附　录

A.1　开发操作型系统 —— 方法之一

M1 —— 初始工程活动

前序的活动：作出建立一个操作型系统的决定。

后继的活动：准备使用已有的代码/数据。

估计时间：时间是不确定的，取决于工程的规模。

通常执行一次还是多次：一次。

特别的考虑：因为这一步骤含义模糊，有无限拖延的趋势。只要系统的 90%(或者更少)在这里定义，系统的开发就应继续进行到下一个阶段。

可以提交：不成熟的系统需求。

- 会谈 —— 会谈的输出是描述系统做什么的“软核心”，通常反映了中层管理者的意见。这种输出的格式是非常自由的。会谈覆盖的领域是不全面的，这成为一个规律。
- 数据收集 —— 此活动的输出可以有多个来源。通常，需求 —— 一般是详细的 —— 不在其他地方收集的部分正是在这里收集的。这是随意的、包罗万象的需求收集活动，它的结果为其他的需求收集活动填补了空白。
- JAD会议输出 —— 此活动的输出是集体讨论的大纲。需求在 JAD会议中形式化的一个好处是自发性和思想的交流，以及不同的人在同一间屋子中关注共同的问题而产生的判断力的聚集。一个或多个 JAD会议的输出是一组形式化的需求，其整体代表了最终用户的需求。
- 战略性商业计划的分析 —— 如果公司有一个战略性商业计划，那么思考这个计划与设计系统的需求怎样联系起来将是有意义的。战略性商业计划的影响能够以多种方法表达出来 —— 设定增长数字、鉴定新的商业线、描述组织结构的变化，等等。所有这些因素，以及其他一些因素，构成要建造的系统的需求。
- 现有系统深刻地影响了新的系统需求。如果已有了相关的系统，最起码也要把新的系统需求和现有系统的接口明确定义。

转化、替代、并行处理以及诸如此类的问题，也是可能涉及的问题。这个活动的输出是现有系统对将要建造系统的需求的影响的说明。

成功的因素：

如果做得恰当，系统的不确定性就会降低，开发工作的范围得到合理的设定，系统的各个部件获得良好的组织。系统的政策性的和技术性的部分同样应该引起注意和定义。

M2 —— 使用现有的代码/数据

前序的活动：系统定义。

后继的活动：规模估计，阶段划分。

估计时间：非常快，即使最大的设计通常也不会超过一星期。

通常执行一次还是多次：一次。

特别的考虑：这个步骤是确保代码和数据可重复使用的最好方法之一。这个步骤对于环境的集成是至关重要的。

在设计好的环境中，每一个工程都有义务做到：

- 尽可能多地使用现有的代码/数据。
- 为将来的工程可以使用现在进行工程的代码和数据做准备。这个步骤的输出是鉴定现有的代码/数据是否可以重复使用，以及鉴定将来的处理需要采取的步骤。

如果现有的代码/数据需要修改，则将这些修改标记为系统开发需求的一个正常的部分。如果现有的代码/数据需要删除，则该删除就成为规格说明的一部分。如果代码/数据需要转换，则该转换就成为开发过程的一部分。

成功的因素：

鉴定可以作为开发基础的现有代码/数据，分辨为准备将来的工作应当做些什么。

M3 —— 规模估计，阶段划分

前序的活动：使用现有的代码/数据。

后继的活动：需求形式化，能力分析。

估计时间：非常快，即使最大的设计通常会在一到两天内完成。

通常执行一次还是多次：一次，之后每个后续的开发阶段会访问这个步骤。

可以提交：开发阶段的标识。

在一般性需求收集完成之后，下一步就是估计系统的规模并且划分开发过程的各个阶段。如果要开发的系统比较大，那么将它分为几个阶段是非常重要的。这样做之后，开发过程被划分成为一些小的可以管理的单元。当然，不同的开发阶段应该被组织成为有意义的序列，这样第二阶段将基于第一阶段，第三阶段将基于第二和第一阶段，以此类推。

这个步骤的输出，在需求很大而应该分段的情况下，将是把通常的需求分为可行的、可管理的阶段。

成功的因素：

继续的增量开发过程是经济和可行的(并且在组织的政策范围内)。

M4——需求形式化

前序的活动：规模估计，阶段划分。

后继的活动：ERD说明书，功能分解。

估计时间：不确定，取决于系统的规模，系统的范围是否定义得好，以及此前的设计具有多少不确定性。

通常执行一次还是多次：每个开发阶段一次。

可以提交：形式化需求说明书。

一旦完成需求收集、规模估计和阶段划分(如果需要的话)，下一步就是对其形式化。在这一步，开发者需要保证：

- 已经收集的需求是完备的，达到合理的可能的程度。
- 需求经过组织整理。

■ 需求是可读的，可理解的，并且在足够低的细节层次是有效的。
■ 需求之间不存在冲突。
■ 需求之间没有重迭。
■ 操作型需求和DSS需求已经分开。

这个步骤的输出是正式的需求定义，准备用于详细设计。

成功的因素：

产生一个简洁的、组织好的、可读的、可操作的、量化的、完备的、可用的需求集合，这也是开发用的一个文档。

CA —— 能力分析

前序的活动：规模估计，阶段划分。

后继的活动：ERD说明书，功能分解。

估计时间：取决于系统的规模，但是如果有评价的工具和专心的计划者，对于适当规模的系统，两到三周是一个合理的估计。

通常执行一次还是多次：每个开发阶段一次。

特别的考虑：能力计划的作用在历史上是混乱的，包括一些不值得特别考虑的不相关的因素。保持开发过程中的能力计划部分集中与切题是很重要的。否则，会成为前进的障碍。

这个被分析的工程享用的资源总量需要在开发的早期阶段被估计。特别要考虑以下几点：

■ 直接存取存储设备(DASD)的消费。
■ 软件需求(包括系统软件、工具、特别定制代码、网络软件、接口软件)。
■ CPU消费。
■ I/O利用。
■ 主存需求。
■ 网络/通道利用。

不仅要分析原始的需求，还应把事务的到达率、峰值 -周期处理、处理模式、响应时间需求、可用性需求以及平均故障时间需求等作为考虑因素。

另外，如果需要定购硬 /软件，那么超前时间和老化时间必须计算在内，以保证当检查应用的时候资源已经到位。

开发的这个阶段的输出是保证需要到位的资源事实上已经到位。

成功的因素：

不用奇怪当需要时资源已经到位，获得资源需要的超前时间，以及需要的大量资源。

PREQ1 —— 技术环境定义

前序的活动：信息处理环境的建立。

后继的活动：规模估计，阶段划分。

估计时间：N/A。

通常执行一次还是多次：N/A。

特别的考虑：偶尔，从头开始建立一个应用系统是必需的，包括定义技术环境。如果是这样的情况，技术定义是在数据 -驱动开发方法的边界之外的。

为了继续下去，技术环境的定义是必要的。如果技术环境此时没有被定义，结果将是产

生系统“颠簸”。简单地说，直到技术环境被定义之后，开始详细的设计才有意义。

某些设计元素超出这点之外，取决于一个或另外一个先前提到的技术基础。至少，下面这些必需确定：

- 使用的硬件平台。
- 使用的操作系统。
- 使用的DBMS。
- 使用的网络软件。
- 使用的开发语言。

不论使用什么硬件、软件和网络，确立下面的内容也是有益的：

- 节点驻留处定义(对网络系统)。
- 记录的系统管理。

成功的因素：

一个强健的、可工作的技术定义满足所开发系统的需求。

D1——实体关系图(ERD，Entity Relationship Diagram)

前序的活动：需求形式化。

后继的活动：数据项目集合说明书。

估计时间：即使最大的系统，如果设计者知道他们要做什么的话两周就足够了。但如果他们不知道，那么需要的时间是不确定的。

通常执行一次还是多次：一次。

可以提交：主要主题领域的标识。

通常的正式需求集合需要标识组成系统的主要主题和这些主要主题之间的关系。主要主题应该处于最高的抽象层次，这是个规则。

典型的主要主题是顾客、产品、事务，诸如此类。对主要主题之间的关系要进行标识，同样还要标识关系的基数。

这个步骤的输出是组成系统的主要主题的标识以及它们之间的关系。

成功的因素：

所有的主要主题已被标识出来，使得不存在领域中的冲突；它们是在最高的抽象层次上标识的。

一个主要的成功因素是只用原始数据建模。另一个成功的因素是在 ERD建模开始之前，已经定义了模型的范围。

D2 —— 数据项目集合(DIS，Data Item Sets)

前序的活动：ERD定义。

后继的活动：性能分析，数据存储定义。

估计时间：每个主题领域需要一个月。

通常执行一次还是多次：每个主题领域一次。

每个主题将来都要分裂(在细节级别上)成为 DIS(数据项目集合)。DIS包括数据的属性，属性的分组和关键字。另外，对数据的“类型”也要标识。这里的其他数据结构包括连接器(代表关系)和第二组数据。这个步骤的输出是对在 D1中标识的主要主题领域添加内容。

成功的因素：

所有类型的主要主题都被标识出来；所有的连接器都被正确地标识；所有的关系都用连接器标识；所有的属性都被标识；所有的属性依照它们对数据组键码的共享关系分组；所有多次出现的属性组和单次出现的属性组分离开来；所有递归关系被设计成最一般情况下所需的形式。这里只能出现原始数据。导出数据在其他环节标识、存储和管理。

D3 —— 性能分析

前序的活动：数据项目集合的开发。

后继的活动：物理数据库设计。

估计时间：每个主题领域需要一个星期，除非主题领域非常巨大。

通常执行一次还是多次：每个主题领域一次。

特别的考虑：如果数据量或处理的数量非常少，这个步骤可以略去。

如果数据量、处理的数量、网络通信量、数据和处理的增长量或处理的峰值周期将引发大量的活动，那么就需要执行这个步骤。如果上述情况一个也不出现，就不需要这个步骤。

这个步骤是处理实际的数据反规范化问题。特别要考虑的是，设计归并表的实践，选择性地引入冗余，创建通用导出数据池，创建数据数组，以及更进一步分离访问概率存在很大差异的数据。

