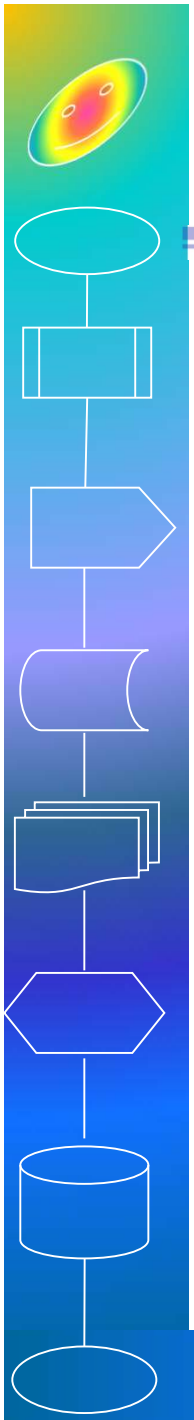


第3章 软件开发过程

先有问题，后有方法！

总结了方法，可以缩短解决类似问题的时间和成本！

主要内容

- 3.1 程序开发基本过程
- 3.2 从程序开发到软件工程化过程
- 3.3 中间产品驱动的过程
- 3.4 瀑布还是迭代？
- 3.5 软件产品的开发过程
- 3.7总结

一个简单的例子

- Y(t) = 5.0sin(t) + 6.0 cos(t) + 7.0 log(t) +0.5t + 4.0user1(ti) + 3.0 user2(ti)

主要内容

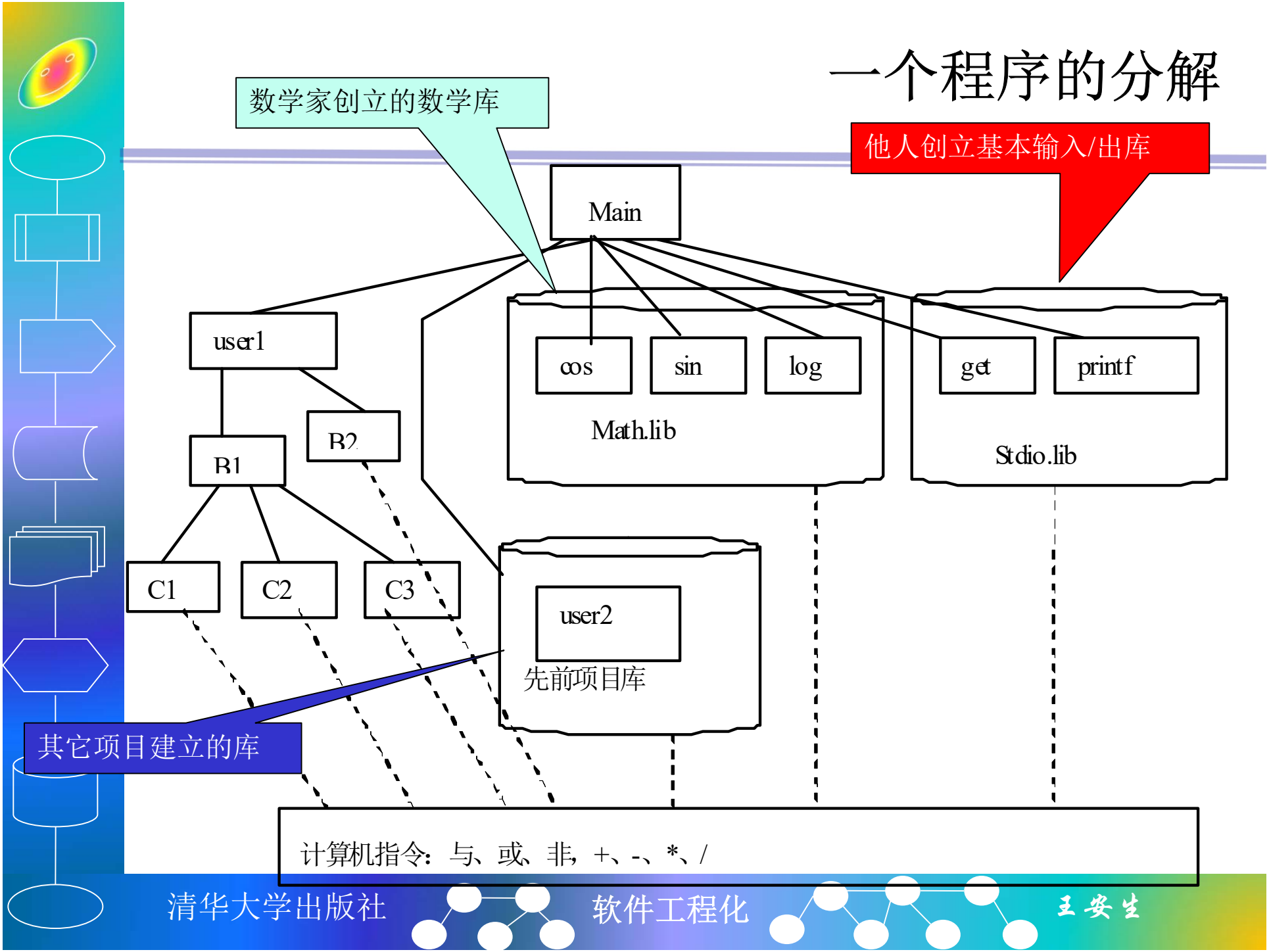
```
#include <math.h>                                          /*第1行*/
#include <stdio.h>                                         /*第2行*/
  main();    /*主程序入口 */                               /*第3行*/
{                                                          /*第4行*/
  Printf (…..)  /* 在屏幕上提示用户输入一个ti */           /*第5行*/
get(ti);   /* 得到ti */                                                  /*第6行*/
/* 计算Y, 依据C编译厂家提供的数学库中的sin、cos和log函数 */             /*第7行*/
Y = 5.0*sin(ti) + 6.0*cos(ti)*+ 7.0*log(ti) +0.5*ti ;                  /*第8行*/
Y= Y +4.0*user1(ti);  /* 计算Y, 调用自个开发的函数user1*/             /*第9行*/
Y= Y +3.0*user2(ti);  /* 计算Y, 调用先前项目开发出的函数user2*/       /*第10行*/
 Printf (…..);  /* 在屏幕显示Y */                                      /*第11行*/
} /*结束返回*/                                                        /*第12行*/
```

将源程序命名为main.c，那么，该例子的建造过程如下：
将main.c源程序编译为main.obj---目标码程序；
将main.obj与所需要的库程序链接(装配)起来，例如，采用如下的命令行：
  Link main.obj + stdio.lib + math.lib

程序开发库和复用

要求的功能	使用已有的库函数	不使用库函数
1. ti 的输入 2. Y的输出	借用C 语言开发环境中的stdio(标准输入输出)库中的get和printf语句，直接完成	必须自己编写端口的输入和输出
3.sin/cos/log计算	借用C 语言开发环境中的math(数学)库中的sin、cos、log函数。	用级数展开，用加减乘除实现sin、cos、log函数。考虑计算精度。
4）user1(ti)	编写User1(ti)，, 考虑计算精度	编写User(ti), 考虑计算精度
5）user2(ti)	调用先前项目的User2(ti)，如果质量合格的话(例如，计算精度符合要求)	

一个程序的分解


数学家创立的数学库
他人创立基本输入/出库
Main
user1
B1
B2
C1
C2
C3
cos
sin
log
Math.lib
get
printf
Stdio.lib
user2
先前项目库
其它项目建立的库
计算机指令：与、或、非，+、-、*、/

程序的开发过程

- 第一步是建立程序库。
  - 一个程序库是将满足一定质量要求的相关程序功能函数聚集在一起，供其他人重复使用的库。程序库可以大大提高后来项目的开发效率，节约成本。
  - 这些库可能会受到程序版权和许可证的限制，产生费用，被称为现货软件 (OTS—Off-The-Shelf)。OTS软件有两种用法，一种是最终用户直接可用，另一种是通过对其功能的调用或二次开发后供最终用户使用。OTS的最常见形式是商业现货(COTS-- commercial OTS)—商业部件----公司之间所产生的软件部件买卖关系。

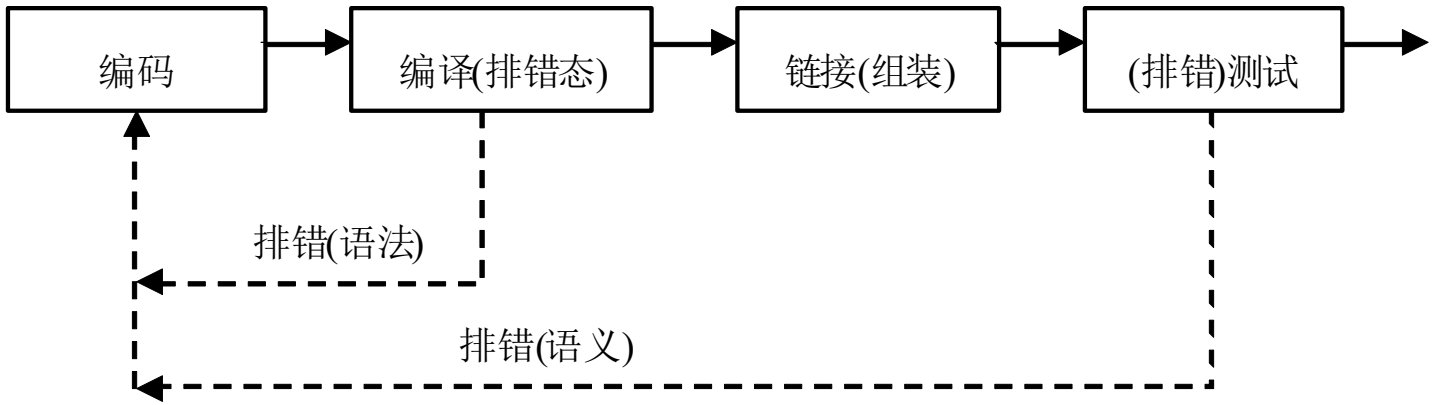
程序的开发过程

- 第二步，
  - 还会用到企业先前项目开发出的软件。虽然软件的原开发者和二次开发者之间没有直接的商业关系，但是，也形成了软件供货者和二次开发使用者之间的关系。二次开发者和原开发者谁对这部分软件的质量负责成为软件开发管理中的一个问题。
- 第三步，
  - 软件项目组在开发过程中必须设计、分解和建立项目组所能公共使用的软件功能函数，并把这些功能聚合在一起形成该项目的软件库。软件库的建立可以在保证项目组最大程度复用代码和功能的同时，降低代码的工作量，提高软件的质量。
- 库至少包括：
  - 1）COTS库—由企业外部提供的软件部件；
  - 2）企业先前项目形成的库；
  - 3）本项目所建立的库。

