# Ch1

1.请谈一下关于软件工程的整体认识和印象。

软件工程终究是服务于人的需求的，编码是实现需求的手段。因此，软件工程是系统地对需求的分析，服务于这个需求的系统的设计，系统的实现，以及后续的维护的总和。因此软件工程呈现出与其它的工程的共性，而不是因为编码这一实现手段的特殊而与其它类型的工程出现本质不同。

2.系统分析师和系统构架设计师有何区别。

系统分析师的工作位于系统架构设计师上层。

系统分析师要负责需求分析、系统设计等位于系统设计及以上层次的任务。相比系统架构设计师，他更贴近业务。

系统架构设计师则更贴近技术，他负责设计技术模型来解决系统分析师提出的目标和问题。

3.应用软件工程（实施分阶段原理等）会增加系统工作量吗？

我认为这是看具体情况的。

使用规范化的软件工程流程必然要为此耗费设计、协调等等工作量，同时降低许多潜在的开发隐患所带来的工作量。而这种开发隐患往往随着项目的增大而增大，所以对于大型项目，应用软件工程实质上是在规避隐患，事半功倍；小型项目则需要权衡利弊。

4.给出一个问题分析的例子，其中问题部分相对简单，但是解决问题的困难在于子问题之间的相互联系。

安排火车时间表。各个火车几点开车几点到达是很容易决定的，但是协调不同火车之间的冲突十分复杂。

5.解释错误、故障和失效之间的区别。举出一个关于错误的例子，并且这个错误导致了需求、设计、代码的故障。举出在需求中存在故障的例子，并且这个故障导致了失效。举出在设计中存在故障的例子，并且这个故障导致了失效。举出在测试数据中存在故障的例子，并且这个故障导致了失效。

错误是由于各种原因而在设计和编码阶段产生的，故障是错误导致的不正确代码、不正确文档等留存在项目中的问题，失效则是在系统在运行时违背了其应有的行为。

例1：甲方要一个在菜市场里买水果的程序，要求乙方“买5个苹果，如果看见卖西瓜的就买1个”，乙方理解需求为“买5个苹果，如果看见卖西瓜的就买1个苹果”并以此进行设计和编码（误解需求）

例2：如例1中乙方错误的理解了甲方的需求，并由于这个对需求的误解导致系统编码错误

例3：以上的例子中乙方对系统的设计由于对需求的误解而错误、故障，并且在最后买水果的时候，看到西瓜时买了1个苹果，违背了甲方的一员

例4：在写测试数据的时候，写了一个“走进了皮鞋店”，而系统设计之初无论如何也没法处理“走进了皮鞋店”的情况，失效了

# Ch2

1.如何称得上一名优秀的程序员？

一个优秀的初级的程序员应该能够较好地完成一个完整过程中的某一部分活动。随着层次的逐渐提升，一个优秀的程序员能掌控的活动应当越来越多，并且对掌控的各个部分能够协调，并自如地让自己在各个不同部分的工作中灵活切换。

2.系统架构师应该具备的素质是什么？

系统架构师应该能根据选择的过程模型的不同设计不同的系统。比如如果使用增量的阶段式开发，那么就应当注重系统的可扩展性，使用迭代的阶段式开发就应该注重系统的大致功能是否全面。

