从需求分析到软件设计

需求的类型

1. functional requirement: required behavior in terms of required activities
2. nonfunctional requirement: quality characteristic that the software must posses
3. design constraint: design decision such as platform or interface components
4. process constraint: restriction on the techniques or resources

和需求相关的人员

client ~ pay for the software to be developed
customer ~ buy the software
user: use the system
domain experts: familiar with the problem that the software must automate
market researchers: determine future trends and potential customers
lawyers or auditors: familiar with goverment, legal requirements
software engineers or other technology experts

获取需求的主要方法

1. Interviewing stake holders
2. Reviewing available documentations
3. Observing the current system (if one exists)
4. Apprenticing with users to learn about user's task in more details
5. Interviewing user or stakeholders in groups
6. Using domain specific strategies, such as Joint Application Design
7. Brainstorming with current and potential users

高质量的需求

1. Making Requirements Testable
2. Resolving Conflicts
3. Characteristics of Requirements

对需求建模

需求分析的两类方法

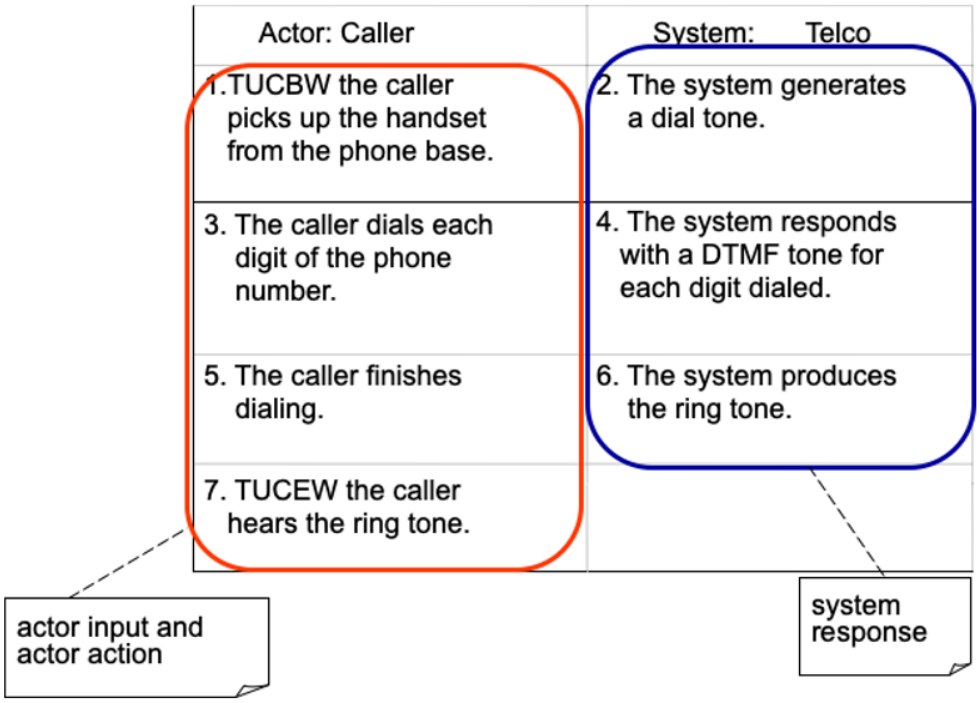
原型化方法 和 建模方法
原型化 ~ 整理用户接口，如界面布局
建模 ~ 有关事件或活动同步约束问题 在逻辑上形成模型