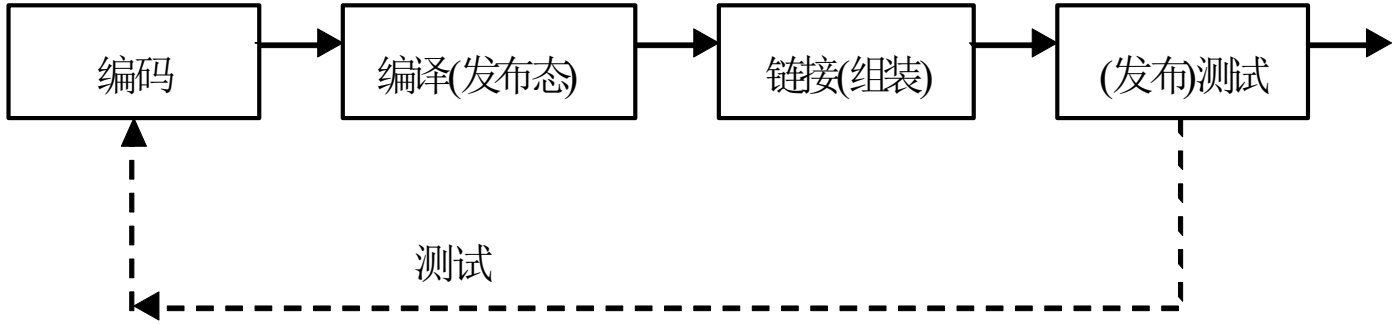
3.2 从程序开发到软件工程化过程

- 3.2.1 软件开发的活动
- 3.2.2 大型软件开发的管理

软件开发的活动

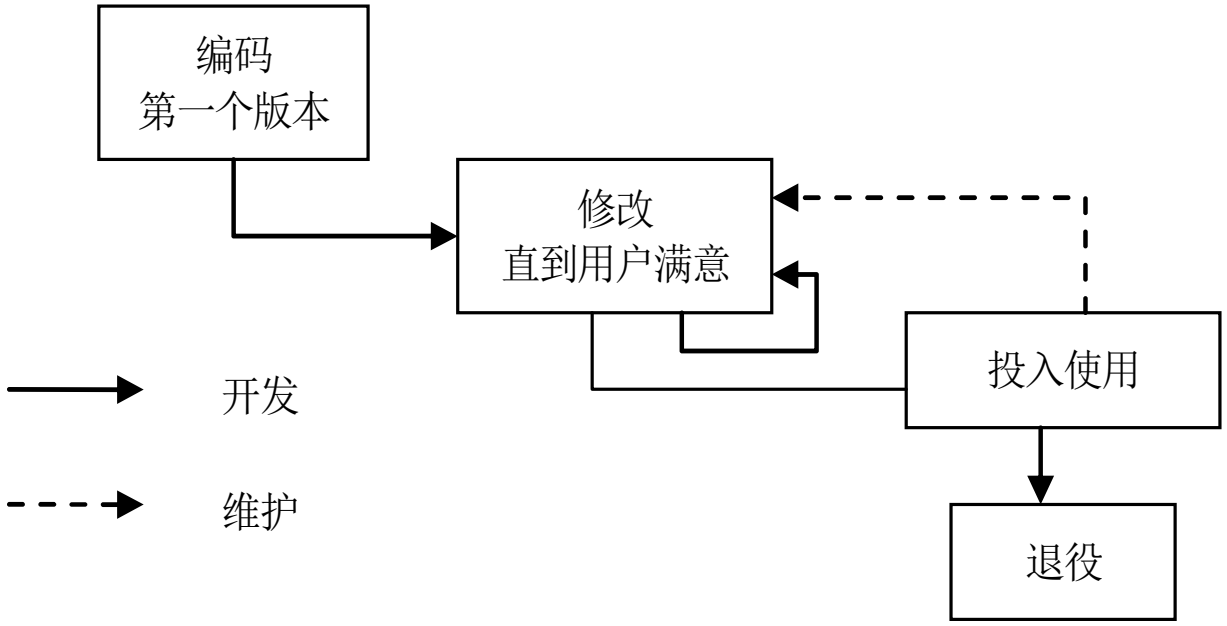


开发过程的基本活动---排错



开发过程的基本活动---发布

Build-and-Fix的软件开发模型

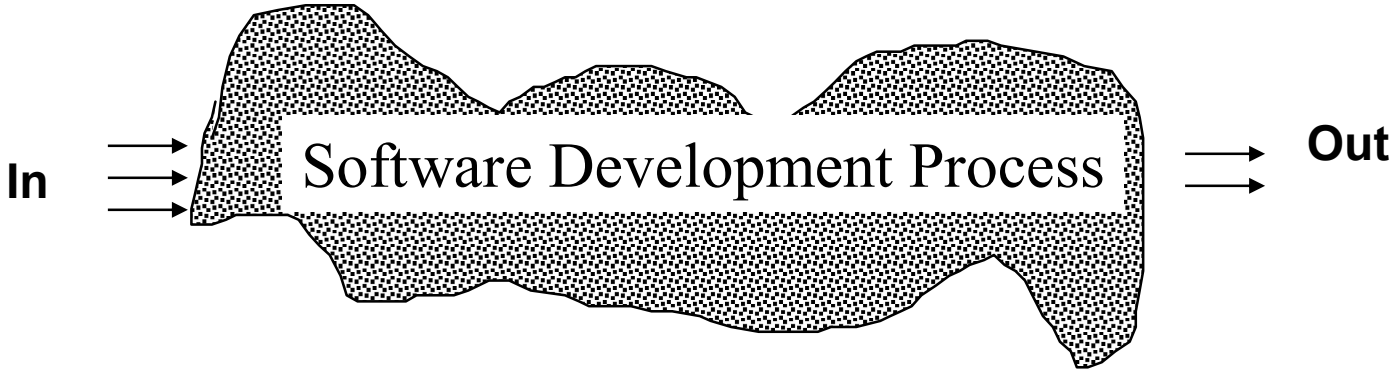


“建造与调试”的模型：

Some persons think that the good software come from debugging and testing. Yes, it is true for a small project, build by 1~3 persons.

Process Visibility

仅仅关注编码、调试、修改的做法，导致只有少数人能看懂工程的进展情况，就像变魔术的黑箱一样，“用户和管理者搞不清代码是如何变出来的”，自然，不知道项目能否按时、按质、按需求完成。



黑箱式的、变魔术式的软件开发是导致项目失败的重要原因！

科学工程实验 PK 伪科学

科学和工程实验

- 关注过程，
- 过程是可重复的、
- 过程是可追溯的
- 过程是可见的
- 过程是可以被测量
- 因此，结果是可重复的、真实的、可信的。
- 重复科学实验过程，就可用于生产

