南京大学
NANJING UNIVERSITY

软件过程及过程模型

Software Process and Process Model

张天
软件工程组
ztluck@nju.edu.cn

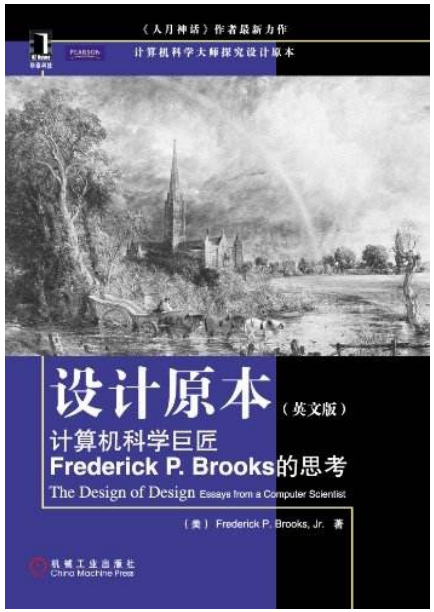
2025年秋季

Questions：

- What is a *software process*?
- What is a *software life-cycle model*?
- What are the *generic framework activities* that are present in every software process?
- What are the *prescriptive process models* and what are their strengths and weaknesses?

- *注：这里有个容易让人迷惑的地方，就是理解**过程模型**和**生命周期模型**两个概念的区别*

推荐一本书：《设计原本》



设计原本

计算机科学巨匠 ***Frederick P Brooks*** *的反思*

Software Process

- What is *Software Process*
  - The software process is the way we produce software.
  - Software engineering practice is a broad array of **principles**, **concepts**, **methods**, and **tools** that you must consider as software is *planned* and *developed*.

  - *注：各种书中给出的定义都不尽相同，但意思是一致的，而一个具体的软件工程实践是需要根据情况定制的！*

Definition using SLCM

- It incorporates the **methodology** with its underlying ***software life-cycle model*** and **techniques**, the **tools** we use, and most important of all, the **individuals** building the software.

- *涉及到这么几个重要的方面：方法学、生命周期模型、技术、工具和开发人员。*

Software Life-cycle Model

- 简而言之：
  - A *life-cycle model* is a description of the *steps* that should be performed when *building* a software product.

- 所有的软件开发方法，必定在两个步骤上一样
  - 一个是要获取需求
  - 另一个是要交付可运行的系统

过程模型和生命周期模型

- 过程模型和生命周期模型是初学者特别容易混淆的概念
  - **Process Model** vs **Software Life-Cycle Models**
- 两者的侧重点不同：
  - 前者以人为主体，从开发过程来看
  - 后者则从软件自身来看
- 但两者的本质是一致的，是从两个不同的视角来看待软件开发

- Life-cycle model
  - waterfall life-cycle model
  - evolution-tree life-cycle model
  - incremental life-cycle models

- Also known as *Process Model*
  - waterfall model
  - evolution(-tree) model
  - incremental models

案例：Winburg市公交系统

- Winburg市公交系统场景描述
  - 每辆公共汽车都有一台只接受美元钞票的收费机
  - 乘客进入公交车时，将钞票插入插槽
  - 票价机器内的传感器扫描账单，机器中的软件使用图像识别算法识别出钞票面值
  - 要求是平均响应时间应少于1秒，而平均准确度至少要达到98%
  - *开放问题思考：你觉得可以开始开发了吗？*

该系统的开发和演化过程

- 软件的开发和演化过程分为6幕
  - 第1幕：实现该软件的第一个版本
  - 第2幕：测试表明，要求确定1美元钞票为有效的平均1秒钟响应时间没能达到
  - 第3幕：对核心部分的图像识别算法进行更换，重新编写代码，达到要求
  - 第4幕：进度落后预算超支，于是增加对钞票识别能力的要求，准备同时卖给自动售货机公司，补偿超支
  - 尾声：几年后，更新硬件，于是更新软件，于是进行了重新开发。

第二幕

- 测试表明，要求确定1美元钞票为有效的平均1秒钟响应时间没能达到，目前只有10秒！
- 高级软件工程师发现了其中的原因
  - 为了获得要求的98%的准确度，在详细设计时，开发经理指示程序员对所有数学计算模块使用双精度数字
- 经过进一步计算和论证发现，即使使用单精度数字，也可以达到规定的98%的精度
- 于是程序员开始对代码部分进行了必要的更改

第三幕

- 在程序员完成工作之前，对系统的进一步测试表明，即使对实现进行了指示性的更改，该系统的平均响应时间仍将超过4.5秒，远远不超过规定的1秒.
- 进一步分析发现，问题在于复杂的图像识别算法
  - 幸运的是，刚刚发现了一种更快的算法
  - 因此使用新算法重新设计并重新实现了售票机软件
  - 这样终于可以成功实现平均响应时间的要求了

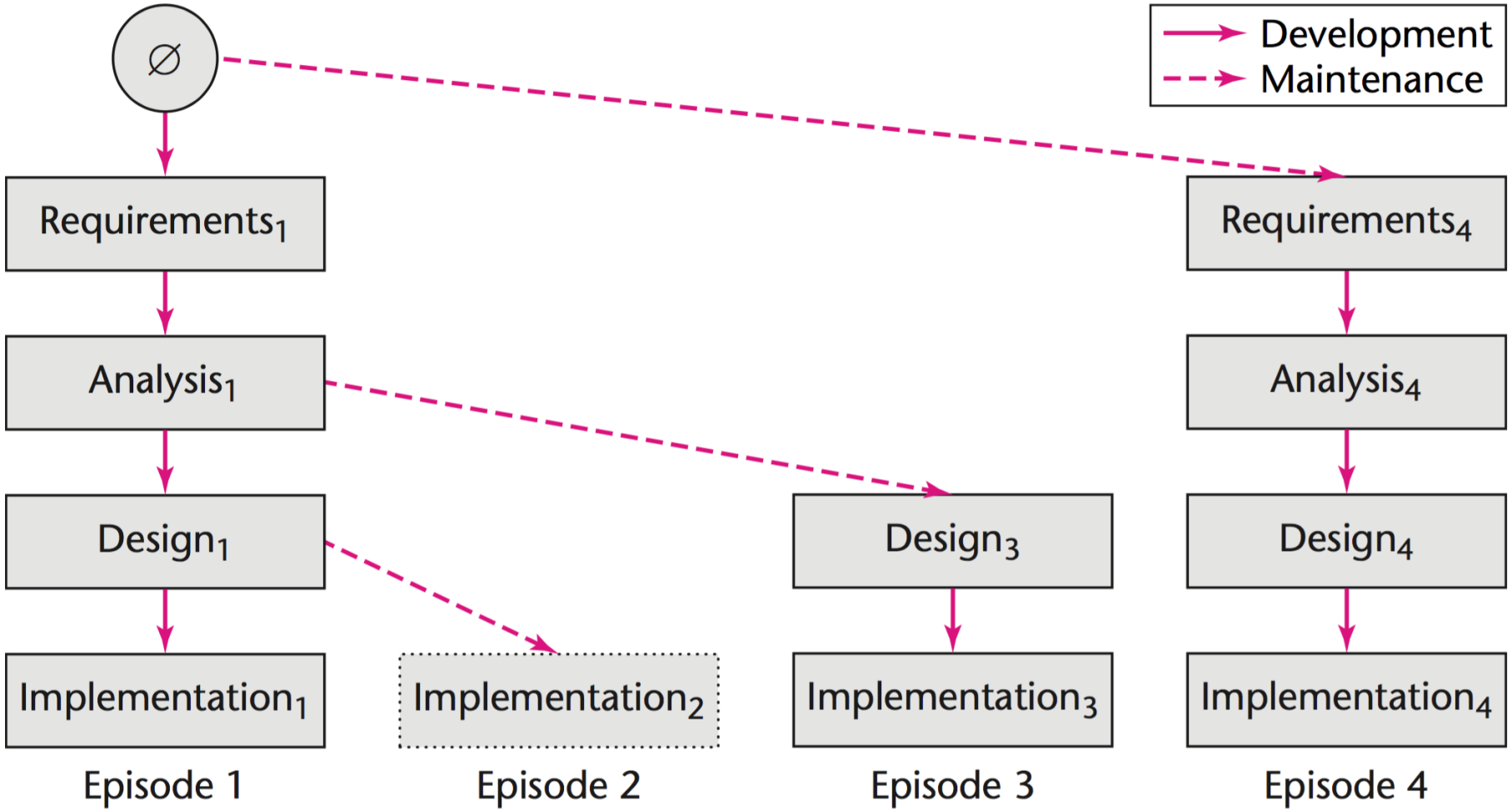
第四幕

- **到目前为止，该项目已大大落后于进度，超出了预算**
- 市长要求软件开发团队尝试尽可能提高系统的美元钞票识别组件的准确性
- 为了满足这一新要求，采用了新设计，将平均精度提高到99.5%以上

