Chapter 4 过程模型Process Models

重点：

4.1.1 The Waterfall Model

4.1.2 Incremental Process Model

4.1.3 Evolutionary Process Model

process model为软件工程工作提供了特定的路线图。 它定义了所有活动、行动和任务的流程、迭代的程度（迭代了几次，迭代的方式(可能是action,task等不断迭代)）、工作产品以及必须完成的工作的组织。

4.1 规定的过程模型Prescriptive Process Models

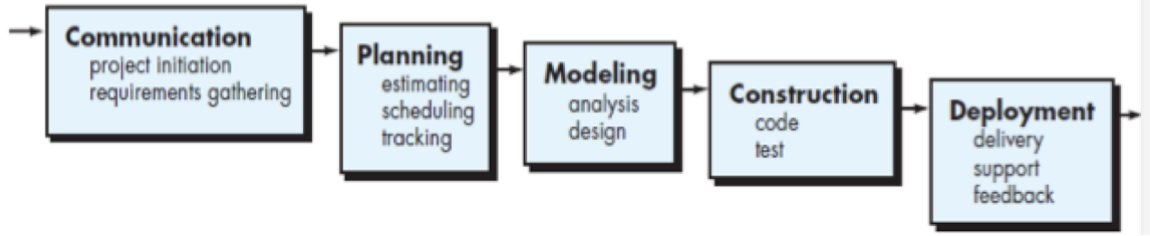
"prescriptive": 规定了一系列过程元素：framework activities, SE actions ,tasks ,work product, quality assurance , change control mechanisms

4.1.1 瀑布模型The Waterfall Model

特点：

1. 按照general/standard process顺序开发，很少迭代：planing是umbrella可以去掉
2. 过程明确，需求清晰
3. 不主张迭代，但也有迭代
4. 用户需要很长时间才能得到成品
5. 某阶段受阻，整个项目都会到blocking states，也就是说，没有形成milestone，一个action，task卡住了就都卡了
6. 线性模型
7. 项目大的时候不要用，适合需求清晰稳定的情况

FIGURE 4.1 The waterfall model



Planning: EST评估→计划→跟踪

瀑布模型的缺点：

waterfall models的问题：

①**实际项目很少遵循模型提出的顺序流程。** 虽然线性模型可以适应迭代，但它是间接这样做的。 因此，随着项目团队的推进，变更可能会导致混乱。（项目发生变更时就不适用，导致混乱）

② **客户通常很难明确地陈述所有要求。** 瀑布模型需要这一点，并且难以适应许多项目开始时存在的自然不确定性。(要求需求明确，然而这点很难做到)

③ 客户要有耐心。 **程序的工作版本要到项目时间跨度的后期才可用。** 重大失误如果在审查工作计划之前未被发现，可能是灾难性的。(比如：waterfall会导致"阻塞状态"，在这种状态下，某些项目团队成员必须等待团队其他成员完成依赖任务。)（容错率低，且在项目后期发现难以修复，and阻塞状态）

4.1.1(extended)V-模型

瀑布模型的变种 (与standard process非常相似)

瀑布模型对软件工程的最大贡献

描述了质量保证行动与与沟通、建模和早期建设活动相关的行动之间的关系。

Unit Testing: 因为需要看底层代码，故依赖Code generation；因为复杂的代码中如何传参，参数指向什么很复杂，需要查看Component Design 和 详细设计规约

Integration Testing：组件与组件的集成，看Component Design；还因为是和子系统之间的协作有关，涉及接口，数据库，软件结构设计，需要参考接口设计，数据设计，软件体系架构设计，故看Architectural Design

System Testing：把编好的程序放在各种集成环境下；首先需要需求分析规约来进行模拟测试（主要看的是需求建模中的功能建模），比如模拟Actor，故需要Requirement Modeling；如果发现失败了，要检查什么接口有问题，故要参考Architectural Design(追溯子系统)

Acceptance Testing: 验收测试是面向用户的，用户不知道你的实现细节而是以需求为依据，所以是需求层面的（需要用到需求规约和需求分析规约），即Requirement Modeling