伪科学和工程

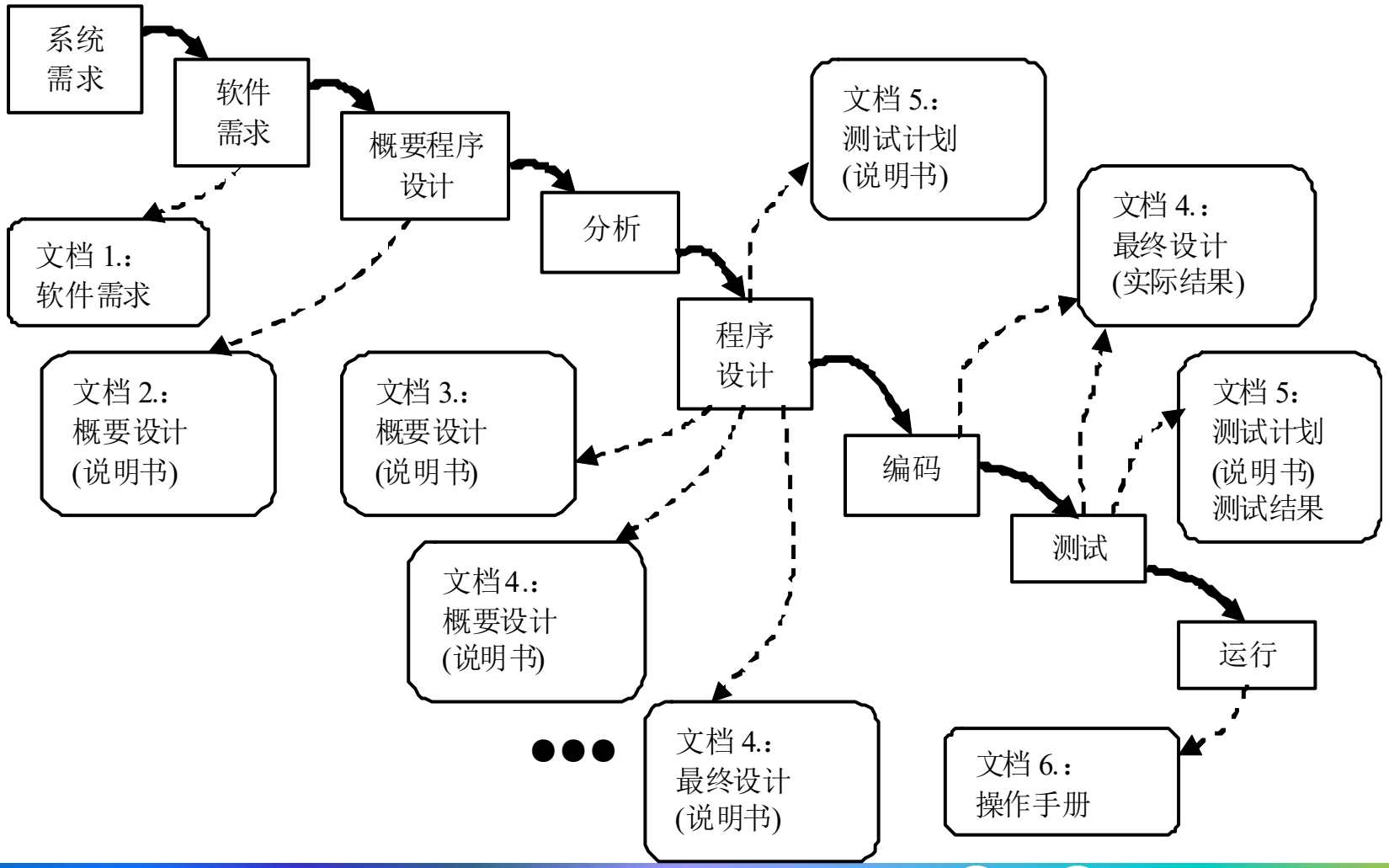
- 仅仅关注结果
- 过程是不可见的黑箱
- 过程是不可追溯的，也不让你追溯，否则就露馅了
- 过程是不可重复的，时刻变化
- 过程是不可测量的
- 因此，无法证实结果真实性
- 但是，自己不承认自己作假！

魔术师

承认过程是不可见的。因此，魔术是真实的。魔术过程是不能批量重复的。如果魔术师真能变出汽车，就让他替代汽车制造厂！

大型软件开发的管理

- Winston Royce总结了9年的开发经验，于1970年发表了《MANAGING THE DEVELOPMENT OF LARGE SOFTWARE SYSTEMS》一文，提出开发大程序的5个步骤：
- **第一步：先进行程序设计。**这一步的工作集中在：1）与程序设计者，而不是分析人员和程序员，一起进行设计；2）设计、定义和分配数据处理模式，即便可能有风险。分配进程和功能，设计数据库，定义数据处理，分配执行时间，定义接口和操作系统的处理模式，描述输入/输出处理，定义概要的操作使用规程；3）编写系统概要文档。
- 描述出对系统的可理解、信息和当前状态。每个工作人员要对系统具有基本的理解。至少有一人对系统全面和深入的理解。
- **第二步：编写设计文档，**包括：1）软件需求文档，2）编写概要程序设计(说明书)，3）编写接口设计(说明书)、最终设计(说明书)、以及测试计划，4）为编码和测试编写最终设计(建造)，5）编写测试计划(说明书)测试结果，6）编写使用手册。

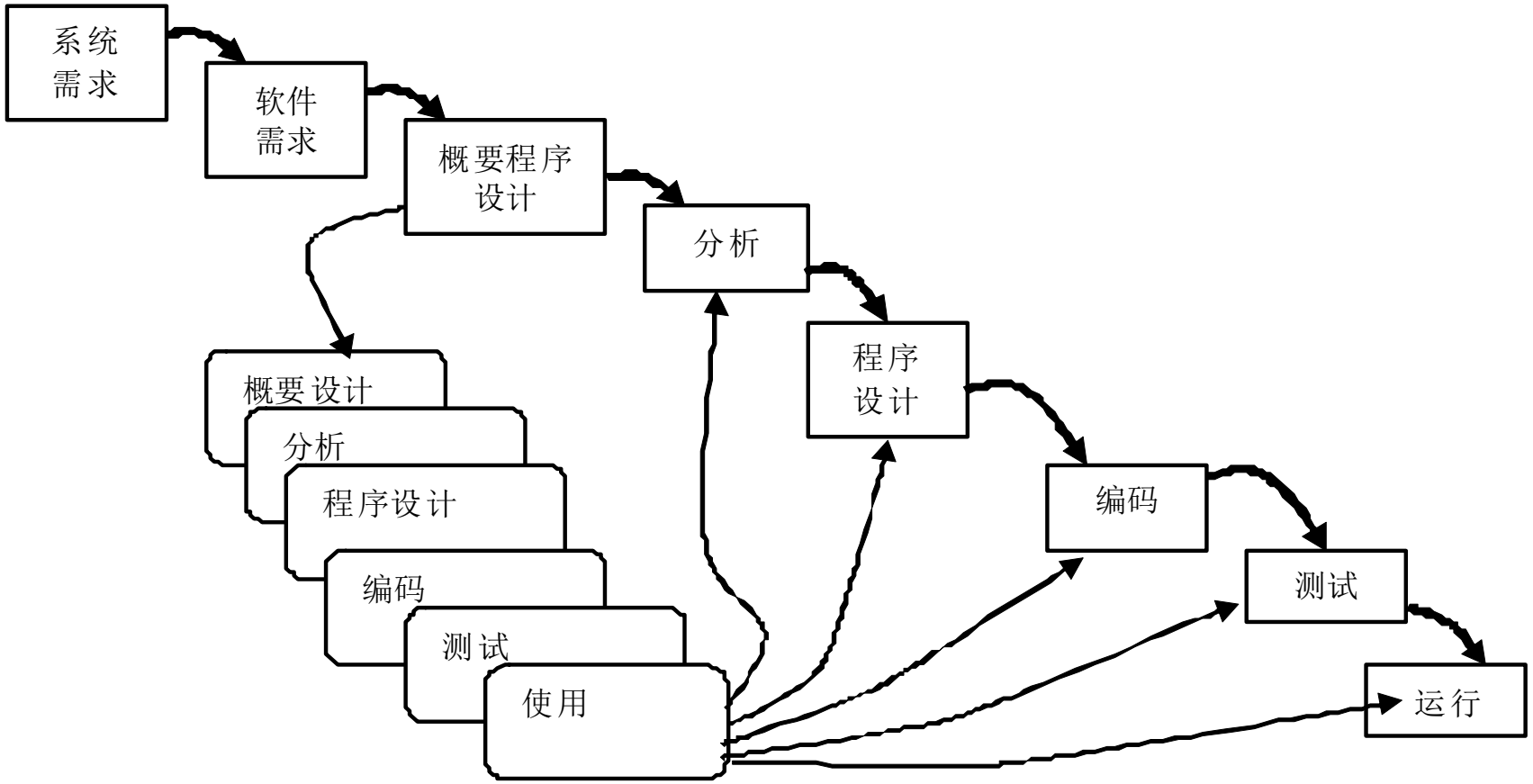

系统
需求
软件
需求
概要程序
设计
分析
文档 5.:
测试计划
(说明书)
文档 4.:
最终设计
(实际结果)
文档 1.:
软件需求
程序
设计
文档 2.:
概要设计
(说明书)
文档 3.:
概要设计
(说明书)
编码
文档 5:
测试计划
(说明书)
测试结果
测试
文档4.:
概要设计
(说明书)
运行
●●●
文档 4.:
最终设计
(说明书)
文档 6.:
操作手册

