软件演化与配置管理

1 软件演化

2 软件维护

3 软件配置管理(SCM)

4 持续集成

软件演化与配置管理

1 软件演化

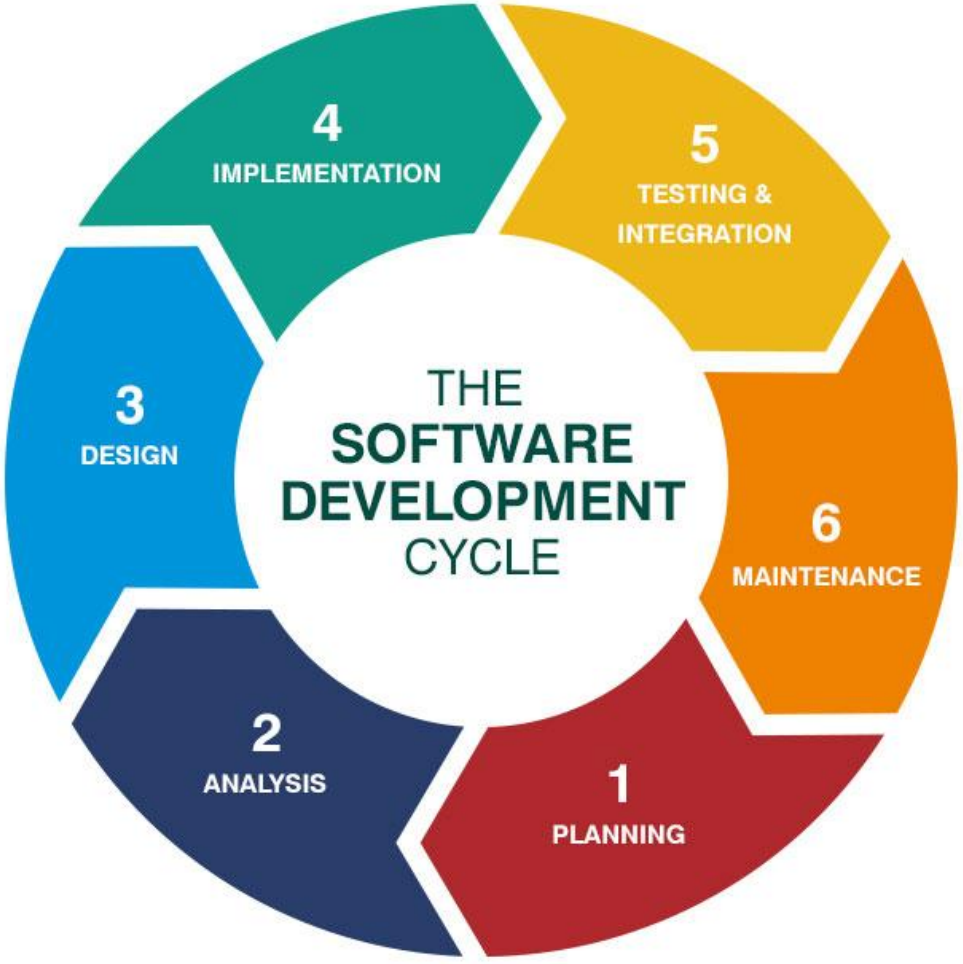
2 软件维护

3 软件配置管理(SCM)

4 持续集成

Lifecycle of a software

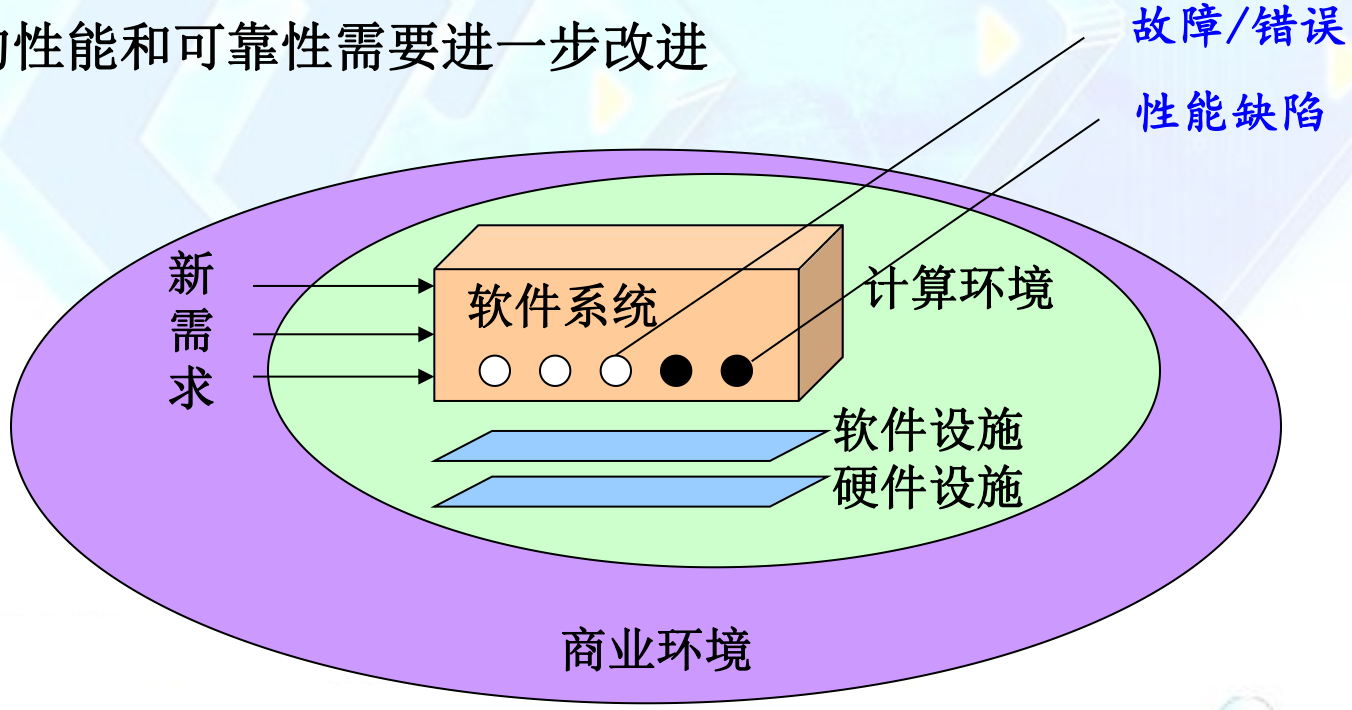
- **Software Development Life Cycle (SDLC): From 0 to 1**