如果这个活动完成，输出将反映一个更加流水线化的设计，对于数据规范化的优点没有或只有一点不利。

成功的因素：

依据数据和访问与更新数据的程序，好的设计使访问和更新更为有效。这个步骤做得合适，可以保证有效地利用资源。

D4 —— 物理数据库设计

前序的活动：性能分析。

后继的活动：伪代码开发。

估计时间：每个表的设计一天。

通常执行一次还是多次：每个表一次。

特别的考虑：如果这个步骤的输入不正确或是含糊不清，这个步骤的工作量将比估计要大得多。

可以提交：表，数据库物理设计。

现在D3和(或)D4的输出用来生成一个物理数据库的设计。输出的一些特性包括：

- 索引。
- 数据的物理属性。
- 划分策略。
- 键码的指明。
- 聚类和交叉。
- 管理变长数据。
- NULL/NOT NULL说明。
- 参照完整性。

这个步骤的输出是实际的数据库的规范说明，说明是针对 DBMS或所采用的任何数据管理软件/规范的。

成功的因素：

正确地完成这个分析步骤，把对数据、性能、更新、访问、可用性、重组织和数据重构等的逻辑设计的所有考虑，转化为可工作的数据库设计。

P1 —— 功能分解

前序的活动：需求形式化。

后继的活动：0层上下文规范说明。

估计时间：取决于系统的规模和不确定性的程度(以及已经确立的范围的稳固性和明确性)。作为一般规则，对合理规模的系统两个星期是足够的。

通常执行一次还是多次：每个开发阶段一次。

从需求文档进入功能分解。功能分解只是把系统完成的大的功能细分为一系列相继的较小的功能(有时细分到称为原语的层次)。

这个过程的输出是大功能的分解，描述从高层次到低层次所进行的不同的活动。

成功的因素：

本节反映的仅是需要设计的功能。这个步骤要考虑到已经作为或将要作为构件块的其他功能。正确地完成这个步骤，将产生可理解的、有组织的、可读的和完备的文档。

P2 —— 0层上下文

前序的活动：功能分解。

后继的活动：1-n层上下文规范说明书。

估计时间：三天。

通常执行一次还是多次：每个开发阶段一次。

功能分解的0层上下文是在最高抽象层次上描述开发的各个主要活动。处理规范说明的 0层上下文对应于数据建模的ERD。

成功的因素：

正确地完成后，这个步骤产生的文档仅说明系统的主要活动及其间的关系。

---

p3 —— 1-n层上下文

前序的活动：0层上下文规范说明。

后继的活动：DFD规范说明。

估计时间：每个功能一天。

通常执行一次还是多次：每个功能一次。

可以提交：完整的功能分解(注意：许多步骤构成这个提交)。

功能分解的其余层次描述所发生的更详细的活动。处理设计中的 1-n层上下文对应于数据设计中的数据项集合(DIS)。

成功的因素：

这个过程正确完成时，所有主要的活动被分解到原语层次。这个分解是有序的、有组织的和完备的，并且与活动流是一致的。

P4 —— 数据流图(DFD)

前序的活动：1-n层上下文规范说明。

后继的活动：算法规范说明。

估计时间：每个活动一个小时(在原语层次)。

通常执行一次还是多次：每个活动一次。

可以提交：每个处理过程的数据流图。

在每一个上下文层次n —— 原语层次 —— 一个DFD被抽取出来。DFD指明一个处理过程的输入和输出，为建立处理过程所需要的存储，以及这个处理过程的简要说明。一个 DFD可以在高于n的上下文层次上完成，前提是可以编写对于那个层次的程序或处理过程。

成功的因素：

每个原语层次的活动的输入和输出都被标识出来，活动流也被标识出来。对数据存储作了概述，每个活动要做的工作做了说明。

P5 —— 算法规范说明；性能分析

前序的活动：DFD规范说明。

后继的活动：伪代码。

估计时间：每个必需说明的原语层次上的活动需要五分钟到两天不等。

通常执行一次还是多次：每个活动一次。

DFD为原语定义的处理过程进一步分解为详细的算法说明。换句话说，在这个步骤，要概述实际出现的每一步处理。

另外，如果性能成为一个问题，程序设计性能的好坏是要考虑的因素。这就要考虑诸如下面的设计技术：

- 把一个长时间运行的程序分解为一系列短程序。
- 要求程序访问较少数量的数据。
- 缩短一个数据段单元的锁定时间。
- 把一个可能更新的锁定改变为只进行访问。

成功的因素：

正确地完成算法说明，把说明需要的所有细节都标识出来，而且只标识说明需要的细节，每次标识一步，并且包含所有可能性。另外，一定要确保和标准工作单元 (SWU)的一致。

P6 —— 伪代码

前序的活动：算法规范说明；物理数据库设计；数据存储设计。

后继的活动：编码。

估计时间：时间长短不一(视前一个活动而定)。

通常执行一次还是多次：每个活动一次(在原语层次)。

算法和程序说明书进一步精炼成为伪代码。设计者确保处理需要的所有数据都可以获得。所有在计算、变换、更新等过程中用到的变量都在这里标识。任何松散的端点都要标识。设计层次上的性能也在这里考虑。这个活动的输出是编码说明，为真正的代码作好准备。

成功的因素：

程序说明的最后一关，包括：

■ 完备性。
■ 执行次序。
■ 所有需要的情况。
■ 所有的意外，包括错误处理和异常条件。
■ 代码结构。

P7 —— 编码
前序的活动：伪代码。
后继的活动：通查。
估计时间：不定，每个活动从一天到两个星期不等。
通常执行一次还是多次：每个活动一次。
可以提交：源代码。

伪代码被翻译成源代码。如果数据已经被恰当地定义了，而且伪代码的说明是完全的，这个步骤就会顺利。这个步骤的输出是源代码。

成功的因素：

完整和高效地把伪代码翻译成代码，包括内部文档。正确完成之后，前面说明的所有需求在这里都得到满足。

P8 —— 通查
前序的活动：编码。
后继的活动：编译。
估计时间：每个活动一小时。
通常执行一次还是多次：每个活动一次。

通查是对代码逐字逐句进行解释。目的是在测试之前发现编写代码的错误(或其他类型的错误)。这个步骤的输出是尽可能免除错误的源代码。

成功的因素：

在编码前检测错误。这个步骤做得好，只剩下非常少的错误需要其他手段来发现。

P9 —— 编译
前序的活动：通查。
后继的活动：单元测试，重点测试。
估计时间：每个活动一个小时或稍少。
通常执行一次还是多次：直到获得一个干净的编译。

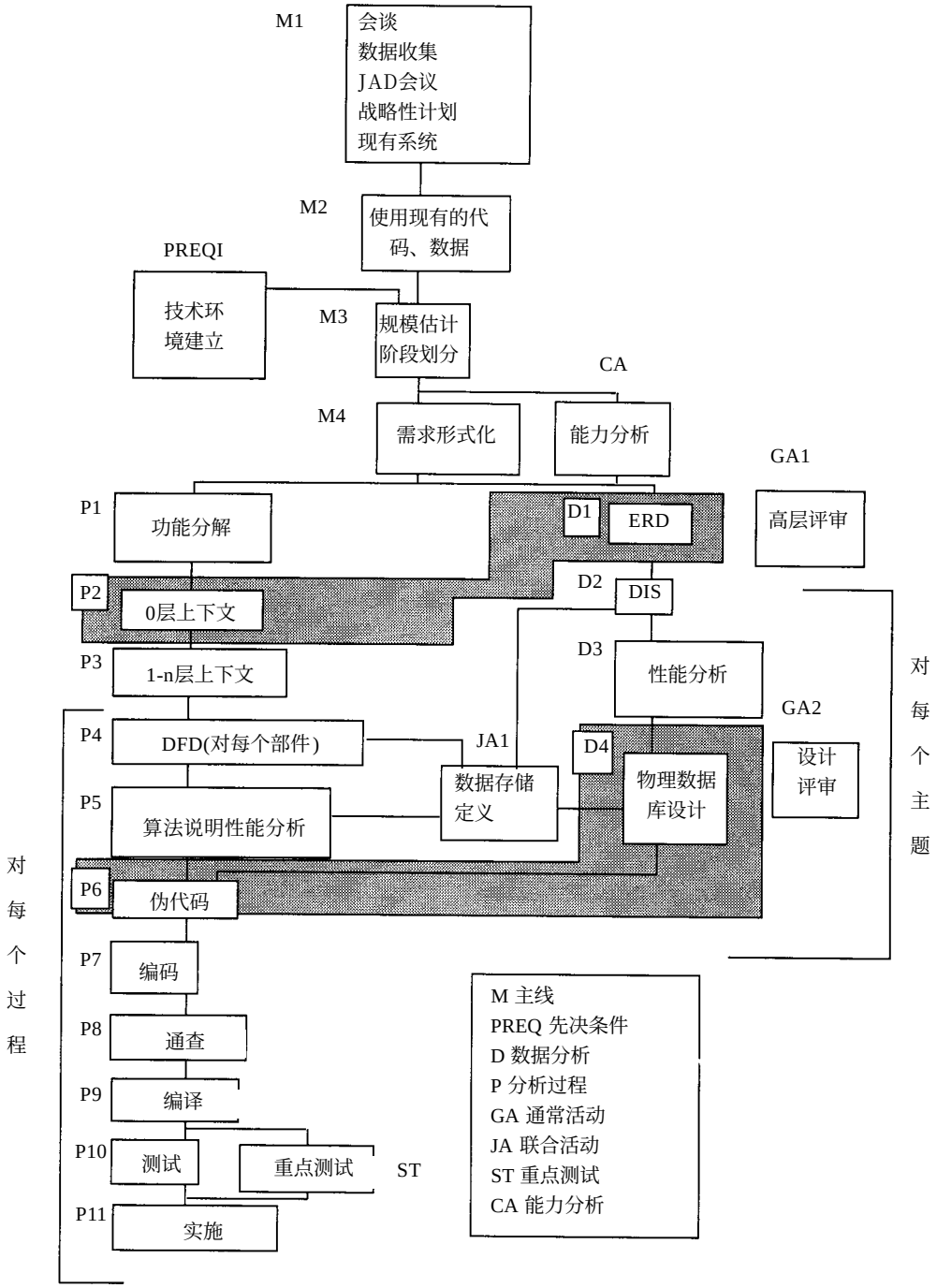
源代码通过编译。所有在编译中发现的错误被改正。这个步骤的输出是编译过的代码，准备用于测试。