3.画一个图，试描述为一次商务旅行购买一张飞机票的过程

确定起止地

查询机票

是否有直达

购买

查询机票

是否有转机

购买

考虑其他交通方式

是

是

否

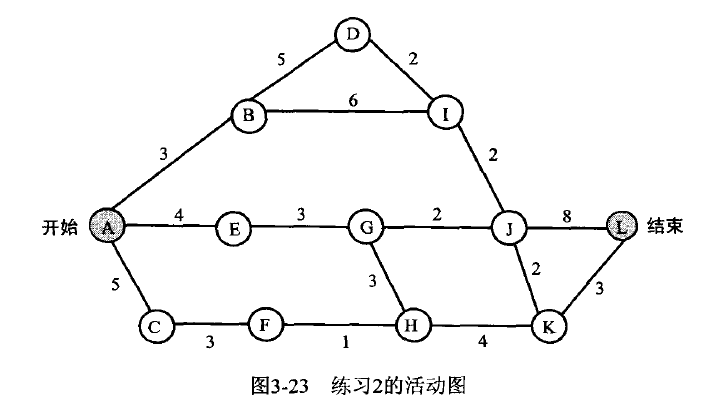
否

4.考虑本章介绍的过程。哪些过程在你对需求变化做出反应时给了你最大的灵活性。

我认为阶段化开发中的增量开发在需求变化时具有最大的灵活性。

# Ch3

2.下图中，对应于图中每条边的数字表示完成这条边代表的活动所需天数。对于每个活动，列出它的前驱，并计算最早开始时间、最晚开始时间和时差。然后确定出关键路径。

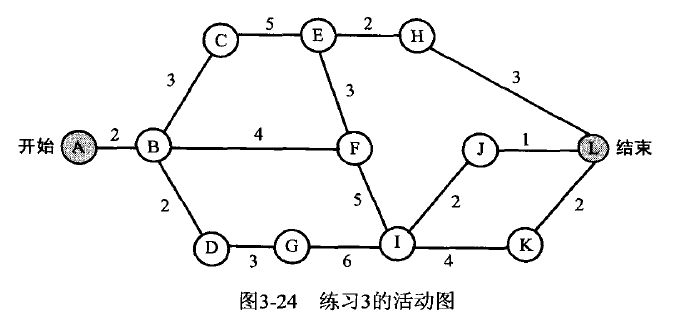


填写的时候就按照逻辑顺序来了

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 活动 | 前驱 | 最早开始时间 | 最晚开始时间 | 时差 |
| AB | / | 1 | 1 | 0 |
| AC | / | 1 | 5 | 4 |
| AE | / | 1 | 3 | 3 |
| BD | AB | 4 | 4 | 0 |
| BI | AB | 4 | 5 | 1 |
| DI | BD | 9 | 9 | 0 |
| IJ | BI DI | 11 | 11 | 0 |
| EG | AE | 5 | 8 | 3 |
| GJ | EG | 8 | 11 | 3 |
| CF | AC | 6 | 10 | 4 |
| FH | CF | 9 | 13 | 4 |
| GH | EG | 8 | 11 | 3 |
| HK | GH FH | 11 | 14 | 3 |
| JK | GJ IJ | 13 | 16 | 3 |
| JL | IJ GJ | 13 | 13 | 0 |
| KL | HK JK | 15 | 18 | 3 |

关键路径是：A-B-D-I-J-L，长度为20

3.找出下图的关键路径



关键路径是：A-B-C-E-F-I-K-L，长度为24

12.很多项目经理根据过去项目中程序员的生产率来计划项目的进度，生产率通常根据单位时间的单位规模来测量。例如，一个组织机构可能每天生产300行代码或每月生产1200个应用点。用这种方法测量生产率合适吗？根据下列事项讨论生产率的测度：

· 用不同的语言实现同样的设计，可能产生的代码行数不同。

· 在实现开始之前不能用基于代码行的生产率进行测量

· 程序员可能为了达到生产率的目标而堆积代码

首先，以生产代码行数是肯定不合理的。首先要说的是，它有一个根本问题，就是软件开发的过程中，编写代码只是一部分。比如：完成同样的需求，好的设计可以降低代码量，但我们不能说它的工作量就小了，因为为了做出这样的设计，程序员可能要付出更多的精力来做好设计。

应用点计数会好一些，但是我认为应用点也是有难有易的，一味地使用数量作为衡量标准也有些不妥。还是应该以更科学的方法。

\*.如果一个案例中涉及到合同管理，项目管理控制和项目沟通等诸多方面，在项目实际运行过程中，出现了甲方随意变更、不配合验收、甲乙双方沟通存在障碍等情形，试问如何从合同管理、过程控制和项目沟通管理三个方面来应对？