用例建模

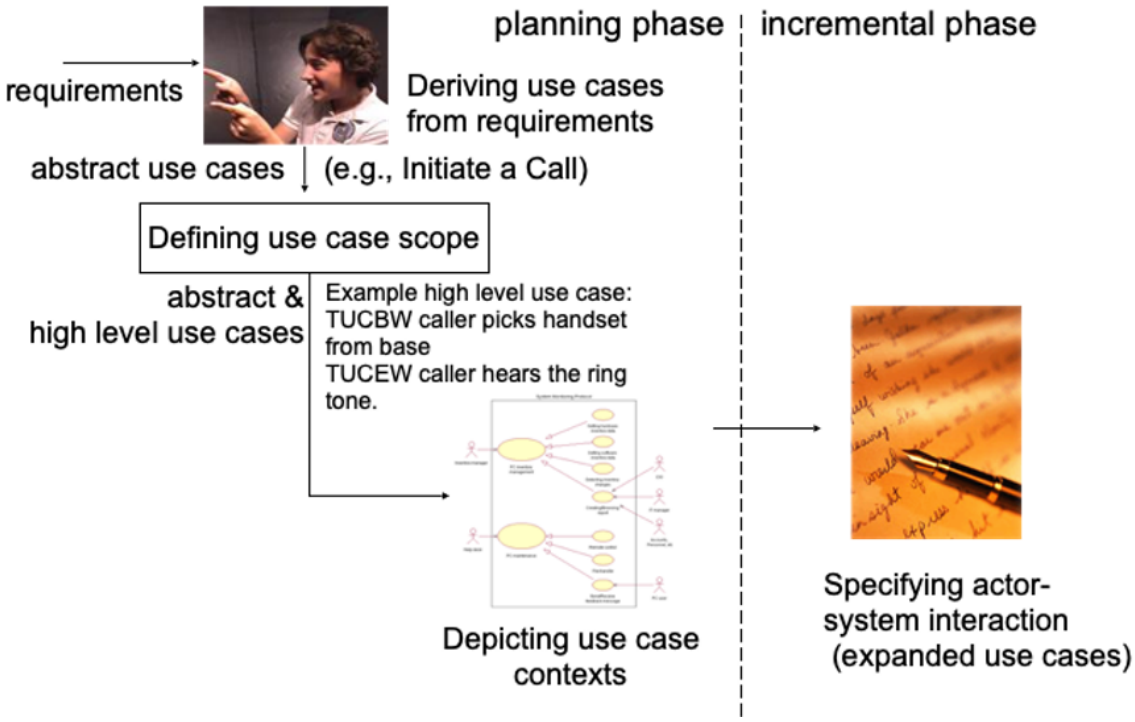
用例 业务过程
用例三个层级

1. 抽象用例 干什么
2. 高层用例 在什么时候开始
3. 扩展用例 将参与者和待开发软件交互过程详细描述出来

扩展用例的两列表格举例



用例建模的基本步骤



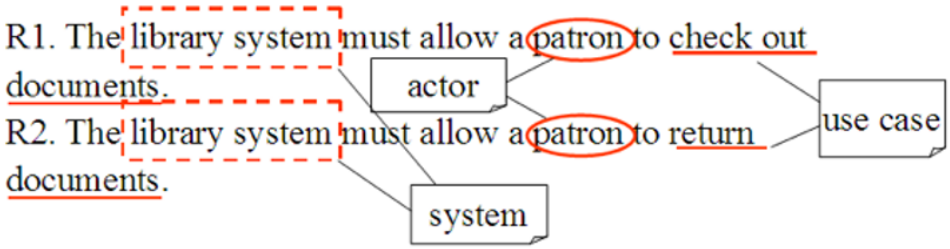
准确提取用例的基本方法 四个必要条件

4. 它是不是业务过程
5. 是不是由某个参与者触发开始