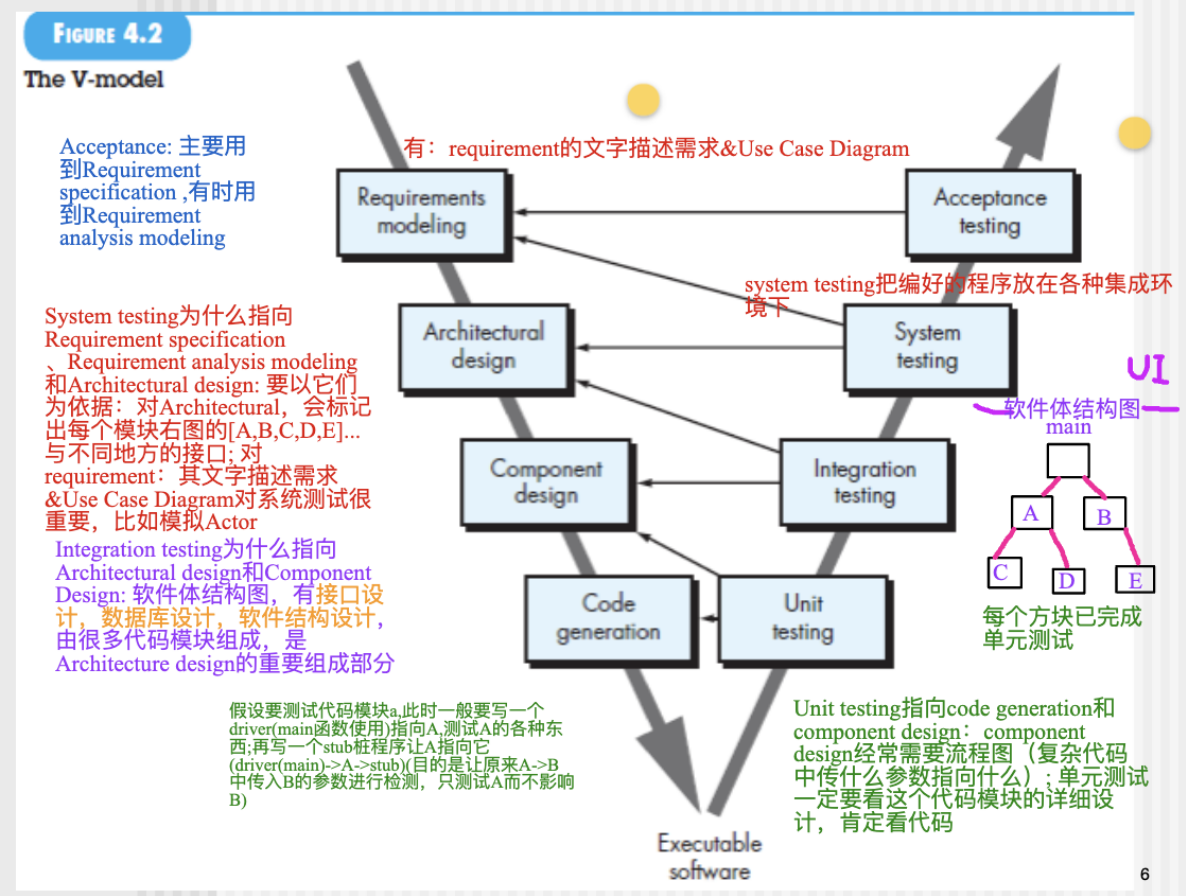
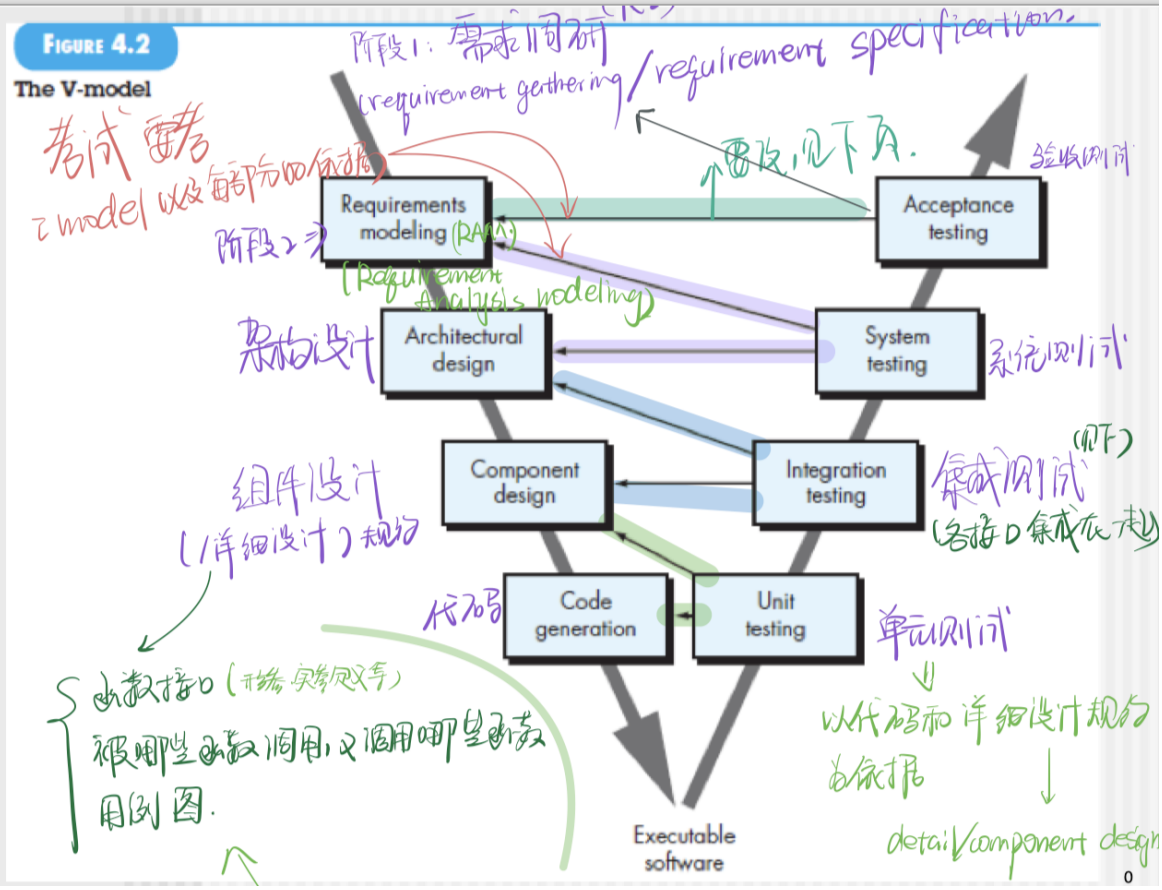
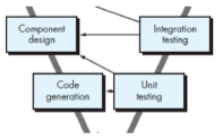