软硬件规格说明的差异

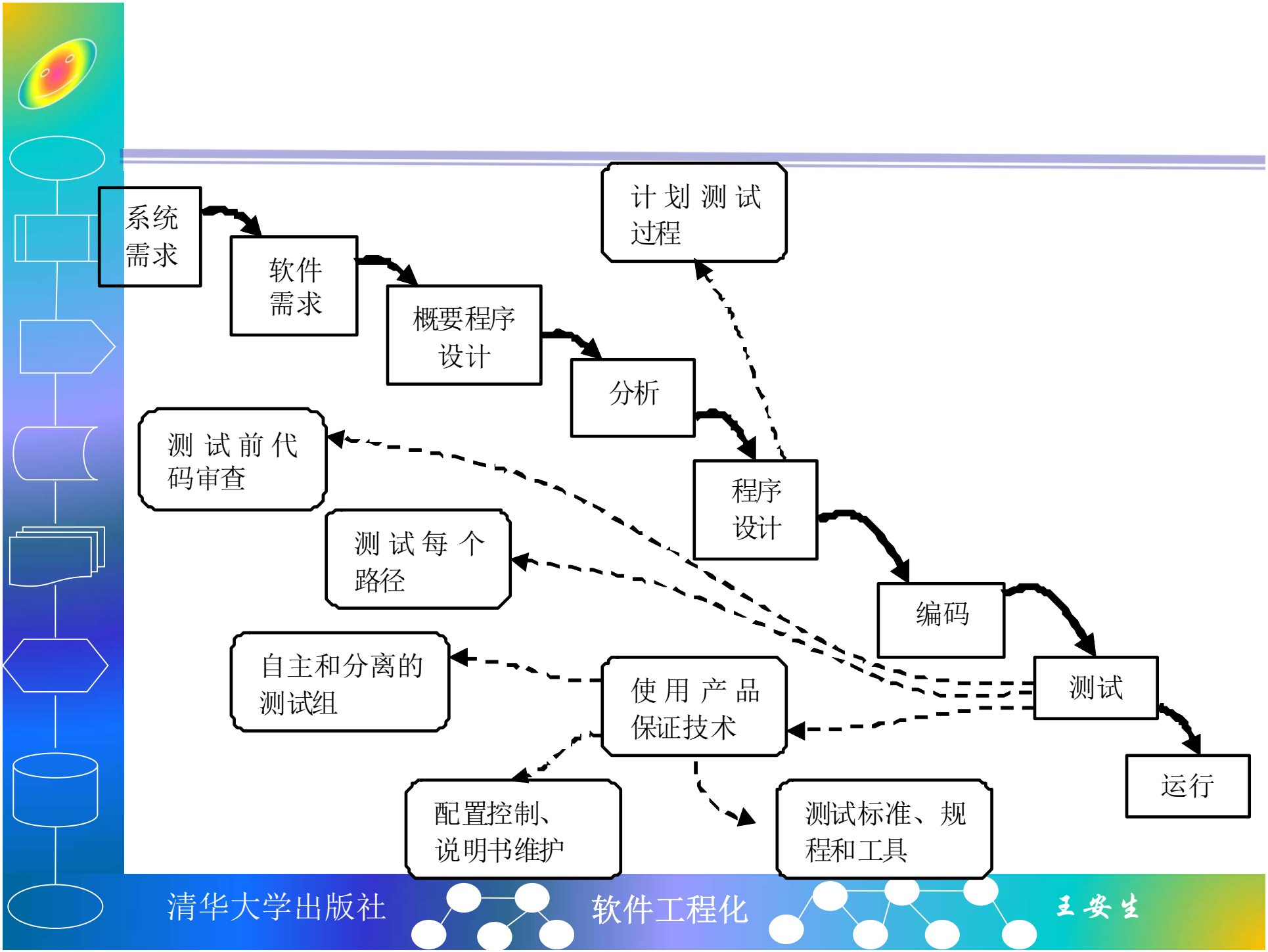
- Royce初步估计：采购5百万美元的硬件装置，需要30页的说明书提供较详细的采购信息。而采购一套5百万美元的软件，大约需要1000页的文档才能理解和控制项目。
  - （1）口头表达是非常含糊的，不足以作为管理和决策的基础。必须用文档建立明确和清晰的完成证据。
  - （2）在软件开发的早期阶段，文档既是规格说明书，也是设计。直到编码阶段，文档、规格说明书和设计都在讲一个事情。如果不编写成书面材料(文档)，就不能让大家完全理解设计，也会造成许多误会，即，缺乏对事情的共同理解(Common Understanding).
  - (3)良好的文档对于开发和测试，乃至后期的运行和重新设计都具有不可估量的价值。文档的价值可以解决经理们和开发人员之间的模糊问题。

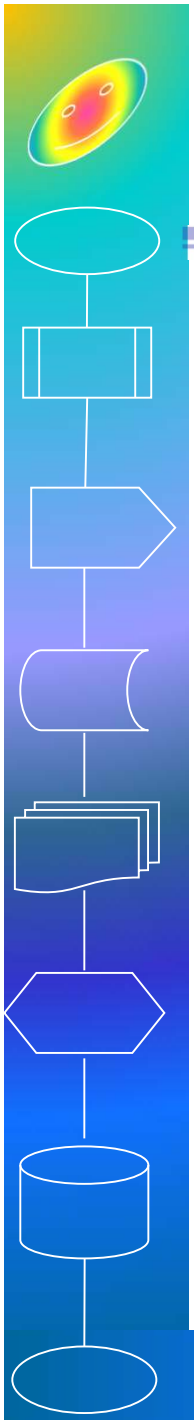
两次迭代

- **第三步：做两次开发，**即，从概要设计、分析、程序设计、编码、测试、到运行，重复两次。第一次关注对最终系统的模拟，后一次形成系统。



- 第四步：计划、控制和监督测试过程。
  - 项目开发过程中人力资源、计算机使用和管理工作大量花费在测试阶段中。
  - 测试发现的问题会导致整个项目的返工。
  - 即使先前的分析、设计、编码等工作都能做好，测试阶段仍有许多对软件进行验证的工作。


系统需求
软件需求
概要程序设计
分析
计划测试过程
程序设计
测试前代码审查
测试每个路径
编码
测试
自主和分离的测试组
使用产品保证技术
运行
配置控制、说明书维护
测试标准、规程和工具

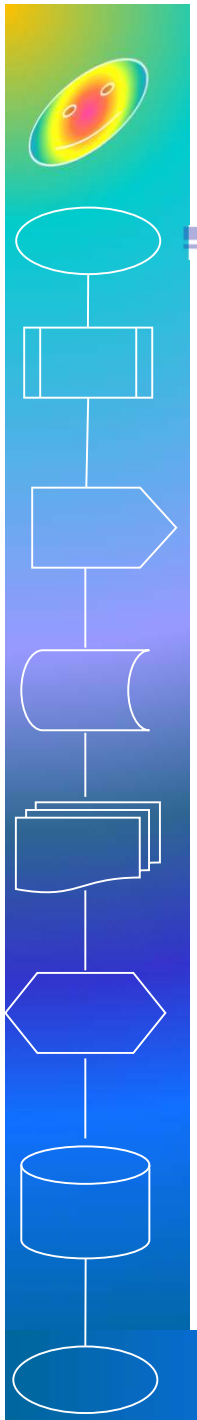


- 第五步：客户参与。
  - 软件开发往往是承包商从客户得到合同后进行的开发。很难用一个软件合同把所有的要求说清楚。
  - 因此，客户参与越早，越容易提交可使用的软件。
  - 客户最起码要参加：
    - “系统需求评审”
    - “概要软件设计评审”
    - 进入编码前的“关键软件设计评审”
    - 以及测试后的“最终软件验收评审”

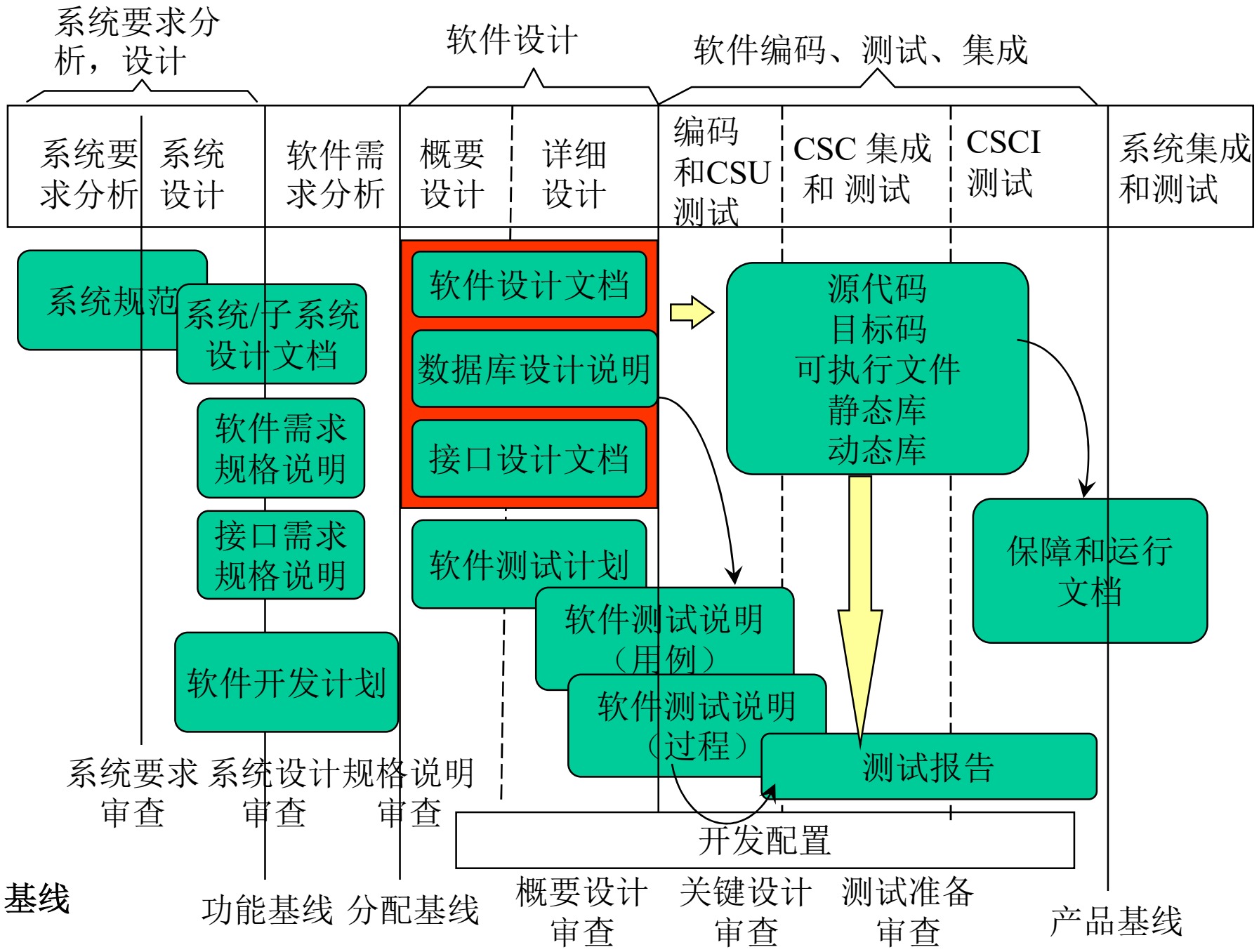
3.3 中间产品驱动的过程

- 3.3.1 中间产品驱动的过程
- 3.3.2 中间产品的意义

3.3 中间产品驱动的过程



- Royce于1970年向人们展示的大规模软件开发流程引起了美国国防部门的重视。
- 作为全球最大的软件用户，美国国防部门认为必须对软件开发商进行有效的管理。
- 特别是把重大软件密集的武器装备系统交付给一个或多个承包商开发时，由于项目的时间周期长，开发人员会发生变动，费用和进度难以控制等原因，美国国防部门必须为项目经理们和承包商提供一个可以执行、可以检查的、审计和评审的、可跟踪的、事后可追溯的项目管理办法。
- Royce提出的大规模软件开发的流程就成为美国国防部门的首选流程。