成功的因素：

为测试准备的编译过的代码。

P10 —— 单元测试。
前序的活动：编译。
后继的活动：实施。
估计时间：差别很大。

通常执行一次还是多次：不定。



图A-1 方法之一

测试编译过的代码。有几个层次的单元测试。最简单的(层次最低的)单元测试是编译过的模块的自测试，确保它的功能能够完成。下一步，把编译过的代码加入到其他代码中使之一

同工作。新的单元测试层次是保证编译过的代码和与之接口的其他模块的集成。当大的模块组准备共同测试时出现第三层单元测试。

这个步骤的输出是测试通过的代码，准备用于执行。

成功的因素：

如果实施是正确的，进入下一个步骤的代码在程序逻辑上就是正确的。而且，在代码进入下一步之前所有情况都测试了，包括错误处理和异常条件。

P11 —— 实施

前序的活动：单元测试；重点测试。

后继的活动：N/A。

估计时间：N/A。

可以提交：满足规范说明的系统。

在实施的过程中有许多活动。实施在一定程度上是一个不断进行的活动。实施的一些典型的活动是：

- 培训，讲授。
- 程序的加载。
- 数据的初始加载。
- 需要时的数据的转换。
- 监测工具的建立。
- 文档编写。
- 恢复，重组过程的建立。

这个步骤(如果这一步真正结束的话)的输出是一个满意的运行系统。

成功的因素：

这个过程完成后，会有一个快乐的用户。

A.2 开发数据仓库 —— 方法之二

DSS1 —— 数据模型分析

前序的活动：承担数据仓库建设的任务。

后继的活动：主题领域分析，“面包箱”分析，数据仓库设计。

估计时间：差别很大，取决于数据模型的状态和质量。

通常执行一次还是多次：一次。

开始时要定义一个数据模型。数据模型需要：

- 标识主要主题领域。
- 清晰地定义模型的边界。
- 把原始数据和导出数据分离。
- 每个主题领域需要标识：
  - 键码。
  - 属性。
  - 属性分组。

- 属性分组之间的关系。
- 多重出现的数据。
- 数据的“类型”。

这个步骤的输出是对机构已经建立了一个可靠的数据模型的确认。如果模型没有满足指定的标准，那么进程就应该停止直至模型达到质量标准。

成功的因素：

数据模型应该具有：

■ 主要主题领域的标识。

■ 每个主要主题有自己的独立的数据定义，包括：

- 数据的子类型。
- 数据的属性。
- 清晰地定义数据的关系。
- 定义数据分组。
- 定义键码。

另外，每个进入数据仓库的数据组应该有 DSS数据和纯操作型数据的描述。所有的 DSS数据有自己的随时间变化的键码，通常指定为高层键码的低位。

DSS2 —— “面包箱”分析

前序的活动：数据模型分析。

后继的活动：数据仓库数据库设计。

估计时间：从一天到两周不等，取决于范围定义得是否好，数据模型定义得是否好，等等。

通常执行一次还是多次：一次。

可以提交：粒度分析。

当数据模型已经分析并且到达足够好的质量，下一步就是“面包箱”分析。“面包箱”分析是通过粗略估计确定 DSS环境的规模。如果数据量成为一个问题，重要的是一开始就应该知道这个情况。“面包箱”分析大致来说就是知道数据仓库要处理多少数据。

“面包箱”分析的输出是简单的 —— 如果数据仓库要容纳大量的数据，那么多层次粒度就需要考虑了。如果数据仓库没有大量的数据，就没有必要考虑多层次粒度。

成功的因素：

对整个数据仓库环境的数据量的估计 —— 以行的数目表示 —— 在一年的限度内和五年的限度内，这就是这个过程的结果。基于这个估计，可以决定是否需要采用多层次粒度。如果需要多层次粒度，那么准确地定义多层次粒度也是这个步骤输出的一部分。

DSS3 —— 技术评价

前序的活动：承担数据仓库建设的任务。

后继的活动：技术环境预备。

估计时间：一周。

通常执行一次还是多次：一次。

可以提交：技术环境评价。

管理数据仓库的技术需求，与管理操作型环境中的数据和处理过程的技术需求和考虑因素是非常不同的。这就是为什么独立的、集中的 DSS数据存储如此流行的原因。

成功的因素：

正确执行后，数据仓库的技术定义满足下列准则：

- 管理大量数据的能力。
- 允许灵活访问数据的能力。
- 根据数据模型组织数据的能力。
- 能够用许多技术接收和发送数据的能力。
- 能够周期地把数据全部加载的能力。
- 能够以一次一个集合或一次一个记录的方式访问数据的能力。

DSS4 —— 技术环境准备

前序的活动：技术评价。

后继的活动：数据仓库设计；载入。

估计时间：一周到一个月。

通常执行一次还是多次：一次。

当数据仓库的体系结构配置已经建立，下一步就是技术上确定如何实现这个配置。这里要考虑的典型问题是：

- 需求DASD的数量。
- 需要什么链路 —— 通过网络或进入网络。
- 预期的处理量。
- 如何使竞争的存取程序之间的处理冲突达到最小限度或减轻这种冲突。
- 由控制数据仓库的技术产生的通信量。
- 由控制数据仓库的技术产生的通信量的性质 —— 或短或长的爆发。

成功的因素：

这个步骤正确执行后，成功的路上就不会有技术上的阻碍了。已经安装、分配、“老化”，而且准备好接收数据的技术部件包括：

- 网络。
- DASD。
- 管理DASD的操作系统。
- 出入数据仓库的接口。
- 管理数据仓库的软件。
- 数据仓库。

DSS5 —— 主题领域分析

前序的活动：数据模型分析。

后继的活动：源系统分析。

估计时间：一天。

通常执行一次还是多次：每个载入工程一次。

可以提交：下一个要建造哪个主题。

要载入的主题领域已经选择了。第一个选择的主题领域必须大到足以有意义，而又小到可以实现。如果有时某个主题领域确实大而且复杂，那么应该选择它的子集来实现。这个步骤的输出是围绕一个主题的工作范围。

成功的因素：

这个阶段正确完成后，输出是下一阶段要载入的数据的定义。最初的几个载入，通常选择小的主题。后来的载入，选择较大的主题，甚至是主题的子集。当正确完成后，选择的载入主题适合开发数据仓库的当前阶段的需要。

DSS6 —— 数据仓库设计

前序的活动：数据模型分析，源系统分析，“面包箱”分析。

后继的活动：程序规范说明。

估计时间：一周到三周。

通常执行一次还是多次：一次。

可以提交：数据仓库的物理数据库设计。

数据仓库是基于数据模型设计的。一些终极的设计特性包括：

- 如果确实需要多层次粒度的话，要做好不同层次粒度的调节。
- 把数据定向到公司的主要主题。
- 只出现原始数据和公共的导出数据。
- 不存在非DSS数据。
- 每个数据记录的时间变化量。
- 在适宜的场合物理反规范化数据(例如保证性能的场合)。
- 在把操作型环境中的数据引入到数据仓库的地方创建人工数据。

这个步骤的输出是一个数据仓库的物理数据库的设计。注意一开始不是整个数据仓库都要详细地设计。最初只要把数据仓库的大结构设计出来就完全可以接受了，在后面的工作中再添入细节。

成功的因素：

这个步骤正确执行后，其结果就是一个具有一定数量数据的数据仓库，这些数据可以用相当有效的方式加载、访问和搜索。

DSS7 —— 源系统分析

前序的活动：主题领域分析。

后继的活动：程序规范说明；数据仓库设计。

估计时间：每个主题领域一周。

通常执行一次还是多次：每个主题领域一次。

可以提交：记录系统的标识。

当要载入的主题已经标识了，下一个活动就是在现有的系统环境中为主题标识源数据。DSS数据有不同的数据源是再正常不过了。集成问题正是这一点上要考虑的。这个步骤考虑的问题如下：

- 当数据从操作型环境转移到数据仓库环境时的键码结构 /键码分辨率。
- 属性。

- 如果有多个来源可以选择应如何处理。
- 如果没有来源可以选择应如何处理。
- 当数据被选择传输给DSS环境时必须做何种变换——编码/解码、转换，等等。

■ 从数据的当前值如何创建时间变化量。

■ 结构——DSS数据结构如何从操作型数据结构中创建。

■ 关系——操作型的关系如何在DSS环境中体现。

这个步骤的输出是数据从操作型环境到 DSS环境的映射。

成功的因素：

在正确执行这个步骤后，服务于数据仓库需要的源系统使用的数据，就会是及时、完备、准确、接近来源和易于访问的，并且与数据仓库需要的结构一致。

DSS8 —— 规范说明

前序的活动：源系统分析，数据仓库设计。

后继的活动：编程。

估计时间：每个抽取/集成程序一周。

通常执行一次还是多次：每个需要编写的程序一次。

当操作型环境和DSS环境的接口勾画出来后，下一步就是以程序说明的形式将它形式化。这里包括的某些主要问题是：

■ 我怎么知道扫描的是什么样的操作型数据？

- 操作型数据打上时间戳了吗？
- 是增量文件吗？
- 有系统日志或审计日志可以使用吗？
- 现有的源代码和数据结构可以改变以产生一个增量文件吗？
- 前映像和后映像文件需要都清除吗？