FIGURE 4.2
The V-model

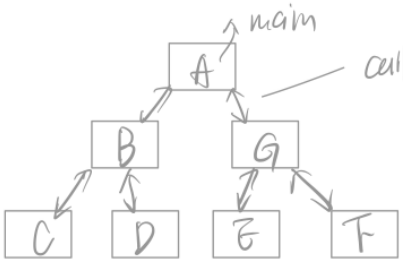


开始：

集成测试以 detail / component design 和 Architectural design 为依据

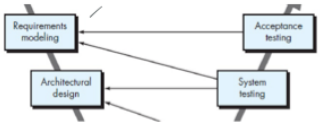


架构设计包括3个部分（第一张PPT笔记）
- 软件体系架构设计
- 数据库设计
- 接口设计 (interface)



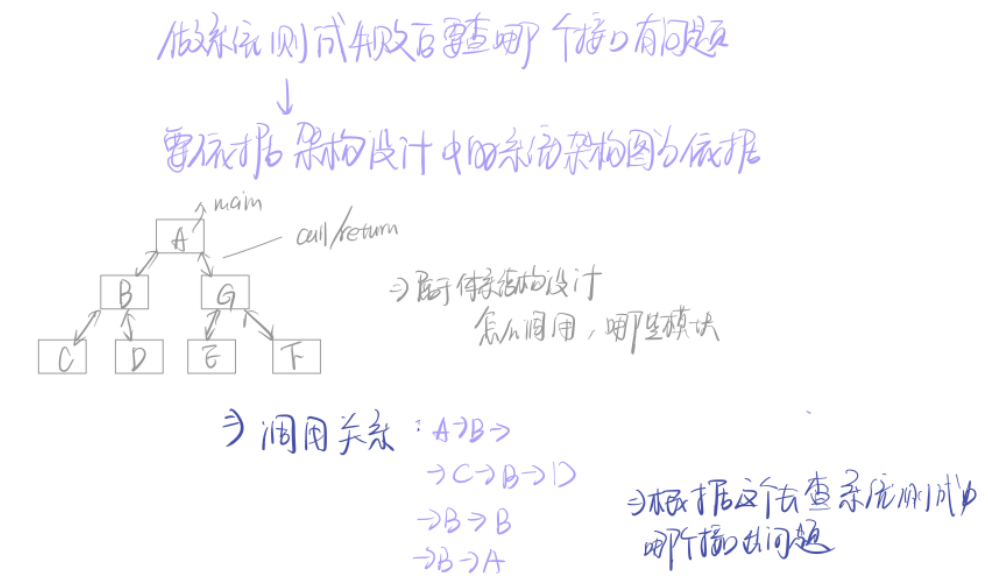
⇒ 软件体系结构设计
怎么调用，哪些模块
知道了这些才可以
集成起来

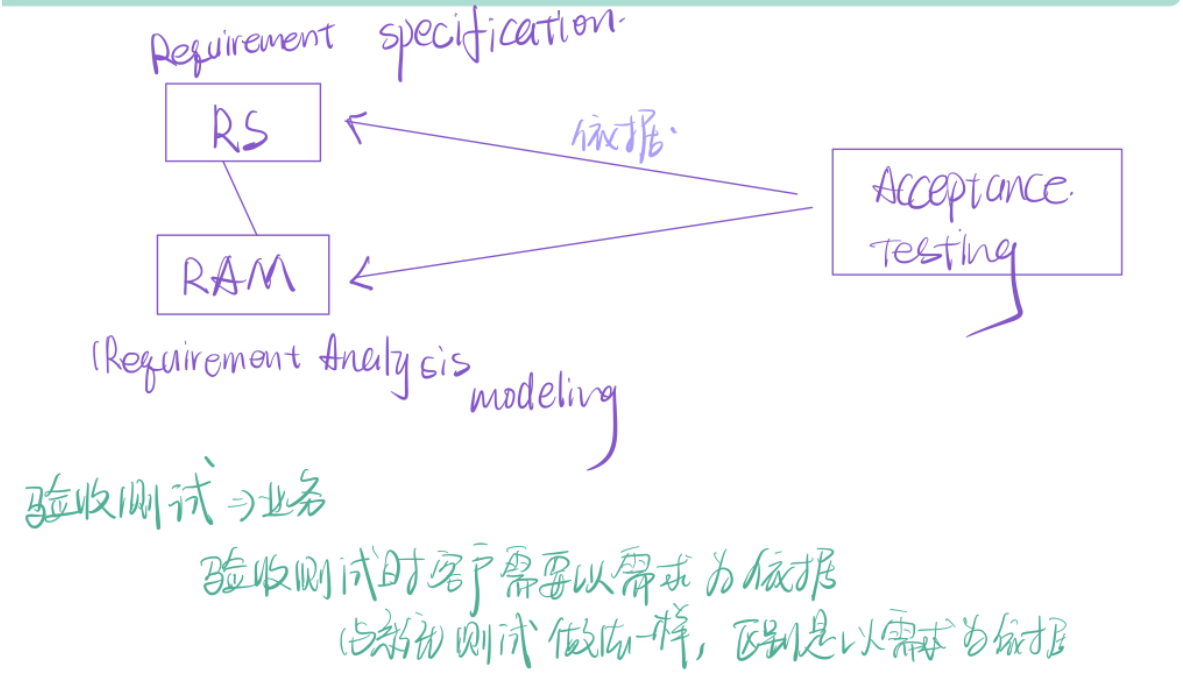
System testing 系统测试以需求分析规约 requirement modeling 和 Architectural design 架构设计 为依据