系统要求分析，设计
软件设计
软件编码、测试、集成
系统要求分析
系统设计
软件需求分析
概要设计
详细设计
编码和CSU测试
CSC 集成和 测试
CSCI测试
系统集成和测试
系统规范
系统/子系统设计文档
软件需求规格说明
接口需求规格说明
软件开发计划
软件设计文档
数据库设计说明
接口设计文档
软件测试计划
软件测试说明（用例）
软件测试说明（过程）
源代码
目标码
可执行文件
静态库
动态库
保障和运行文档
测试报告
开发配置
系统要求审查
系统设计审查
规格说明审查
基线
功能基线
分配基线
概要设计审查
关键设计审查
测试准备审查
产品基线

中间产品的意义

- 中间产品质量是最终产品质量的依据。

- 在软件的开发过程中，如果能够实现“第一次就作对”，即，所有活动输出的中间产品都是正确的。那么，最终产品当然也就没有错误，就是正确或零缺陷的。

- 但是, 对于软件开发来讲，很难做到“第一次就作对”。解决的办法是在每个阶段活动后进行评审，尽可能找出该阶段工作中的错误。“尽可能”就意味着不能够完全消除该阶段的错误。其原因在于，当前阶段所产生错误可能要等到后期的一些阶段才能被发现，而不是当前阶段结束时的评审。中间产品的缺陷越多，最终产品质量也会越差。

中间产品的意义

- 中间产品的另一个作用是增加工作的复用
- 一个软件生产企业内部、多个生产企业之间、客户等的要求、以及系统的运行维护和升级改造，都需要对以前的工作成果进行复用。
- 一个项目的需求、设计、实现、测试文档和方法，以及项目的工作流程可以作为下一个项目参考
- 。阶段性工作产品被相互参考和复用，正像机械和建筑图纸一样，需求文档、软件设计文档和方案可以被复用，代码可以被部分复用，测试用例和测试过程可以被复用。

中间产品的意义

- 中间产品可以极大地降低返工工作量。
- 如果没有中间产品，一旦发现错误，就必须全部返工，因为开发队伍搞不清缺陷是哪个阶段引入的。这样的返工工作量是巨大的。特别是，在人员流动的情况下，更无法追溯缺陷发生的根源。
- 通过对中间产品的缺陷追溯，可以把缺陷的发生把定位到某个阶段或文档。因此，而不必都追溯到原始的用户需求，更不需要追溯到原先承担此工作的已经离开的员工。

瀑布模型过程的特征(1)

- 各个阶段活动：
  - 被追溯，文档是事后追溯的依据
  - 中间产品(文档和代码等)的质量可被检查
- 活动与人无关
  - 工厂的工人一定是可被替换的，
  - 软件开发是生产活动，不能完全依赖“天才”程序员-----软件工厂(日本人率先提出)
- 质量证据
  - 最终产品质量取决于中间的产品质量
  - 发生事故后，可以追溯到人和活动责任

瀑布模型过程的特征(2)

- 过程是可重复的：
  - 客户(如国防部)把项目交给任何厂家，都能按时、按质完成，且经费受控
  - 要求每个开发商都按规定的过程和活动执行
- 明确的里程碑，便于检查和评审：
  - 中间工作和活动的质量
  - 中间成本的控制
  - 工作任务的完成情况
  - 避免风险(捐款外逃、不可能完成等)，管理方可以及时停止项目，寻找新的解决方案

瀑布模型的作用

- 明确了各个阶段
  - 活动、工作和责任
  - 输入(准入条件)
  - 输出(结果)
- 规定各个阶段工作的
  - 质量
  - 时间
  - 人力资源
  - 其他资源(工具、外购软件等)

瀑布模型过程的特征----不足(3)

- 过程是线性的：
  - 虽然每个阶段都要求严格的检查，总是不可避免错误遗漏，直到后期阶段才被发现
  - 起初有些需求，客户和开发者都没想到，或
  - 都没想明白，随着开发的深入，才认识到或逐步清晰
- 因为，工作计划是线性的，安排的很紧凑，你的开发队伍已经没有时间和资源再做返工！导致：
  - 项目彻底失败(遗漏主要功能)
  - 能用，但离客户要求差的太远
  - 能用，性能不稳定，但不敢用，.....
- 改变开发过程.....

3.4 瀑布还是迭代?

- 3.4.1 增量式模型
- 3.4.2 渐进式模型
- 3.4.3 螺旋式模型

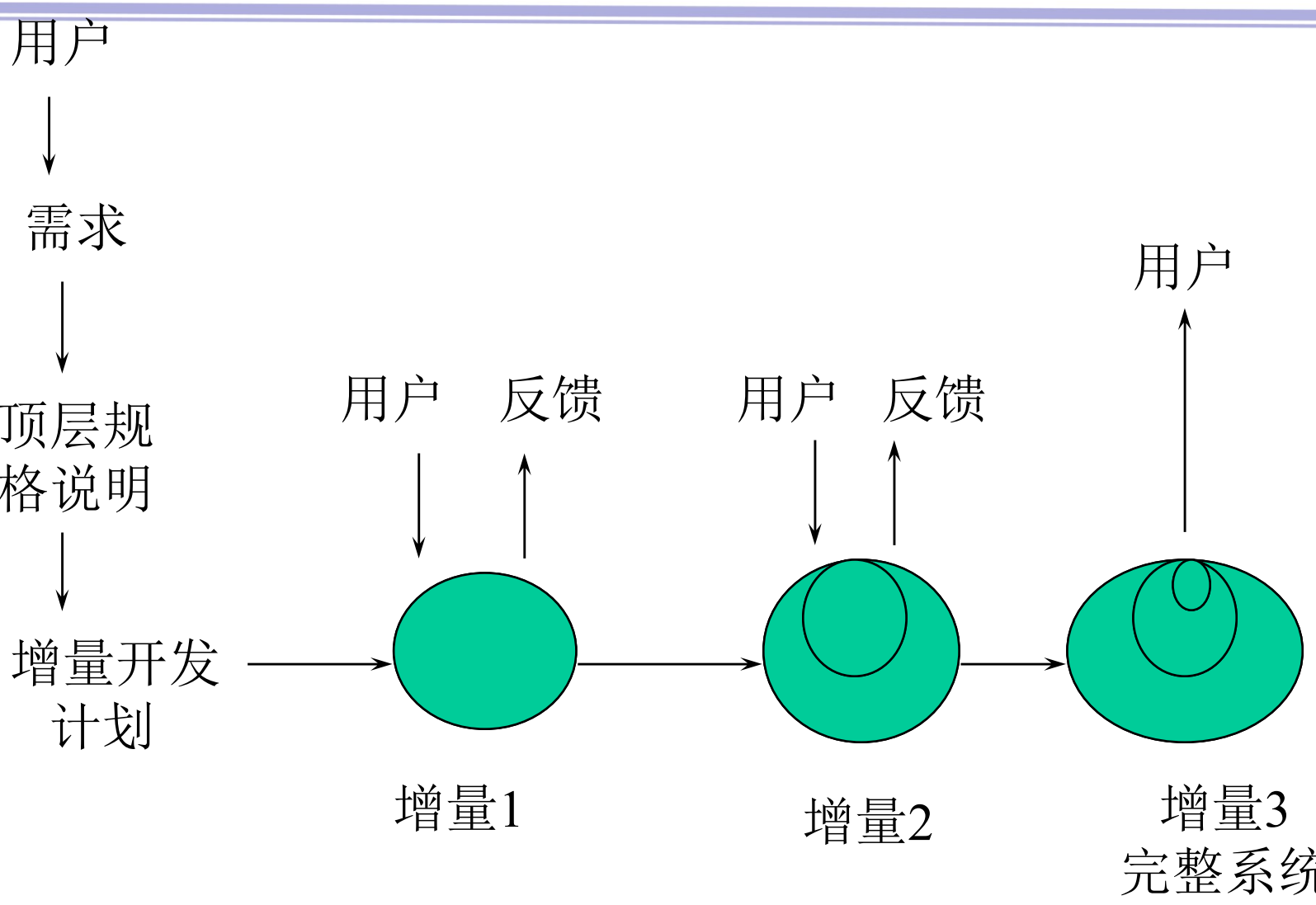
瀑布还是迭代？

- 大多的教科书，包括美国军方、英国国防部门等定义的国防系统软件开发过程要求都是以瀑布式(顺序性)模型为例进行论述。
- 这种论述导致了教育界的一种对“瀑布式模型”的误解：“瀑布模式是标准的开发过程”
- 瀑布式模型强调每个阶段完全正确后才能进入下一个阶段。而在实际中这是很难做到。由于系统或软件的需求总是不能完全被搞清楚，直到系统进入到测试或试用阶段、甚至是实际使用时，用户才发现“所开发出的软件不是我们所需要的”。
- 美国军方强调“开发方可以采用最有益于项目的、经工业界实践证明的开发过程”