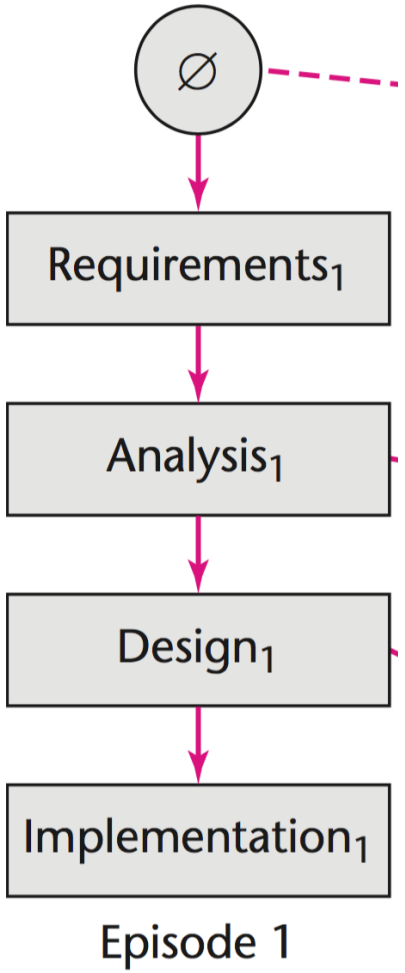
尾声

- 几年后，售票机内的传感器已过时，需要用较新的型号替换
- 原有的软件系统也就无法使用了，需要重新开发
- 大家发现新硬件设备的功能和性能都远远优于之前的，于是相应的功能就可以更丰富
- 软件的复杂度也大大提高，于是有人建议使用新的软件架构来开发
- 相应的编程语言也一起跟着更换了

Winburg 案例生命周期模型的演化树

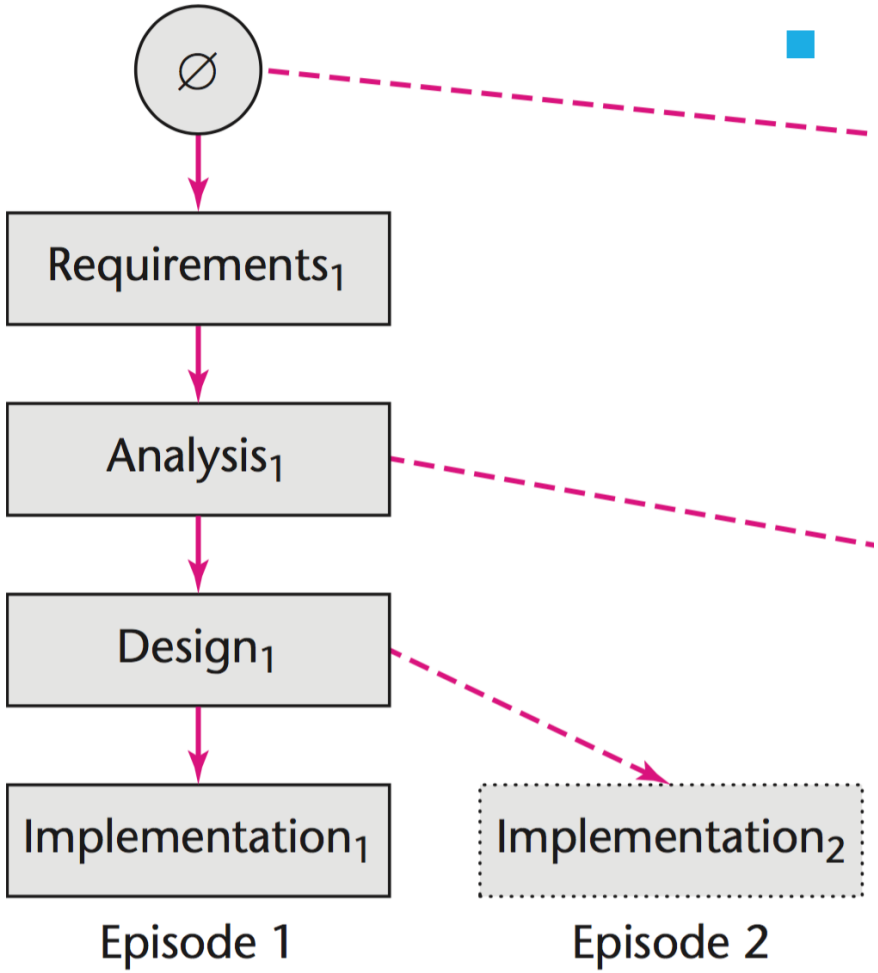


Winburg 案例生命周期模型的演化树



- 最左边的方框代表第1幕
  - 该系统是从零开始开发的
  - 依次遵循需求（Requires1），分析（Analysis1），设计（Design1）和实现（Implementation1）.
  - 接下来，如前所述，对该软件的第一版进行的试验表明，无法实现1秒的平均响应时间，因此必须修改该实现.
  - 修改后的实现显示为Implementation2.