软件演化与配置管理

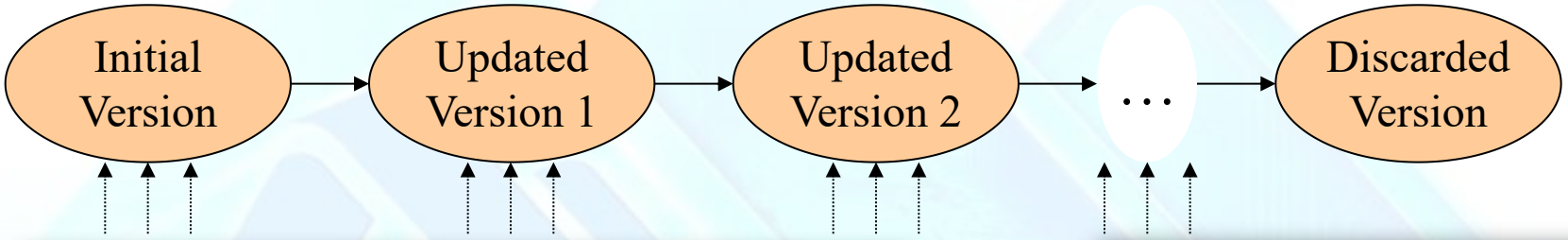
软件演化

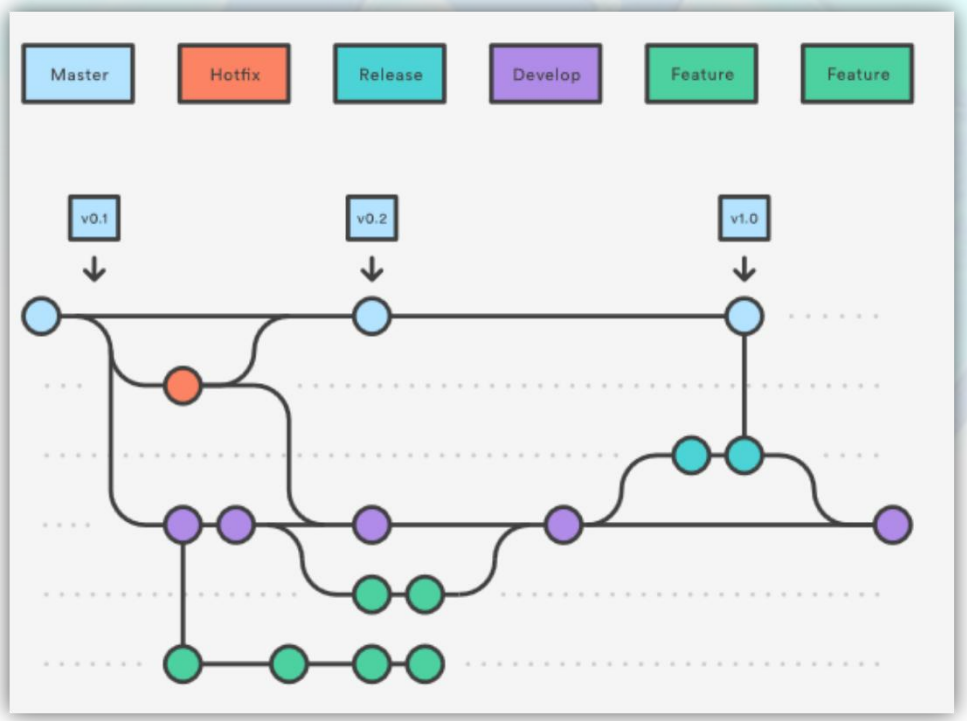
- 软件在使用过程中，新的需求不断出现
- 商业环境在不断地变化
- 软件中的缺陷需要进行修复
- 计算机硬件和软件环境的升级需要更新现有的系统
- 软件的性能和可靠性需要进一步改进