■ 一旦扫描后我如何存储输出？

- DSS 数据用预分配、预格式化吗？
- 添加数据吗？
- 替换数据吗？
- 在DSS环境进行更新吗？

这个步骤的输出是实际的程序规范说明，用于把数据从操作型环境引入数据仓库中。

成功的因素：

当正确完成后，这个步骤将会使数据的抽取和集成尽可能地高效和简单。 (这很少会是一个顺利、简单的过程。)

DSS9 —— 编程

前序的活动：规范说明。

后继的活动：载入。

估计时间：每个抽取/集成程序一周。

通常执行一次还是多次：每个程序一次。

可以提交：抽取、集成和时间—透视程序转换。

这个步骤包含了所有编程的标准活动，例如：

- 开发伪代码。
- 编码。
- 编译。
- 通查。
- 各种形式的测试 —— 单元测试，重点测试。

成功的因素：

当正确完成后，这个步骤生成的代码将是高效、有文档说明、易于改变、准确和完备的。

DSS10 —— 载入

前序的活动：编程，技术环境预备。

后继的活动：数据仓库的使用。

估计时间：N/A。

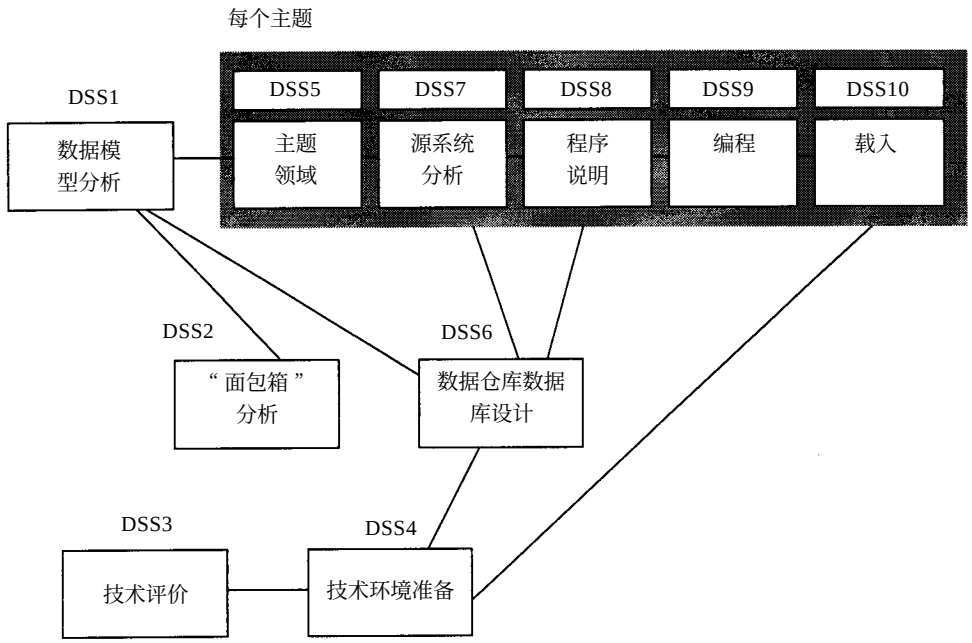
通常执行一次还是多次：N/A。

可以提交：可用的数据仓库。

这个步骤需要的仅仅是执行前面开发的DSS程序。这里考虑的问题有：

- 载入的频率
- 清除载入数据
- 老化载入数据(例如，运行帐务汇总程序)
- 管理多层次粒度
- 更新活样本数据(如果活样本数据表已经建立)

这个步骤的输出是已载入的具有功能的数据仓库。



图A-2 方法之二

成功的因素：

当正确完成后，这个步骤的结果是一个可访问的、可理解的、能够为 DSS群体的需要服务的数据仓库。

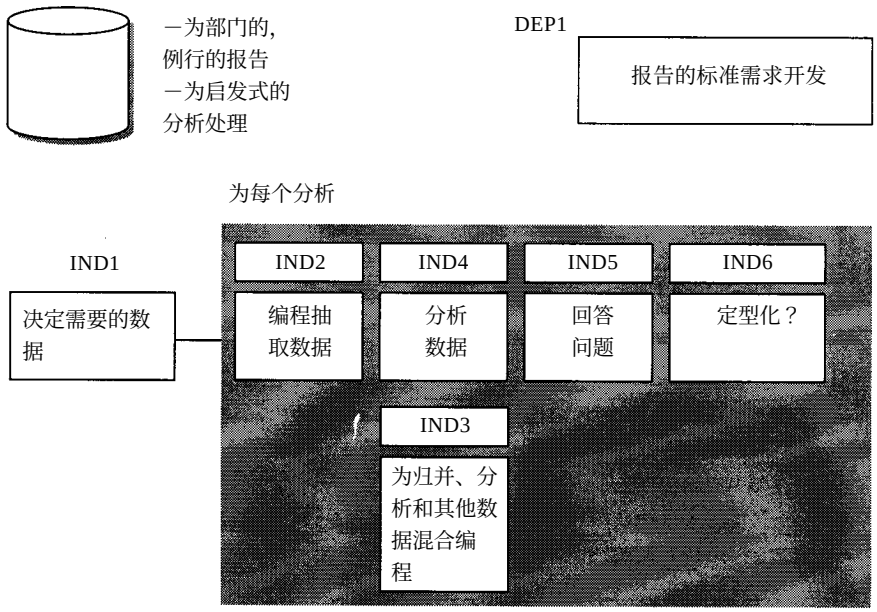
A.3 启发式处理 —— 方法之三

在体系结构设计环境中开发的第三个阶段是为分析的目的使用数据仓库的数据。当数据载入到数据仓库环境中，就可以着手使用了。

在这个层次进行的开发和环境其他部分的开发有本质的不同。第一个大的不同是这个阶段的开发过程总是从数据开始，即从数据仓库中的数据开始。第二个不同是在开发过程的开始不知道需求是什么。第三个不同(其实是第一和第二个因素的副产品)是处理以极其重复的启发式的方式完成。在其他类型的开发中，总是存在一定数量的重复。但是在数据仓库开发后开始的DSS部件开发，重复的整个性质改变了。处理中的重复在分析开发过程中是普通而且重要的部分，比在其他场合要多得多。