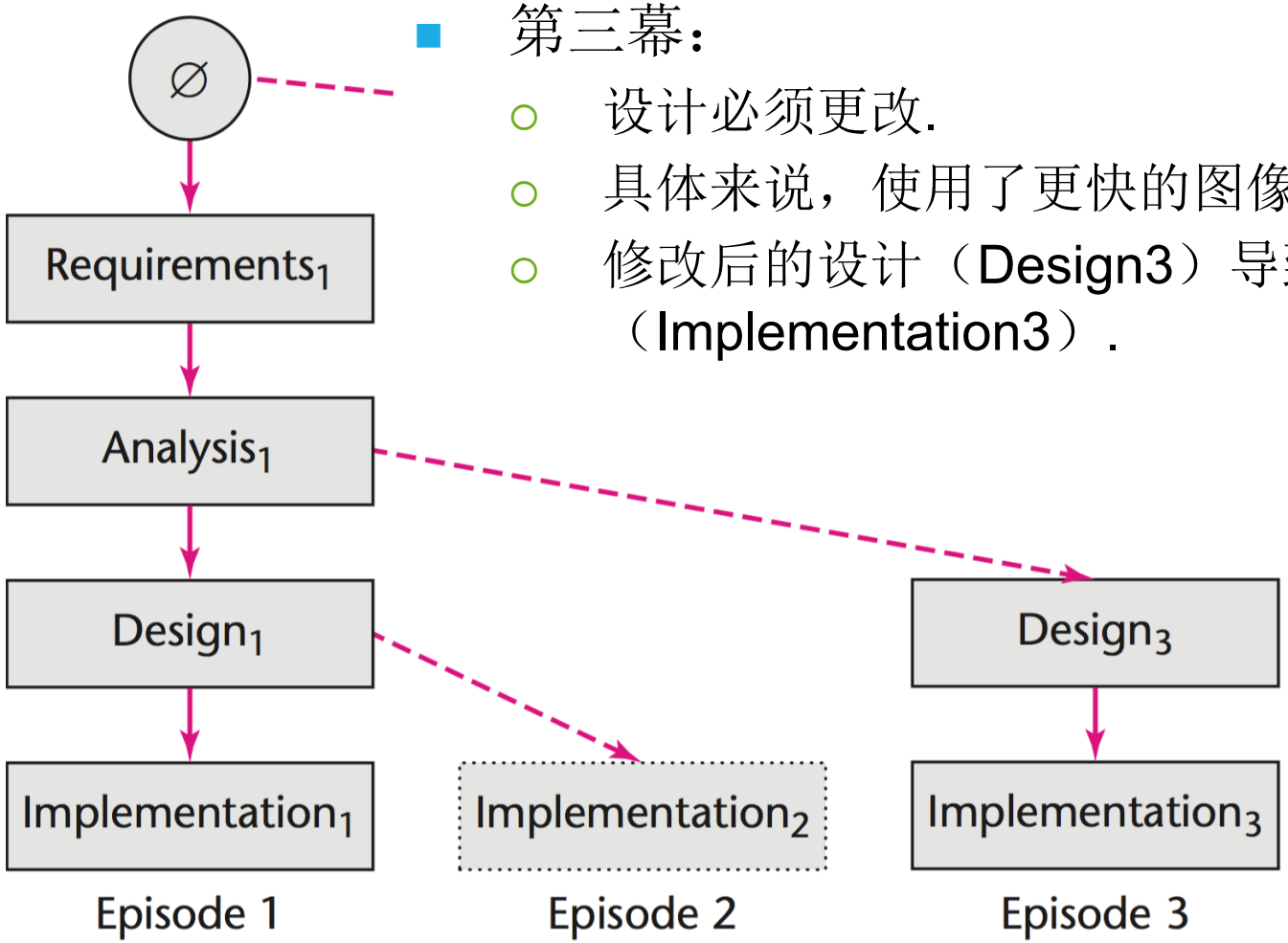
Winburg 案例生命周期模型的演化树



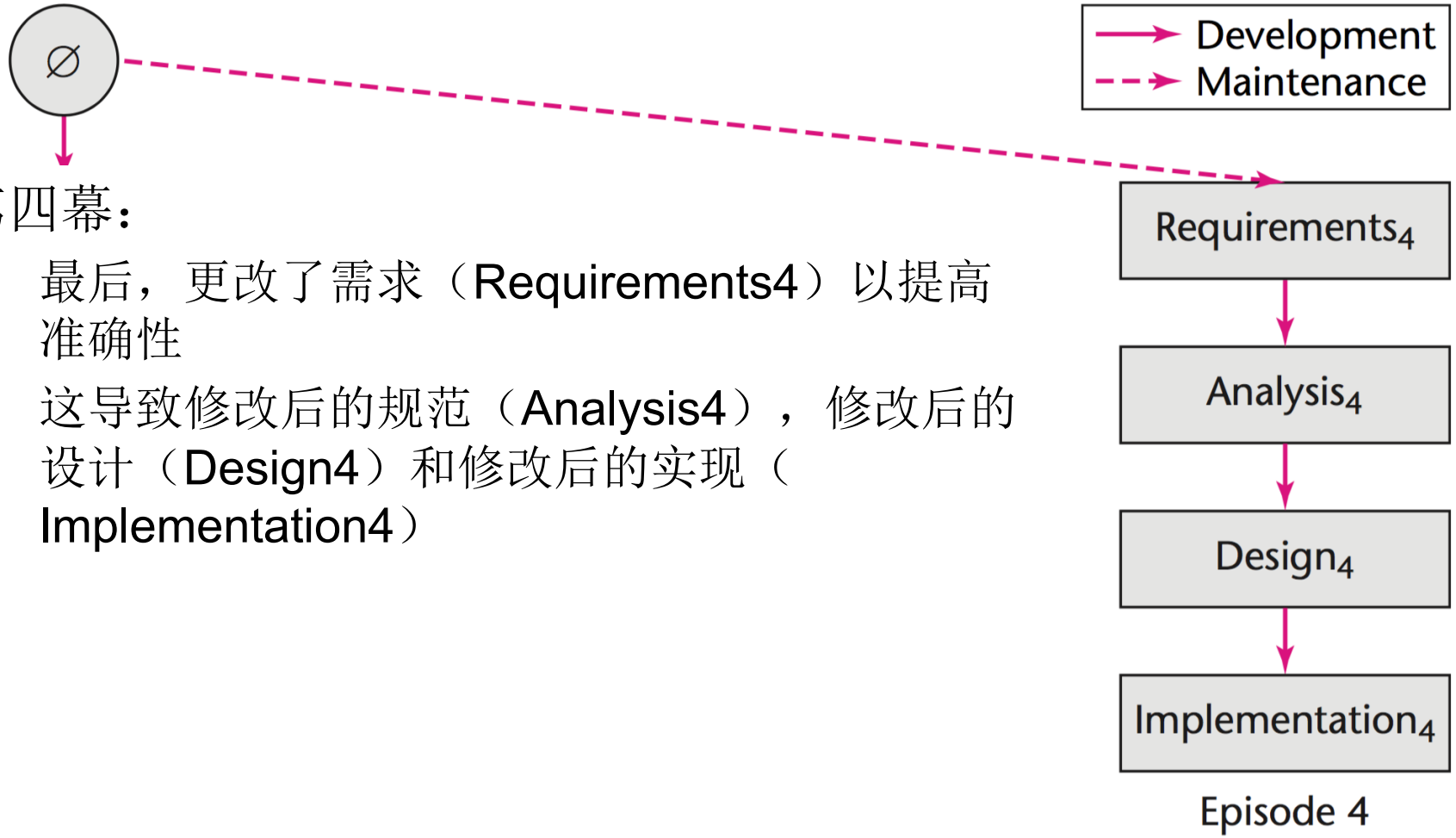
- 第二幕
  - 修改后的实现显示为Implementation2.
  - 但是，Implementation2从未完成
  - 这就是为什么用虚线绘制表示Implementation2的矩形的原因

Winburg 案例生命周期模型的演化树

- 第三幕：
  - 设计必须更改.
  - 具体来说，使用了更快的图像识别算法.
  - 修改后的设计（Design3）导致修改后的实现（Implementation3）.



Winburg 案例生命周期模型的演化树



- 第四幕：
  - 最后，更改了需求（Requirements4）以提高准确性
  - 这导致修改后的规范（Analysis4），修改后的设计（Design4）和修改后的实现（Implementation4）

- 在图中，实线箭头表示显影，虚线箭头表示维护
- 例如，当在第3集中更改设计时，Design3将Design1替换为Analysis1的设计

Baseline

- 在每个情节的结尾，我们都有一个基线/基准，即一套完整的工件
  - 工件（**artifact**）是软件产品的组成部分
- 该案例有四个基线:

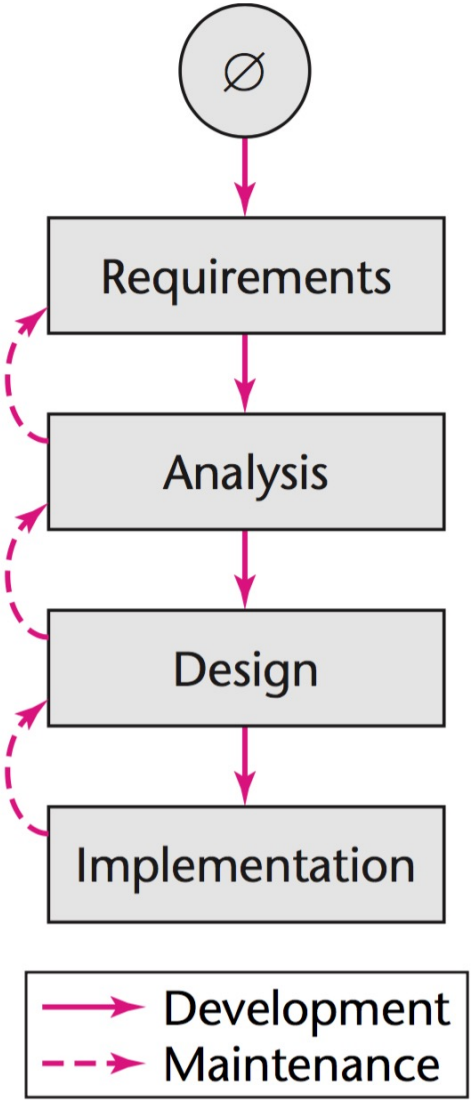
At the end of Episode 1: Requirements1, Analysis1, Design1, Implementation1

At the end of Episode 2: Requirements1, Analysis1, Design1, Implementation2

At the end of Episode 3: Requirements1, Analysis1, Design3, Implementation3

At the end of Episode 4: Requirements4, Analysis4, Design4, Implementation4

Winburg 的瀑布式生命周期模型



- 瀑布生命周期模型的简化版本
  - 可以将这种经典的生命周期模型视为具有反馈回路的线性模型.
  - 如果在设计过程中发现由于需求错误引起的故障，则按照向上的虚线箭头，软件开发人员可以从设计追溯到分析，进而追溯到需求并在此处进行必要的更正

简单总结前面两个模型

- 瀑布模型当然可以用来代表Winburg mini案例研究
- 但是，与演化树模型不同，它无法显示事件的顺序.
- *注：瀑布模型的最大优点就是其简单而本质地体现了软件开发的主线过程*

增量和迭代

- Why
  - **Miller's Law**（米勒法则）：At any one time, we humans are capable of concentrating on only approximately seven chunks.

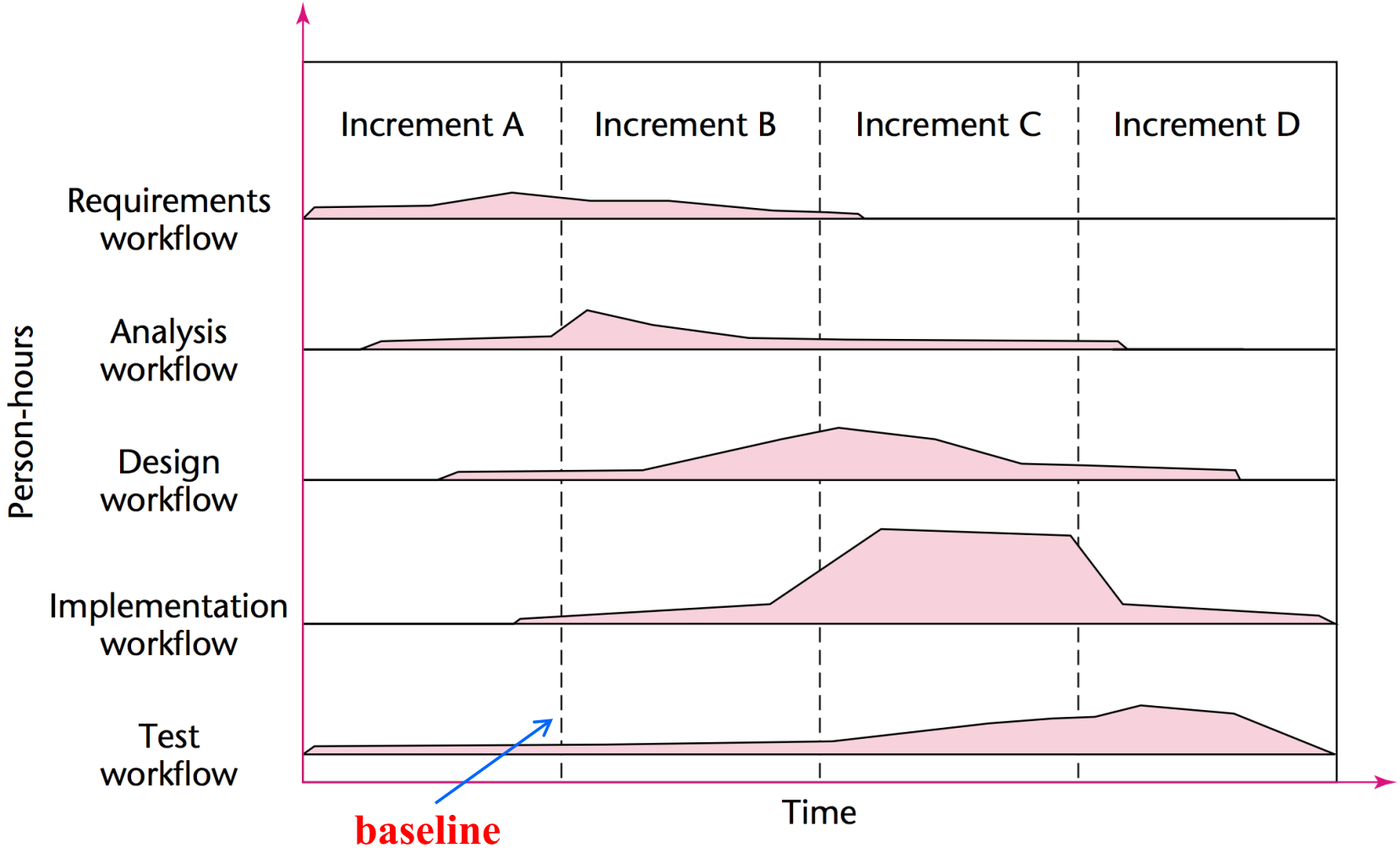
    （*在任何时候，人类最多可以将注意力集中在* ***7*** *件事情上*）
  - 应对此限制的一种可行办法就是，对于要处理的所有信息，我们可以采取逐步求精（ stepwise refinement ）的方式.