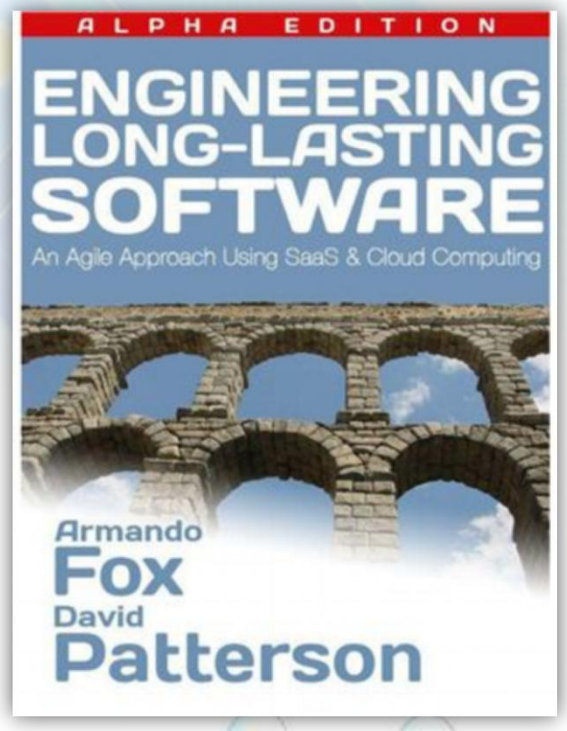


Lifecycle of a software

- **Multiple versions in the life of a software: From 1 to *n***

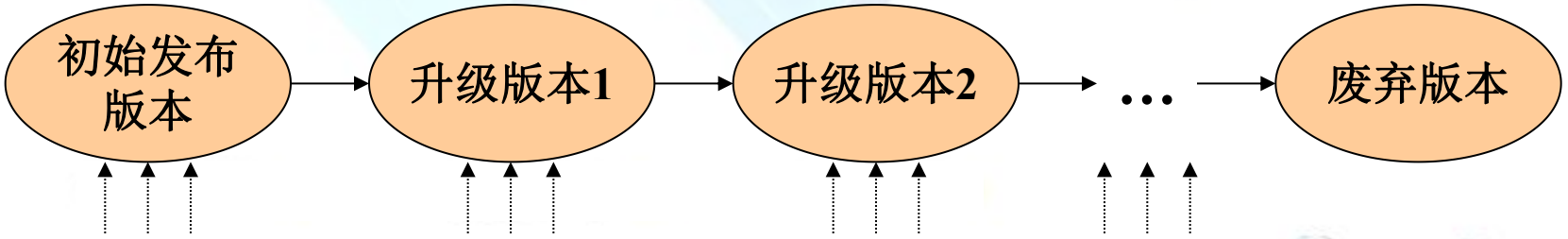






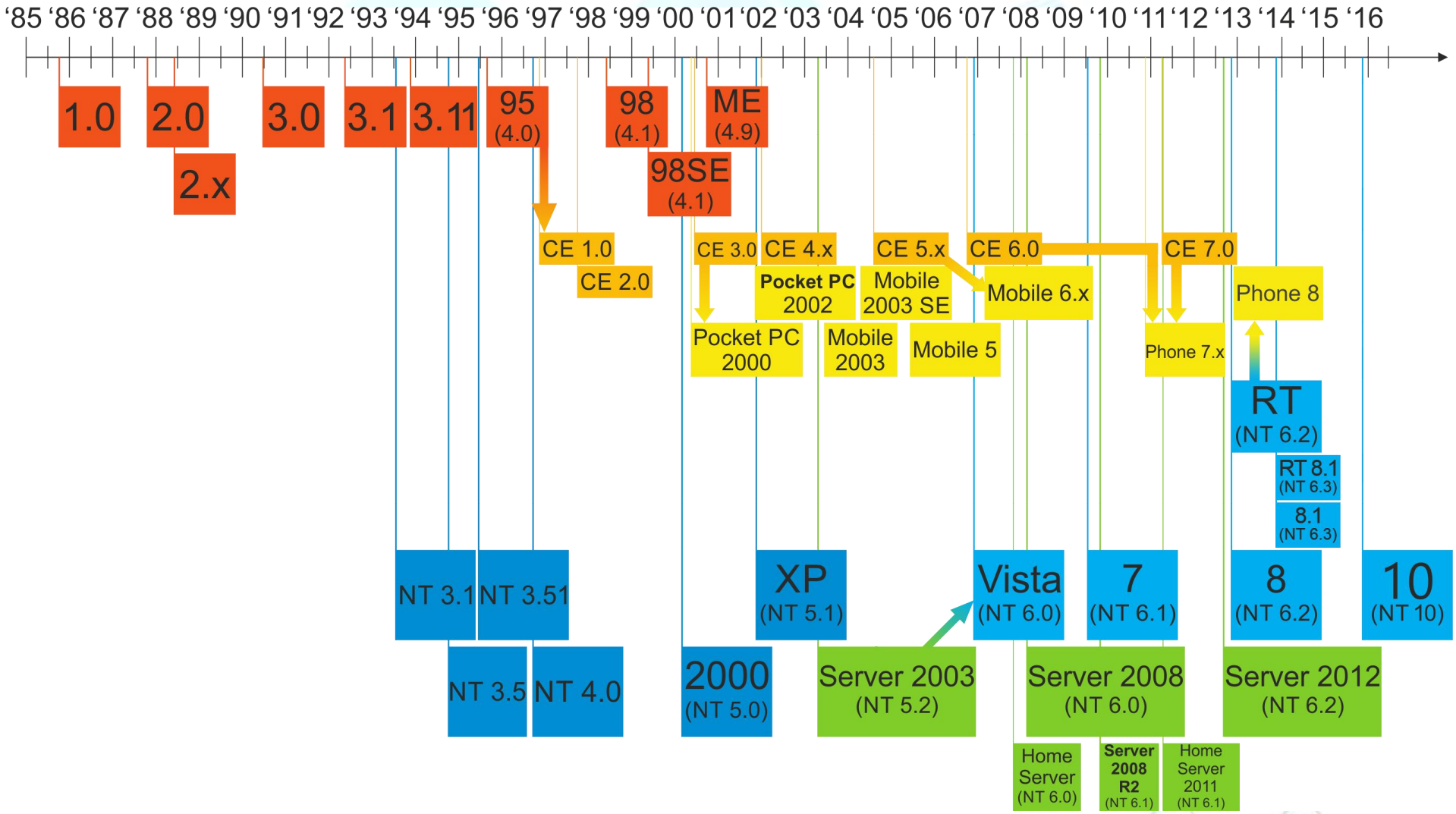
软件演化的Lehman定律

- 持续变化(continuing change)
  - 现实世界的系统要么变得越来越没有价值，要么进行持续不断的变化以适应环境的变化
  - 环境变化产生软件修改，软件修改又继续促进环境变化
- 复杂度逐渐增大(increasing complexity)
  - 当系统逐渐发生变化时，其结构和功能将变得越来越复杂，并逐渐难以维护并失去控制，直至无法继续演化，从而需要大量额外的资源和维护工作来保持系统的正常运行
  - 软件修改会引入新的错误，造成故障率的升高



软件演化与配置管理

Example 1: Microsoft Windows (1985-2016)



Example 2: WeChat (2011-2016)

微信 6.5.2 for iOS 全新发布 2016-12-20
微信 6.5.1 for iOS 全新发布 2016-12-12
微信 6.3.30 for iOS 全新发布 2016-11-03
微信 6.3.28 for iOS 全新发布 2016-10-25
微信 6.3.27 for iOS 全新发布 2016-09-22
微信 6.3.25 for iOS 全新发布 2016-09-05
微信 6.3.24 for iOS 全新发布 2016-08-29
微信 6.3.23 for iOS 全新发布 2016-08-01
微信 6.3.22 for iOS 全新发布 2016-06-28
微信 6.3.19 for iOS 全新发布 2016-06-06
微信 6.3.15 for iOS 全新发布 2016-03-21
微信 6.3.13 for iOS 全新发布 2016-02-01
微信 6.3.9 for iOS 全新发布 2015-12-29
微信 6.3.8 for iOS 全新发布 2015-12-11
微信 6.3.7 for iOS 全新发布 2015-11-24
微信 6.3.5 for iOS 全新发布 2015-10-19
微信 6.2.5 for iOS 全新发布 2015-09-10
微信 6.2.4 for iOS 全新发布 2015-08-04
微信 6.2.2 for iOS 全新发布 2015-06-17
微信 6.2 for iOS 全新发布 2015-05-25
微信 6.1.5 for iOS 全新发布 2015-04-27
微信 6.1.4 for iOS 全新发布 2015-03-30
微信 6.1.1 for iOS 全新发布 2015-02-09
微信 6.1 for iOS 全新发布 2015-01-19
微信 6.0.2 for iOS 全新发布 2014-12-20
微信 6.0.1 for iOS 全新发布 2014-11-06
微信 6.0 for iOS 全新发布 2014-09-30

微信 5.4 for iOS 全新发布 2014-08-14
微信 5.3.1 for iPhone 全新发布 2014-06-23
微信 5.3 for iPhone 全新发布 2014-05-08
微信 5.2.1 for iPhone 全新发布 2014-03-21
微信 5.2 for iPhone 全新发布 2014-01-26
微信 5.1 for iPhone 全新发布 2013-12-20
微信 5.0.3 for iPhone 全新发布 2013-10-24
微信 5.0 for iPhone 全新发布 2013-08-05
微信 4.5 for iPhone 全新发布 2013-02-05
微信 4.3.3 for iPhone 全新发布 2013-01-04
微信 4.3.2 for iPhone 全新发布 2012-10-30
微信 4.3.1 for iPhone 全新发布 2012-9-25
微信 4.3 for iPhone 全新发布 2012-9-5
微信 4.2 for iPhone 全新发布 2012-7-19
微信 4.0 for iPhone 全新发布 2012-4-19
微信 3.5 for iPhone 全新发布 2011-12-20
微信 3.1 for iPhone 全新发布 2011-10-27
微信 3.0.1 for iPhone 全新发布 2011-10-17
微信 3.0 for iPhone 全新发布 2011-10-01
微信 2.5.3 for iPhone 全新发布 2011-09-13
微信 2.5.2 for iPhone 全新发布 2011-08-30
微信 2.5.1 for iPhone 全新发布 2011-08-17
微信 2.5 for iPhone 全新发布 2011-08-03
微信 2.2 for iPhone 全新发布 2011-06-30
微信 2.1 for iPhone 全新发布 2011-06-08
微信 2.0 for iPhone(语音版) 全新发布 2011-05-10
微信 1.3 for iPhone(测试版) 全新发布 2011-04-06
微信 1.2 for iPhone(测试版) 全新发布 2011-03-21
微信 1.1 for iPhone(测试版) 全新发布 2011-03-10
微信 1.0 for iPhone(测试版) 全新发布 2011-01-21

Example 3: 亿发智能进销存软件 (2014-2019)

- 2018年12月06日_3010413
- 2018年12月12日_3010414
- 2019年01月02日_3010415
- 2019年01月06日_3010416
- 2019年01月08日_3010417
- 2019年01月14日_3010418
- 2019年01月22日_3010419
- 2019年01月25日_3010420
- 2019年02月15日_3010421
- 2019年02月22日_3010422
- 2019年02月25日_3010423
- 2019年02月26日_3010424
- 2019年03月02日_3010425
- 2019年03月05日_3010426
- 2019年03月14日_3010427
- 2019年03月20日_3010428
- 2019年03月30日_3010430
- 2019年04月08日_3010431
- 2019年04月15日_3010432
- 2019年04月17日_3010433
- 2019年04月18日_3010434
- 2019年04月28日_3010435
- 2019年05月11日_3010436
- 2019年05月13日_3010437
- 2019年05月13日_3010439
- 2019年05月29日_3010440
- 2019年06月19日_3010441
- 2019年06月20日_3010442
- 2019年07月10日_3010443
- 2019年07月11日_3010445
- 2019年07月18日_3010446
- 2019年08月01日_3010447
- 2019年08月21日_3010448
- 2019年09月11日_3010449.2
- 2019年10月12日_3010450
- IIS设置说明.txt
- 历史版本列表