requirement modeling 需求建模
- 数据建模
- 功能建模
- 行为建模

⇒ 在系统测试中作为依据





4.1.2 增量过程模型

场景：

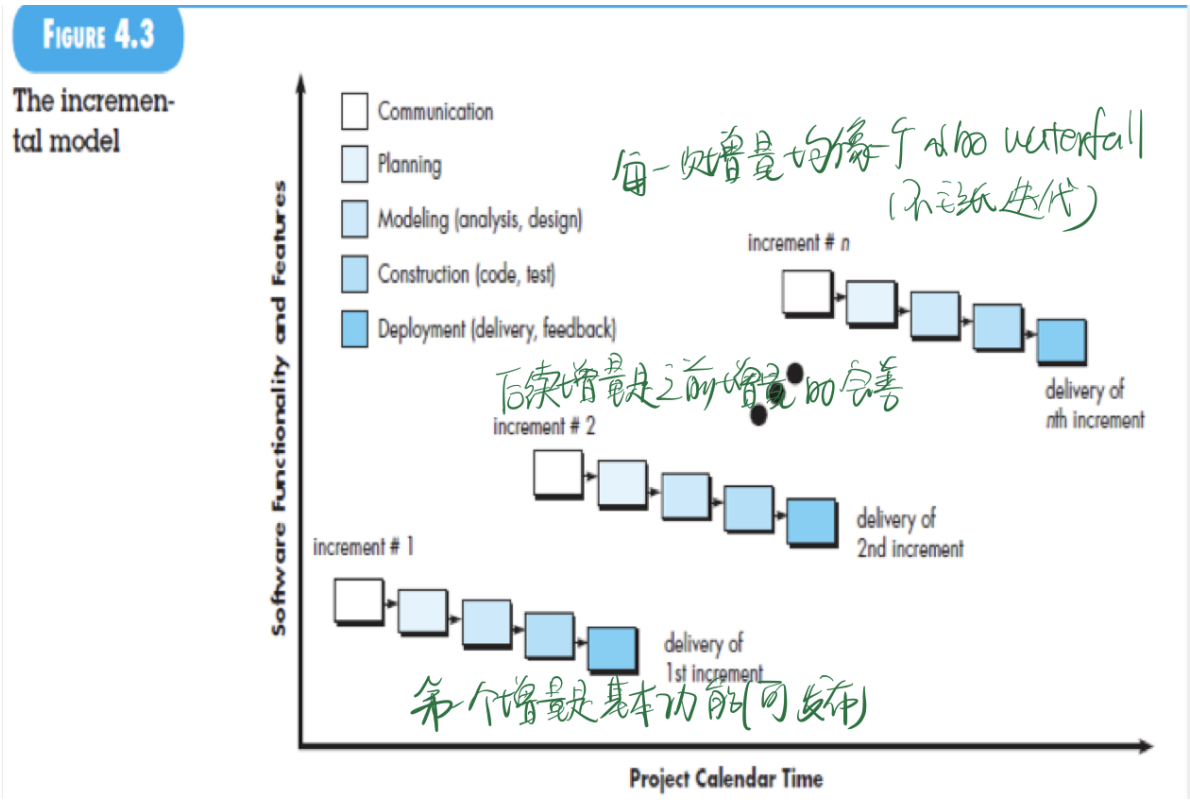
1. Non-linear开发工作的整体范围排除了一个纯粹的线性过程。
2. Rapid Raw Product v1.0 可能迫切需要向**用户快速提供**一组有限的软件功能，然后在以后的软件版本中完善和扩展该功能。
3. Update v2.0++++ 有部分功能**不成熟需要后期才增量补充**；
4. Preview Function 市场要求**先发布一部分功能**，后续功能增量补充；
5. Parallel 一些内容**交叉进行**

结合了waterfall模型和并行处理流parallel

当使用增量模型时，**第一个增量通常是核心产品。** 也就是说，**解决了基本要求**，但许多补充功能（一些已知，另一些未知）仍未交付。 核心产品由客户使用（或经过详细评估）。 作为使用和/或评估的结果，为下一个增量制定计划。

特点：

- 每一次增量都生成可发布的产品increment producut，第一个增量形成核心基本产品，后续增量形成完善版本
- 每次增量是一个waterfall，一次增量内不主张迭代
- 不同增量间可并行开发（比如：第2个做coding时，第3个可做需求建模）——纵轴重合



4.1.3 演化过程模型Evolutionary Process Model

强调模型迭代iteration

场景：

1. Complex系统**复杂**，模型演化**需求变更**；
2. Preview紧凑的市场期限**要求一个功能有限的版本**需要应对市场和企业的压力；
3. Requirement一系列核心产品和系统**需求明确**，但产品和系统**拓展的细节还没有被设计出来**