我已乙方（也就是我）的角度来思考问题

合同管理阶段：应当在合同中就说明确定需求的时间点和验收流程，并且规定改需求和更改验收流程所需要的付出的代价

项目管理阶段：写好文档，确定好里程碑，并且可以考虑分阶段验收

项目沟通：应当量化工作进度，让作为外行的甲方可以无疑义地确定工作进度

# Ch4

1. 如何拒绝需求分析过程中某些不合理的用户需求？

我认为首先要分类

-技术上不现实的需求：以专业角度否定不合理的需求，因为这种问题是没有余地的

-可实现但是从我方角度来讲不合适：面对这种情况时，其实是不能确定需求最终的合理性的，应当以需求为唯一准绳，不能涉及技术方面。

2. 分析阶段类的大致种类有边界类、实体类、控制类。请简述其具体含义（包括画法）。

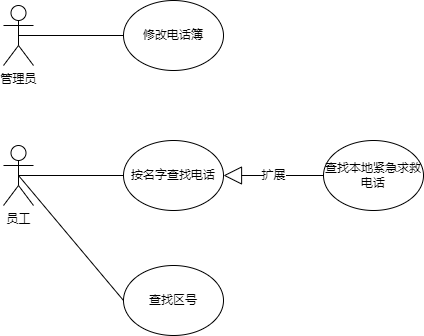
边界类：两个对象之间彼此交互使用的接口，系统与外界交互的门户

实体类：系统中的实体，存储系统中具体数据的类

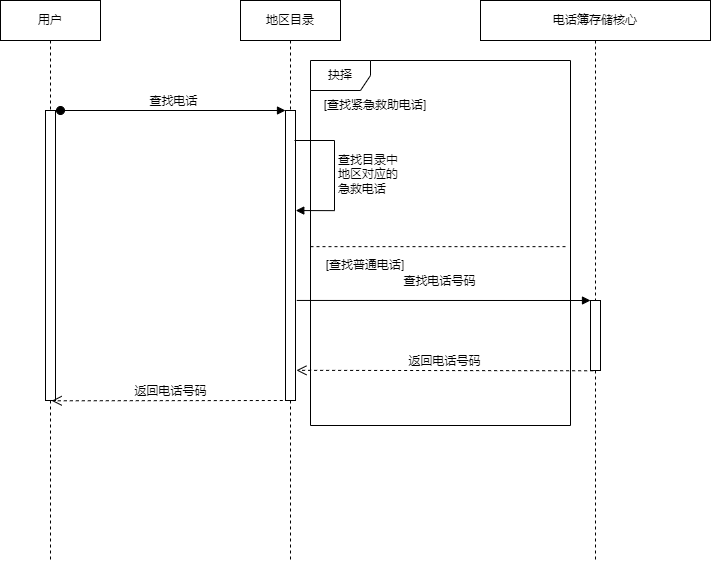
控制类：将来自用户的操作转化为对系统中其他类的操作的类

12.针对用联机电话号码簿替换电话公司给你的号码簿这样的问题，编写一套UML模型（用例图、MSC图、类图）。给出名字时，要求该电话号码簿应该能够提供电话号码；它还应该能够列出不同地区的区号，并给出你所在地区的紧急求救电话号码。

