第二章 软件需求分析

2.1 软件需求分析的基本概念

软件需求分析也称为需求分析工程，是软件生命期中重要的一步，也是决定性的一步。在可行性分析阶段，对开发新系统的基本思想和过程进行了初步分析和论证，对系统的基本功能、性能及开发时间的限制，人员安排、投资情况等作出了客观的分析。

在需求分析阶段，要对经过可行性分析所确定的系统目标和功能作进一步的详细论述，确定系统“做什么？”的问题。

2.1.1 软件需求分析的任务

软件需求分析关系到软件系统开发的成败，是决定软件产品质量的关键。只有通过需求分析才能把软件功能和性能的总体概念描述为具体的软件需求规格说明，从而奠定软件开发的基础。要在可行性分析的基础上，进一步确定用户的需求。

需求分析的基本任务是：

要准确地定义新系统的目标，为了满足用户需求，回答系统必须“做什么”的问题。获得需求规格说明书。

为了更加准确地描述需求分析的任务，Boehm 给出软件需求的定义：

研究一种无二义性的表达工具，它能为用户和软件人员双方都接受，并能够把“需求”严格地、形式地表达出来。

对于大、中型的软件系统，很难直接对它进行分析设计，人们经常借助模型来分析设计系统。模型是现实世界中的某些事物的一种抽象表示，抽象的含义是抽取事物的本质特性，忽略事物的其他次要因素。因此，模型既反映事物的原型，又不等于该原型。模型是理解、分析、开发或改造事物原型的一种常用手段。例如，建造大楼前常先做大楼的模型，以便在大楼动工前就能使人们对未来的大楼有一个十分清晰的感性认识，显然，大楼模型还可以用来改进大楼的设计方案。

由于需求分析方法不同，描述形式不同。图 2.1 描述了需求分析一般的实现步骤。

⑴获得当前系统的物理模型。物理模型是对当前系统的真实写照，可能是一个由人工操作的过程，也可能是一个已有的但需要改进的计算机系统。首先是要对现行系统进行分析、理解，了解它的组织情况、数据流向、输入输出，资源利用情况等，在分析的基础上画出它的物理模型。

⑵抽象出当前系统的逻辑模型。逻辑模型是在物理模型的基础上，去掉一些次要的因素，建立起反映系统本质的逻辑模型。

⑶建立目标系统的逻辑模型。在分析目标系统与当前系统在逻辑上的区别，建立符合用户需求的目标系统的逻辑模型。

⑷补充目标系统的逻辑模型。对目标系统进行补充完善，将一些次要的因素补充进去，例如出错处理。

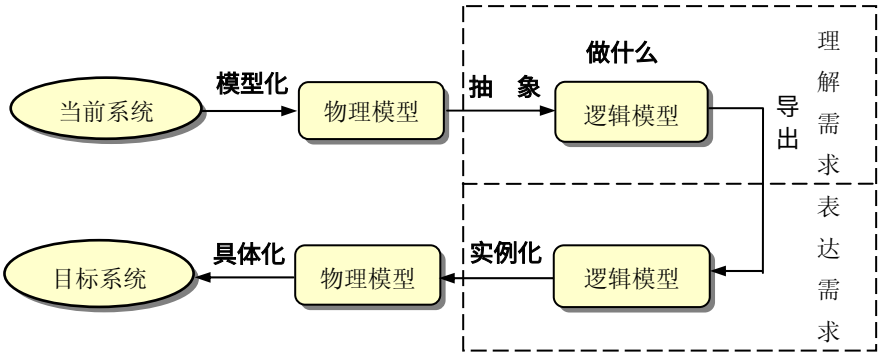


图 2.1 需求分析步骤

根据上述分析得知，需求分析的具体任务是：

1．确定系统的综合要求

- 确定系统功能要求—这是最主要的需求，确定系统必须完成的所有功能。
- 确定系统性能要求—应就具体系统而定，例如可靠性、联机系统的响应时间、存储容量、安全性能等。
- 确定系统运行要求—主要是对系统运行时的环境要求；如系统软件、数据库管理系统、外存和数据通信接口等。
- 将来可能提出的要求—对将来可能提出的扩充及修改作预准备。

2．分析系统的数据要求

软件系统本质上是信息处理系统，因此，必须考虑：

- 数据 （需要哪些数据、数据间联系、数据性质、结构）
- 数据处理 （处理的类型、处理的逻辑功能）

3．导出系统的逻辑模型—通常系统的逻辑模型用 DFD 图来描述。

4．修正系统的开发计划—通过需求对系统的成本及进度有了更精确的估算，可进一步修改开发计划。

2.1.2 需求分析的过程

需求分析阶段的工作，可以分为以下四步：

1．问题识别

双方确定问题的综合需求。这些需求包括功能需求（最主要的需求）、性能需求、环境需求和用户界面需求，另外还有可靠性、安全性、保密性、可移植性和可维护性等方面的需求。

2．分析与综合

导出软件的逻辑模型。

3．编写文档

⑴编写“需求说明书”，把双方共同的理解与分析结果用规范的方式描述出来

⑵编写初步用户使用手册，

⑶编写确认测试计划，

⑷修改完善项目开发计划。

4．分析评审

作为需求分析阶段工作的复查手段，应该对功能的正确性、完整性和清晰性，以及其他需求给予评价。

2.1.3 软件需求分析的原则

近几年来已提出许多软件需求分析与说明的方法，每一种分析方法都有独特的观点和表示方法，但无论哪种分析方法都适用下面的基本原则：

1．能够表达和理解问题的信息域和功能域

对于计算机程序处理的数据，其信息域包括信息流(如图 2.3)，即数据通过一个系统时的变化方式、信息内容和信息结构，而功能域反映上述三方面的控制信息。

2．能够对问题进行分解和不断细化，建立问题的层次结构。

3．需要给出系统的逻辑视图和物理视图。

软件需求的逻辑视图给出的是软件要达到的功能和要处理信息之间的关系，而不是实现的细节。

软件需求的物理视图给出的是处理功能和信息结构的实际表现形式，这往往是由设备本身决定的。

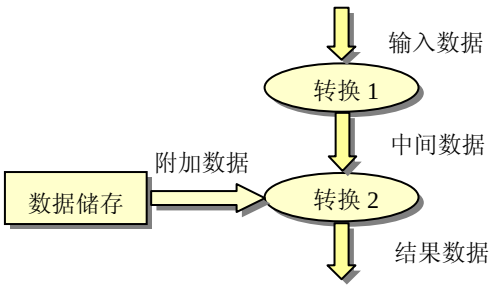


图 2.3 信息流

在图 2.3 中，输入数据首先转换成中间数据，然后转换成输出数据。在此期间可以从已有的数据存储（如磁盘文件或内存缓冲）中引入附加数据。对数据进行转换是程序中应有的功能或子功能。两个转换功能之间的数据传递就确定了功能间的接口。