George Miller

The Magical Number Seven

生命周期的迭代和增量

The construction of a software product in four increments



Note that

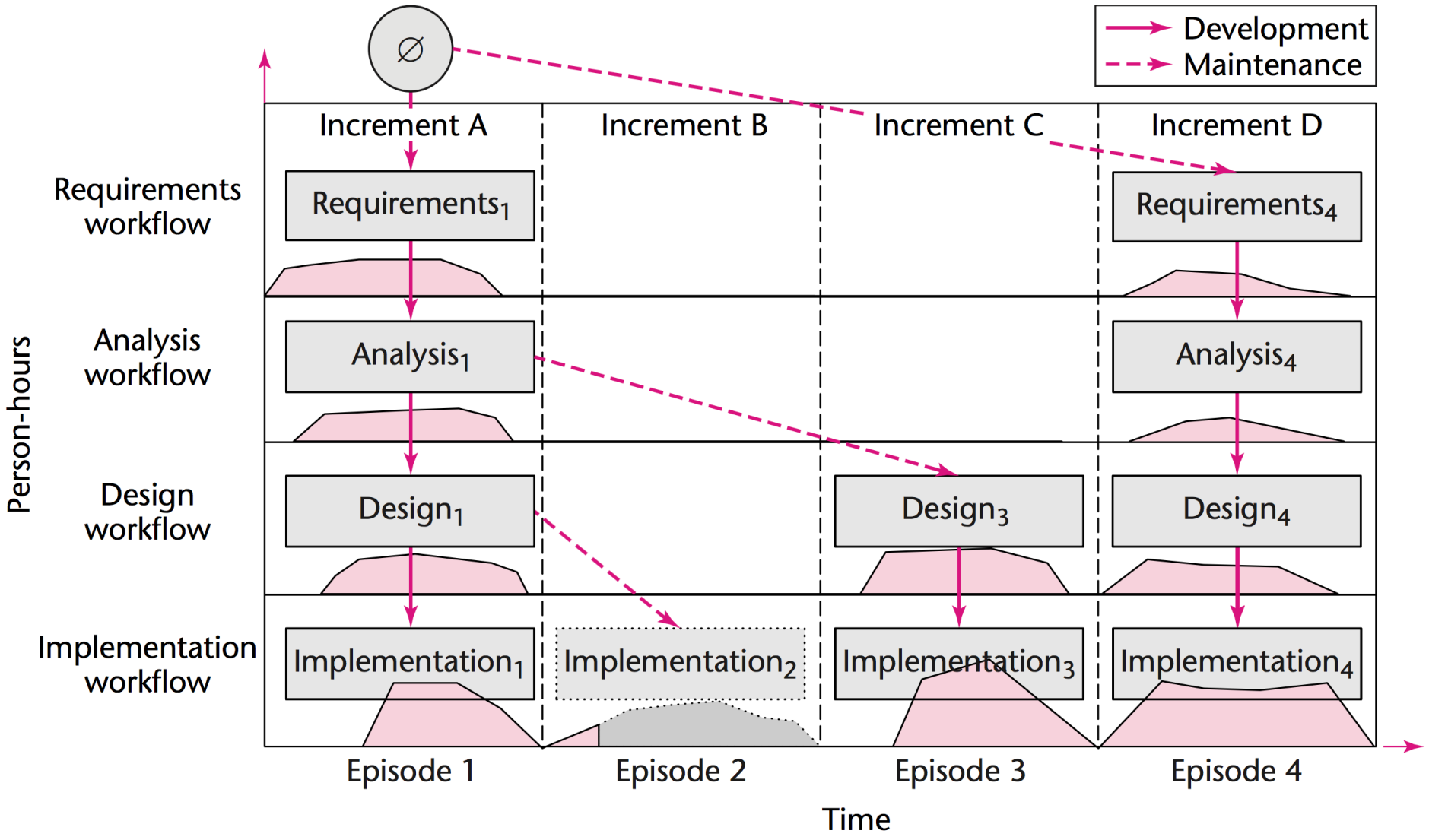
- 这里的迭代和增量的概念跟目前软件开发中所使用的概念略微有些出入，直观上讲
  - 增量（Incrementation）：指对于产品的功能上、模块上的增加
  - 迭代（Iteration）：指增量部分仍然使用相同的流程来开发

Five Core Workflows

- There are five **core workflows** (activities) performed over the entire life cycle.
  - Requirements workflow
  - Analysis workflow
  - Design workflow
  - Implementation workflow
  - Test workflow

- *注：在后面的过程框架部分还要再细讲*

The evolution-tree life-cycle model for the Winburg mini case study superimposed on the iterative-and-incremental life-cycle model.



- The above model shows the evolution-tree model of the Winburg mini case study superimposed on the iterative-and-incremental model
  - Increment A corresponds to Episode 1, Increment B corresponds to Episode 2, and so on.
  - The three dashed arrows of the evolution-tree model show that each increment constitutes maintenance of the previous increment.

Discussion 1

- 根据上面的Winburg小案例，讨论一下软件过程和模型
  - 关于模型的维度问题
  - 关于不同模型的易用性
  - 模型对问题进行刻画的能力

Discussion 2

- 根据上面的Winburg小案例，讨论一下软件过程和模型
  - 关于模型的维度问题
  - 关于不同模型的易用性
  - 模型对问题进行刻画的能力

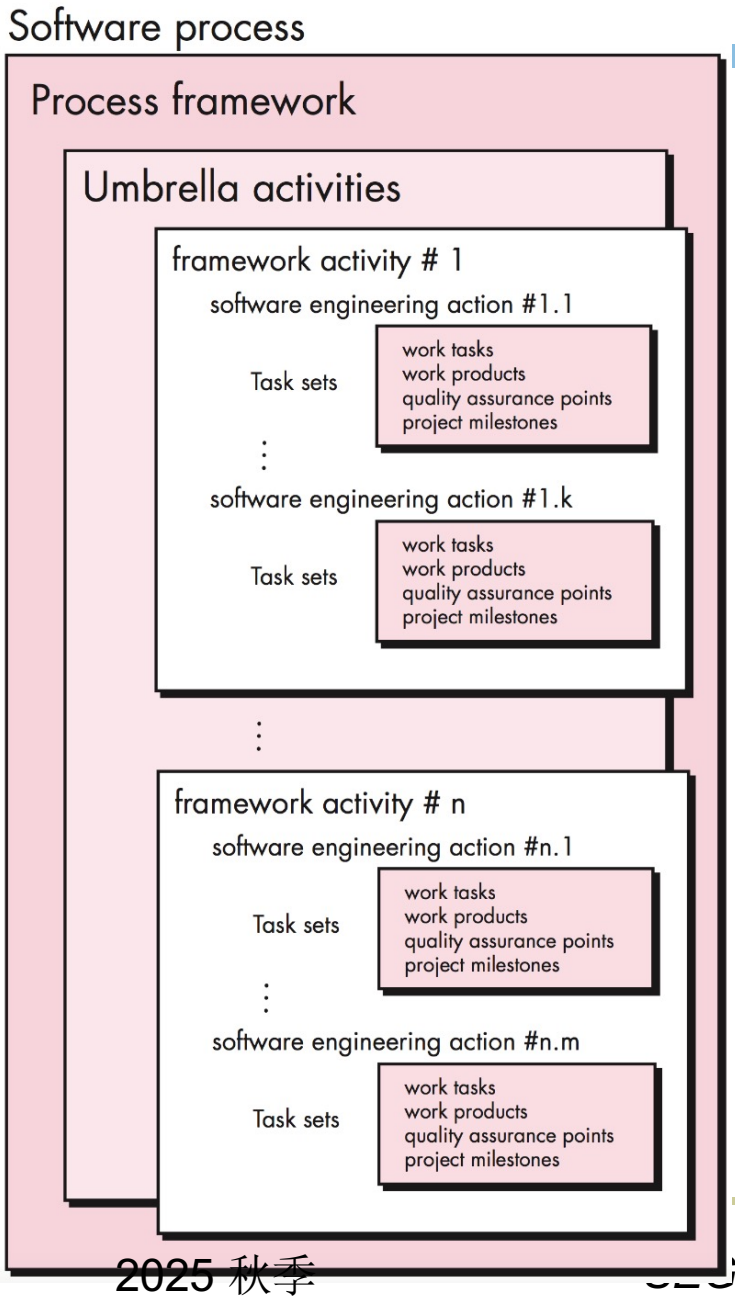
A generic process model

- A generic process framework for software engineering defines **five** framework activities:
  - Communication
  - Planning
  - Modeling
  - Construction
  - Deployment
- In addition, a set of umbrella activities are applied throughout the process