- 2016年09月21日_3010255
- 2016年09月21日_3010305
- 2016年09月26日_3010305_2
- 2016年09月27日_3010306
- 2016年09月27日_3010306_2
- 2016年09月28日_3010307
- 2016年09月28日_3010308
- 2016年09月28日_3010308_2
- 2016年10月08日_3010309
- 2016年10月09日_3010310
- 2016年10月10日_3010311
- 2016年10月12日_3010312
- 2016年10月14日_3010313
- 2016年10月15日_3010314
- 2016年10月24日_3010315
- 2016年11月01日_3010316
- 2016年11月04日_3010317
- 2016年11月08日_3010318
- 2016年11月09日_3010319
- 2016年11月10日_3010320
- 2016年11月17日_3010321
- 2016年11月21日_3010322
- 2016年11月25日_3010323
- 2016年12月01日_3010324
- 2016年12月18日_3010325
- 2016年12月20日_3010326
- 2016年12月22日_3010327
- 2017年01月07日_3010328
- 2017年01月11日_3010329
- 2017年01月23日_3010330

例子：银行业的困境

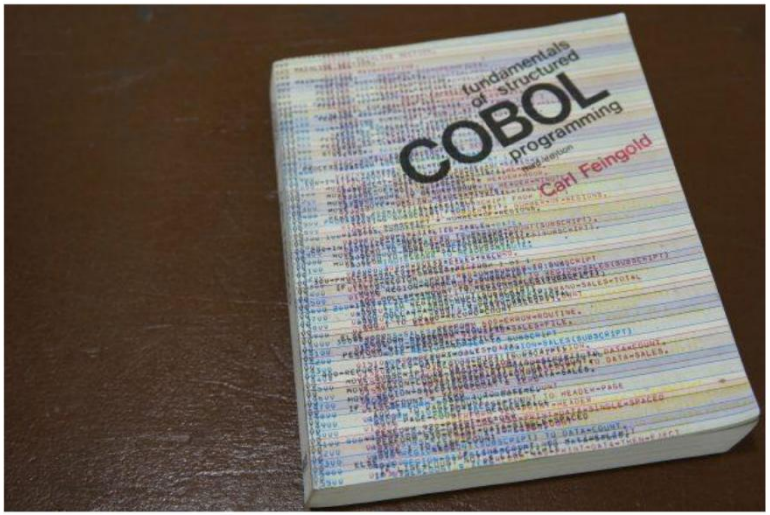
- 目前不少银行/保险公司所使用的核心业务处理系统甚至仍在使用1980年代所开发的COBOL语言书写的程序，用的还是VSAM文件系统
- 随着银行/保险行业标准和会计准则的更新和新产品的推出，这些原有的系统已经无法支持这些新变化
- 但由于现在已经无法找到能够完全理解这些核心系统的程序人员(例如COBOL早已很少使用)，所以不少银行/保险公司往往受困于此却无计可施

软件演化与配置管理

美国75岁COBOL工程师助企业维护老旧系统

为古老程序语言打拼，美国 75 岁 COBOL 工程师助企业维护老旧系统

TechNews 2017-04-13 10:12



不少人都想开发手机程序，学习 Java 或者 Swift 已成为潮流。不过在美国，拥有近 60 年历史的 COBOL 仍有一定地位，可惜精通 COBOL 的人并不多，一些人也步入老年。

75 岁 COBOL 工程师组公司，兼职咨询者多为退休人士

COBOL 全名 Common Business Oriented Language，于 1960 年正式发布，主要用于金融业的电脑系统。虽然推出至今已近 60 年，重要程度大不如前，但碍于种种原因，不少系统仍然是 COBOL。加上网络银行的普及，以现代程序语言编写的手机程序和服务需要与旧式的 COBOL 系统互相配合，间接产生 COBOL 的人才需求。

但众所周知，COBOL 是一种古老的程序语言，现在已很少人学。在美国 COBOL 人才已经不多，余下的已步入老年，所以当系统出现问题时，只有少数人懂得修理。

Bill Hinshaw 现年 75 岁，有 32 个孙子和曾孙，于 1960 年代开始写程序，精通 COBOL。他接受《路透社》访问时表示，数年前本来计划退休，但前客人不断致电他求救，于是他在 2013 年成立 COBOL Cowboys，助企业联系 COBOL 工程师。不过 Hinshaw 称，公司有 20 个兼职咨询者，很多都达到退休年龄，只有少数“年轻人”，但这些“年轻人”也已有 4、50 岁。

具经验的 COBOL 工程师会被聘请修复系统故障、编写程序手册，或者磨合新旧系统，时薪可超过 100 美元。对企业而言，这项成本似乎很高，但与更换系统的成本及可能产生的风险相比，还是九牛一毛。

只有原开发者最清楚自己设计的 COBOL 系统

巴克莱银行前行政总裁 Antony Jenkins 表示，现在大型的金融机构都是购并而成，因此银行打算更换旧系统时，问题已超越了“人才短缺”。“来自不同世代的旧系统叠床架屋，有时更互相交织”。

IBM 表示，他们已开设训练课程，12 年间培训逾 180,000 个 COBOL 开发者，所以业界其实已经意识到他们不能永远依赖早晚会离开的老员工。不过 COBOL 系统变化非常大，开发者鲜有制作指南，让其他人难以维护系统。

Hinshaw 称，一些他在 1970 年代为银行开发的软件到现在仍在使用，有时系统问题只能由原开发者解决，这是管理人员经常打给他的原因。

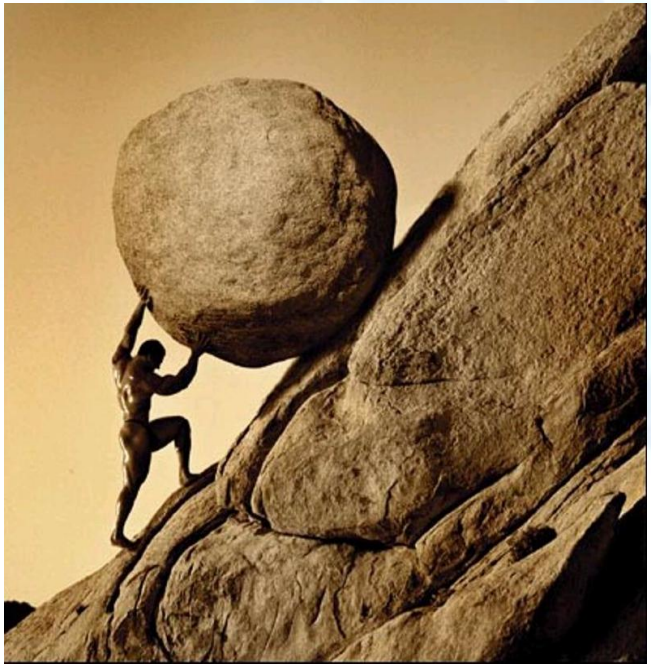
一些企业已开始转型。以澳洲联邦银行为例，他们在 2012 年透过 SAP 更新核心银行平台；瑞典北欧银行也会在 2020 年采取行动，不过转型期间，银行需要重新找回被辞退的老 COBOL 工程师。其中一个年届 60 岁的 COBOL 工程师在 2012 年被辞退，换上另一批较年轻、便宜、接受新程序语言教育的员工，但在 2014 年重新被邀请为承包商，解决系统问题。

“重新被银行召回是对我的肯定。”

软件演化与配置管理

现代版的西西弗斯和吴刚

- 当今的软件开发人员正是现代版的西西弗斯和吴刚，他们面对的是快速变化的技术和无休无止、越来越复杂的需求
- 必须寻求一种更好的软件研发方法论，以支持软件的持续演化



西西弗斯置顶巨石

吴刚伐桂

软件演化的处理策略

- 软件维护(Software Maintenance)
  - 为了修改软件缺陷或增加新的功能而对软件进行的变更
  - 软件变更通常发生在局部，不会改变整个结构
- 软件再工程(Software Re-engineering)
  - 为了避免软件退化而对软件的一部分进行重新设计、编码和测试，提高软件的可维护性和可靠性等
- 前者比后者的力度要小