2.1.4 需求分析方法

不同的开发方法，需求分析的方法也有所不同，常见的分析方法有：**1．功能分析方法**

将系统看作若干功能模块的集合，每个功能又可以分解为若干子功能,子功能还可继续分解。分解的结果已经是系统的雏形。

2．结构化分析方法

是一种以数据、数据的封闭性为基础，从问题空间到某种表示的映射方法，由数据流图（DFD 图）表示。在下一节，将对结构化分析方法进行具体介绍，

3．信息建模法

是从数据的角度对现实世界建立模型的。大型信息系统通常十分复杂，很难直接对它进行分析设计，人们经常借助模型来设计分析系统。

模型是开发过程中的一个不可缺少的工具。信息系统包括数据处理、事务管理和决策支持。实质上，信息系统可以看成是由一系列有序的模型构成的，这些有序模型通常为：功能模型、信息模型、数据模型、控制模型和决策模型，所谓有序是指这些模型是分别在系统的不同开发阶段、不同开发层次上建立的。建立信息系统常用的基本工具是 ER 图。

4．面向对象的分析方法

面向对象的分析方法（OOA）的关键是识别问题域内的对象，分析它们之间的关系，并建立起三类模型：对象模型、动态模型和功能模型。有关 OOA 的方法将在第四、五章进行讨论。

2.2 结构化分析（SA）方法

结构化开发方法（Structured Developing Method）是现有的软件开发方法中最成熟，应用最广泛的方法，主要特点是快速、自然和方便。结构化开发方法由结构化分析方法（SA 法）、结构化设计方法（SD 法）及结构化程序设计方法（SP 法）构成的。

结构化分析（Structured Analysis，简称 SA 法）方法是面向数据流的需求分析方法，是 70 年代末由 Yourdon,Constaintine 及 DeMarco 等人提出和发展，并得到广泛的应用。它适合于分析大型的数据处理系统，特别是企事业管理系统。

SA 法也是一种建模的活动，主要是根据软件内部的数据传递、变换关系，自顶向下逐层分解，描绘出满足功能要求的软件模型。

2.2.1 SA 法概述

1．SA 法的基本思想

结构化分析（Structured Analysis，简称 SA 法）是面向数据流的需求分析方法，是 70 年代由 Yourdon,Constaintine 及 DeMarco 等人提出和发展，并得到广泛的应用。

结构化分析方法的基本思想是“分解”和“抽象”。

分解：是指对于一个复杂的系统，为了将复杂性降低到可以掌握的程度，可以把大问题分解成若干小问题，

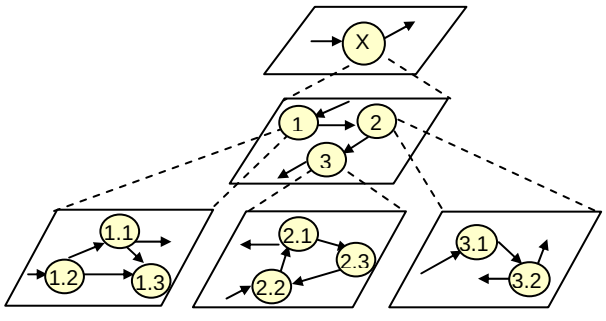


图 2.4 自顶向下逐层分解

然后分别解决。（如图 2.4）

图 2.4 是自顶向下逐层分解的示意图。顶层抽象地描述了整个系统，底层具体地画出了系统的每一个细节，而中间层是从抽象到具体的逐层过渡。

抽象：分解可以分层进行，即先考虑问题最本质的属性，暂把细节略去，以后再逐层添加细节，直至涉及到最详细的内容，这种用最本质的属性表示一个自系统的方法就是“抽象”。

2．SA 法的步骤

⑴建立当前系统的“具体模型”；

系统的“具体模型”就是现实环境的忠实写照，即将当前系统用 DFD 图描述出来。这样的表达与当前系统完全对应，因此用户容易理解。

⑵抽象出当前系统的逻辑模型；

分析系统的“具体模型”，抽象出其本质的因素，排除次要因素，获得用 DFD 图描述的当前系统的“逻辑模型”。

⑶建立目标系统的逻辑模型；

分析目标系统与当前系统逻辑上的差别，从而进一步明确目标系统“做什么”，建立目标系统的“逻辑模型”（修改后的 DFD 图）。

⑷为了对目标系统作完整的描述，还需要考虑人机界面和其它一些问题。

3．SA 法的描述工具

⑴ 分层的数据流图

⑵ 数据词典

⑶ 描述加工逻辑的结构化语言、判定表或判定树。

2.2.2 数据流图

数据流图（Data Flow Diagram，简称 DFD）是描述系统中数据流程的图形工具，它标识了一个系统的逻辑输入和逻辑输出，以及把逻辑输入转换逻辑输出所需的加工处理。

1．数据流图的图符数据流图有以下 4 种基本图形符号：

图 2.5 DFD 图的基本符号

箭头表示数据流，圆或椭圆表示加工。双杠或者单杠表示数据存储，矩形框表示数据的源点或终点，即外部实体。

⑴数据流 是数据在系统内传播的路径，由一组成固定的数据项组成。除了与数据存储（文件）之间的数据流不用命名外，其余数据流都应该用名词或名词短语命名。数据流

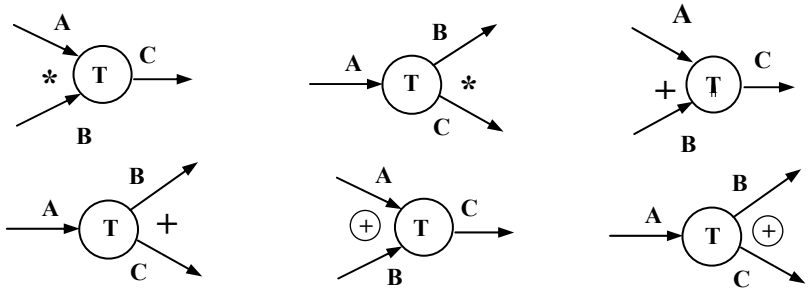
可以从加工流向加工，也可以从加工流向文件或从文件流向加工，也可以从源点流向加工或从加工流向终点。

⑵ 加工　也称为数据处理，它对数据流进行某些操作或变换。每个加工也要有名字，通常是动词短语，简明地描述完成什么加工。在分层的数据流图中，加工还应有编号。

⑶ 数据存储　指暂时保存的数据，它可以是数据库文件或任何形式的数据组织。流向数据存储的数据流可理解为写入文件，或查询文件，从数据存储流出的数据可理解为从文件读数据或得到查询结果。

