遗产系统，针对其不同部分和所期望的系统之间的距离，所使用的演化策略是不同的。主要可以有以下三种方法：

1. **维护**（Maintenance）：直到最近几年，维护还一直是和系统演化有关的唯一的可行性活动。遗产系统在开发完成之后，都一直处于被维护的过程中，到它被替换为止。对于要进行再工程的系统中，需要改动并不大的部分，则只需采用维护的策略即可。即：不会对系统进行大的实质性的改动，只是对某些小的细节进行一些修改和修正。

2. **改造**（Transformation）：改造则是比维护要更进一步的演化活动。根据是否需要依赖于对遗产系统内部细节的理解，可以把改造分为白盒改造和黑盒改造两种。白盒改造是包含一个通过逆向工程的过程来对各个模块及其内部细节进行深度理解的过程。而黑盒改造只是包含一个强调对模块的接口进行浅度理解的逆向工程过程。相比之下，白盒改造所需的技术难度显然要比黑盒改造要大，而且它对遗产系统状况的要求也要更高一些，否则就无法进

行深度的程序理解了。

3. **替换**(Replacement):替换适用于那些已经错误百出、病入膏肓的遗产系统。比如说,它所使用的支撑软件再进行继续的维护已经不太现实,或是其运行的硬件平台已经更换。这时候,就应当拿一个新开发的系统来直接替换掉原来的已经没有利用价值的系统。

事实上,很多情况下,针对一个具体的系统,所要使用的策略应该是组合策略,即对一个系统的各个子系统使用不同的策略。对于遗产系统的某一部分,可能进行维护就足以满足要求,而另外的一部分则需要进行白盒改造或黑盒改造,剩下的一部分则可以完全用新开发的系统来替换。通常情况下,也只有这样的组合策略才是最合理的,也是性能价格比最优的。

完成系统演化之后,就得到了期望中的新系统。一次再工程过程的所有活动也就结束了。当然,随着时间的推移,新系统会慢慢变成遗产系统,也就需要进行新的一轮再工程的活动。可以看到,软件系统以这种连续演化的方式生存下去以符合在本文开始提到的新的软件生存周期模型的要求。

## 三、新课题

众所周知,信息产业的发展是日新月异的。每一种新的软件开发方法、新的软件技术的出现都可能对(遗产)软件系统产生重大影响,从而会给再工程提出新的研究课题。再工程必须要适应不断发展变化的软件技术的进步,才能够继续保持自己的优势,在遗产系统到未来系统的演化过程中发挥更好的作用。下面将着重讨论构件技术和分布对象技术可能会对再工程活动带来的影响。

### 3.1 构件技术对再工程的影响

基于构件的软件开发(CBSD)致力于通过组装现有软件构件的方式来建造大型软件系统。通过增强系统的灵活性和可维护性,这种方法可以降低软件开发成本,加快系统组装的速度,减轻大型系统支持和升级所带来的维护负担。这一方法建立在这样一个前提之上,即大型软件系统的某些部分在类似系统中经常出现,以致于具有共性的部分应该只需要编写一次,具有共性的系统应该通过复用来实现。基于构件的软件开发体现了 Fred Brooks 所提出的"用购买代替建造"(buy, don't bulid)的思想[3]。基于构件的软件开发有时也被称为基于构件的软件工程(CBSE)[6-7]。

构件技术在近年来得到了广泛的研究和应用。基于构件的软件工程可以提高软件开发的抽象级别。从机器语言到汇编语言到高级语言的进化已经从很大程度上提高了软件的生产率,但是已经走到了尽头。接下来自然的进化就是再一次提高抽象的级别,通过组装构件来完成一项任务,而不是都从头来书写这些构件。

但是完全利用构件来开发大的软件系统仍然有很大的风险和障碍,这其中有社会的、经济的、和官僚的原因,同时也有技术上的障碍。软件的演化开发需要消除这些障碍以使得构件能够更加容易地被加入、取走、和替换。正如人们认识到可演化开发的好处一样,作为 CBSE 的最初原料的构件也会被更好地进行再工程,更容易地合并到已有设计中去。可互换的软件零件的时代虽然还不成熟,但是终将成为一个在近期内得到应用的技术。

构件技术对再工程的影响主要有两个方面:

第一,再工程能够为构件技术的方法传播和使用提供支持和促进作用。显然,基于构件的系统是容易进行演化的系统,也就是更容易进行再工程的系统。如果从再工程的方面考虑的话,自然希望遗产系统是应用基于构件的软件开发方法开发的。因此,随着再工程深入人心,将促进构件技术的推广和使用。

第二,构件技术的发展也使得再工程过程要进行相应的改变以适应基于构件的遗产系统的演化。毕竟通过构件组装的系统和原来一行一行代码开发的系统间存在很大的差别。在对这样的系统进行再工程的过程中,必须充分考虑其有别于普通系统的特殊之处。在进行系统理解和系统演化的过程中,可能会应用和其它系统不同的策略来进行。

### 3.2 分布对象技术对再工程的影响

分布对象技术正越来越受到人们的重视。它的发展有两个最主要驱动因素:网上计算(Web computing)和中间件(middleware)[18]。而今,在诸如 Netscape 和 Internet Explorer 这样的浏览器支持下,用户只需轻轻地按下鼠标按钮,其客户机就可以访问全球任何地方的服务器。