软件演化与配置管理

1 软件演化

2 软件维护

3 软件配置管理(SCM)

4 持续集成

软件维护

- 软件维护
  - (ANSI/IEEE) 在软件产品发行和投入运行使用之后对其的修改，以改正错误，改善性能或其他属性，从而使产品适应新的环境或新的需求

软件维护的类型

- 纠错性维护：修改软件中的缺陷或不足
- 适应性维护：修改软件使其适应不同的操作环境，包括硬件变化、操作系统变化或者其他支持软件变化等
- 完善性维护：增加或修改系统的功能，使其适应业务的变化
- 预防性维护：为减少或避免以后可能需要的前三类维护而提前对软件进行的修改工作

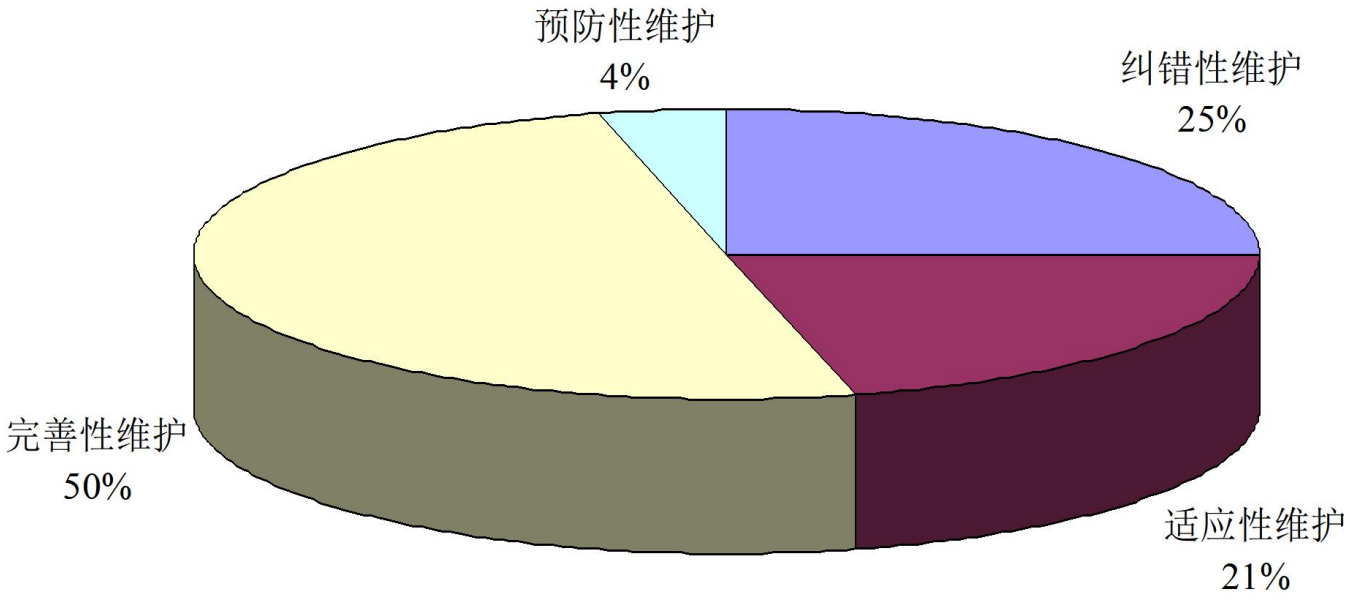
纠错性维护、适应性维护

- 纠错性维护(Corrective Maintenance):
  - 在软件交付使用后，因开发时测试的不彻底、不完全，必然会有部分隐藏的错误遗留到运行阶段
  - 这些隐藏下来的错误在某些特定的使用环境下就会暴露出来
  - 为了识别和纠正软件错误、改正软件性能上的缺陷、排除实施中的误使用，应当进行的诊断和改正错误的过程就叫做纠错性维护
- 适应性维护(Adaptive Maintenance):
  - 在使用过程中，外部环境(新的硬、软件配置)和数据环境(数据库、数据格式、数据输入/输出方式、数据存储介质)可能发生变化
  - 为使软件适应这种变化，而去修改软件的过程就叫做适应性维护

完善性维护、预防性维护

- 完善性维护(Perfective Maintenance)
  - 在软件的使用过程中，用户往往会对软件提出新的功能与性能要求
  - 为了满足这些要求，需要修改或再开发软件，以扩充软件功能、增强软件性能、改进加工效率、提高软件的可用性
  - 这种情况下进行的维护活动叫做完善性维护
- 预防性维护(Preventive Maintenance)：为了提高软件的可维护性、可靠性等，为以后进一步改进软件打下良好基础
  - 定义为“采用先进的软件工程方法对需要维护的软件或软件中的某一部分(重新)进行设计、编制和测试”

软件维护的类型



小结

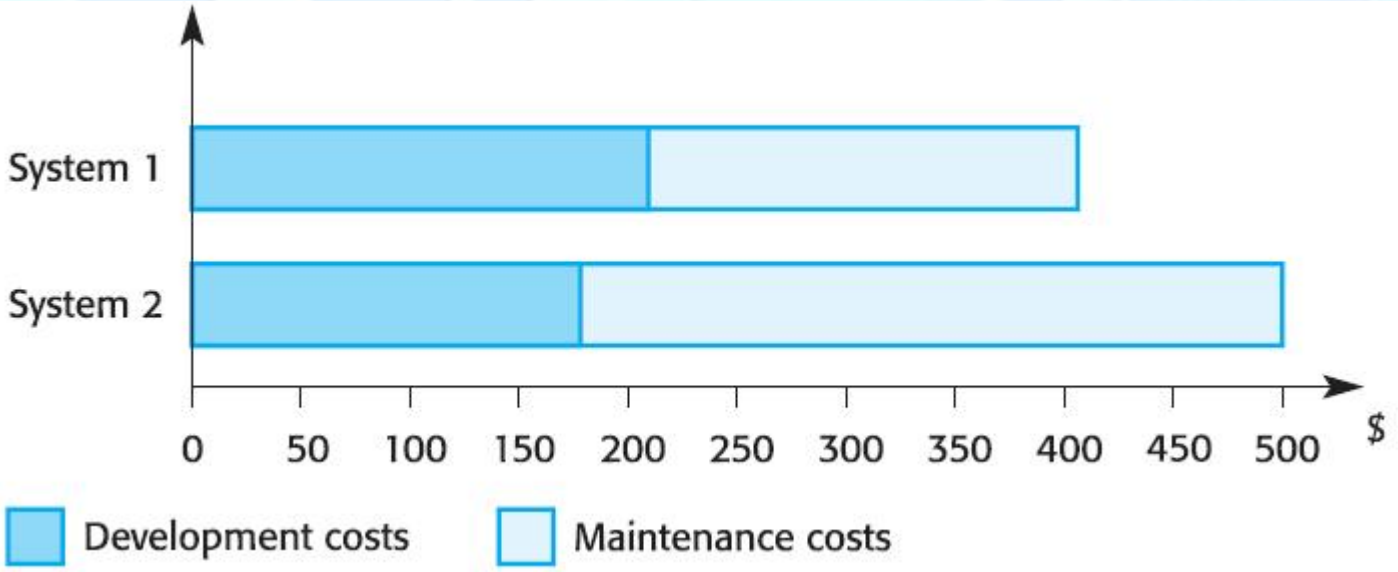
- 实践表明，在几种维护活动中，完善性维护所占的比重最大，即大部分维护工作是改变和加强软件，而不是纠错
  - 完善性维护不一定是救火式的紧急维修，而可以是有计划、有预谋的一种再开发活动
  - 来自用户要求扩充、加强软件功能、性能的维护活动约占整个维护工作的50%
- 软件维护活动所花费的工作占整个生存期工作量的70%以上，这是由于在漫长的软件运行过程中需要不断对软件进行修改，以改正新发现的错误、适应新的环境和用户新的要求，这些修改需要花费很多精力和时间，而且有时会引入新的错误