两种常见的演化过程模型(都是迭代的）：

4.1.3.1 原型模型Prototype

特点：快速

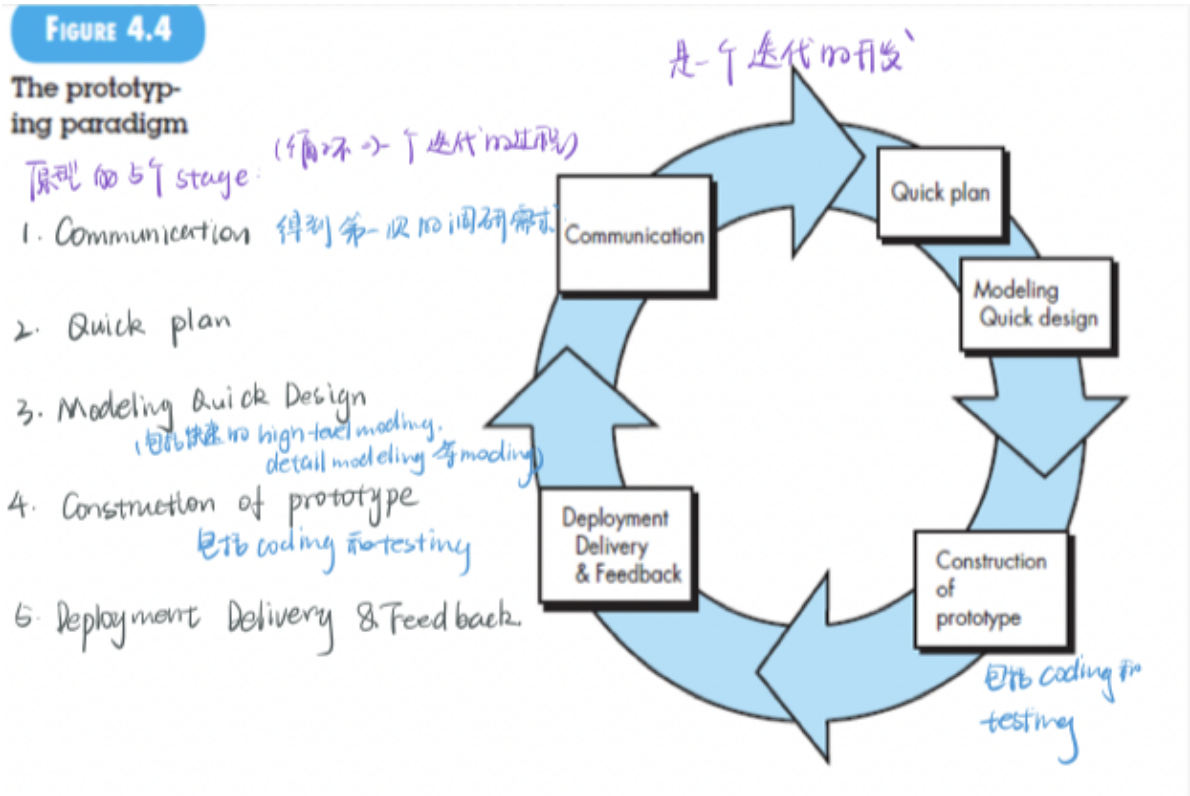
为什么要用Prototype(很可能考)

一般它是一个可以确定需求identifying software requirement的方法

描述：客户定义了一组软件的**一般目标但没有具体需求**时；开发者不知道具体开发细节时，可帮助您和其他利益相关者更好地了解在需求模糊时要构建的内容。

迭代发生在原型被调整以满足不同利益相关者的需求时，同时使您能够更好地了解需要做什么。

啥都快，不过Deployment模块貌似没有support环节?