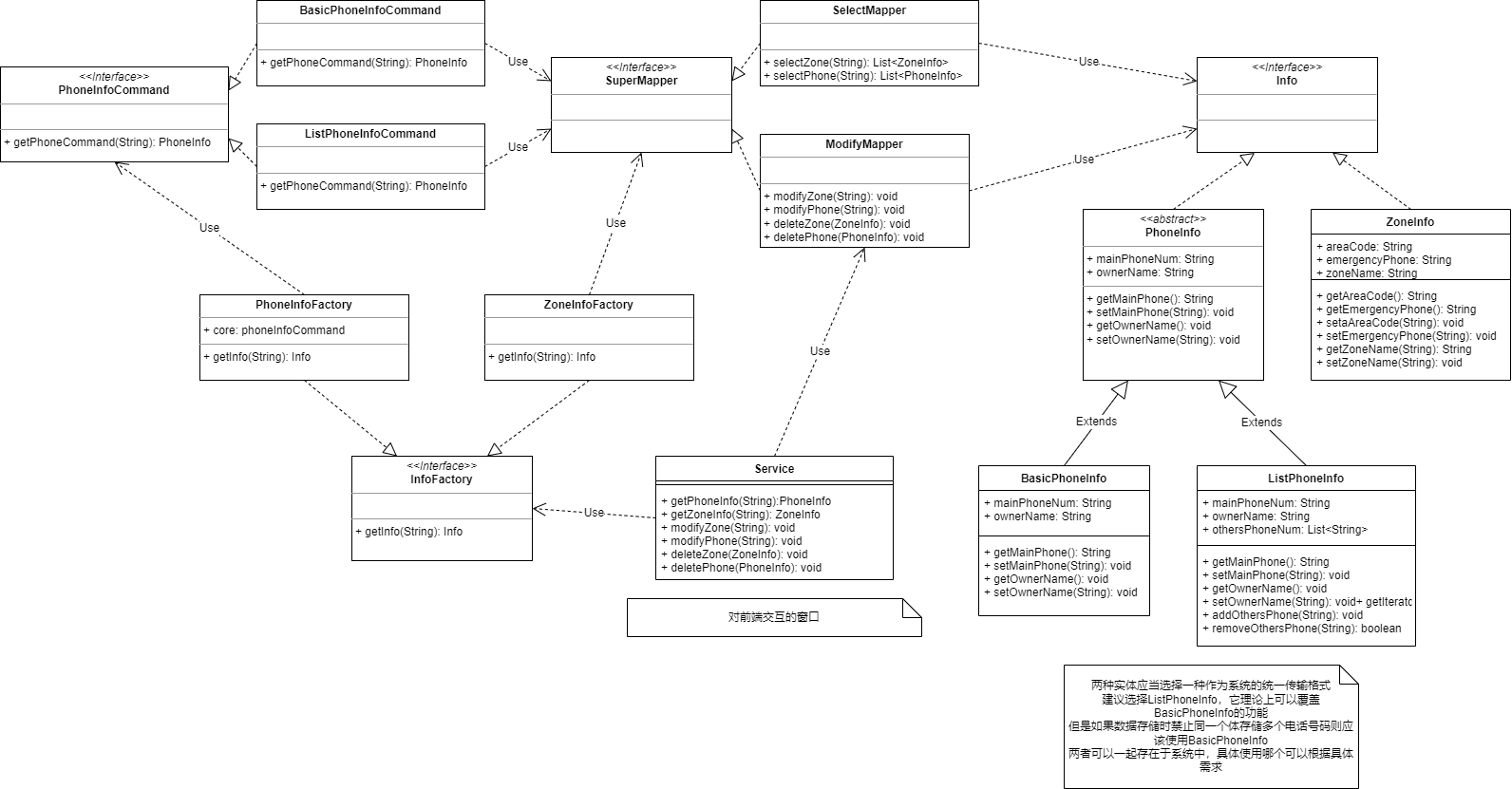
用例图



MSC图-查询号码的过程



类图



Sp：设计一款网络日记APP。（画像课堂作业）

首先，“网络日记”代表什么？

这里就有两种：不让别人看的日记，正正经经的电子日记记事本；让别人看的日记，那属于是换了个名字的博客（正经人谁写日记.jpg）。

如果是前者，那么我们实际上是要做一个功能更多的在线笔记本，那么一定要注重功能多样和复杂性的权衡，应当以轻量、易用为基础，扩充贴图、书签等美化功能；

如果是后者，那么我们做的根本就不是一个日记，而是做一个分享社区，那么就需要社区类的功能。

# Ch5

4.给出一个用原型开发但并没有节省大量开发时间的例子

高精运算的软件，各个运算逻辑紧密耦合，难以使用原型开发进行分块原型化，整体原型化又会使原型过程过于冗长

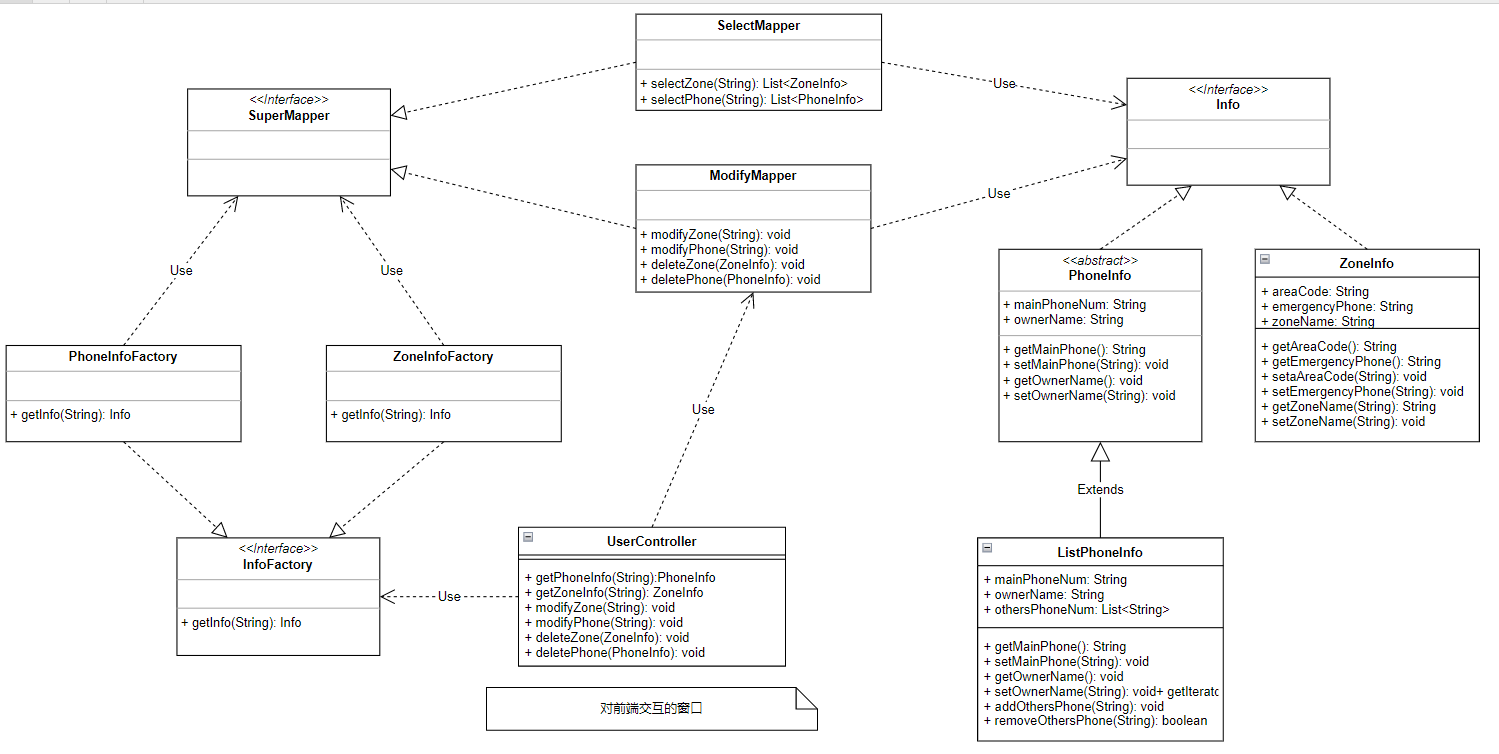
5.列举出最适合原型设计的系统特征

- 系统可以被分为比较小的模块，分开攻坚

- 系统有不确定之处，比如有多种方案需要权衡，或有未确定的功能需要测试

12续

分块介绍类的设计，和最开始的类图已经有不同了



DAO：

<<Interface>>SuperMapper：没有实际内容，所有DAO层对象的接口，我认为各个Mapper间可能会在将来有需要实现的共性，所以预先设计了

SelectMapper implements SuperMapper：用于实现各种查询方法，不允许有任何修改操作

方法：

+ selectZone(String): List<ZoneInfo>

选择区域信息

+ selectPhone(String): List<PhoneInfo>

选择电话信息

ModifyMapper implements SuperMapper：用于实现各种修改方法，不允许有任何查询操作

方法：

+ modifyZone(ZoneInfo): void

修改区域信息，如果不存在则新增，存在则修改

+ modifyPhone(PhoneInfo): void

修改电话信息，逻辑同上

+ deleteZone(ZoneInfo): void

删除区域信息

+ deletePhone(PhoneInfo): void

删除电话信息

POJO：

统一使用lombok，具有get、set等方法

<<Interface>>Info：没有实际内容，所有POJO对象的接口，理由同SuperMapper

<<abstract>>PhoneInfo implements Info：抽象类，作为后续电话信息类的父类(当然，后续可以有结构更复杂的存储类)