6. 是不是显示的或隐式地终止于某个参与者
7. 是不是为某个参与者完成了有用的业务过程

准确提取用例的基本方法

- 举个例子，以我们常见的图书馆系统为例，有如下两个需求：

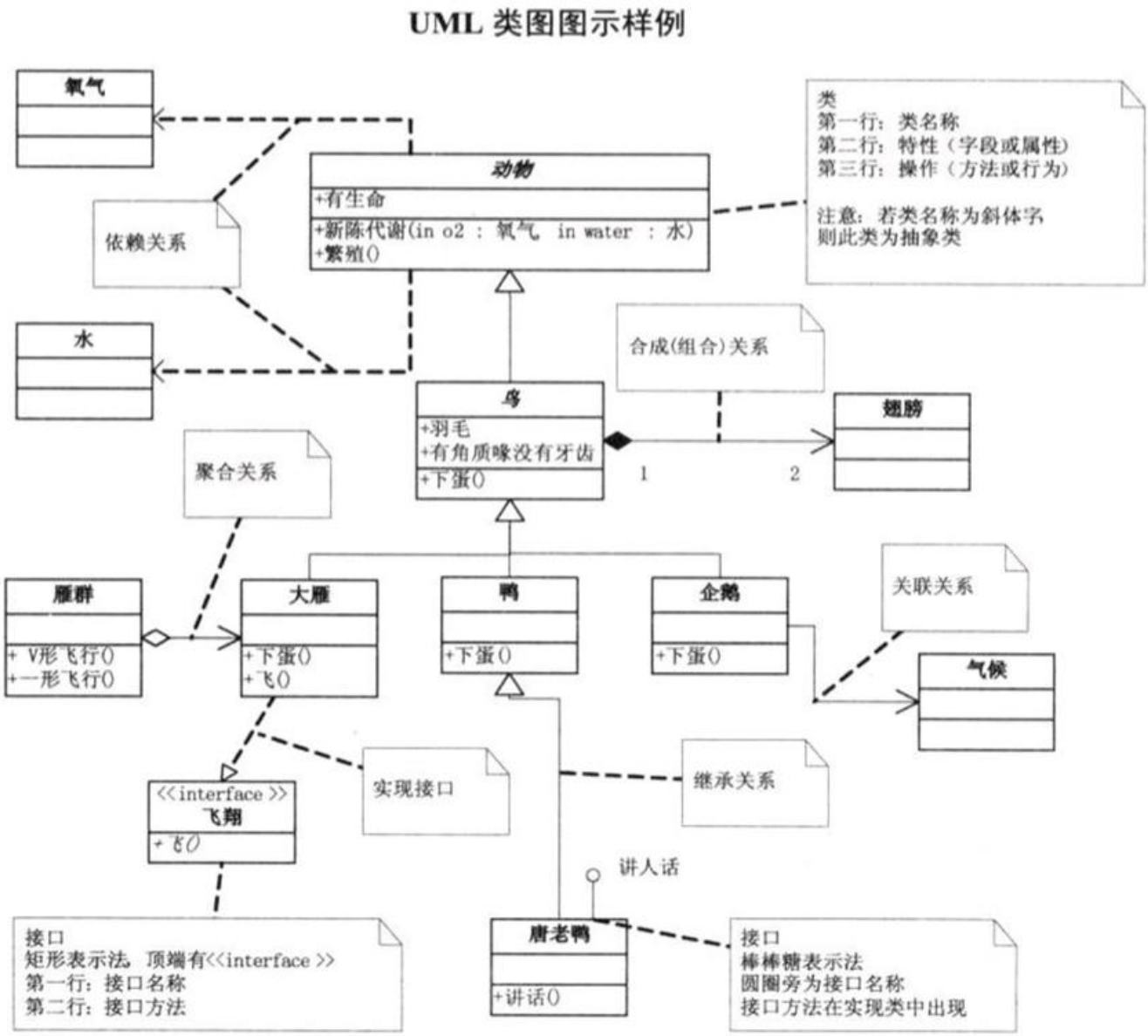


根据准确提取用例的基本方法，很容易可以得到：

System: Library System
Actor: Patron
Use Cases:
UC1: Checkout Document
UC2: Return Document