Umbrella Activities

- Typical umbrella activities:
  - project tracking and control
  - risk management
  - quality assurance
  - configuration management
  - technical reviews

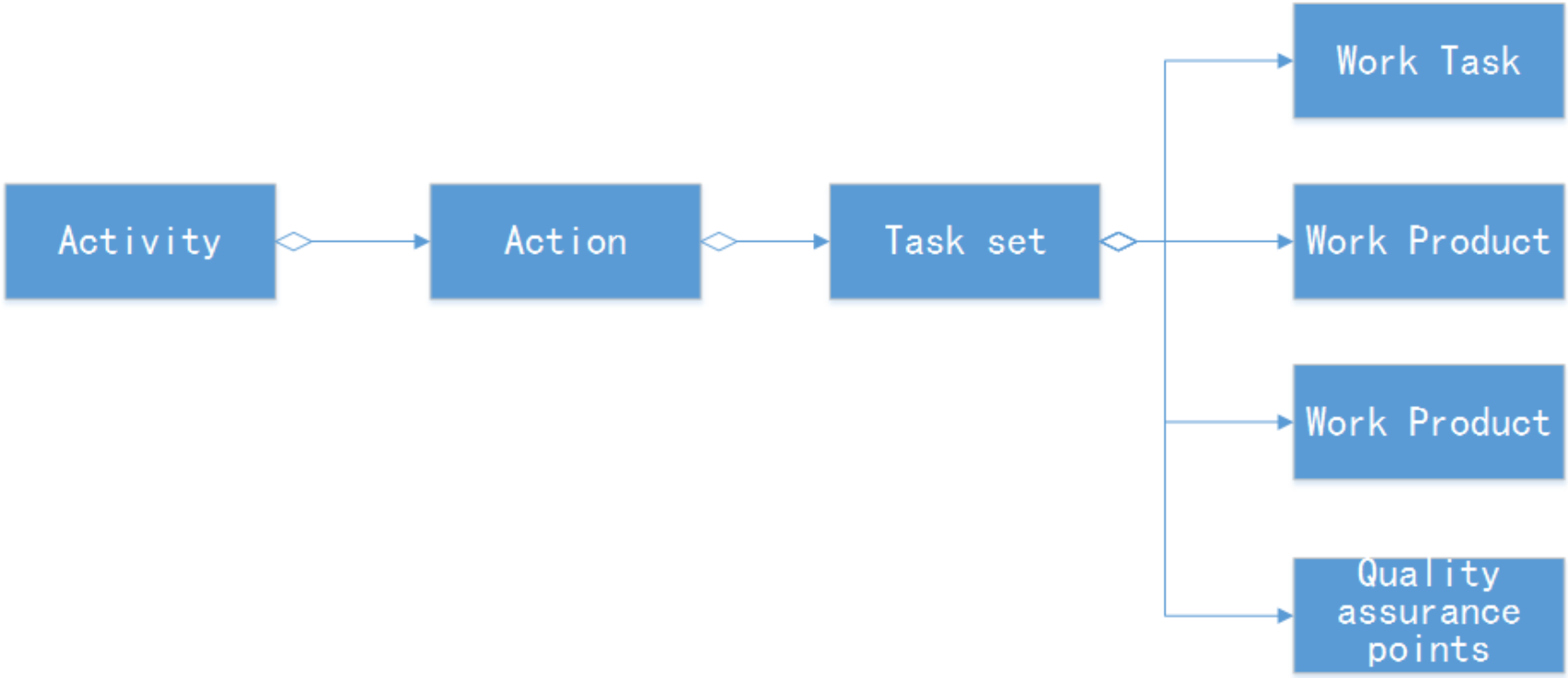
A software process framework



- Key concepts:
  - framework activity
  - software engineering action
  - task sets
    - software engineering work tasks
    - work products
    - quality assurance points
    - project milestones

Relationship

- Relationships between them are:



Task Set

- A task set defines the actual work to be done to accomplish the objectives of a software engineering action.

- For example
  - Requirements gathering is an important software engineering action that occurs during the communication activity.

A *small*, relatively *simple* project

- The *task set* for requirements gathering might look like this:
  1. Make a list of stakeholders for the project.
  2. Invite all stakeholders to an informal meeting.
  3. Ask each stakeholder to make a list of features and functions required.
  4. Discuss requirements and build a final list.
  5. Prioritize requirements.
  6. Note areas of uncertainty.

A *larger*, more *complex* software project

- It might encompass the following *work tasks*:
  1. Make a list of stakeholders for the project.
  2. Interview each stakeholder separately to determine overall wants and needs.
  3. Build a preliminary list of functions and features based on stakeholder input.
  4. Schedule a series of facilitated application specification meetings.
  5. Conduct meetings.
  6. Produce informal user scenarios as part of each meeting

7. Refine user scenarios based on stakeholder feedback.
8. Build a revised list of stakeholder requirements.
9. Use quality function deployment techniques to prioritize requirements.
10. Package requirements so that they can be delivered incrementally.
11. Note constraints and restrictions that will be placed on the system.
12. Discuss methods for validating the system.

Comparing two above

- Both of these task sets achieve “requirements gathering”
- they are quite different in their *depth* and *formality*
- The software team chooses the task set that will allow it to achieve the goal of each action and still *maintain quality and agility*.

Prescriptive Life-Cycle Models

- Code-and-Fix Life-Cycle Model
- Waterfall Life-Cycle Model
- Rapid-Prototyping Life-Cycle Model

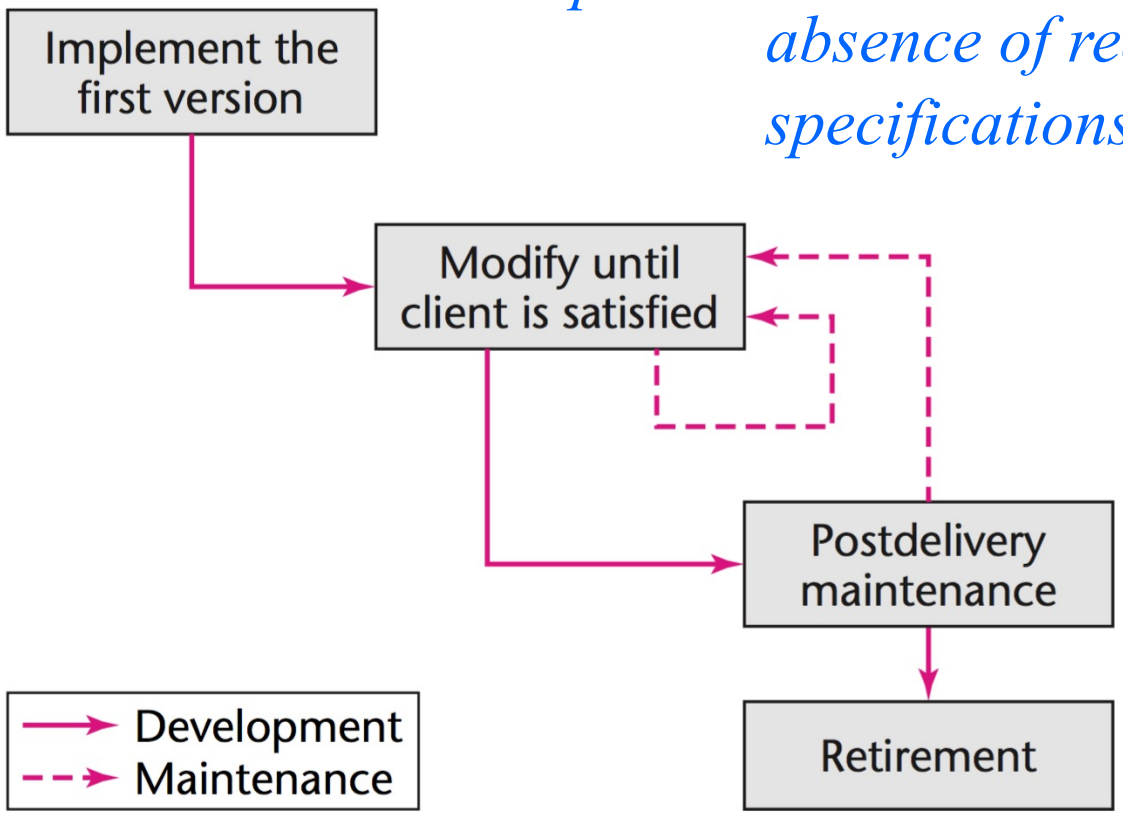
Code-and-Fix Life-Cycle Model

- The product is implemented without requirements or specifications, or any attempt at design.