⑷ 数据源点和终点　是软件系统外部环境中的实体（包括人员、组织或其他软件系统），统称为外部实体。一般只出现在数据流图的顶层图中。

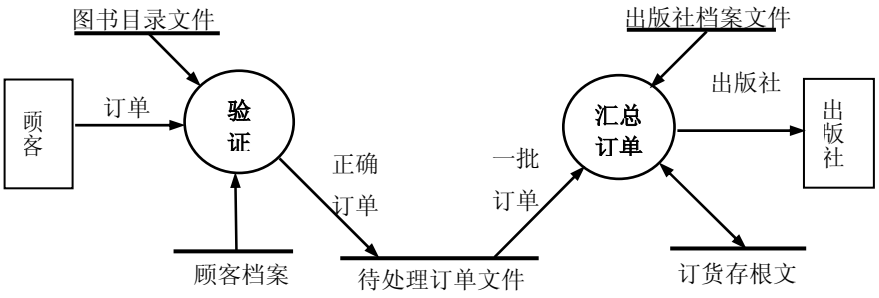
还有一些辅助的图例：



其中：＊　表示与，＋　表示或，⊕ 表示互斥

图 2.6 DFD 图的辅助符号

例 1：画出图书预定系统的 DFD 图。现有一图书预定系统，接收由顾客发来的订单，并对订单进行验证，验证过程是根据图书目录检查订单的正确性，同时根据顾客档案确定是新顾客还是老顾客，是否有信誉。经过验证的正确订单，暂存放在待处理的订单文件中。对订单进行成批处理，根据出版社档案，将订单按照出版社进行分类汇总，并保存订单存根，然后将汇总订单发往各出版社。



图书预定系统的 DFD 图

画图步骤是：

⑴ 首先确定外部实体（顾客、出版社）及输入、输出数据流（订单、出版社订单）。

⑵ 再分解顶层的加工（验证订单、汇总订单）。

⑶ 确定所使用的文件（图书目录文件、顾客档案等 5 个文件）。

⑷ 用数据流将各部分连接起来，形成数据封闭。

特别要注意的是：数据流图不是传统的流程图或框图，数据流也不是控制流。数据流图是从数据的角度来描述一个系统，而框图则是从对数据进行加工的工作人员的角度来描述系统。数据流图中的箭头是数据流，而框图中的箭头则是控制流，控制流表达的是程序执行的次序。

下图是培训中心管理系统的数据流图，由于只有一层，因此分解的加工较多不易理解，而且如果其中某个加工较复杂，例如编号为 3 的加工“付款”和编号为 7 的加工“复审”仍很复杂，一时难以理解，如果不继续分解下去，直到每个加工都足够简单易于理解为止，则会影响需求分析结果的可读性。

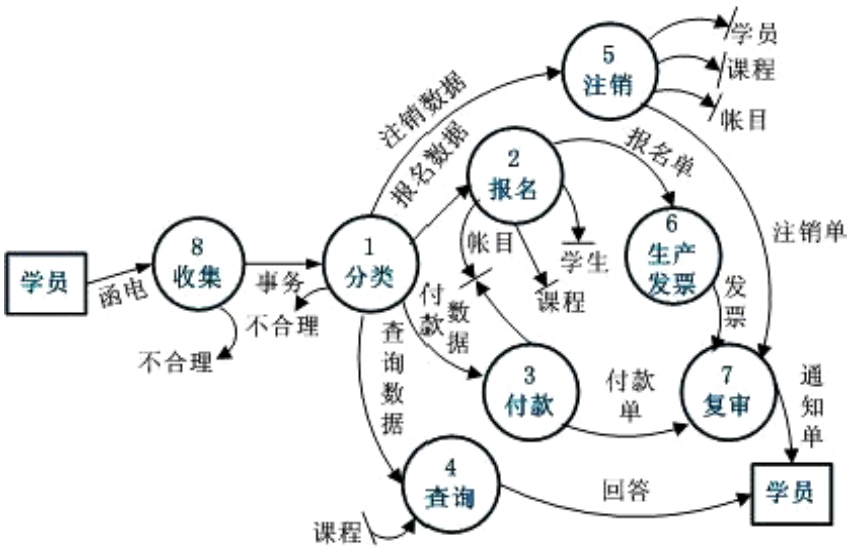


图 2.8 培训中心管理系统的 DFD 图

2．画分层 DFD 图的方法

如图 2.8 所示，如果系统规模较大，仅用一个 DFD 图难以描述，会使得系统变得复杂，且难以理解。为了降低系统的复杂性，采取“逐层分解”的技术，画分层的 DFD 图。

画分层 DFD 图的一般原则是：“先全局后局部,先整体后细节,先抽象后具体”。通常将这种分层的 DFD 图,分为顶层、中间层、底层。顶层图说明了系统的边界,即系统的输入和输出数据流，顶层图只有一张。底层图由一些不能再分解的加工组成，这些加工都已足够简单，称为基本加工。在顶层和底层之间的是中间层。中间层的数据流图描述了某个加工的分解，而它的组成部分又要进一步分解。 画各层 DFD 图时，应“由外向内”。

画分层 DFD 图的具体步骤：

⑴ 先确定系统范围，画出顶层的 DFD 图。

⑵ 逐层分解顶层 DFD 图，获得若干中间层 DFD 图。

⑶ 画出底层的 DFD 图。

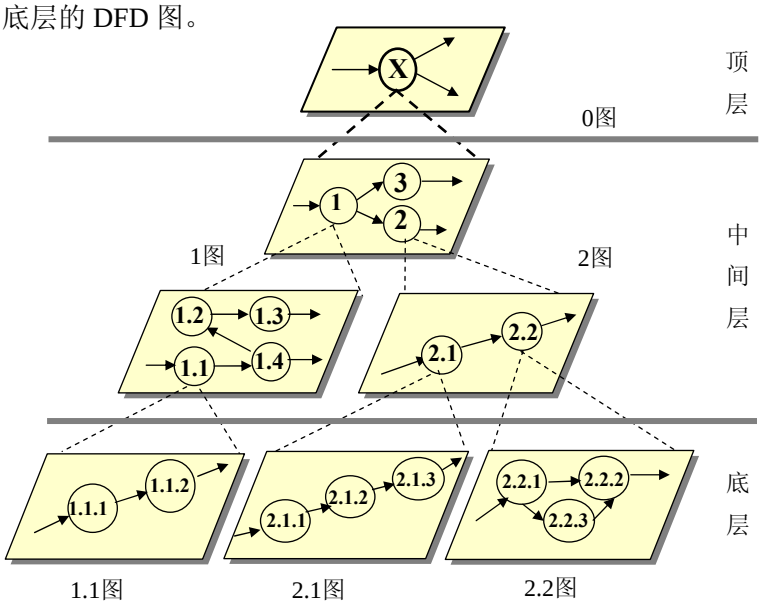


图 2.8 DFD 图

在画分层数据流图时，首先遇到的问题就是应该如何分解？不能够一下子把一个加工分解成它所有的基本加工，一张图中画出过多的加工是使人难以理解的，但是如果每次只是将一个加工分解成两个或三个加工，又可能需要分解过多的层次，也会影响系统的可理解性。

一个加工每次分解成多少个子加工才合适呢?