用例图基本画法

1. 关联关系 ~ 不强调关联
2. 单向关联 ~ 参与者“知道”用例，用例“不知道”参与者
3. 包含关系
4. 扩展关系



业务领域建模

开发人员获取业务领域知识的过程。

业务领域建模基本步骤

1. 收集应用业务领域的信息
2. 头脑风暴
3. 概念分类
4. 画出UML类图

头脑风暴具体

业务领域概念分类方法

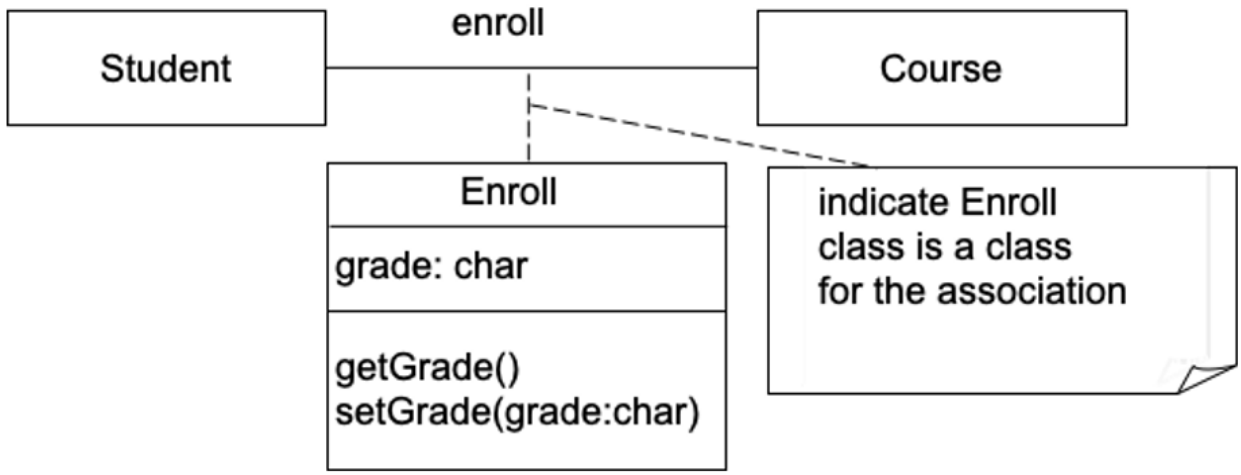
对象有独立存在，而属性不能独立存在。

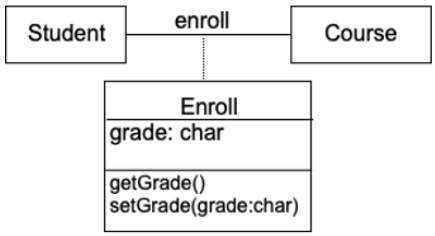
业务数据建模

关联类及数据模型

关联关系时业务数据建模的关键

关联类及其UML类图画法





Student表

student_id	name	...	...
0001	Alex	...	...
0002	Eric	...	...
...	...	...	...

Course表

course_id	name	...	...
0001	OOSE	...	...
0002	AI	...	...
...	...	...	...

Enroll表

student_id	course_id	grade	...
0001	0001	A	...
0001	0002	A	...
0002	0001	B	...
...	...	...	...

关系数据模型的MongoDB设计与实现

MongoDB ~ 文档数据库 基于分布式 用类似JSON格式的文档来存储数据

Rich JSON Documents

MongoDB中存储的JSON格式文档范例如下所示。每一个JSON文档对应一个ID即下图中“_id"的值，除“_id"外的数据按照“key：value”的方式可以任意定义数据的结构。

```
{
  "_id": "5cf0029caff5056591b0ce7d",
  "firstname": "Jane",
  "lastname": "Wu",
  "address": {
    "street": "1 Circle Rd",
    "city": "Los Angeles",
    "state": "CA",
    "zip": "90404"
  },
  "hobbies": ["surfing", "coding"]
}
```

ACID transaction

原子性(Atomicity) 一致性(Consistency) 隔离性(Isolation) 持久性(Durability)