- Instead, the developers simply throw code together and rework it as many times as necessary to satisfy the client.

simple

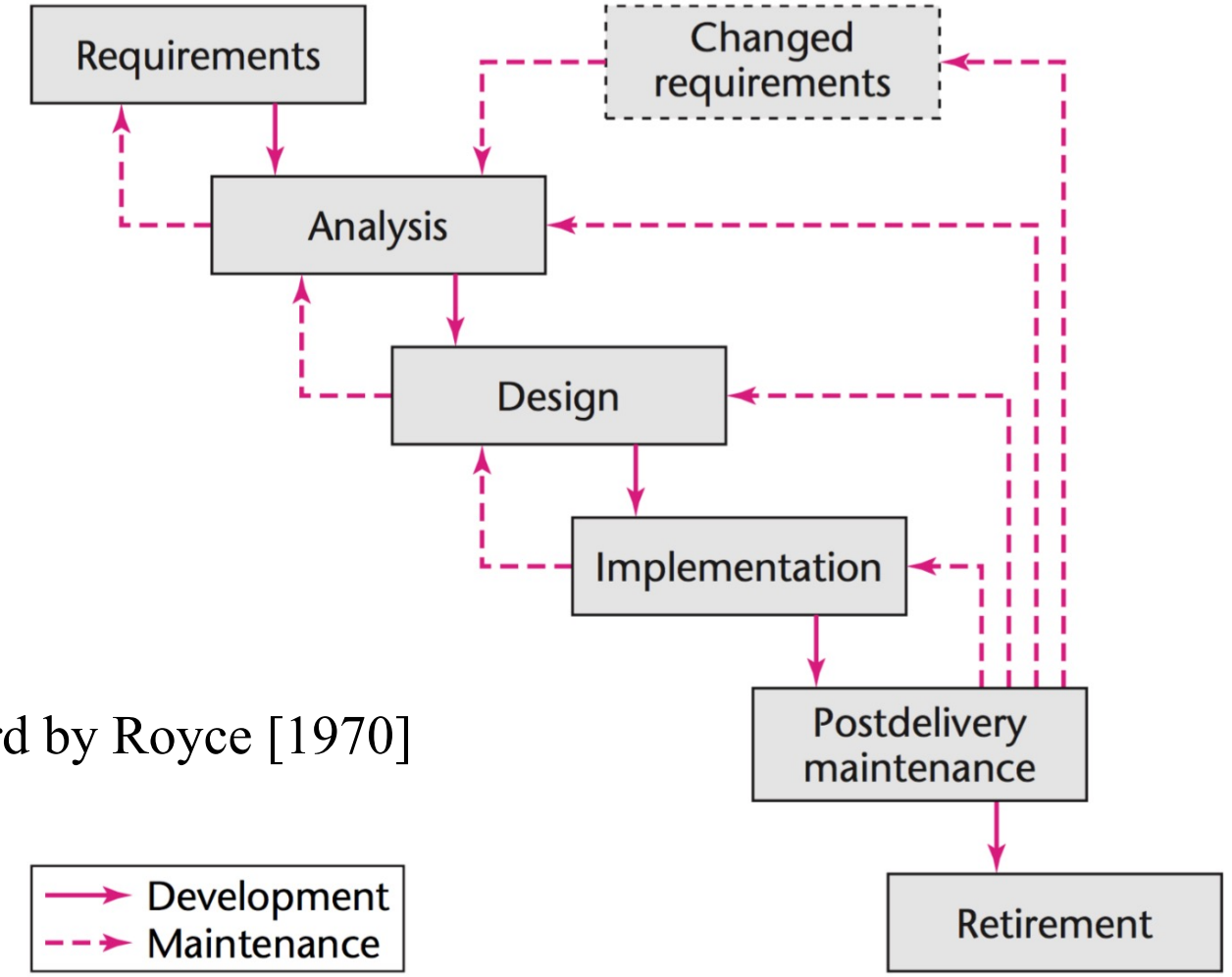
absence of requirements, specifications, and design



high cost

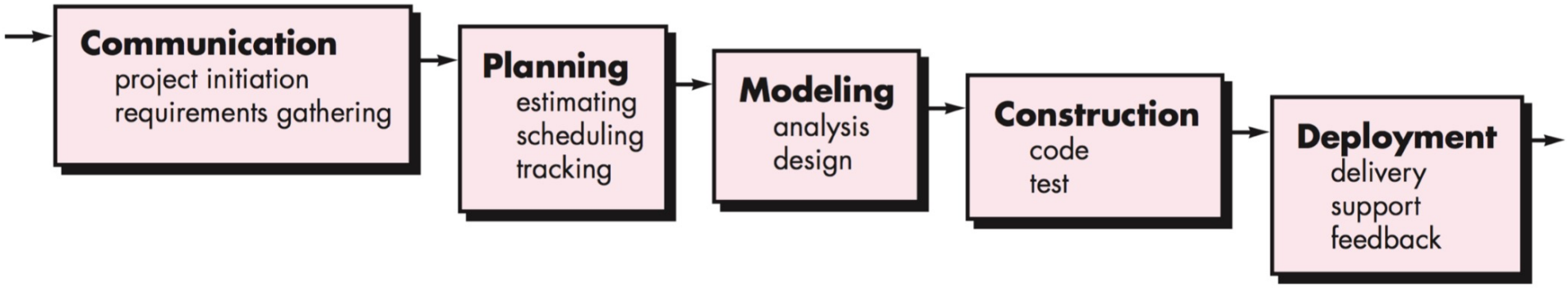
The code-and-fix model is the *easiest* way to develop software—and by far the *worst* way.

Waterfall Life-Cycle Model



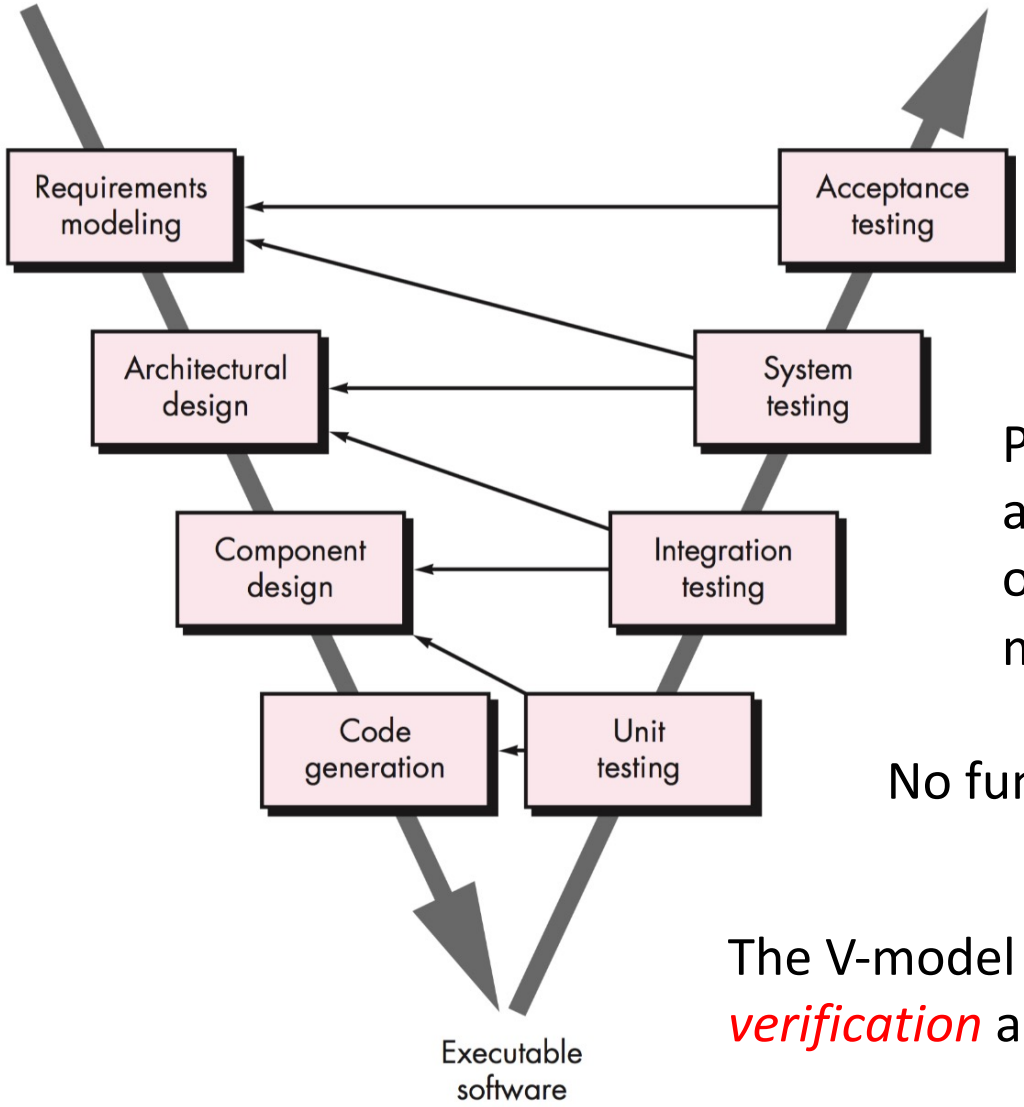
Put forward by Royce [1970]