根据经验“最多不要超过 7 个”。统计结果证明，人们能有效地同时处理 7 个或 7 个以下的问题，但当问题多于 7 个时，处理效果就会下降。当然也不能机械地应用，关键是要使数据流图易于理解。

同时还有几条原则可供参考：

- 分解应自然，概念上要合理、清晰。
- 只要不影响数据流图的“易理解性”，可以适当地多分解成几部分，这样分层图的层数就可少些。

一般来说，在上层可以分解得快些，而在中、下层则应分解得慢些，因为上层是一些综合性的描述，“易理解性”相对地说不太重要。

下节我们以一个实例来说明画分层 DFD 图的方法。

2.2.3 实例—医院病房监护系统

实例：画出“医院病房监护系统”的分层 DFD 图。

如图 2.9 所示，病症监视器安置在每个病床，将病人的组合病症信号（例如由血压、脉搏、体温组成）实时地传送到中央监护系统进行分析处理。在中心值班室里，值班护士

使用中央监护系统对病员的情况进行监控，监护系统实时地将病人的病症信号与标准的病症信号进行比较分析，当病症出现异常时，系统会立即自动报警，并打印病情报告和更新病历。同时，根据医生的要求随时打印病人的病情报告，系统定期自动更新病历。

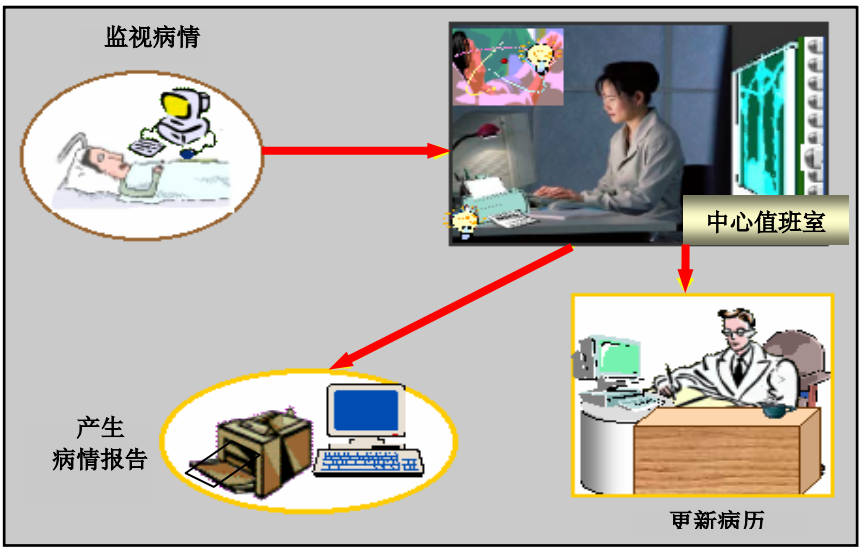


图 2.9 医院病房监护系统的场景

经过初步的需求分析，得到系统的主要功能要求：

⑴监视病员的病症（血压、体温、脉搏等）

⑵定时更新病历

⑶病员出现异常情况时报警。

⑷随机地产生某一病员的病情报告。

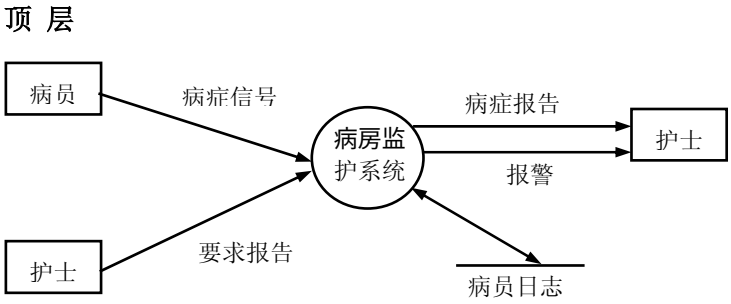


图 2.10 病房监护系统顶层 DFD 图

根据分析得到的系统功能要求，画出医院病房监护系统的分层 DFD 图，首先画顶层的 DFD 图（图 2.10）。顶层确定了系统的范围，其外部实体为病员和护士。

在顶层 DFD 图的基础上再进行分解，得到第一层的 DFD 图（图 2.11）。

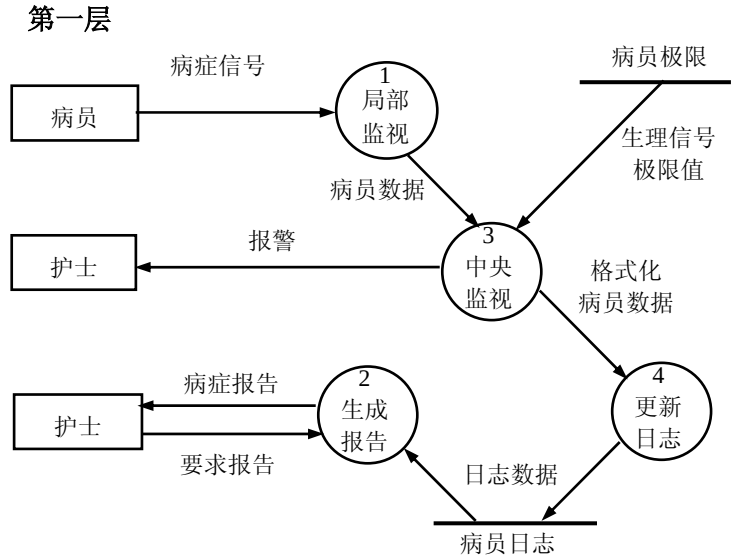


图 2.11 病房监护系统一层 DFD 图

第一层分解为局部监视、生成报告、中央监视、更新日志 4 个加工。这层的分解是关键，是根据初步的需求分析所得到系统主要功能要求来进行分解的。

1. 加工“局部监视”，是用于监视病员的病症（血压、体温、脉搏等），因此该加工一定有来自病员的输入数据流“病症信号”，输入的病症信号是模拟信号，经过“局部监视”加工后，转换为数字信号，因此该加工应该有数值型的输出数据流“病员数据”。

2. 加工“中央监视”，显然是系统中最重要的加工。它要接受来自局部监视的病员数据，同时要将病员数据与标准的“病员极限文件”中的“生理信号极限值”进行比较，一旦超过，立即报警。为了定时更新病历，还需要输出“格式化病员数据”。

3. 定时更新病历的功能就由加工“更新日志”完成，它接收由加工“中央监视”输出的已格式化病员数据，对数据进行整理分类等加工后写入“病员日志文件”（病历），日志数据通常是以压缩的二进制代码存储的。

4. 为了实现根据医生要求，随机地产生某一病员的病情报告，需要从病员日志文件中提取病员日志数据，由加工“生成报告”进行格式转换，生成并打印输出“病症报告”。

在第一层分解的基础上，应对 4 个加工进行进一步分解，以 4 个加工中最重要的加工“中央监视”为例，进行第二层分解。如图 2.12 所示，为了将病员数据与生理信号极限值进行比较，首先要将来自“局部监视”加工的病员数据进行开解。在进行“格式化病员数据”加工时，还要由时钟加入日期和时间。