3.4.1 增量式模型

- 增量式(incremental)开发是指在开发过程中，先实现那些需求明确的增量。随着系统开发进展，人们会对一些需求不明确的需求逐渐清晰起来，那么在后续的第二次，第三次等迭代中，可以更容易地实现这些需求功能。
- 由于需求的实现是一个个“增量”叠加的，将这种模型称为增量式的开发。

3.4.1 增量式模型



案例1--IBM FSD：

- IBM FSD正式采用IID的项目始于1972年。所承担的项目是一个生命有关的系统，多达1百万行代码的美军Trident潜艇的命令与控制系统。O’Neill任项目经理，他和同事们计划词用IID开发模式，IBM的管理层具有远见，批准了该项目采用IID。最终获得了成功，得到了IBM杰出贡献奖(Outstanding Contribution Award)。因为该系统要求在确定的工期内交付，否则，IBM FSD将面临每天10万美元的罚款。
- 该项目组将项目分为4个时间箱(timebox)，每次迭代6个月。重要工作仍然是前期的需求说明工作，迭代不同于当今敏捷方法所建议(几周的)时间范围(见第18章敏捷方法)。虽然，大部分“反馈意见驱动”的进化主要发生在需求阶段，O’Neill特别指出：IID方法也是一个适合于的复杂和高风险项目的大规模开发。

迭代式的案例

- **案例2--也是在1972年：**
  - IBM FSD竞争对手TRW也将IID方法用于到主要的项目----100万美元TRW/军用国防软件项目。该项目始于1972年1月，TRW开发队伍采用5次迭代。
  - 第一次迭代集中一个目标，5次迭代。迭代没有规定严格的时间箱。开发队伍仍然把前期的规格说明作为重点工作，但在每次的迭代中重点响应上次迭代中出现的问题。

- **案例3--1970年代的中期：**
  - IBM FSD开发轻型飞机多用途系统(LAMPS-- Light Airborne Multipurpose System)，是美国海军舰载直升机武器系统的一部分。需要4年工期，200个人年工作量，开发3百万行代码，集成了分布在直升机和军舰的8个不同的上7百万行程序和数据。LAPMS分为45个时间箱的迭代(每次迭代一个月)交付。Mills对此项目的总结是：“每次都能按财政预算及时交付。

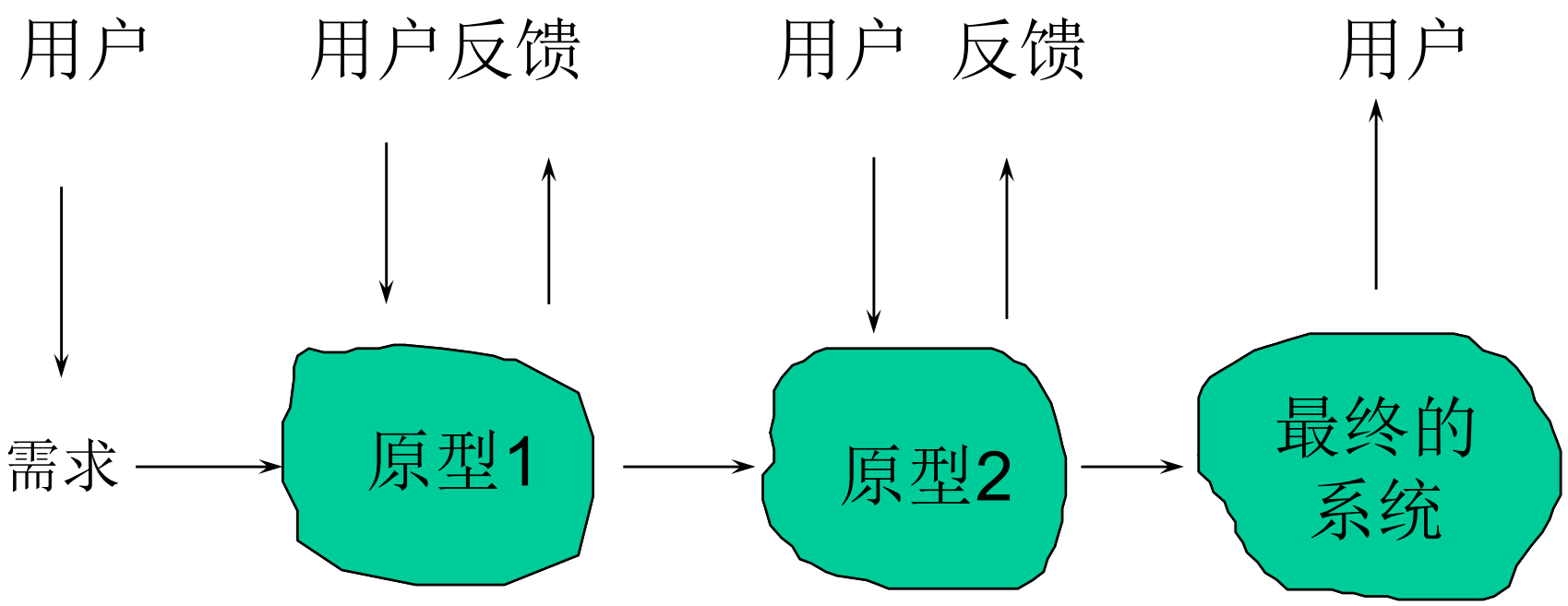
经典迭代开发的优势

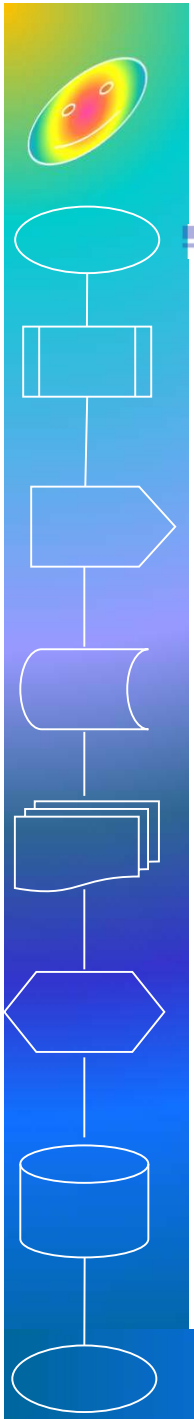
- “迭代式增强的基本思想是增量式地开发软件，让开发者能够从早期的开发中、系统的增量、交付的版本中学到经验。从系统的开发和使用中学习一切可能学到的东西。过程中的关键是从系统需求的简单子集实现开始，通过迭代增强和进化后续的版本，直到系统被实现。每次迭代中，对设计进行修改，并增加新的功能要求。”
- ------Vic Basili 和Joe Turner，1975年

3.4.2 渐进式模型

- 渐进式(evolutional)开发，或进化式开发，与增量式开发一样也是需要多次迭代的过程。所不同的是，许多软件的需求不能完全区分出所谓的“需求明确部分”和“需求不明确部分”。
- 这种情况下，很难采用增量式开发，因为不能断定哪些增量是清楚的。

3.4.2 渐进式模型





- 渐进(Evolution)是一个保持稳定性的技能。如果能把复杂系统分成许多小步骤实现，且每步都可以清晰地测量出成果，以及可以重新处理先前“失误”之处的话，肯定能够更好地实现该系统。开发队伍就有机会从前面的失败中获得真实世界的经验，校正发现的错误。早期的项目推动了渐进式开发的发展。

----1976年，Tom Gilb

- 案例4：
- 1977年，IBM FSD引入三叉式IID方法，包括：在每次迭代的后期集成所有的软件部件---McHenry把它称为“集成工程”。一些集成队伍成员和Mills(IBM员工)在集成活动中担当顾问的角色。集成工作涉及到了2500个FSD的软件工程师[7]。

- 案例5：
  - NASA的载人航天是早期渐进式开发的有一个重要案例。从1977年到1980年，IBMFSD开发队伍在31个月中采用了17次迭代，平均每次迭代间隔是8周。主要的原因是避免瀑布式模式不能适应开发过程中需求不断变化的要求。
- 由于规模、复杂性、渐进性(改变需求)的自然规律，就必须承认理想的软件生命周期(瀑布模型)不能被严格的使用…，需要将基于小增量发布的实现方法与理想的生命周期结合，实现总的软件包。

- 问题
  - 缺乏过程的可见性
  - 系统通常不能够很好的结构化
  - 可能需要特殊技巧 (例如，快速原型语言)
- 应用
  - 中小规模的交互式系统
  - 大系统的一部分（例如，用户接口）
  - 生命周期较短的系统

自然过程和社会过程

- 自然过程
  - 自然科学家研究自然世界的运动过程，并对其建立模型。然后利用这种模型构造出符合自然规律的系统，例如，化学反应过程就是典型自然过程的例子，牛顿三大定律是反应物体运动的自然规律，Maxwell方程是电磁场现象的描述，爱因斯坦的质能互相转换也是自然过程规律。
- 自然过程是人类必须遵循的自然规律。违背自然规律，会给环境和人类带来灾难！