属性：

+ mainPhoneNum: String

主电话号码

+ ownerName: String

电话的主人

ListPhoneInfo extends PhoneInfo：具有一个List，存储主电话主人所具有的其他非主要号码

属性：

+ mainPhoneNum: String

+ ownerName: String

+ othersPhoneNum: List<String>

存储更多电话的List

方法：

+ addOthersPhone(String): void

+ removeOthersPhone(String): Boolean

ZoneInfo implements Info：存储这个城市的区号、急救电话、名称

属性：

+ areaCode: String

区号

+ emergencyPhone: String

急救电话

+ zoneName: String

城市名

工厂：

<<Interface>> InfoFactory：获取Info对象的所有工厂的接口

方法：

+ getInfo(String): Info

规定的获取信息的方法

PhoneInfoFactory implements InfoFactory：获取电话信息的工厂，依赖于Mapper实现getInfo方法

ZoneInfoFactory implements InfoFactory：获取区域信息的工厂，依赖于Mapper实现getInfo方法

Controller层：

使用springboot

UserContoller：提供给用户，用于获取信息。get方法依赖于Factory获取信息并返回，但剩下的修改方法目前还直接依赖于mapper

方法：

+ getPhoneInfo(String):PhoneInfo

+ getZoneInfo(String): ZoneInfo

+ modifyZone(String): void

+ modifyPhone(String): void

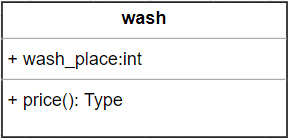
+ deleteZone(ZoneInfo): void

+ deletePhone(PhoneInfo): void

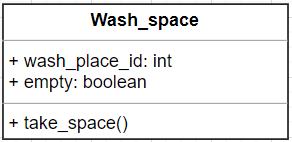
# Ch6

1.皇家汽车服务站经理Manny准备扩展他们的服务，使之包括洗车服务。这将是一个自动化系统。客户选择洗车的方式并注明车的类型。系统计算费用并在控制面板上显示应付金额。然后客户支付洗车费。支付结束后，如果洗车正忙的话，系统就会提示客户必须等待，否则，系统提示客户将车驶进洗车间。请问，如何对图6-21所示的类图进行修改才能提供这种新服务。

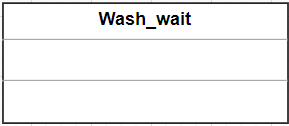
增加service的子类Wash，表示洗车服务



增加Inventory下的类Wash\_space，表示洗车位



增加message的子类Wash\_wait，表示洗车等待信息



# Ch7

# Ch8

1.如果您来带领一个测试团队，您会做哪些工作？

-分析需求，根据需求、项目规模规划整体测试计划，规定测试和文档的规范

-为不同的部分规划不同的测试方法：是否用自动化测试？黑盒or白盒？集成的顺序？

-根据重要程度的不同安排人手，并投入到最重要的部分的测试工作

2.对于需求文档，如何进行测试？

-根据需求文档本身确定流程图、输入域

-“小”用白盒，“大”用黑盒

-抽离地看待系统，我们的目的是找到错误并修正它，要尽可能地找到各种各样的用例，覆盖种种情况，努力找错。

-科学地、理性地进行测试用例设计（黑盒白盒都是），尽量根据测试设计原则进行设计而不是凭感觉，这有助于我们覆盖各种情况

3.图8-22说明了一个软件系统中的构建层次。分别描述用自底向上方法、自顶向下方法、改进的自顶向下方法、一次性集成方法、三明治方法和改进的三明治方法对构件进行集成测试的顺序

\*每行代表一步，代表一批可以同时进行的测试，每一个框代表一个测试

自底向上：

[ L ] [ G ] [ H ] [ M ] [ N ] [ J ] [ K ] [ E ]

[ F L ] [ M N I ] [ C H ]

[ B F G ] [ D I J K ]

[ A B C D E ]

自顶向下：

[ A ]

[ A B C D E ]

[ A ~ K ]

[ A ~ N ]

改进的自顶向下：

[ A ]

[ B ] [ C ] [ D ] [ E ]

[ A B C D E ]

[ F ] [ G ] [ H ] [ I ] [ J ] [ K ]

[ A ~ K ]

[ L ] [ M ] [ N ]

[ A ~ N ]

一次性集成方法

[ A ] ~ [ N ]

[ A ~ N ]

三明治方法

\*目标层为BCDE

[ L ] [ G ] [ H ] [ M ] [ N ] [ J ] [ K ] [ E ] [ A ]

[ F L ] [ I M N ] [ C H ]

[ B F G ] [ D I J K ]

[ A B C D E ]

改进的三明治方法

\*目标层为BCDE

[ L ] [ G ] [ H ] [ M ] [ N ] [ J ] [ K ] [ E ] [ A ] [ I ] [ F ] [ C ]

[ F L ] [ I M N ] [ C H ] [ B ] [ D ]

[ B F G ] [ D I J K ]

[ A B C D E ]