第二层：加工“中央监视”分解

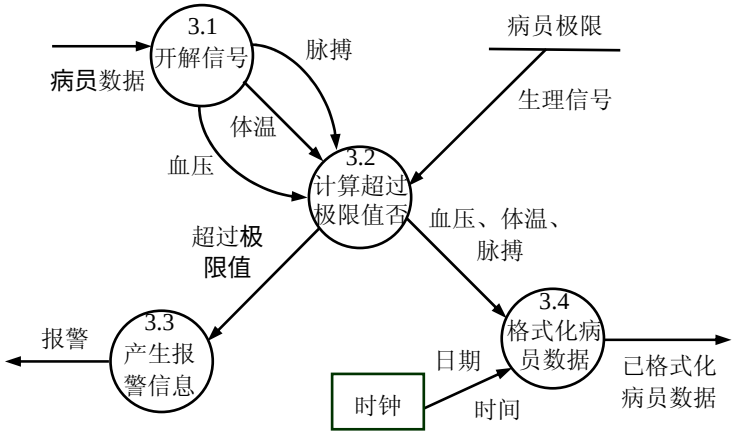


图 2.12 加工“中央监视”分解

2.2.4 分层 DFD 图的改进

分层数据流图是一种比较严格又易于理解的描述方式，它的顶层描绘了系统的总貌，底层画出了系统所有的细部，而中间层则给出了从抽象到具体的逐步过渡。

1．画分层 DFD 图的基本原则

⑴．数据守恒与数据封闭原则

所谓数据守恒是指加工的输入输出数据流是否匹配，即每一个加工既有输入数据流又有输出数据流。或者说一个加工至少有一个输入数据流，一个输出数据流。

⑵加工分解的原则

自然性：概念上合理、清晰；

均匀性：理想的分解是将一个问题分解成大小均匀的几个部分；

分解度：一般每一个加工每次分解最多不要超过 7 个子加工,应分解到基本加工为止。

⑶子图与父图的“平衡”父图中某个加工的输入输出数据流应该同相应的子图的输入输出相同(相对应)，分层数据流图的这种特点称为子图与父图“平衡”。

例 1 考察下图中子父图的平衡

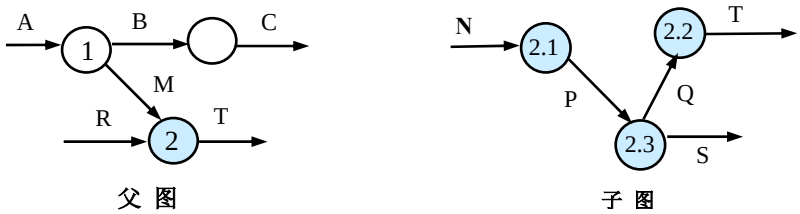


图 2.13 子图与父图

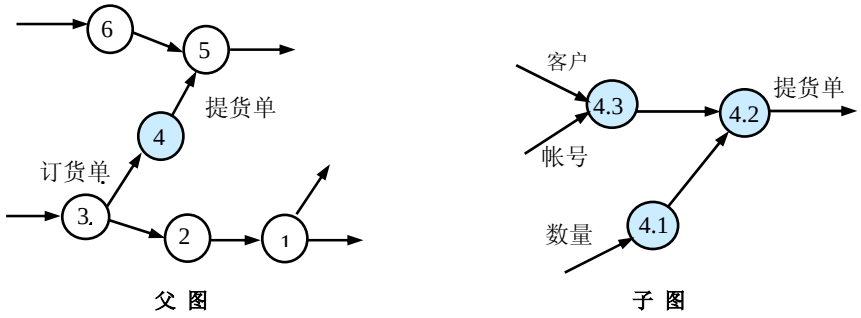


图 2.14 子图与父图的平衡

显然，图 2.13 中子图与父图不平衡。子图是父图中加工 2 的分解，加工 2 有输入数据流 R 和 M，输出数据流 T，而子图则只有一个输入数据流 N，却有两个输出数据流 T 与 S。图 2.14 中，子图是父图中加工 4 的分解，虽然表面上加工 4 只有一个输入数据流“订货单”，而子图却有三个输入数据流，但是如果“订货单”是由“客户”、“帐号”和“数量”三部分组成，即有如下数据条目：订货单 =客户+帐号+数量（2.2.5 数据词典），则子、父图平衡。

⑷合理使用文件

当文件作为某些加工之间的交界面时，文件必须画出来，一旦文件作为数据流图中的一个独立成份画出来了，那么它同其它成份之间的联系也应同时表达出来。

理解一个问题总要经过从不正确到正确，从不确切到确切的过程，需求分析的过程总是要不断反复的，一次就成功的可能性是很小的，对复杂的系统尤其如此，因此，系统分析员应随时准备对数据流图进行修改和完善，与用户取得共识，获得无二义性的需求，才能获得更正确清晰的需求说明，使得设计、编程等阶段能够顺利进行，这样做是必须和值得的。

2．分层 DFD 图的改进

DFD 图必须经过反复修改，才能获得最终的目标系统的逻辑（目标系统的 DFD 图）。改进的原则与画分层 DFD 图的基本原则是一致的，可从以下方面考虑 DFD 图的改进：

⑴ 检查数据流的正确性

① 数据守恒

② 子图、父图的平衡

③ 文件使用是否合理。特别注意输入/出文件的数据流。

⑵ 改进 DFD 图的易理解性

① 简化加工之间的联系（加工间的数据流越少，独立性越强，易理解性越好）。

② 改进分解的均匀性。

③ 适当命名（各成分名称无二义性，准确、具体）。

2.2.5 数据词典

分层数据流图只是表达了系统的“分解”，为了完整地描述这个系统，还需借助“数据词典”(data dictionary)和“小说明”对图中的每个数据和加工给出解释。

对数据流图中包含的所有元素的定义的集合构成了数据词典。它有四类条目：数据流、数据项、文件及基本加工。在定义数据流或文件时，使用表 2-1 给出的符号。将这些条目按照一定的规则组织起来，构成数据词典。

表 2-1 在数据词典的定义中出现的符号

符　号	含　义	例及说明
=	被定义为	
+	与	X=a + b 表示 X 由 a 和 b 组成
[•••|•••]	或	X=[a | b] 表示 X 由 a 或 b 组成
{•••}	重复	X={a} 表示 X 由 0 个或多个 a 组成
m{•••}n 或 $\{\bullet\bullet\bullet\}_m^n$	重复	X=2{a}6 或 $x=\{a\}_2^6$ 表示重复 2—6 次 a
(•••)	可选	X=(a) 表示 a 可在 X 中出现，也可不出现
“•••”	基本数据元素	X=“a” 表示 X 是取值为字符 a 的数据元素
••	连接符	X=1••8 表示 X 可取 1 到 8 中的任意一个值

1．数据流条目

给出了 DFD 中数据流的定义，通常对数据流的简单描述为列出该数据流的各组成数据项。

例：数据流“乘客名单”由若干“乘客姓名”、“单位名”和“等级”组成，则词典中的“乘客名单”条目是：

乘客名单＝{乘客姓名＋单位名＋等级}

又如：报名单＝姓名＋单位名＋年龄＋性别＋课程名

也可以对数据流进行较详细的描述，如下例：

例：某查询系统中，有个名为“查询”的数据流，目前“查询”有三种类型，即“顾客状况查询”、“存货查询”和“发票存根查询”，预计至 1990 年底还将增加 3 至 4 种其他类型的查询。系统每天约需处理 2 000 次查询，每天上午 9：00—10：00 是查询的高峰，此时约有 1 000 次查询。上述信息都是“用户要求”的一部分，在分析阶段应该认真收集，并记录在词典的有关条目中，所以“查询”条目描述如下。