在DSS部件开发中的步骤可以分为两类 —— 重复出现的分析(有时称为“部门”的或“功能”的分析)和真正的启发式处理(“个别”的层次)。

图A-3显示了在数据仓库开始载入后的开发步骤。



图A-3 方法之三

A.4 启发式DSS开发 —— 方法之四

DEP1 —— 重复的标准开发 —— 为进行重复性的分析处理(通常称为提交标准报告)，通常的需求-驱动处理出现了。这意味着下列的步骤(前面描述的)会重复进行：

M1 —— 会谈，数据收集，JAD，战略计划，现有系统。

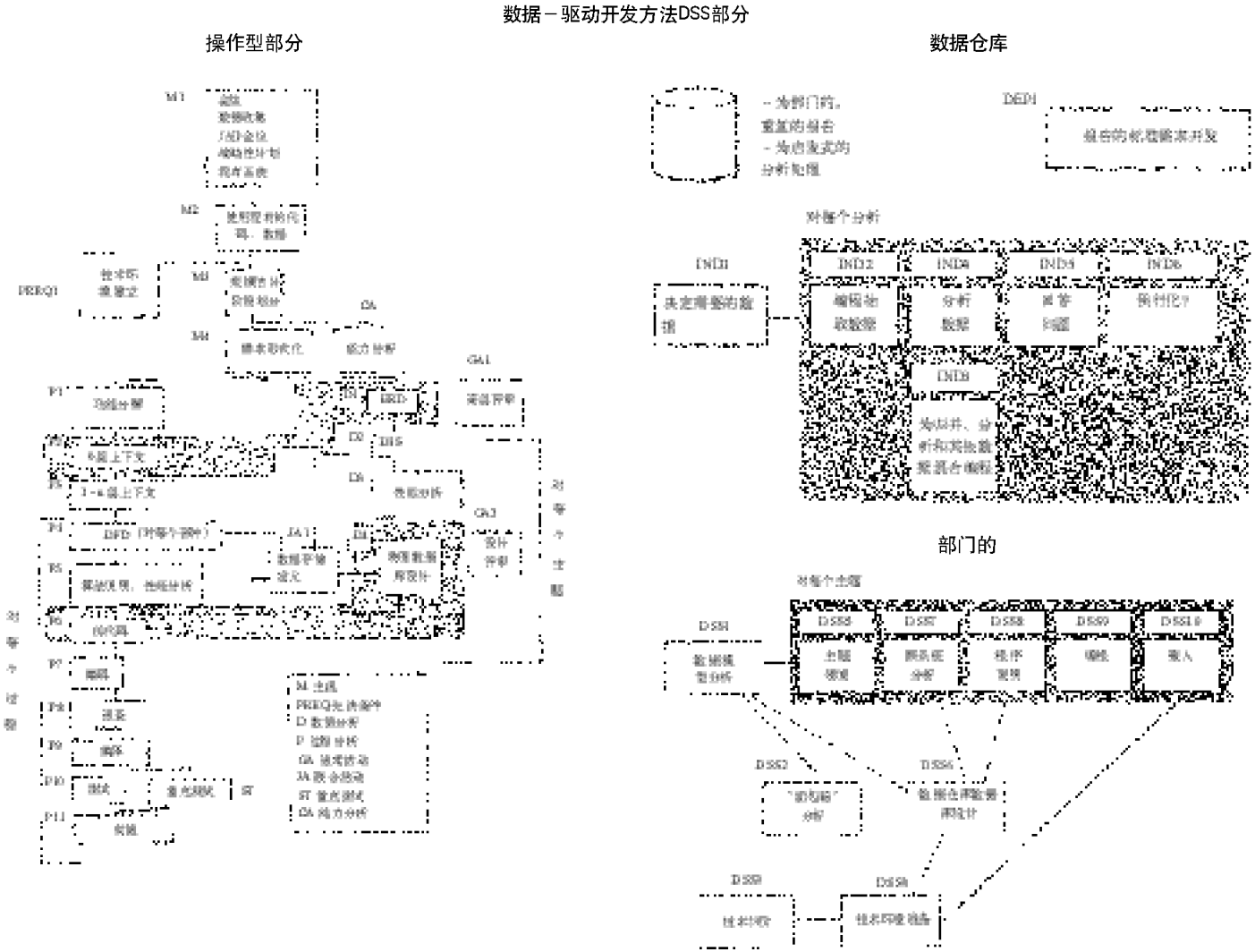
M2 —— 规模估计，阶段划分。

M3 —— 需求形式化。
P1 —— 功能分解。
P2 —— 0层上下文。
P3 —— 1-n层上下文。
P4 —— 每个部件的DFD。
P5 —— 算法规范说明；性能分析。
P6 —— 伪代码。
P7 —— 编码。
P8 —— 通查。
P9 —— 编译。
P10 —— 测试。
P11 —— 实现。
另外，至少下列的部分将会在合适的时间进行：
GA1 —— 高层评审。
GA2 —— 设计评审。
开发的数据分析部件没有必要做，因为开发者从数据仓库中做数据分析。
这个活动的输出是在规范基础上产生的报告。
成功的因素：
当正确完成后，这个步骤保证满足规范报告的需要。这些需要通常包括：
■ 管理报告。
■ 会计报告。
■ 关键因素指示符报告。
■ 市场报告。
■ 销售报告。
可预知的和重复性的信息需要在这个功能中满足。
注意，对于高度迭代的处理，存在成功的因素，但是是由处理过程在整体上满足的。因为需求没有预先定义，每次迭代的成功是带一定主观性的。

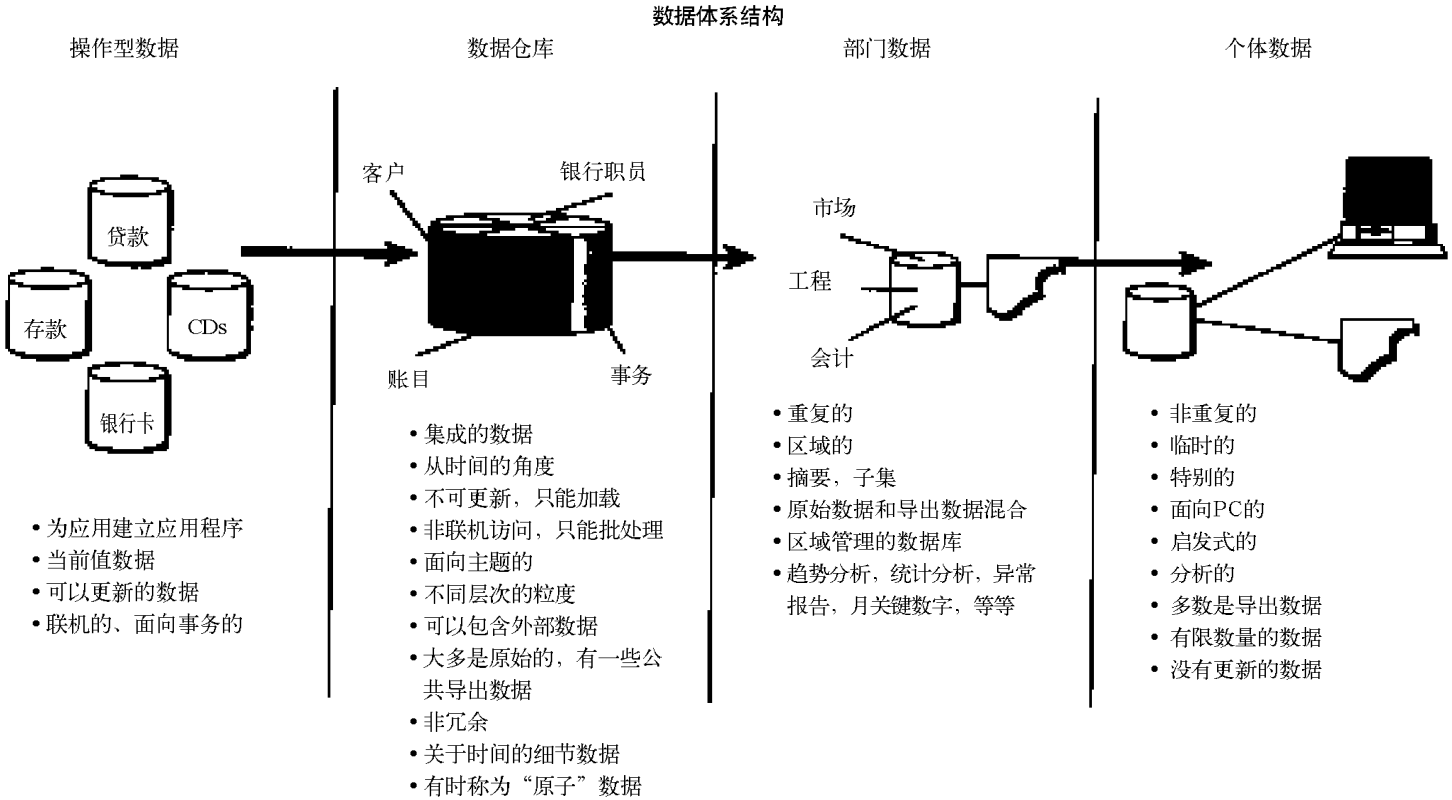
IND1 —— 确定需要的数据
在这里，把数据从数据仓库中选择出来，以满足报告需求的潜在的应用。当开发者在推测方面是训练有素的，就可以理解这个活动最初的前两三次，只能检索到一部分所需的数据。
这个活动的输出是为将来的分析选择好数据。

IND2 —— 编写抽取数据的程序
当为分析处理选择好了数据，下一步就是编写程序来访问和剥离数据。这个程序应该写得容易修改，因为可以预见程序在很多情况下要运行，修改，然后再运行。
可以提交：从数据仓库中为DSS分析抽出的数据。

IND3 —— 混合，归并，分析
数据选择好了就准备分析。通常这意味着编辑数据，和其他数据混合，提炼。



图A-4　方法之四，数据－驱动开发方法



图A-4　方法之四，数据－驱动开发方法(续)

与所有其他的启发式过程一样，可以预见这个程序应该编写得易于修改，因此可以快速再运行。这个活动的输出是完全可用于分析的数据。

可以提交：和其他相关数据的分析。

IND4 —— 分析数据

当数据被选择和准备好后的问题就是“所得的结果满足分析人员的需要吗？”如果结果不满足，则进行另一次迭代过程。如果结果满足，就开始准备最终的报告。

IND5 —— 回答问题

最终报告的生成通常是在处理过程的多次循环后得到的。最终的总结很少能在一次分析循环中得到。

IND6 —— 定型化

最后的问题是考虑是否把生成的最终报告定型化。如果这个报告需要重复地运行，那么把这个报告作为需求集合提交并且作为一种定期重复操作重建这个报告，就是很有意义的了。

A.5 总结

不同的活动彼此之间的关系以及与数据体系结构的概念之间的关系如图 A-4中所示。

A.6 选择的主题

图形是描述数据-驱动开发方法的属性的最好方法。图 A-5显示出数据模型是数据-驱动方法的核心。

数据模型关系到操作型数据的设计，数据仓库中数据的设计，操作型数据的开发和设计过程，以及数据仓库的开发和设计过程。图 A-5还显示了同一数据模型如何和每个活动和数据库关联。

数据模型是标识跨越不同应用的通用性的关键。但是有人会问，“识别处理过程的通用性也很重要吗？”

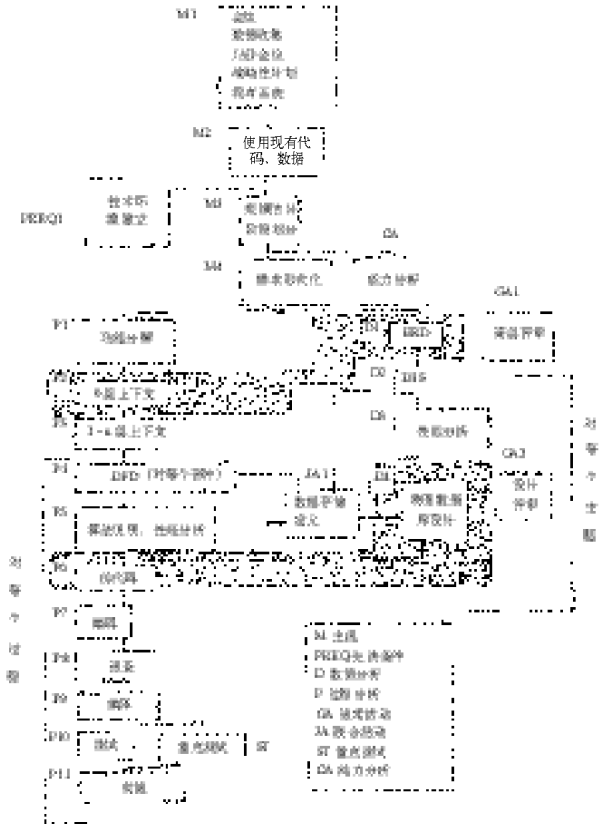
回答是当然的，识别跨越应用处理的通用性是很重要的。但是试图集中于处理的通用性有几个问题 —— 处理的改变更快于数据，处理趋向于紧密地混合通用的和独自的处理以至于它们经常不可分离，而且典型的处理分析经常在设计的范围设置一个人为的小边界。数据本质上比处理更加稳定。数据分析的范围比处理模型的范围更容易扩大。因此，把聚焦于数据作为识别通用性的要旨很有意义。另外，这里的假设是如果发现数据的通用性，那么这个发现会导致对应的处理过程的通用性。