Modeling One-to-Few

针对个人需要保存多个地址进行建模的场景下使用内嵌文档是很合适，可以在person文档中嵌入addresses数组文档

```
> db.person.findOne()
{
  name: 'Kate Monster',
  ssn: '123-456-7890',
  addresses : [
     { street: '123 Sesame St', city: 'Anytown', cc: 'USA' },
     { street: '123 Avenue Q', city: 'New York', cc: 'USA' }
  ]
}
```

One-to-Many

- 以产品零件订货系统为例。每个商品有数百个可替换的零件，但是不会超过数千个。这个用例很适合使用间接引用---将零件的objectid作为数组存放在商品文档中(在这个例子中的ObjectID我使用更加易读的2字节，正常是由12个字节组成的)

```
> db.parts.findOne()
{
    _id : ObjectID('AAAA'),
    partno : '123-aff-456',
    name : '#4 grommet',
    qty: 94,
    cost: 0.94,
    price: 3.99
```

```
> db.products.findOne()
{
    name : 'left-handed smoke shifter',
    manufacturer : 'Acme Corp',
    catalog_number: 1234,
    parts : [     // array of references to Part documents
        ObjectID('AAAA'),    // reference to the #4 grommet above
        ObjectID('F17C'),    // reference to a different Part
        ObjectID('D2AA'),
        // etc
    ]
```

```
  // Fetch the Product document identified by this catalog number
> product = db.products.findOne({catalog_number: 1234});
   // Fetch all the Parts that are linked to this Product
> product_parts = db.parts.find({_id: { $in : product.parts } } ).toArray() ;
```

One-to-Squillions

- 我们用一个收集各种机器日志的例子来讨论一对非常多的问题。由于每个mongodb的文档有16M的大小限制，所以即使你是存储ObiectID也是不够的。我们可以使用很经典的处理方法“父级引用”---用一个文档存储主机，在每个日志文

```
> db.hosts.findOne()
{
    _id : ObjectID('AAAB'),
    name : 'goofy.example.com',
    ipaddr : '127.66.66.66'
}

>db.logmsg.findOne()
{
    time : ISODate("2014-03-28T09:42:41.382Z"),
    message : 'cpu is on fire!',
    host: ObjectID('AAAB')       // Reference to the Host document
}
```

```
  // find the parent ‘host’ document
> host = db.hosts.findOne({ipaddr : '127.66.66.66'});  // assumes unique index
   // find the most recent 5000 log message documents linked to that host
> last_5k_msg = db.logmsg.find({host: host._id}).sort({time : -1}).limit(5000).toArray()
```

One-to-Squillions

- 我们用一个收集各种机器日志的例子来讨论一对非常多的问题。由于每个mongodb的文档有16M的大小限制，所以即使你是存储ObjectID也是不够的。我们可以使用很经典的处理方法“父级引用”---用一个文档存储主机，在每个日志文档中保存这个主机的ObjectID。

```
> db.hosts.findOne()
{
    _id : ObjectID('AAAB'),
    name : 'goofy.example.com',
    ipaddr : '127.66.66.66'
}

>db.logmsg.findOne()
{
    time : ISODate("2014-03-28T09:42:41.382Z"),
    message : 'cpu is on fire!',
    host: ObjectID('AAAB')      // Reference to the Host document
}
```

```
  // find the parent ‘host’ document
> host = db.hosts.findOne({ipaddr : '127.66.66.66'});  // assumes unique index
   // find the most recent 5000 log message documents linked to that host
> last_5k_msg = db.logmsg.find({host: host._id}).sort({time : -1}).limit(5000).toArray()
```

反范式化

在一个读比写频率高的多的系统里，反范式是有使用的意义的。如果你很经常的需要高效的读取冗余的数据，但是几乎不去变更他的话，那么付出更新上的代价还是值得的。更新的频率越高，中种设计方案的带来的好处越少。

三种设计方案：内嵌，子引用，父引用

1. 一对很少且不需要单独访问内嵌内容的情况下可以使用内嵌多的一方
2. 一对很多且很多的一端内容因为各种理由需要单独存在的情况下可以通过数组的方式引用多的一方的。
3. 一对非常多的情况下，请将一的那端引用嵌入进多的一端对象中。