Java 计算是网上计算的一个杰出代表,它由一个面向对象的语言、相关的类库、以及 Java 虚拟机(JVM)组成。Java applets 提供了在网页中添加交互和动画功能的能力,增加了人们对网络技术的兴趣。可以说,Java 计算就好像是在一个通用机器上实现了网上的移动对象(mobile objects)。

中间件是在一个分布式系统中,增强构件之间集成的技术。它是一种不管通讯协议、系统结构、操作系统、数据库和其它应用服务的差异,而使得应用程序的组成元素可以通过网络连接来进行互操作的软件[15]。这种技术提供了在 Java applets 和新的或

遗产信息系统之间的面向对象的连接。中间件技术的一个突出的例子就是CORBA[13]。

分布对象技术从很大程度上正在改变着软件系统随时间而演化的方式。原来，再工程的重点是通过理解遗产系统，获取对它的基本功能的了解，从而保证能够把系统变得更加健壮，并且更易于演化。但是，分布对象技术、封装策略、和Web正在改变再工程的重点。特定形式的程序理解的开销/收益比(cost/benefit)在很长时间内都会保持不变，而对遗产系统或其子系统使用封装策略的开销/收益比却在迅速地降低。所以，再工程策略正在将它的重点从深度的程序理解转而考虑分布对象技术带来的影响。

在分布对象技术的带动下，可以看到，大粒度复用正在慢慢露出很有前途的征兆，而且也已经有了不少现实的例子。这就使得人们可以开发更大的系统，并且可以使得遗产系统在更低的开销下进行演化。这就意味着在遗产系统的演化过程中，黑盒改造的策略相对于白盒策略来说；应当受到更多的重视。除非深度程序理解领域有惊人的突破，那么决策的天平将会转向浅度的接口理解[18]。

**结束语** 再工程作为一种重要的软件工程技术，已经慢慢地为人们所接受。它为遗产系统的演化提供了理论和方法上的支持。在利用遗产系统开发新系统的时候，如果能够合理地按照再工程的框架中所规定的活动来进行，会在很大程度上降低开发费用，缩短工期，并且能够降低一定的风险。

同时，再工程的理论和技术也需要随着其它软件技术的进展而发展。面对诸如利用构件技术和分布对象技术开发的新形式的遗产系统，需要新的对策，才能始终如一地保持再工程的特点和优势。

## 参考文献


1 Arnold R S. Tutorial on software reengineering. In: CSM'90: Proceeding of the 1990 Conference on Software Maintenance. San Diego, California, Nov., 1990. 26～29

2 Arnold R S. Software Reengineering. IEEE Computer Society Press, 1993

3 Brooks F P Jr. No Silver Bullet: Essence and Accidents of Software Engineering, Computer, 1987. 20 (4). 10～19

4 Barbara I Santa. Back to the Future Through Reengineering. Joint Logistics Commanders Joint Policy Coordinating Group on Computer Management, Santa Barbara, CA, November, 1992

5 Bergey John, et al. A Reengineering Process Framework. Software Engineering Institute, Canegie Mellon Univ., Pittsburgh, PA, 1996

6 Brown Alan W. Preface: Foundations for Commponent-Based Software Engineering, vii-x. Component-Based Software Engineering. In: Selected Papers from the Software Engineering Institute. Los Alamitos, CA: IEEE Computer Society Press, 1996

7 Brown Alan W, et al. Engineering of Component-Based Systems, 7-15. Component-Based Software Engineering. In: Selected Papers from the Software Engineering Institute. Los Alamitos, CA: IEEE Computer Society Press, 1996

8 Chichofsky E, Cross I J H. Reverse Engineering and Design Recovery: A Taxonomy, IEEE Software, January 1990: 13～17

9 Feiler P H. Reengineering: An Engineering problem (CMU/SEI-93-SR-5, ADA267117), Pittsburgh, Pa.: Software Engineering Center, CMU, July 1993

10 Application reengineering: [Technical Report GPP-208]. Guide International Corp., 1989

11 Witschurke R. Wiederverwendung in der Informationsverarbeitung: Re-use, Re-engineering, Reverse Engineering, ISSN 0943-1624, Institut fur Software und Systemtechnik, Universitat Dortmund, Germany, December, 1992

12 Leonard-Barton Dorothy. Implementation as Mutual Adaptation of Technology and Organization. Research Policy, 1988, 17(5)

13 Object Management Group. The Common Object request Broker: Architecture and Spceification, Rev 2.0 (OMG Document 96-03-04). 1995

14 Przybylinski S R, et al. Software technology Transition. In: the 13th. Intl. Conference on Software Engineering, Austin, TX, May, 1991

15 Rymer J. The Muddle in the Middle. Byte, April, 1996

16 Tilley Scott R, Smith Dennis B. Perspectives on Legacy Systems Reengineering (Draft) [online]. Reengineering Center, Software Engineering Institute, Carnegie Mellon University

17 Tilley Scott R, et al. From Software Rehosting to System Reengineering. In: Intl. Conference on Software Maintenance. Monterey, CA November, 1996. 4～8

18 Weiderman N, et al. Implications of Distributed Object Technology for Reengineering: [CMU/SEI-97-TR-005]. Pittsburgh, Pa.: Software Engineering Center, CMU, 1993