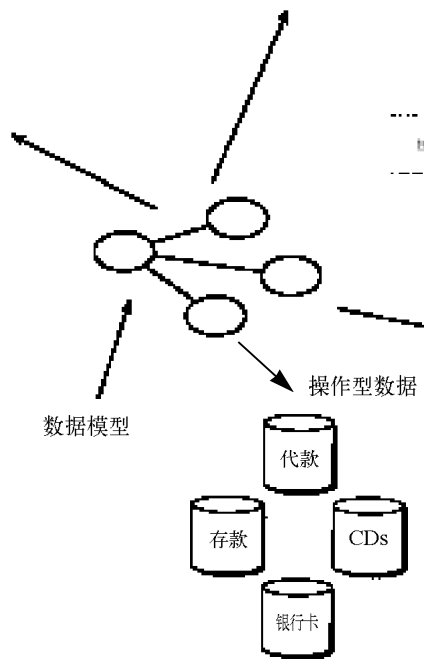
因此，数据模型 —— 跨越所有应用，反映企业的观点 —— 是标识和统一数据和处理通用性的基础。

A.6.1 提交

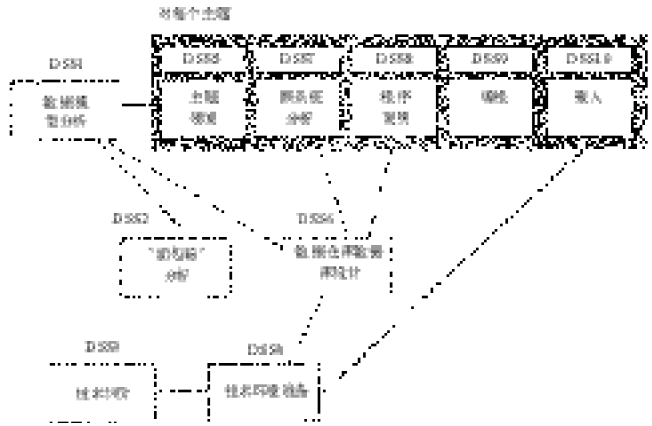
数据－驱动开发方法的步骤包括一个提交。事实上，一些步骤和另一些步骤共同构成一个提交。然而对于大部分情况，方法的每个步骤都有独自的提交。

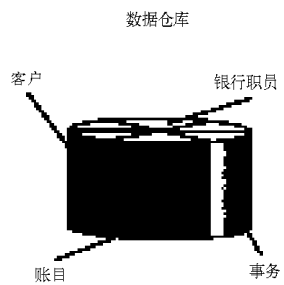
操作型系统的开发的处理过程分析部件的提交在图 A-6中显示。





- 为应用建立应用程序
- 数据当前值
- 数据可以更新
- 联机的、面向事务的

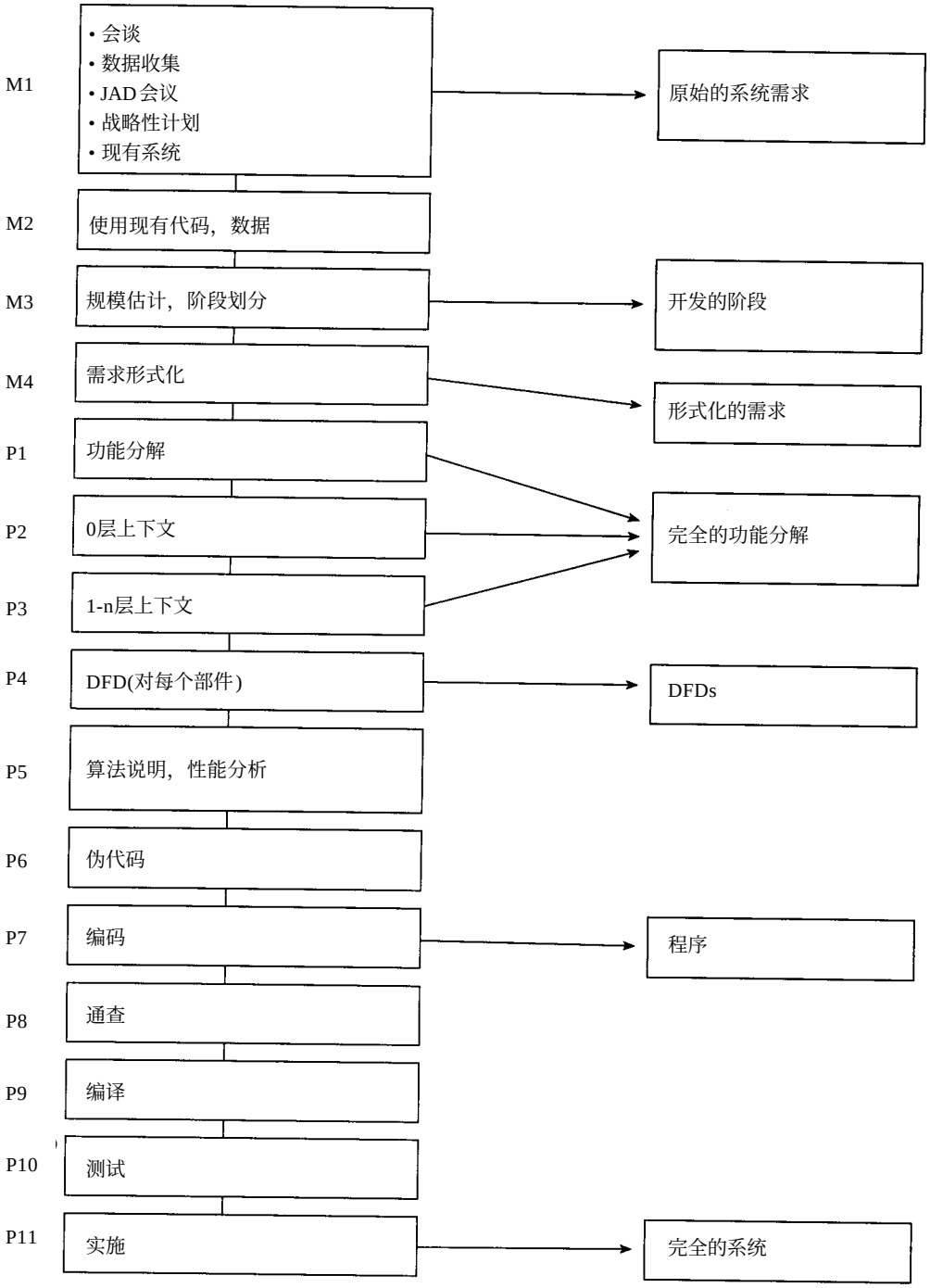




- 集成的数据
- 从时间的角度
- 不可更新，只能加载
- 非联机访问，只能批处理
- 面向主题的
- 不同层次的粒度
- 可以包含外部数据
- 大多是原始的，有一些公共导出数据
- 非冗余
- 关于时间的详细数据
- 有时称为“原子”数据