数据流名：查询

简　述：系统处理的一个命令

别　名：无

组　成：[顾客状况查询|存货查询|发票存根查询]

数据量：2000 次/天

峰　值：每天上午 9：00—l0：00 有 l000 次

注　释 ：　至 1990 年底还将增加 3 至 4 种查询

2．文件条目

给出某个文件的定义，文件的定义通常是列出文件记录的组成数据流，还可指出文件的组织方式。

例：某销售系统的订单文件：

订单文件＝订单编号＋顾客名称＋产品名称＋订货数量＋交货日期

3．数据项条目

给出某个数据单项的定义，通常是该数据项的值类型、允许值等。

例如：帐号= 00000 ~ 99999 ；　存款期=[1 | 3 | 5] （单位：年）**4．加工条目**

加工条目就是“加工小说明”。由于“加工”是 DFD 图的重要组成部分，一般应单独进行说明。

因此，数据词典是对数据流图中所包含的各种元素定义的集合。它对的 4 类条目：数据流、数据项、文件及基本加工进行了描述，是对 DFD 图的补充。

2.2.6 加工逻辑说明

加工逻辑又称为“加工小说明”，对数据流图中每一个不能再分解的基本加工都必须有一个“加工小说明”给出这个加工的精确描述。小说明中应精确地描述加工的激发条件、加工逻辑、优先级、执行频率和出错处理等。加工逻辑是其中最基本的部分，是指用户对这个加工的逻辑要求。

对基本加工说明有三种描述方式：结构化语言，判定表和判定树。在使用时可以根据具体情况，选择其中一种方式对加工进行描述。

1．结构化语言

结构化语言是介于自然语言（英语或汉语）和形式语言之间的一种半形式语言，它是自然语言的一个受限制的子集。自然语言容易理解，但容易产生二义性，而形式化语言精确、无二义性，却难理解，不易掌握。结构化语言则综合二者的优点，在自然语言的基础上加上一些约束；一般分为两层结构：外层语法较具体，为控制结构（顺序、选择、循环），内层较灵活，表达“做什么”。

特点：简单、易学、少二义性，但不好处理组合条件。

例如：外层可为以下结构：

1．顺序结构

2．选择结构

IF–THEN-ELSE;　CASE-OF-ENDCASE；

3．循环结构

WHILE-DO; REPEAT-UNTIL

结构化语言举例：

例一 “确定能否供货”的加工逻辑：

根据库存记录

IF 订单项目的数量<该项目库存量的临界值

THEN 可供货处理

ELSE 此订单缺货，登录，待进货后再处理

ENDIF

例二 根据当前流动资金值确定贬值数。

IF the Current–Capital–Value is less then $1000

Then

Set Depreciated–Amount to Current–Capital–Value.

Set Current–Capital–Value to zero.

Otherwise

Set Depreciated–Amount to 10% of Current–Capital–Value.

Reduce Current –Capital-Value by 10%.

结构化语言特点:

简单，易学，少二义性。不好处理组合条件。

二. 判定表

判定表是一种二维的表格，常用于较复杂的组合条件。通常由四部分组成，如表 2-2。可以处理用结构化语言不易处理的，有较复杂的组合条件的问题。

表 2-2

条件框	条件条目
操作框	操作条目

条件框 — 条件定义。

操作框 — 操作的定义。

条件条目 — 各条件的取值及组合。

操作条目 — 在各条件取值组合下所执行的操作。

例如对商店每天的营业额所收税率的描述：

营业额X （¥）	00≤X<500	5000 ≤X<10000	X≥10000
税 率	5%	8%	10%

判定表的特点：

可处理较复杂的组合条件，但不易理解.不易输入计算机。

例：一图书销售系统，其中一加工为“优惠处理”，条件是：顾客的营业额大于 1000 元，同时必须信誉好，或者虽然信誉不好，但是 20 年以上的老主顾。

分析：共有 3 个判定条件，有 8 种可能的组合情况，其中用 Y 表示满足条件，N 表示不满足条件，X 表示选中判定的结论。其判定表如图 2.15。对图 2.15 进行化简，例如情况 1、2 可以合并，即只要是营业额大于 1000 元，同时信誉好，无论是否大于 20 年的老主顾，均可以优惠处理。化简后的判定表为图 2.16。

	1	2	3	4	5	6	7	8
≥1000 元	Y	Y	Y	Y	N	N	N	N
信誉好	Y	Y	N	N	Y	Y	N	N
>20 年	Y	N	Y	N	Y	N	Y	N
优惠处理	X	X	X					
正常处理				X	X	X	X	X

Y-满足条件 N-不满足条件 X-选中判定

图 2.15判定表

化简后

	1	2	3	4	5
≥1000 元	Y	Y	Y	N	N
信誉好	Y	N	N	Y	N
>20 年	—	Y	N	—	—
优惠处理	X	X			
正常处理			X	X	X

图 2.16 化简后的判定表

判定表的特点：

可处理较复杂的组合条件，但不易理解.不易输入计算机。

3．判定树

特点：描述一般组合条件较清晰。不易输入计算机。

例 1 仍然以图书销售系统的处理为例，其判定树如图 2.17 所示。

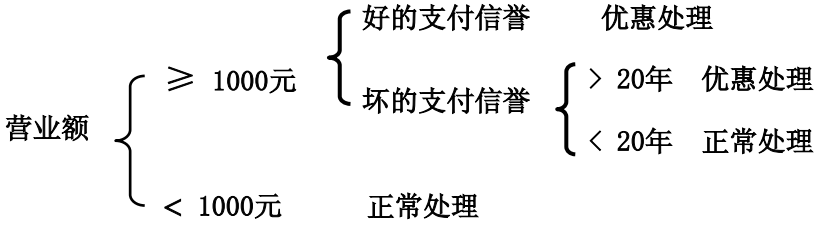


图 2.17 判定树

例 2：计算个体与国营用户的不同折扣量。

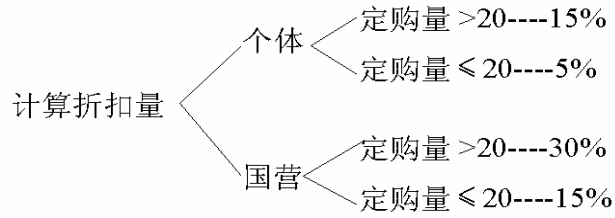


图 2.18 计算折扣量的判定树

判定树特点：描述一般组合条件较清晰。不易输入计算机。

2.3 原型化方法

按照传统的瀑布模型进行软件开发时，是将软件开发这样一个充满回朔的过程硬性地割裂开，虽然强调各个阶段的复审，但由于用户所提出的需求往往是模糊的，因此很难得到一个完整精确的规格说明，这将直接影响到后期的开发。尤其是对于某些试验性、探索性的项目，更是难于得到一个准确、无二义性的需求。针对结构化方法的主要缺点，推出了原型化方法。