自然过程和社会过程

社会过程----人为定义和组织的过程

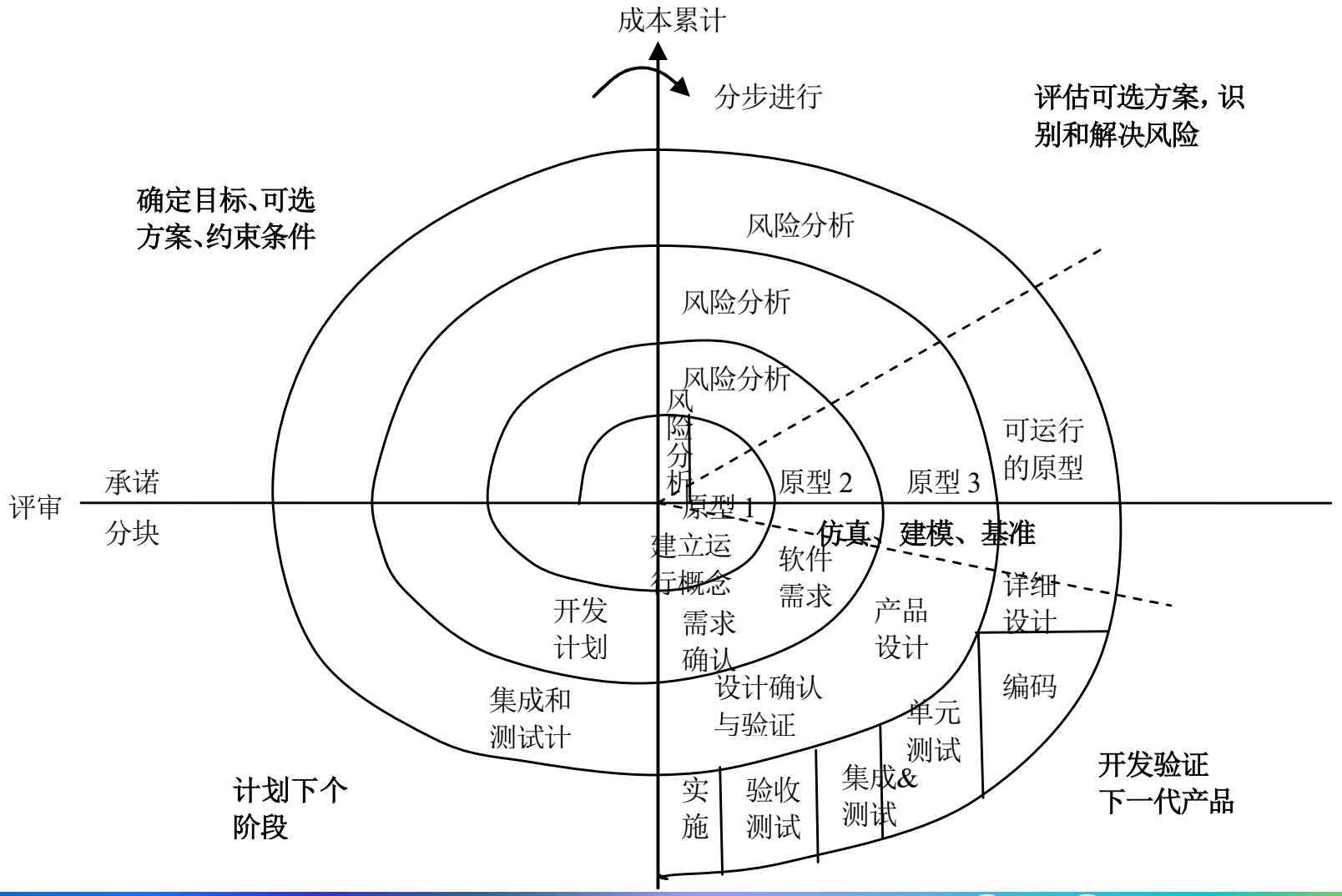
- (社会)过程是人为定义的，“用于产生某结果的一整套操作、一系列的活动、变化以及作为最终结果的功能(韦氏大词典)”。
  - 例如，“摩尔定律”是一个社会过程规律，世界上只有为数不多的半导体生产厂商能够满足摩尔定律，虽然摩尔定律是每个半导体厂商的追求的目标。半导体厂商如果不提高生产率，就不会满足摩尔定律。
  - 又例如，普遍规律是“奴隶社会->封建社会->资本主义->社会主义->共产主义社会”。
  - 但是，中国长达2000年的“封建社会”。乾隆年间全球GDP的25%~30%是中国的。鸦片战争和甲午战争直接导致半封建半殖民地社会，（成功的革命）直接进入社会主义社会，目前是全球第二大经济体。

改变软件开发这个社会过程

- 社会过程是可改变
  - 解放思想是改变社会过程的基础。
    - “以为上了书的就是对的，文化落后的中国农民至今还存着这种心理。”—毛泽东《反对本本主义》
    - 如果你只遵循条文，就教条主义
- 工程创新包括
  - 技术和工艺，例如，
    - 编译效率提高，算法更快等等
  - 生产过程，
    - 这是过程观点关注的问题

3.4.3 螺旋式模型

- **螺旋式(Spiral)**开发与渐进式开发一样。不同点是要强调商业上的风险。特别是那些时间周期长(多达数年)、人力资源耗费多的、面向市场的软件项目，期望每次迭代都要对项目的商业风险进行分析，以避免系统最终实现后，已经没有市场需求。
- 1985年，TRW公司的Barry Boehm发表“A Spiral Model of Software Development and Enhancement”。其文中的几个重要口号是：“生命周期的概念是有害的”，“瀑布模型已经死亡”等。
- Boehm给出了螺旋式模型，图中分为四个象限，螺旋线围绕着四个象限不断迭代和扩展。


成本累计
分步进行
评估可选方案，识别和解决风险
确定目标、可选方案、约束条件
风险分析
风险分析
风险分析
风险分析
可运行的原型
承诺
评审
分块
原型 2
原型 3
原型 1
仿真、建模、基准
建立运行概念
软件需求
详细设计
开发计划
需求确认
产品设计
编码
集成和测试计
设计确认与验证
单元测试
集成&测试
实施
验收测试
计划下个阶段
开发验证下一代产品

3.5 软件产品的开发过程

- 3.5.1 策略和原则的建立
- 3.5.2 定义产品和开发过程
- 3.5.3 开发产品与装箱
- 3.5.4 与黑客方法的差别
- 3.5.5 方法的优点

微软---业界的榜样

- 微软希望的是高度灵活、企业化创新公司的产品开发方法，而不是普通意义上标准化的开发方法。
- 从1980年代后期，微软采用“同步-稳定”的方法建造了当今几乎垄断市场的Excel、Office、Publisher、 Windows 95、Windows NT、Words、等产品。

3.5.1 策略和原则的建立

- 从管理角度看，微软在1996财政年就有20550个员工，250种产品，年收入是87亿美元。其软件产品的复杂程度也是可想而知的。例如Windows 95有11百万行代码，200多个程序员和测试人员参加开发。
- 在1990年代中期，微软的研究人员花了两年半的时间，对38个关键人物(包括Bill Gates)进行会谈，评审了数千页的保密项目文档和项目结束后的报告。通过这些研究，提出了：
  - 两个定义产品和开发过程策略，以及一组原则
  - 微软的“同步-稳定”开发方法。

微软的“同步与稳定”模型

里程碑 1(头 1/3 特征)
开发(设计、编码、原型)
可用性实验
私有发布测试
每日建造
特征调试
特征集成
代码稳定(No Server bugs)
Buffer time (20%-50%)

里程碑 2(2/3 特征)
开发(设计、编码、原型)
可用性实验
私有发布测试
每日建造
特征调试
特征集成
代码稳定(No Server bugs)
Buffer time

里程碑 3(最后的全集)
开发
可用性实验
私有发布测试
每日建造
特征调试
特征集成
特征完成
代码完成
代码稳定
Zero bug 发布
上市

How Micosoft Builds Software

Michael A. Cusumano, Richard W. Selby
Communications of the ACM archive
Volume 40 , Issue 6 (June 1997)

Pages: 53 - 61

3.5.2 定义产品和开发过程

- 策略1：“通过进化特征和固定资源，集中创新性”
- 并通过下面5个原则实现此策略。
  - 1）将大项目分解为多个里程碑周期（总项目工期的20%-50%作为缓冲时间），而不分割产品维护组。
  - 2）使用“远景陈述”和特征说明大纲指导项目。
  - 3）依据用户活力和数据挑选出特征，并排序。
  - 4）演化出一个模块化和水平化的设计体系结构，并将产品结构映射为项目结构。
  - 5）依据个人承诺，分成多个小任务，并固定项目资源。

3.5.3 开发产品与装箱

- 策略2：“并行地做每个事情，并经常保持同步”。
- 并通过下面的5个原则实现此策略:
  - 1）并行队伍工作，但保持同步，每天排错；
  - 2）每个主要平台和产品有多个版本，总有产品可以装箱；
  - 3）在一个开发场地，讲“共同语言”；
  - 4）随着建造过程不断测试产品；
  - 5）使用度量数据，判断里程碑的完成和产品发布情况。

开发人员每天的工作

- 微软把每天的建造过程分为几个步骤。
  - 首先，开发人员要从集中的源代码版本中检出自己的私有源代码备份，改变这些私有代码实现其所需功能。
  - 其次，建立产品的一个含有新特征的私有建造(build)，并对其进行测试。
  - 第三，把私有代码拷贝与源代码的主拷贝进行对比检查。
  - 对比检查过程包括自动回归测试，以便测试出新改变的特征对产品其它地方的影响或错误。开发人员通常每周两次“检入”自己的代码放到主代码库中，而不是每天都做。
- 需要良好的版本控制机制支持！！！

3.5.4 与黑客方法的差别

- 梆、梆、梆、梆、.....
  - 不断地敲击键盘----黑客方法
  - 黑客文化对于PC和互联网上的软件开发方法并不是一件坏事。
  - 至今，仍然有大量的“黑客”，夜以继日地在“梆梆”地敲键盘，并没有任何的预先计划。
- 黑客方法造就了像 Lotus 1-2-3、WordPerfect、Word和Excel等早期的版本，并被人们仅仅乐道。