关系数据模型的MongoDB设计与实现总结

- 1、优先考虑内嵌，除非有什么迫不得已的原因。
- 2、需要单独访问一个对象，那这个对象就不适合被内嵌到其他对象中。
- 3、数组不应该无限制增长。如果many端有数百个文档对象就不要去内嵌他们可以采用引用ObjectID的方案；如果有数千个文档对象，那么就不要内嵌ObjectID的数组。该采取哪些方案取决于数组的大小。
- 4、不要害怕应用层级别的join：如果索引建的正确并且通过投影条件限制返回的结果，那么应用层级别的join并不会比关系数据库中join开销大多少。
- 5、在进行反范式设计时请先确认读写比。一个几乎不更改只是读取的字段才适合冗余到其他对象中。
- 6、在mongodb中如何对你的数据建模，取决于你的应用程序如何去访问它们。数据的结构要去适应你的程序的读写场景。

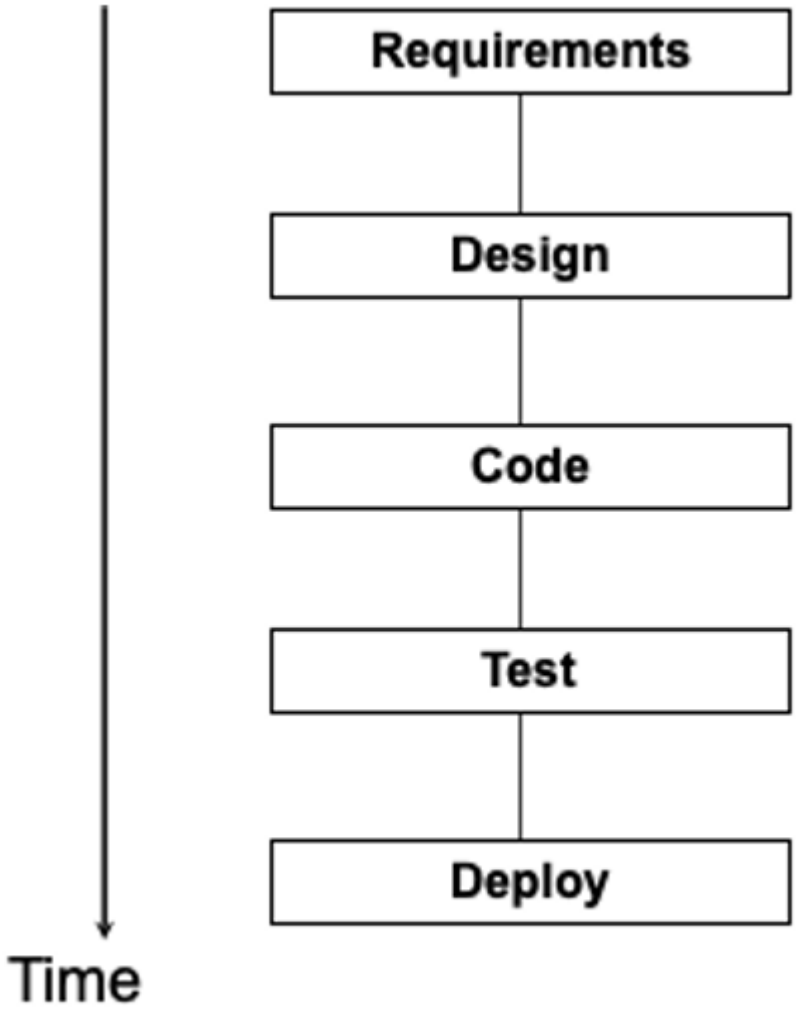
业务概念原型

业务概念原型

- 概念是人对能代表某种事物或发展过程的特点及意义所形成的思维结论。