什么是原型化方法（Prototyping Method）？

原型是软件开发过程中，软件的一个早期可运行的版本，它反映了最终系统的部分重要特性。

原型化方法的基本思想是花费少量代价建立一个可运行的系统，使用户及早获得学习的机会，原型化方法又称速成原型法（Rapid Prototyping）。强调的是软件开发人员与用户的不断交互，通过原型的演进不断适应用户任务改变的需求。将维护和修改阶段的工作尽早进行，使用户验收提前，从而使软件产品更加适用。

2.3.1 软件原型的分类

通常，原型是指模拟某种产品的原始模型。在软件开发中，原型是软件的一个早期可运行的版本，它反映最终系统的主要特性。例如，建造大楼前常先做大楼的模型，以便在大楼动工前就能使人们对未来的大楼有一个十分清晰的感性认识，显然，大楼模型还可以用来改进大楼的设计方案。

快速原型法(rapid prototyping)是近年来提出的一种以计算机为基础的系统开发方法，它首先构造一个功能简单的原型系统，然后通过对原型系统逐步求精，不断扩充完善得到最终的软件系统。原型就是模型，而原型系统就是应用系统的模型。它是待构筑的实际系统的缩小比例模型，但是保留了实际系统的大部分性能。这个模型可在运行中被检查、测试、修改，直到它的性能达到用户需求为止。因而这个工作模型很快就能转换成原样的目标系统。

由于软件项目的特点和运行原型的目的不同，原型有两种不同的类型：

1．废弃（throw away）型

先构造一个功能简单而且质量要求不高的需求规格模型，针对这个模型系统反复进行分析修改，形成比较好的设计思想，据此设计出更加完整、准确、一致、可靠的最终系统。系统构造完成后，原来的模型系统就被废弃不用。

2．追加（add on）型

先构造一个功能简单而且质量要求不高的模型系统，作为最终系统的核心，然后通过不断地扩充修改，逐步追加新的要求，最后发展成为最终系统。

采用什么形式主要取决于软件项目的特点和开发者的素质，以及支持原型开发的工具和技术。要根据实际情况加以决策。

注意，在使用原型化方法进行软件开发之前，必须明确使用原型的目的，从而决定分

析与构造内容的取舍。

原型法的主要优点在于它是一种支持用户的方法，使得用户在系统生存周期的设计阶段起到积极的作用；它能减少系统开发的风险，特别是在大型项目的开发中,由于对项目需求的分析难以一次完成，应用原型法效果更为明显。

原型法的概念既适用于系统的重新开发，也适用于对系统的修改；快速原型法要取得成功，要求有象第四代语言（4GL）这样的良好开发环境/工具的支持。原型法可以与传统的生命周期方法相结合使用，这样会扩大用户参与需求分析、初步设计及详细设计等阶段的活动，加深对系统的理解。

2.3.2 快速原型开发模型

快速原型法的工作模型如图 2.19 所示，按以下步骤循环执行：

1．快速分析

快速确定软件系统的基本要求，确定原型所要体现的特性（总体结构，功能，性能、界面等）。

2．构造原型

根据基本规格说明，忽略细节，只考虑主要特性，快速构造一个可运行的系统。有三类原型：用户界面原型，功能原型，性能原型。

3．运行和评价原型

用户试用原型并与开发者之间频繁交流，发现问题，目的是验证原型的正确性。

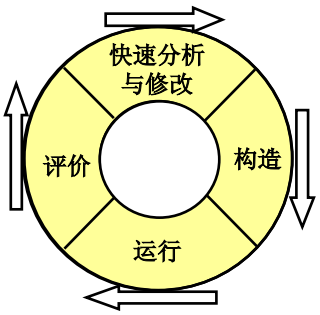


图 2.19 快速原型工作模型

4．修正与改进

对原型进行修改，增删。

对原型生存期的模型进一步细化，图 2.20 是细化的快速原型开发模型，根据该模型原型化开发方法可分为 9 个阶段，各阶段都可能有反复。

⑴ 快速分析

在分析者和用户的紧密配合下，快速确定软件系统的基本要求。

⑵ 构造原型

在快速分析的基础上，根据基本规格说明，尽快实现一个可运行的系统。

⑶ 运行和评价原型

目的是验证原型的正确程度，进而开发新的并修改原有的需求。

⑷ 修改和改进

根据修改意见进行修改。

⑸ 判定原型完成

经过修改或改进的原型，达到参与者的一致认可，则原型开发的迭代过程可以结束。

⑹ 判断原型细部是否说明

判断组成原型的细部是否需要严格地加以说明。

⑺ 原型细部的说明

对于那些不能通过原型说明的所有项目，仍需要通过文件加以说明。

⑻ 判定原型效果

⑼ 整理原型和提供文档

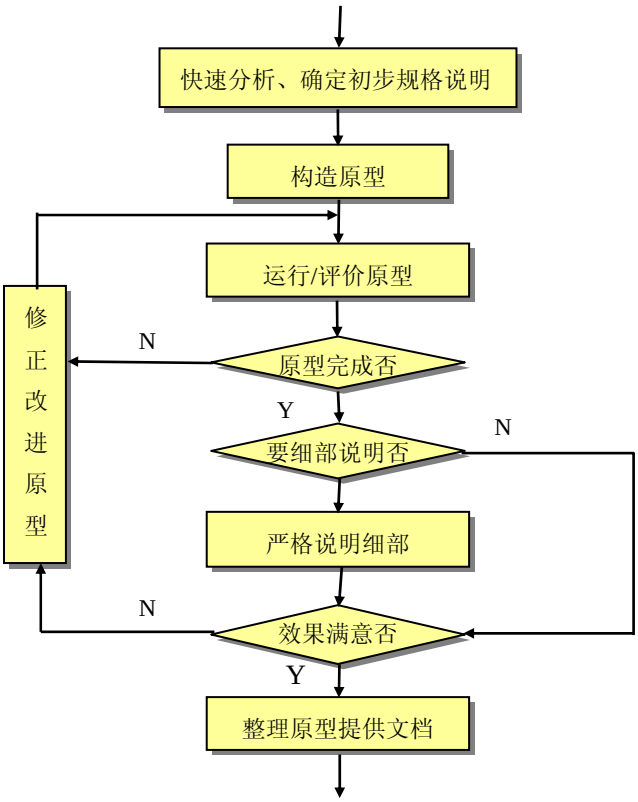


图 2.20 细化的原型模型

快速建立系统原型进行系统的分析和构造有如下优点：

1．增进软件开发人员和用户对系统需求的理解。便于将用户模糊的功能需求明确化。

2．为用户提供了一种强有力的学习手段。

3．易于确定系统的性能，是理解和确认软件需求规格说明的工具。

4．按照 *RCP* 法建立的原型即为最终的产品。

2.4 系统动态分析

一个软件系统在其运行过程中，构成系统的各元素（对象）的状态在改变，对象之间的联系也随时间在改变。例如：在操作系统中，进程 3 种基本状态的变化如图 2.21 所示。为了直观地分析系统的动作，从特定的视点来描述系统的动态特性，必须采用动态的需求分析方法。