软件维护的内容

- 程序维护
  - 根据使用的要求，对程序进行全部或部分修改
  - 修改以后，必须书写修改维护报告
- 数据维护
  - 数据维护指对数据有较大的变动：如安装与转换新的数据库，或者某些数据文件或数据库出现异常时的维护工作，如文件的容量太大而出现数据溢出，需要数据归档、迁移、备份等
- 硬件维护
  - 硬件人员应加强设备的保养以及定期检修，并做好检验记录和故障登记工作

软件维护的成本

- 软件的维护成本极其昂贵
  - 业务应用系统：维护费用与开发成本大体相同
  - 嵌入式实时系统：维护费用是开发成本的4倍以上



软件维护的典型困难

- 软件维护中出现的大部分问题都可归咎于软件规划和开发方法的缺陷：
  - 软件开发时采用急功近利还是放眼未来的态度，对软件维护影响极大，软件开发若不严格遵循软件开发标准，维护就会遇到许多困难
- 例如：
  - 读懂原开发人员写的程序通常相当困难
  - 软件人员的流动性，使得软件维护时，很难与原开发人员沟通
  - 没有文档或文档严重不足
  - 软件设计时，欠考虑软件的可修改性
  - 频繁的软件升级，要追踪软件的演化变得很困难，使软件难以修改

造成困难的根本原因

- 很难甚至不可能追踪软件的整个创建过程
- 很难甚至不可能追踪软件版本的进化过程，软件的变化没在相应文档中反映出来
- 理解他人的程序非常困难，当软件配置不全、仅有源代码时问题尤为严重
- 软件人员流动性很大，维护他人软件时很难得到开发者的帮助
- 软件没有文档、或文档不全、或文档不易理解、或与源代码不一致
- 多数软件设计未考虑修改的需要(有些设计方法采用了功能独立和对象类型等一些便于修改的概念)，软件修改不仅困难而且容易出错
- 软件维护不是一项有吸引力的工作，从事这项工作令人缺乏成就感

不好的维护所造成的代价

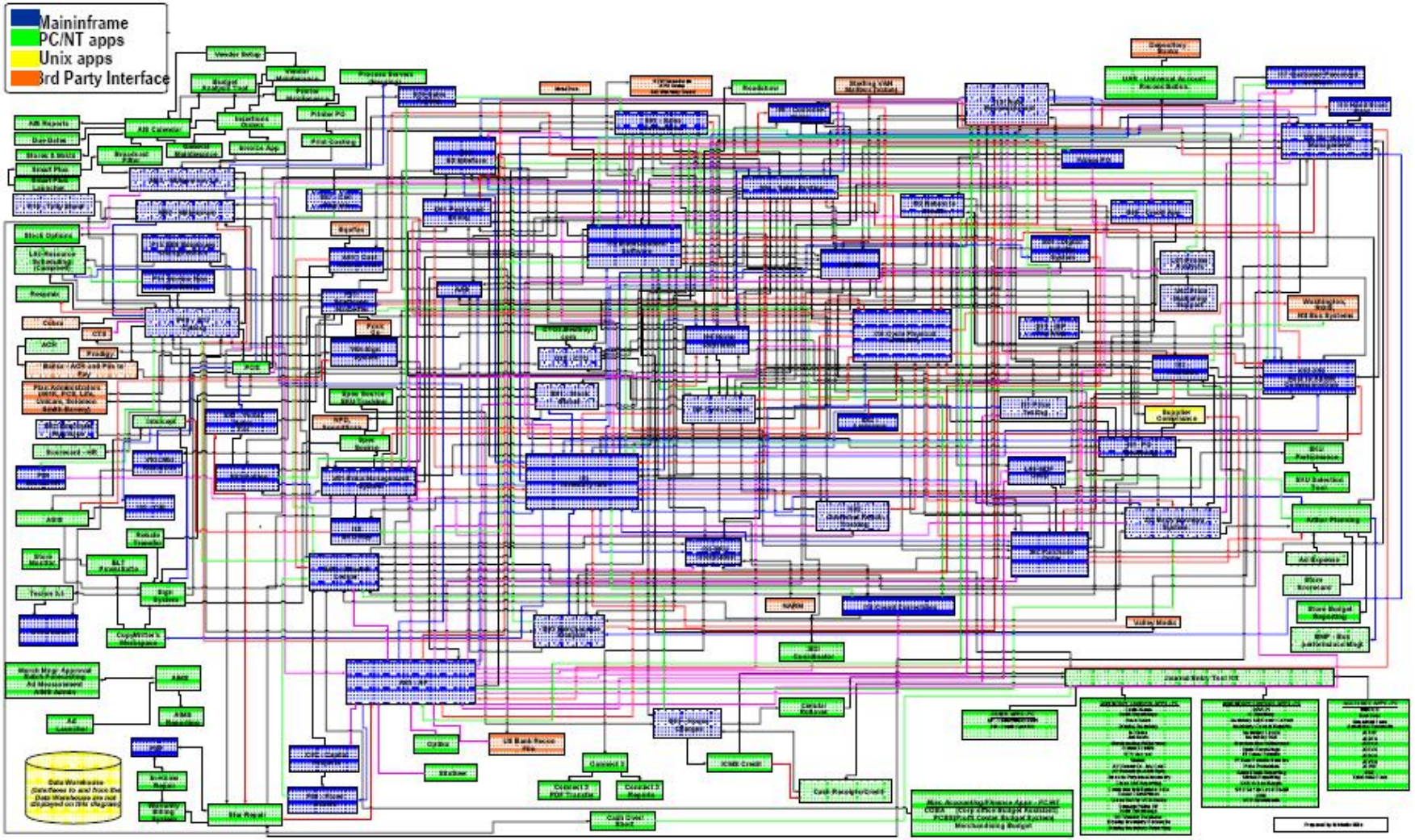
- 可用的资源必须供维护任务使用，以致耽误甚至丧失了开发的良机
- 当看来合理的有关改错或修改的要求不能及时满足时将引起用户不满
- 由于维护时的改动，在软件中引入了潜伏的故障，从而降低了软件的质量
- 当必须把软件工程师调去从事维护工作时，将在开发过程中造成混乱
- 生产率的大幅度下降

遗留系统

- 遗留系统(legacy system)
  - “已经运行了很长时间的、对用户来说很重要的、但是目前已无法完全满足要求却不知道如何处理的软件系统”
- 特点
  - 现有维护人员没有参与开发
  - 不具备现有的开发规范
  - 文档不完整，修改记录简略

软件演化与配置管理

遗留系统



遗留系统

- 更换遗留系统(Legacy System)是有风险的
  - 遗留系统几乎没有完整的描述
  - 业务过程和遗留系统的操作方式紧密地“交织”在一起
  - 重要的业务规则隐藏在软件内部
  - 开发新软件本身是有风险的
- 变更遗留系统的问题
  - 系统的不同部分是由不同的团队实现的
  - 系统的部分或全部是用一种过时不用的语言编写
  - 文档不充分或过时
  - 经过多年维护，系统结构可能已经破坏，理解设计难度大