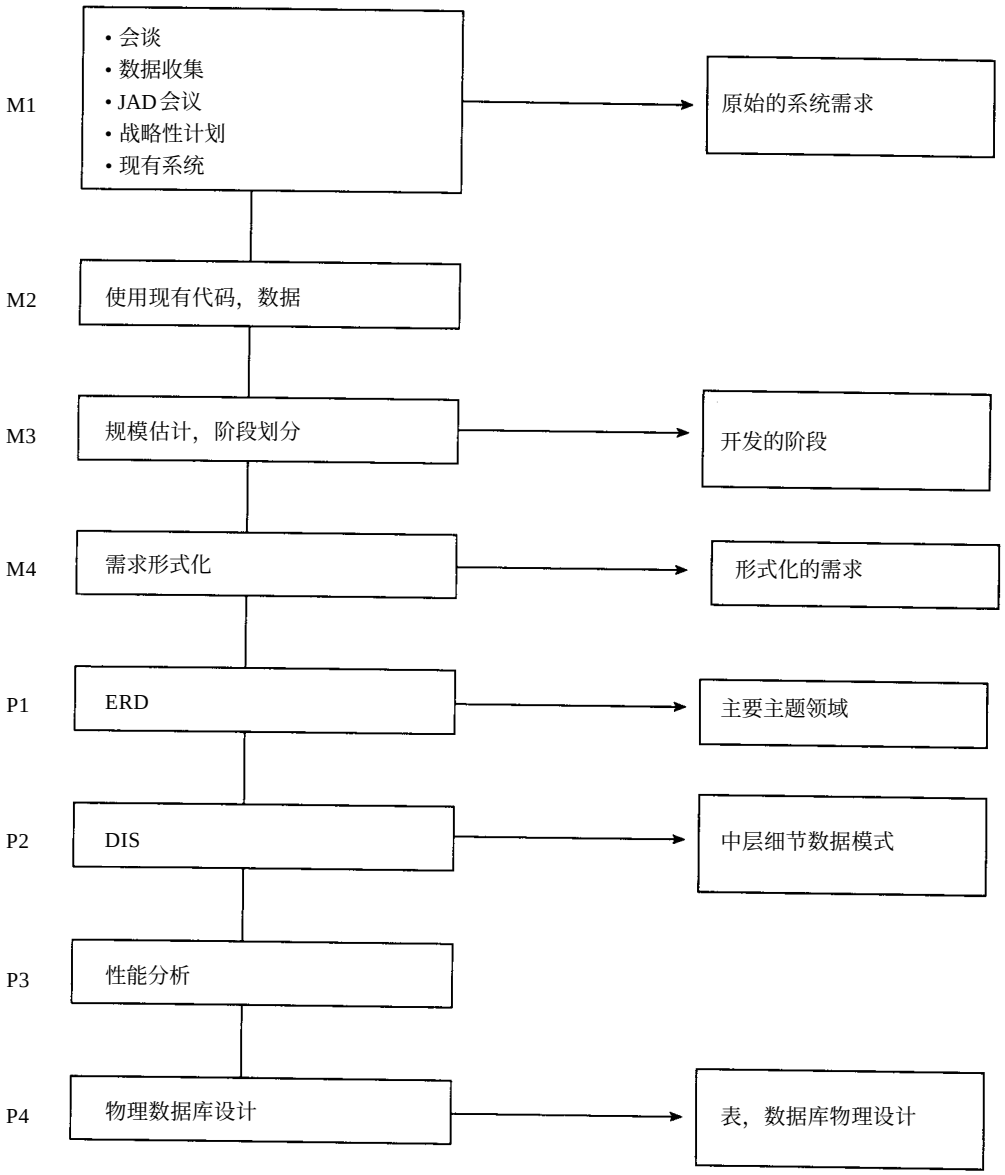
图A-5 方法之五

图A-6显示会谈和数据收集处理的提交是一个原始的系统需求的集合。决定什么代码 /数据可以重用的分析和规模估计 /阶段划分步骤产生一个描述开发阶段的提交。



图A-6 方法之六，开发生命周期中的提交

需求形式化的活动产生(不要惊讶)一个正式的系统规范说明的集合。功能分解活动的结果是提交一个完整的功能分解。



图A-7 方法之七，操作型数据分析的提交

DFD定义的提交是一个描述已经被分解的功能的 DFD集合。通常， DFD集合表现出分解的初始层次。

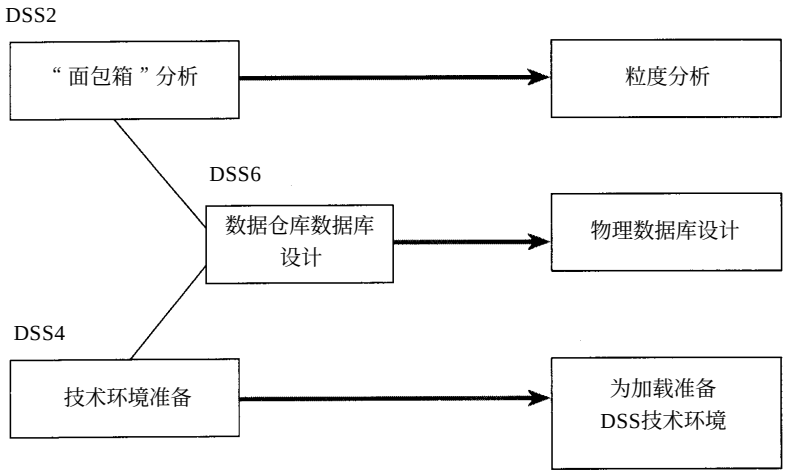
编码活动产生的提交是程序。最后，实施活动产生一个完整的系统。

图A-7显示操作型系统数据分析的提交。

会谈和数据收集过程、规模估计和阶段划分活动以及需求形式化定义产生的提交和前面讨论的一样。

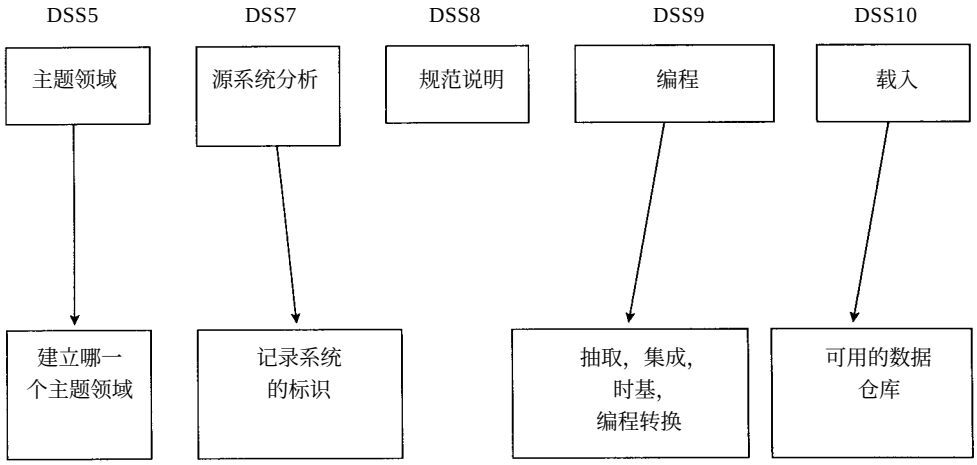
ERD活动的提交是标识主要主题领域和它们之间的关系。 DIS活动的提交是每个主题领域的完全特征化和规范化的描述。物理数据库设计的最后提交是实际的表或数据库设计，准备在数据库管理系统中给出定义。

数据仓库开发工作的提交在图A-8中显示，“面包箱”分析的结果是粒度和容量的分析。和数据仓库数据库设计相关的提交是数据仓库表的物理设计。和技术环境准备相关的提交是建立数据仓库存在的技术环境。注意这个技术环境可能是也可能不是操作型系统存在的同一环境。



图A-8 方法之八，预备的数据仓库提交

在反复进行的基础上，数据仓库载入活动的提交在图 A-9中表示，其中显示出主题领域分析的提交 —— 数据仓库的每次载入 —— 是选择一个主题(或可能是主题的子集)来载入。



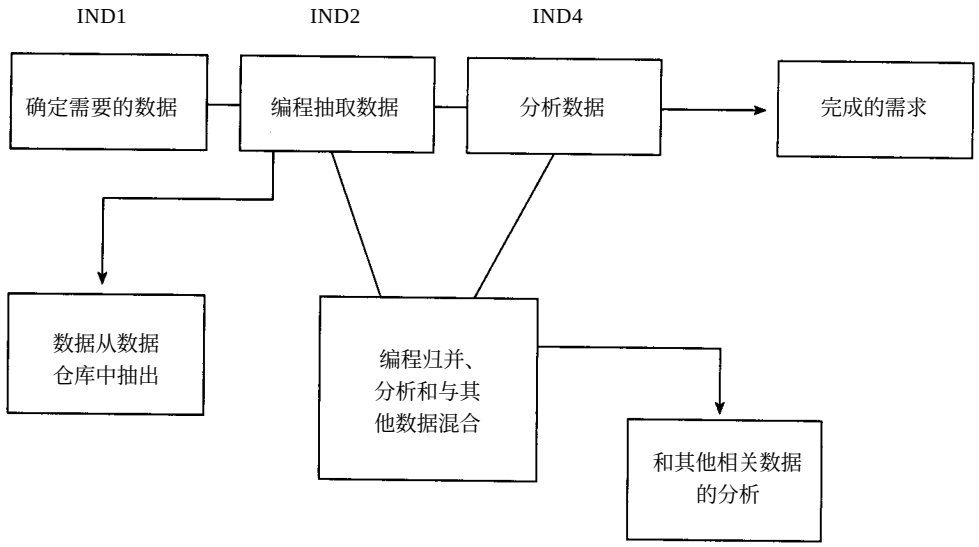
图A-9 方法之九，数据仓库开发步骤中的提交

源系统分析的提交是被考虑的主题领域的记录系统的标识。编程阶段的提交是抽取、集成和把数据从当前值改变到时间变化量的程序。

数据仓库的载入的最后提交是数据仓库真正的载入。已经指出过载入数据到数据仓库是一个不断进行的活动。

处理过程的启发式层次的提交不像在开发的操作型和数据仓库层那样容易定义。这个阶段分析处理的启发式特性更形式化。然而，图 A-10显示出一些基于数据仓库的启发式处理的提交。

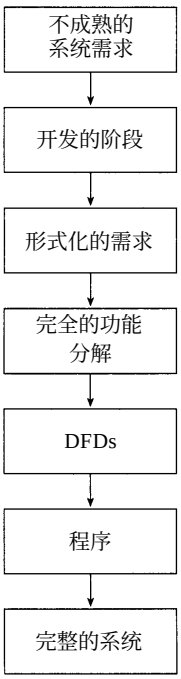
图A-10显示从数据仓库中抽出的数据是抽取程序的结果。后来的分析步骤提交的是基于已经提炼的数据的进一步分析。最后数据分析提交的是满足条件的 (与可理解的)需求。



图A-10 方法之十，启发式层次处理的提交

A.6.2 一个提交的线性流程

除了启发式处理，可以期望一个线性的提交流程。图 A-11显示了执行数据-驱动开发方法的过程分析部件产生的提交的例子。

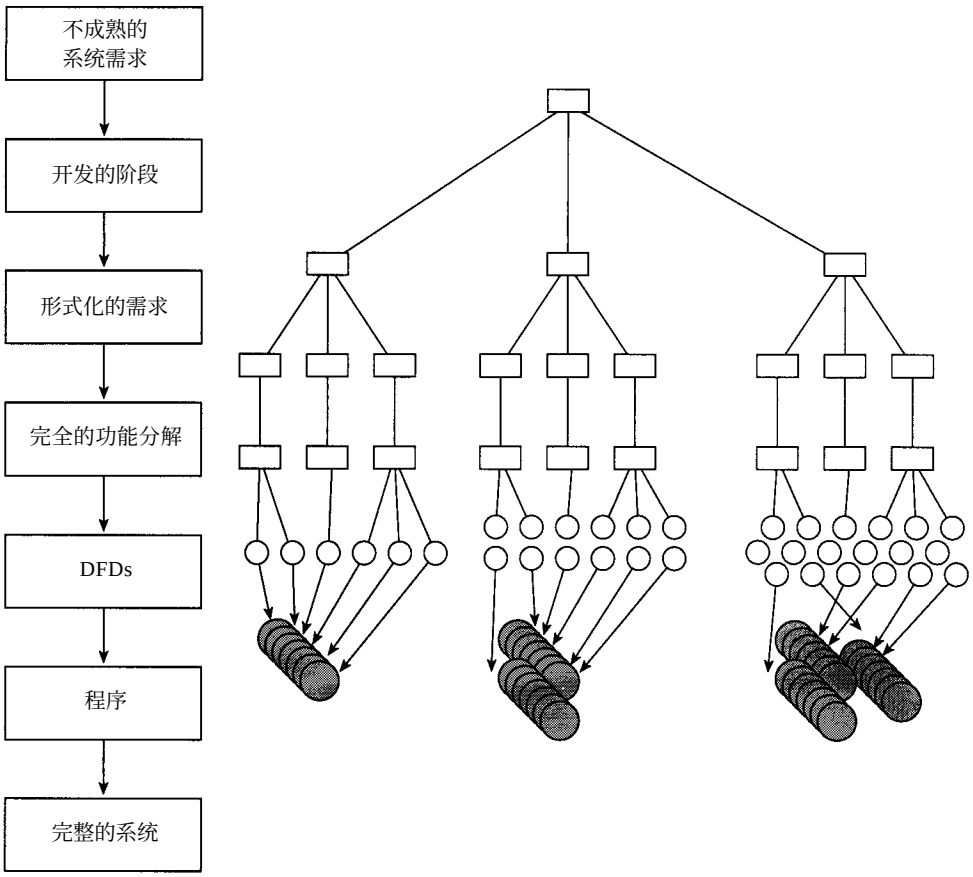


图A-11 方法之十一，操作型处理分析提交的线性流程

确实存在产生提交的线性流程的原因。然而，线性流程掩盖了两个重要方面：

- 提交通常以重复的方式产生。
- 在任何给定的层次有多重提交。换句话说，任何层次的提交能够在下一个低的层次上产生多个提交，如图A-12所示。

图A-12显示单一的需求定义导致三个开发阶段。每个开发阶段经过需求定义形式化再进行分解。从分解中，多个活动被标识了，并为每个活动创建一个 DFD。接下来，每个DFD产生一个或多个程序。最终，这些程序构成了完整系统的构架。