Prototype的不足之处：

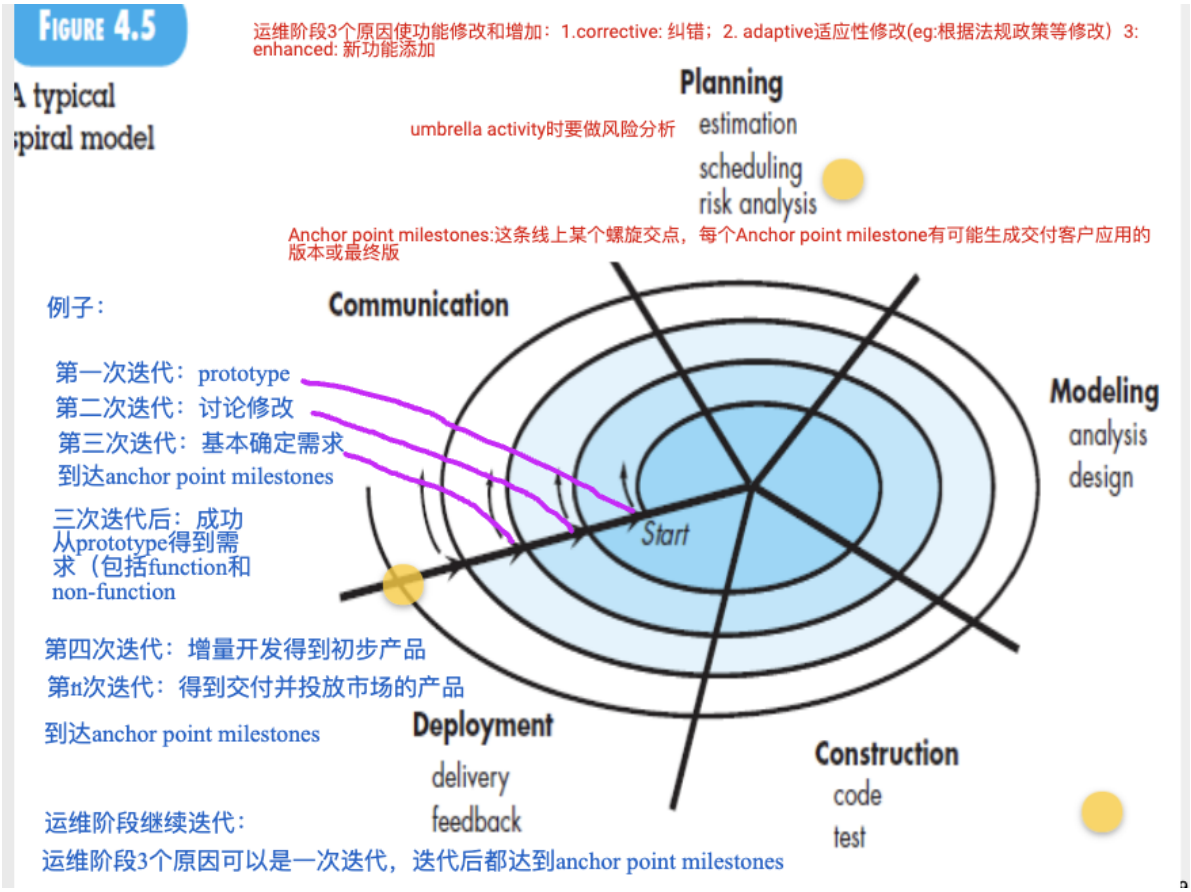
- （代码能跑了，但是static的没啥用）成型但不一定可靠：stakeholder看到的软件可运行，但不知道原型是固定在一起的，不知道在急于使其工作时您**没有考虑整体软件质量或长期可维护性。**
- （代码都是垃圾的的但能用，结果没改）错用：经常会在实现上做出妥协，以便让原型快速工作。不适当的操作系统或编程语言可能仅仅因为可用和已知而被使用； 一个低效的算法可能会被简单地实现来展示能力。 一段时间后，您可能会对这些选择感到满意，而**忘记了它们不合适的所有原因。**

Prototype主要是用来确定需求，真正开发时可能会被抛弃掉，换成更加注重质量的model；没有被抛弃的部分会成为真正系统的一部份

真正的开发要做到质量控制SQA

4.1.3.2 螺旋模型Spiral Model

特点：适合**大型项目，风险较高的项目，需求不明确的项目(big,risky,unclear)**，将**原型设计的迭代性质与瀑布模型的受控和系统方面相结合。** 它为快速开发越来越完整的软件版本提供了潜力。



解释：

每次迭代circuit后会得到一个work product

CPMCD:这5个步骤是可裁减的；

比如：第一次迭代(circuit)主要为了得到需求规约，则Communication后4个可以减去，最后得到一个Requirement Specification；后面的迭代则都用到了后4个，但详细程度重要程度也不同(比如:第二次迭代得到Prototype，第三次可能会完成部分功能得到可发布的一个版本，第四次进行功能完善等)

其他process models: 在软件交付时结束；

spiral model: 在整个软件的生命周期中都适用