软件演化与配置管理

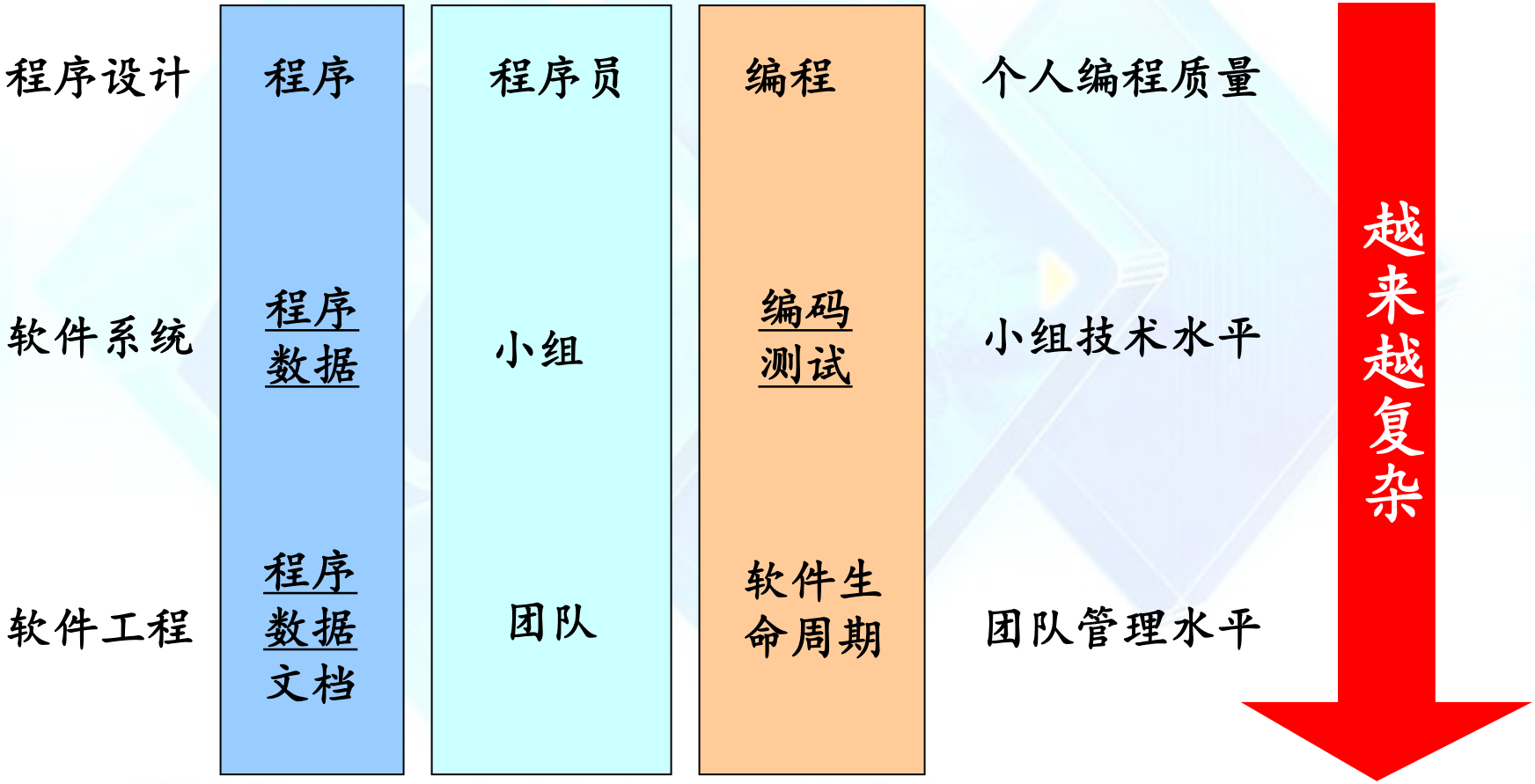
1 软件演化

2 软件维护

3 软件配置管理(SCM)

4 持续集成

软件配置管理的背景

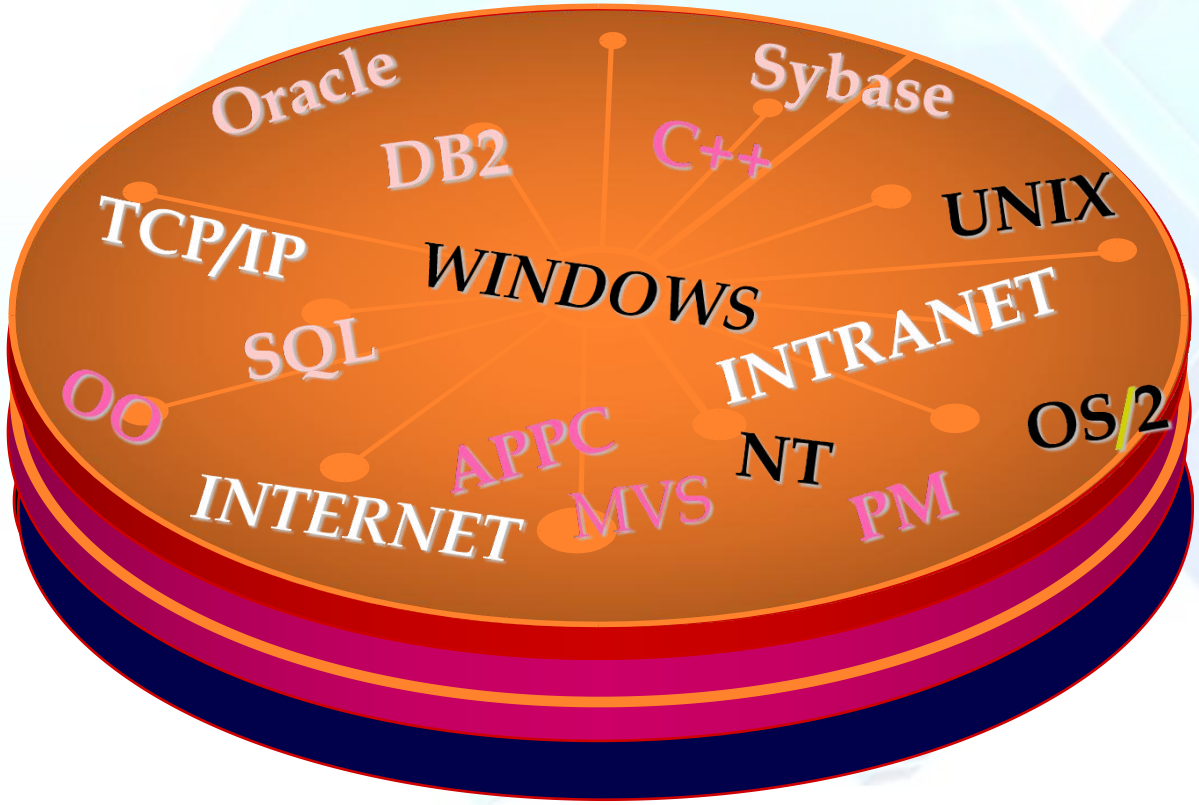

程序设计
程序
程序员
编程
个人编程质量
软件系统
程序
数据
小组
编码
测试
小组技术水平
软件工程
程序
数据
文档
团队
软件生
命周期
团队管理水平
越来越复杂

软件开发过程中面临的困境

- 缺乏对用户需求进行有效的管理和追踪的工具
- 产品升级和维护所必需的程序和文档非常混乱
- 代码可重用性差从而不能对产品进行功能扩充
- 开发过程中的人员流动经常发生
- 软件开发人员之间缺乏必要的交流
- 由于管理不善致使未经测试的软件加入到产品中
- 用户与开发商没有有效的产品交接文档