常用的动态需求分析方法有状态迁移图、时序图、Petri 网等。本节将介绍状态迁移图和 Petri 网的基本概念。

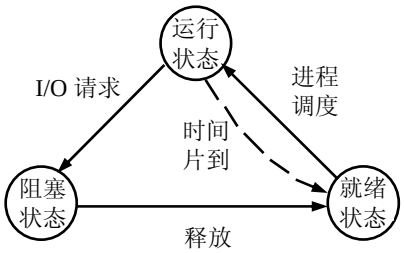


图2.21 进程的状态迁移

2.4.1 状态迁移图

状态迁移图是描述系统的状态如何相对应于外部的信号进行迁移的一种图形表示。

在状态迁移图中，用圆圈“◯”表示可得到的系统状态，用箭头“⟶”表示从一种状态向另一种状态的迁移。在箭头上写上导致迁移的信号或事件的名字。

例：如图 2.22 所示。

状态迁移图表示的关系还可用表格的形式表达，这样的表格成为状态迁移表。例如图 2.23 所示，是与 2.22 等价的状态迁移表。

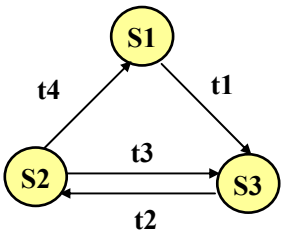


图 2.22 状态迁移图

状态 / 事件	S1	S2	S3
t1	S3		
t2			S2
t3		S3	
t4		S1	

图 2.23 状态迁移表

在图 2.22 的状态迁移图中，系统可取得的状态＝S1，S2，S3，事件＝t1，t2，t3，t4。事件 t1 将引起系统状态 S1 向状态 S 3 迁移，事件 t2 将引起系统状态 S3 向状态 S2 迁移等。

图 2.23 是状态迁移表，第 1 列 t i 表示事件，表中第 i 行第 j 列的元素是一个状态，它

从现在的状态 j 因事件 i 而要移动到下一个状态。例如：第 3 行的元素 t3（事件），将引起状态 S2 向状态 S3 迁移。

如果系统复杂，可以把系统状态迁移图分层表示，这种分层的状态迁移图，不仅对系统的状态及其状态之间的转变进行清晰的描述，还可对某些状态进行进一步的细化。图 2.24 中，在确定了状态 S1，S2，S3 之后，再对状态 S1 进行细化，得到细化的状态 S11 和 S12。

状态迁移图的优点：第一，状态之间的关系能够直观地捕捉到；第二，由于状态迁移图的单纯性，很容易建立相应的分析工具。

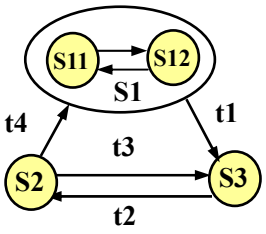


图2.24 状态迁移图的网

2.4.2 PETRI 网

Petri 网的思想是由 C.A.Petri 提出来的。在硬件和软件系统的开发中，它适用于描述与分析相互独立、协同操作的处理系统。在软件需求分析与设计阶段都可以使用 Petri 网。

1．基本概念

Petri 网简称 PNG（petri net graph），是一种有向图，Petri 网由四种元素构成，结点与迁移的输入、输出。它有两种结点：

⑴ 位置（place）：符号为圆圈“○”，可用来表示系统的状态，也可解释为使事件工作的条件，或使之工作的要求。如图 2.25,一组位置 P 为｜P_1，P_2，P_3，P_4，P_5｜。

⑵ 迁移（transition）：符号为短直线“—”或“｜”，可用来表示系统中的事件。如图 2.25 中，一组迁移 T 为｜t_1, t_2, t_3, t_4｜。

图中的有向边表示对迁移的输入，或由迁移的输出。

符号“ ⟶| ”：表示事件发生的前提，即对迁移（事件）的输入。

符号“ |⟶ ”：表示事件的结果，即由迁移（事件）的输出。

通常把迁移的启动称为激发，它是迁移的输出，只有当作为输入的所有位置的条件都满足时才能引起激发。

如图 2.25，给出了一个 PNG 的例子，它表示了一个处于静态状态的系统。只给出系统中各个状态通过迁移而表示出来的相互关系。通过 Petri 网的执行才能完成系统及其行为的完整模型。

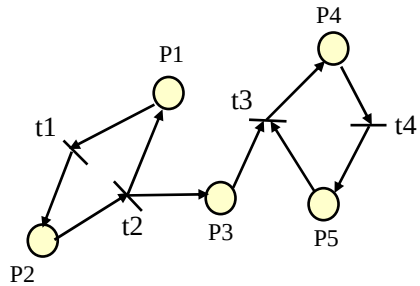
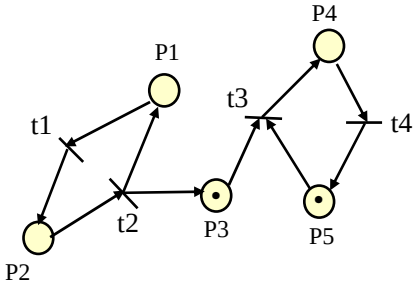


图 2.25 Petri 网

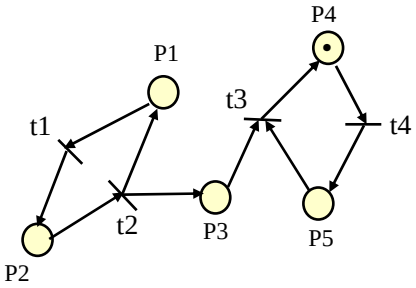


2.26 加标记的 Petri 网

为了描述系统的一种动态的行为，引入了标记（token，也称为令牌）的概念。图 2. 26 表示加了标记的 Petri 网。在图 2.26 中，表示位置 P3 与 P5 的圆圈中间点了一个黑点，称之为标记。标记在位置上的出现表明了处理要求的到来。

当激发产生的结果有几个时，将随机地选择一个结果输出，并把作为结果的位置状态加上标记。标记在 PNG 中的游动，就出现了“状态的迁移”。

在图 2.26 中，P3 与 P5 上都出现了标记，标明这两个状态都有了处理要求，亦即迁移（事件）t3 激发的两个前提都已具备，迁移 t3 激发。作为执行的结果，位置 P3 与 P5 上的标记移去，标记迁移到了位置 P4 上。



2.28 标记的迁移

2．建立 Petri 网的方法

1．确定系统的独立成分,主要指确定位置；

2．确定这些成分动作的前提和结果（迁移）；

3．根据条件，确定成分之间的相互关系，建立 Petri 网；

4．通过建立可达树，对 Petri 网进行分析，确定系统的特性；

根据需要重复上述①～④步，修改模型，直到满意为止。

软件需求分析的工作，是软件开发人员与用户密切配合，充分交换意见，达到对需求分析一致的意见。作为开发人员一方，进行需求工作的主要是是系统分析员和系统工程师等，他们处于用户和高级程序员之间，负责沟通用户和开发人员的认识和见解，起着桥梁作用，是需求分析的主要角色。

需求分析阶段的最终任务是要完成目标系统的需求规格说明，确定系统的功能和性能，为后阶段的开发打下基础。

需求分析的方法和技术因具体的系统而定，常用的有 SA 法，原型法，OOA 法等。