Waterfall Process Model



- The *waterfall model,* sometimes called the *classic life cycle*, suggests a systematic, *sequential* approach
- begins with *customer specification of requirements* and progresses through *planning*, *modeling*, *construction*, and *deployment*

The V-model



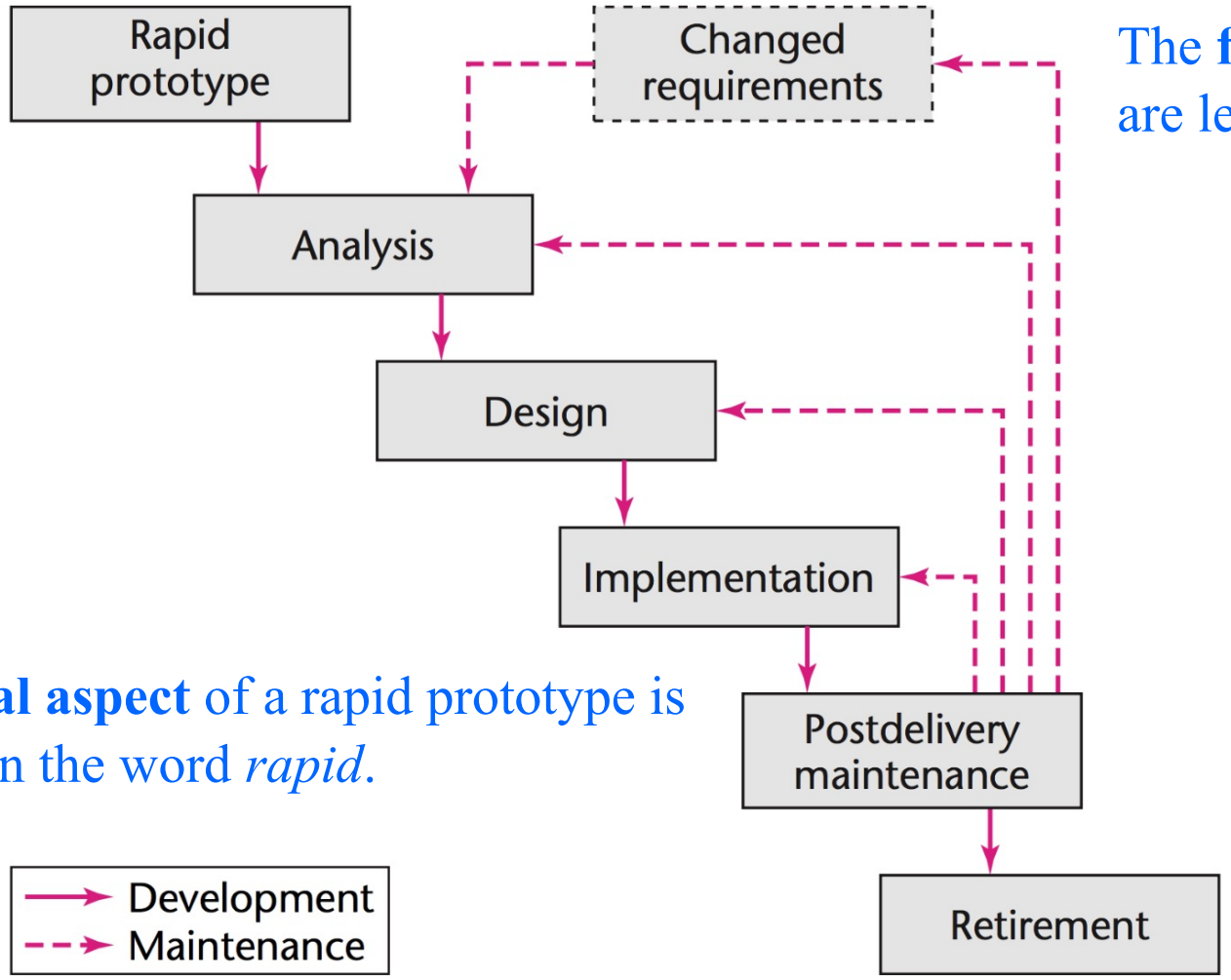
A variation of the waterfall model is called the *V-model* [Buc99].

Performing a series of tests (quality assurance actions) that validate each of the models created as the team moved down the left side.

No fundamental difference

The V-model provides a way of visualizing how *verification* and *validation* actions are applied.

Rapid-prototyping Life-cycle Model



The **feedback loops** are less needed

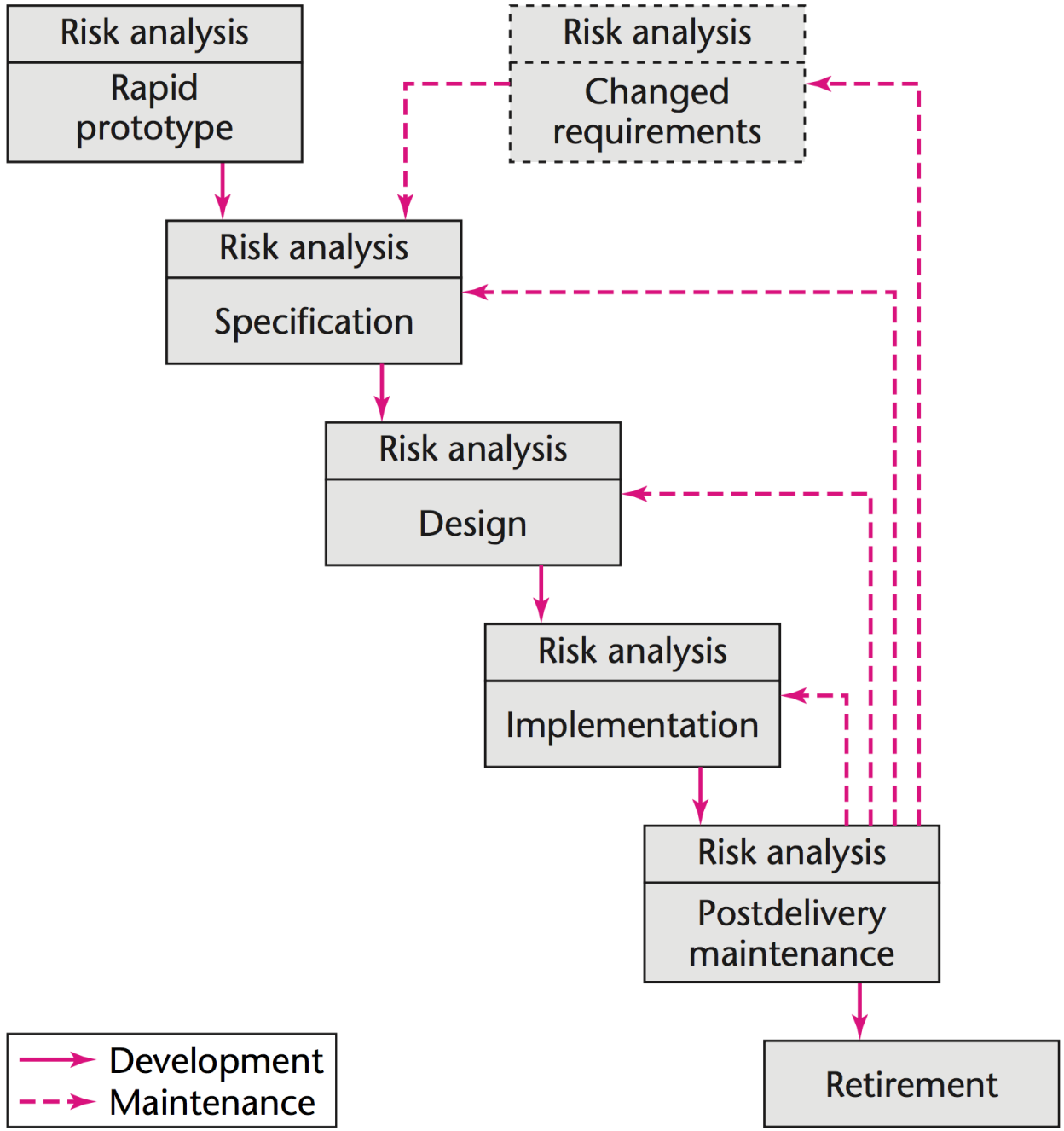
An **essential aspect** of a rapid prototype is embodied in the word *rapid*.

- The first step is to build a *rapid prototype* and let the client and future users *interact* and *experiment* with the rapid prototype.

- Once the client is satisfied, the developers can draw up the ***specification document*** with some assurance that the product meets the client's real needs.