黑客文化与软件工厂

- 但是，随着代码行的不断增加到数十万行、上百万行的代码后，黑客方法就行不通了。远远超出个人的能力。

- 依靠个人的才能，难以战胜有组织的团队，无论你多聪明！

黑客文化与软件工厂

- 微软每天对产品建造的同步和稳定、以及周期性的里程碑处的稳定和测试，给人一种“黑客开发方法”的感觉。
- 根本的差别是微软的方法更强调是产品开发中灵活的组织结构。

将黑客文化与有序工业生产结合

- 微软是将“黑客文化”引领到有序开发的成功企业，主要增加足够的结构，以便于开发PC机的软件产品。
- “同步-稳定方法”成就了微软。微软所实施的非常像许多企业那样的增量或迭代、以及并行工程。
- 采纳了其它公司所引入的软件工程实践（如测试技术），并发明了许多技术方法（包括利用积累的历史度量数据分析缺陷的趋势，并安排项目进度，后期讨论。

将黑客文化与有序工业生产结合

- 微软引入的是结构化的“类黑客”的方法进行软件产品开发，让其能够适应小规模和大规模的产品开发。

- 很好地把(黑客)创新与大规模组织所要求的有序开发方式进行了有机结合，产生出具有市场竞争的优势：其产品“足够好”，而不是等到产品“完美”才发布；通过增量式的对特征的演化来改进产品；然后将多种产品版本销售给客户并不断的升级。
  - **--------It is Microsoft’s culture!**

3.5.5 方法的优点

同步-稳定	瀑布式的顺序开发
产品开发和测试并行进行	按顺序进行
前景陈述	建造产品前，完成“冻结的”规格说明和详细设计
区分特征的优先级，按3或4个里程碑实施子项目建造	企图同时建造产品的每个部分
频繁同步(每日建造)，并立即同步(里程碑)	在项目的后期进行一次大的集成和系统测试
“固定”发布和装箱日期，多个发布周期	在每个项目周期内，做到特征和产品“完美”
在开发过程中，让客户不断反馈	反馈意见作为下一个项目的输入
进行产品和过程设计，让大团队像小团队一样工作	个体组成的大团队分散在各个功能部门工作。

“足够好”还是“高可靠”？

- 微软的这些优势主要体现在面向大众化的、具有广大市场的软件产品的建造上。所体现的是“足够好”，而不是“完美” “安全关键”和“任务关键”等特定项目性要求。
- 当你更看重“完美” “安全关键”和“任务关键”等时，是否仍采用这种方法呢？
  - NO！
- 民航、航天、高铁、银行/金融等软件的开发是另一个故事！

3.6计算机辅助与模型驱动的软件工程

- 3.6.1 计算机辅助软件工程与工具
- 3.6.2 模型驱动的软件工程

3.6.1 计算机辅助软件工程与工具

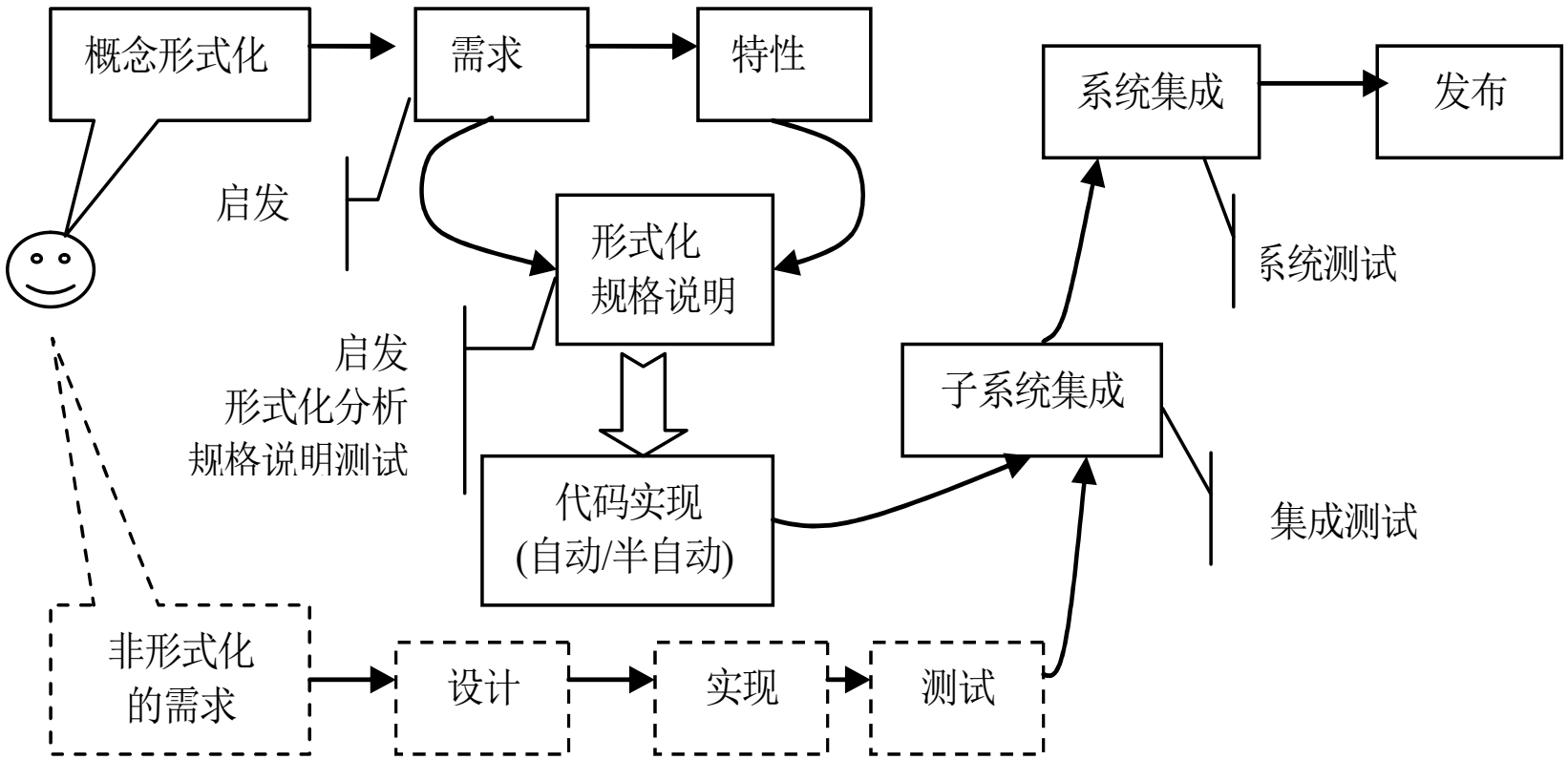
- 基于CASE进行软件开发的活动包括：
  - 用图形化的建模技术描述系统的需求规格说明或软件设计；
  - 使用数据字典，理解设计中的各个实体之间的信息交换和连接关系；
  - 依据模型自动或半自动生成所需的代码；
  - 使用建造工具，编译、链接出可运行二进制代码；
  - 调试和测试可运行的软件。

软件工具划分为：

- 1）计划类工具，例如进度安排的工具、成本和工作量估算工具等；
- 2）编辑工具，例如文本编辑器、框图编辑器、字处理工具等；
- 3）变更管理工具，如对需求跟踪工具、控制修改的系统等；
- 4）原型工具，如用户界面描述和生成工具、界面动画等
- 5）方法知识工具，如设计编辑器、数据字典、代码生成器等；
- 6）语言处理工具，如编译器、解释器等；
- 7）代码分析工具，如源代码的交叉引用的产生器、代码静态分析器和动态执行器；
- 8）测试工具，如测试数据生成器、文件比较器；
- 9）调试工具，如交互调试系统(Debugging Systems)
- 10）文档编制工具，如常见的字处理工具、图形、图像编辑器等
- 11）逆向工程工具，如交叉引用系统、程序结构梳理与结构化重建系统等。

3.6.2 模型驱动的软件工程

- MDE方法的创立和支持者们认为，软件开发的首要认为是建立模型，而不是急于写出代码。
- 建立的MDE主要有两个方面的工作：
  - （1）建立领域特定的建模语言(DSML—Domain-Specific Modeling Language)。用DSML描述软件的结构、行为和需求，例如，软件无线电、电子航空计算、在线金融服务、仓储管理、以及中间件等。
  - （2）开发转换引擎和生成器。用它们分析模型的特殊性，然后集成各类构件，例如已有的源代码、仿真部件，XML描述等，最后生成整个软件的代码[14]。


概念形式化
需求
特性
系统集成
发布
启发
形式化
规格说明
系统测试
启发
形式化分析
规格说明测试
子系统集成
代码实现
(自动/半自动)
集成测试
非形式化
的需求
设计
实现
测试

3.7 总结

- No Silver Bullet！”。
- 不存在一个适用于所有软件项目或软件企业的“万能”的软件开发过程和方法，完全解决软件工程面临的危机。
- 软件开发过程模型都来源与对开发过程的总结和提炼。
- 面向项目的开发与面向产品的开发过程具有很大的差别。
- 采用计算机辅助软件工程环境可以提高软件生产率。面向不同领域和行业要求，需要不同的需求描述方法，由此需要面向行业和领域的软件工程

Homework

- 针对下面各系统，提出合理的软件开发过程模型，并给出理由：
  - 1）汽车刹车的控制系统；
  - 2）学院的一个学生管理系统；
  - 3）一个航班订票系统；
  - 4）一个程序设计课的项目，要求在两个小时内完成；
  - 5）开发一个类似于Excel的制表软件，并打算到市场上销售。