- 概念原型是一种虚拟的、理想化的软件产品形式。

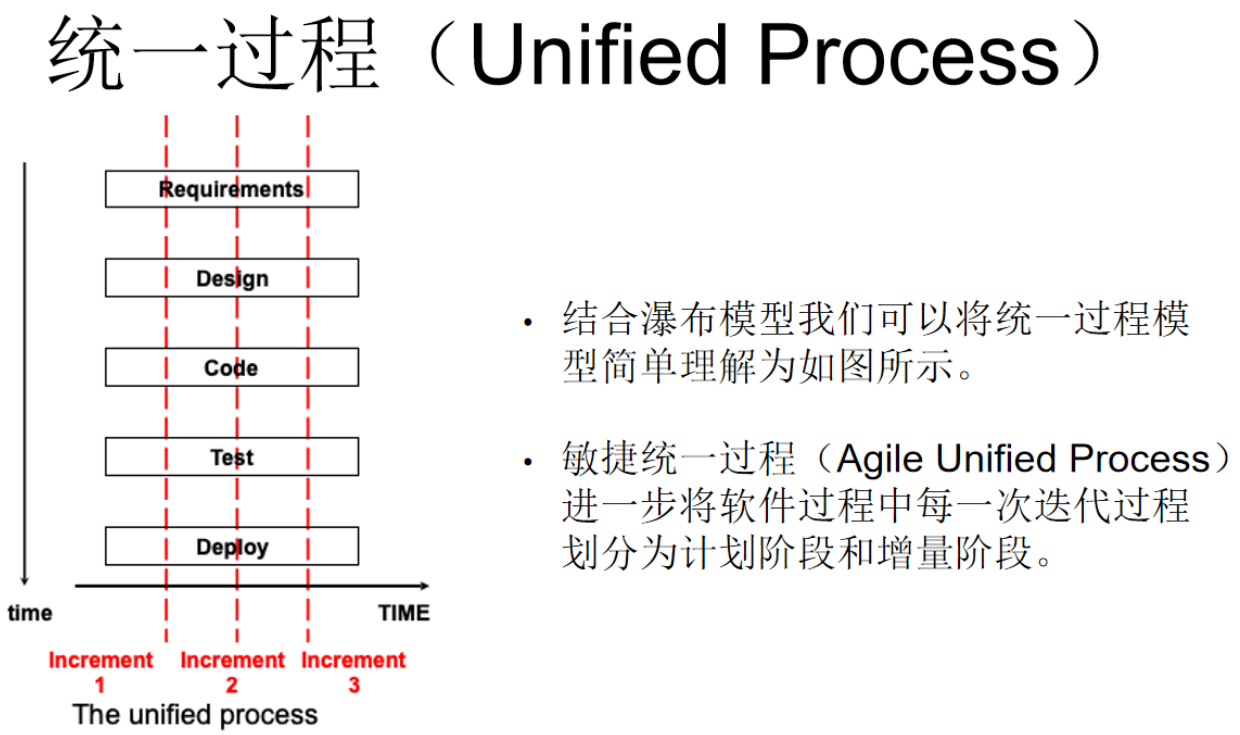
从需求分析到软件设计

瀑布模型



The waterfall process

统一过程 核心要义：用例驱动、以架构为中心、增量且迭代的过程。



敏捷统一过程的四个关键步骤

1. 确定需求
2. 通过用例的方式满足这些需求
3. 分配用例到各增量阶段
4. 具体完成各增量阶段所计划的任务 **敏捷统一过程增量阶段**
5. 用例建模
6. 业务领域建模
7. 对象交互建模
8. 形成设计类图
9. 软件的编码实现和软件应用部署

   对象交互建模

对象交互建模（Object Interaction Modeling）

- 对象交互建模的基本步骤
- 找出关键步骤进行剧情描述（scenario）
- 将剧情描述（scenario）转换成剧情描述表（scenario table）
- 将剧情描述表转换成序列图的基本方法
- 从分析序列图到设计序列图
- 一个完整用例的对象交互建模

从分析序列图到设计序列图