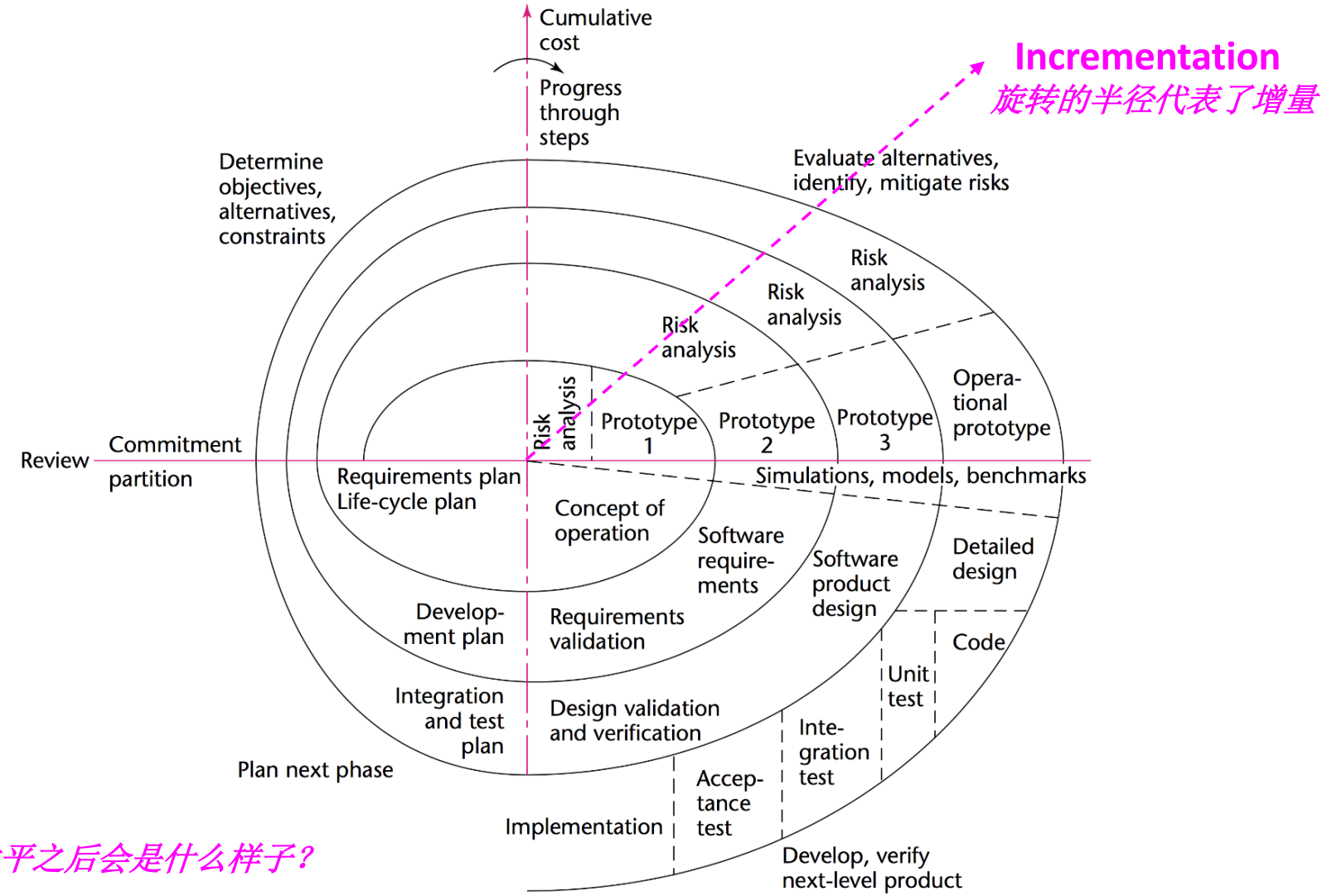
Spiral Life-Cycle Model

- The idea of ***minimizing risk*** via the use of ***prototypes*** and other means is the idea underlying the spiral life-cycle model [Boehm, 1988].
- A *simplified* way of looking at this life-cycle model is as a *waterfall* model with each phase preceded by risk analysis
- Spiral Model ≈ Prototyping Model + Waterfall Model



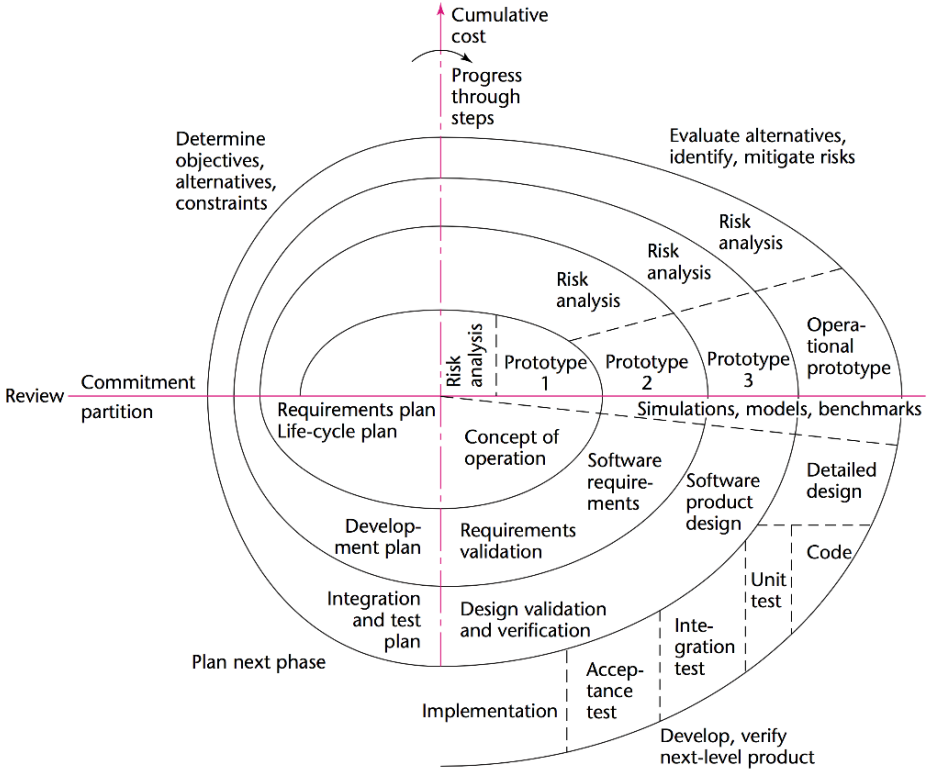
核心思想就是利用阶段性瀑布模型，再控制每个阶段瀑布开始时的风险（用原型）

Full spiral life-cycle model

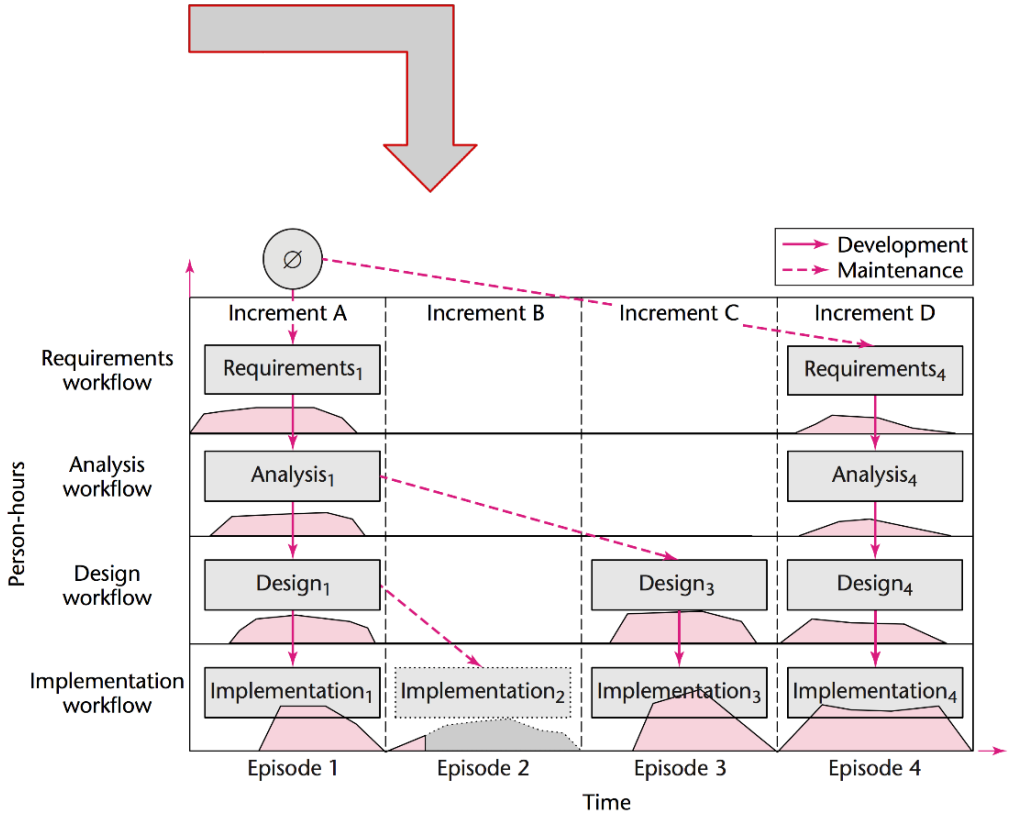


思考:
将这个螺旋拉平之后会是什么样子?

Understanding Spiral Model



Two dimensions：
Incrementation & Iteration



Reference

- Object-Oriented and Classical Software Engineering (8 edition), Stephen Schach, McGraw-Hill Education, 2010
- Software Engineering: A Practitioner's Approach (8 edition), Roger S. Pressman, McGraw-Hill Education, 2014