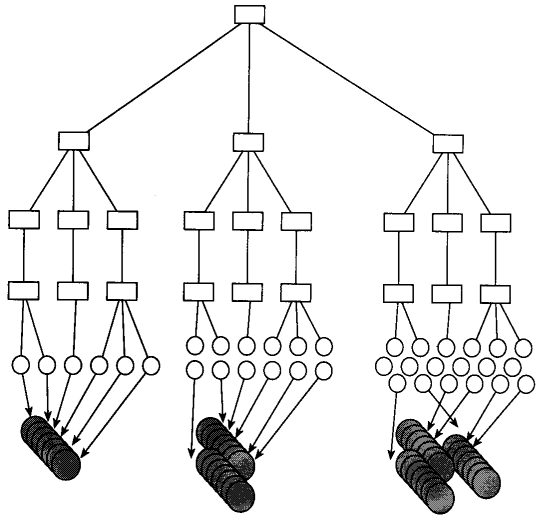


图A-12 方法之十二，提交通常在低层次扩散为多个提交

A.6.3 估计开发需要的资源

看一下图A-12，当准确地指明了要产生多少个提交后，就能够合理地估计开发过程需要多少资源了。

图A-13显示了一个简单的技术，首先定义每个层次的提交，这样整个提交的数目就可以知道了。然后建立每个提交所需的时间乘以每个提交的资源，产生一个需要雇员资源量的估计。



图A-13 方法之十三，估计系统的开发时间

A.7 SDLC/CLDS

前面的讨论提到这样一个事实，操作型系统在一种系统开发生命周期下创建，而 DSS系

图A-14 方法之十四

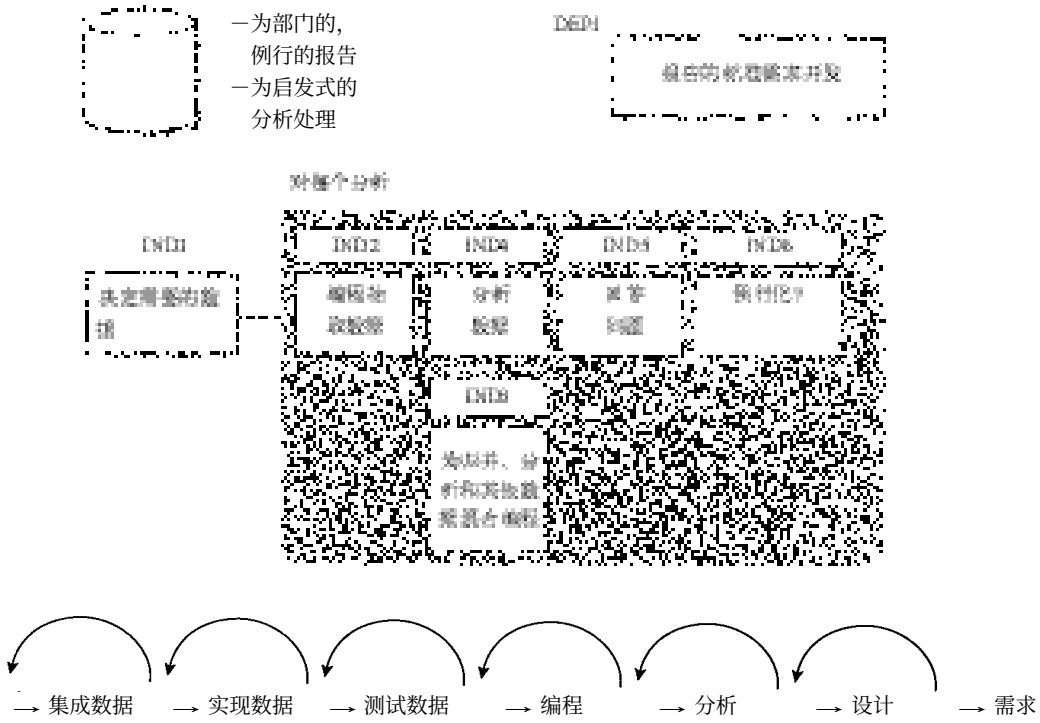
统在另一种系统开发生命周期下创建。图 A-14显示操作型系统的开发生命周期，需求是起点。下面的活动包括分析、设计、编程、测试、集成、实施和维护。

和DSS系统相关的系统开发生命周期在图 A-15显示，其中DSS处理从数据开始。当分析的数据取得后(通常使用数据仓库)，接下来是编程、分析等后续步骤。 DSS数据的开发生命周期的结束是一个可理解的需求。

A.7.1 数据字典

图A-16描述了数据字典所扮演的角色。

在操作型处理中的ERD开发和文件编制、 DIS开发、物理数据库设计和编码等活动中，数据字典起着核心的作用。数据字典在数据仓库开发的世界中的数据模型分析、主题领域选择、源系统选择(记录标识系统)和编程中也起着重要作用。

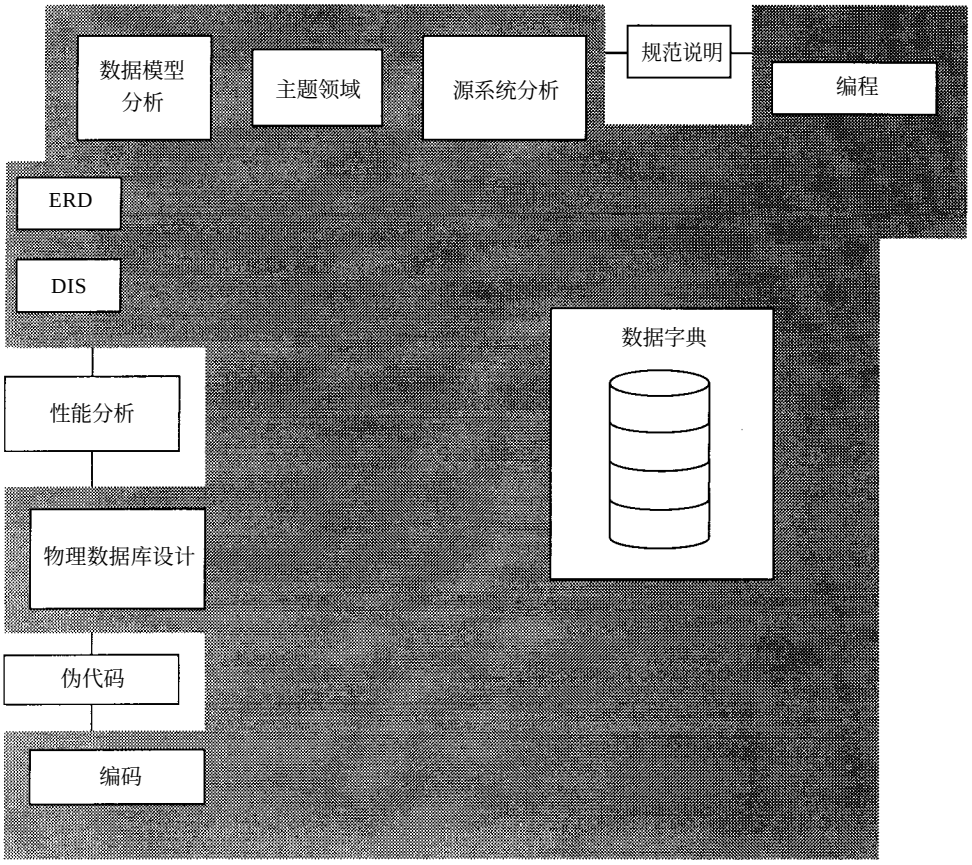


图A-15 方法之十五

A.7.2 现有系统怎样？

极少情况下，开发工作是在没有现有系统储备下全新进行的。现有系统确实没有给数据 -驱动开发方法的DSS部件带来任何问题。找到现有系统中的记录系统作为数据仓库数据的基础是最通常的事情。

然而，对操作型环境中的现有系统需要作点说明。第一种使用现有操作型系统的方法是试图在其上建造系统。如果这样是可能的，就会更富有生产力。但是很多情况不能在现有操作型系统之上建造。



图A-16 方法之十六，数据仓库开发

第二种方法是试图修改现有操作型系统。有些情况可行，但大多数情况不可行。

第三种方法是大规模替换并且加强现有的操作型系统。在这种情况下，现有操作型系统除了能为收集需求服务外别无用处。

大规模替换的一个变种是对现有操作型系统的一部分或全部进行转化。这要有一个基本限制，这就是现有系统小而且简单。现有操作型系统越大、越复杂，系统转换的可能性就越小。