软件演化与配置管理

开发环境的复杂性



- 多操作系统
- 多开发工具
- 网络化
- 团队方式
- 异地开发

缺乏管理所造成的问题

软件生产达不到规模化

缺少有效的沟通机制

成员间缺少沟通

人员流动

硬件配置

- 你要购买一台计算机，需要考虑哪些配置？
  - CPU类型与主频
  - 缓存大小
  - 内存容量与频率
  - 硬盘容量与转速
  - 显示器大小与分辨率
  - …

软件配置

- 软件配置(software configuration)：由在软件工程过程中产生的所有信息项构成，它可以看作该软件的具体形态(软件配置项)在某一时刻的瞬间映像
- 软件配置管理(Software Configuration Management, SCM)

SCI = 软件配置项

软件配置管理的含义

- “协调软件开发从而使得混乱减到最小的技术称为软件配置管理；它是对开发团队正在开发软件的修改进行标识、组织和控制的技术，目的是使错误量降至最低，并使生产率最高。”

——Wayne Babich

《SCM Coordination for Team Productivity》
（SCM协调团队生产力）

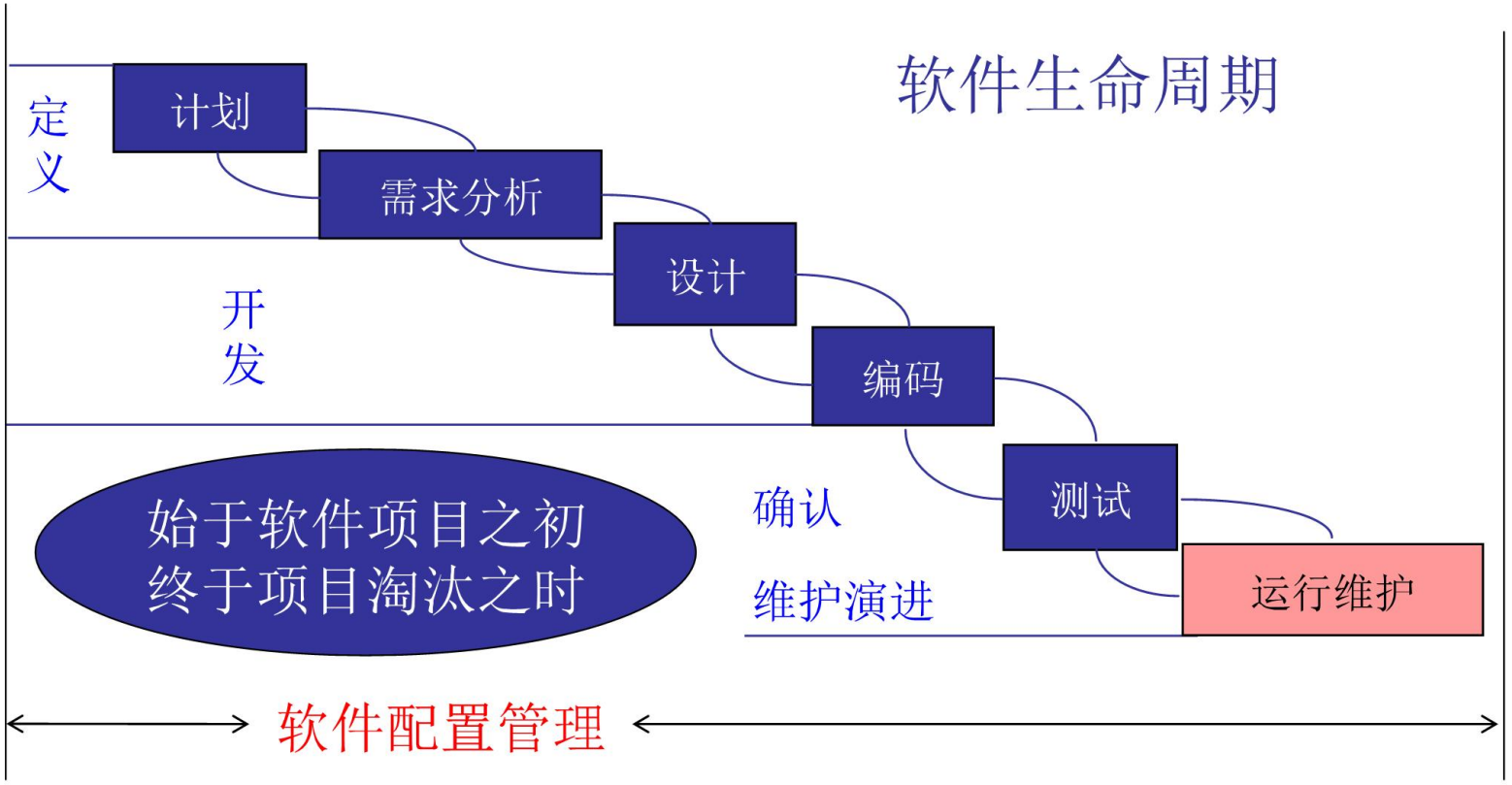
- “配置管理能够系统的处理变更，从而使得软件系统可以随时保持其完整性，因此称为‘变更控制’，可以用来评估提出的变更请求，跟踪变更，并保存系统在不同时刻的状态。”

——Steve McConnel

《Code Complete》（代码大全）

软件配置管理的特点

- SCM贯穿整个软件生命周期与软件工程过程



软件配置管理的目标

- 目标：
  - 标识变更
  - 控制变更
  - 确保变更的正确实现
  - 向开发组织内各角色报告变更

总结：当变更发生时，能够提高适应变更的容易程度，并且能够减少所花费的工作量

SCM的基本元素

- 配置项(Configuration Item, CI)
- 基线(Baseline)
- 配置管理数据库(CMDB)
- 最终硬件库(Definitive Hardware Store, DHS)
- 最终软件库(Definitive Software Library, DSL)

(1) 配置项

- 软件过程的输出信息可以分为三个主要类别：
  - 计算机程序(源代码和可执行程序)
  - 描述计算机程序的文档(针对技术开发者和用户)
  - 数据(包含在程序内部或外部)

  这些项包含了所有在软件过程中产生的信息，总称为软件配置项(SCI)

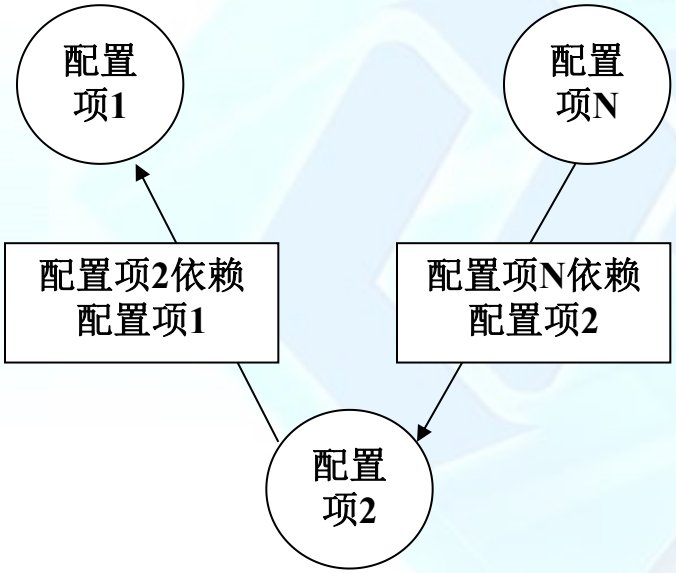
- SCI是软件全生命周期内受管理和控制的基本单位，大到整个系统，小到某个硬件设备或软件模块

配置项

- SCI具有唯一的名称标识和多个属性：
  - 名称
  - 描述
  - 类型(模型元素、程序、数据、文档等)
  - 项目标识符
  - 关联关系
  - 变更和版本信息

软件演化与配置管理

配置项之间的依赖关系

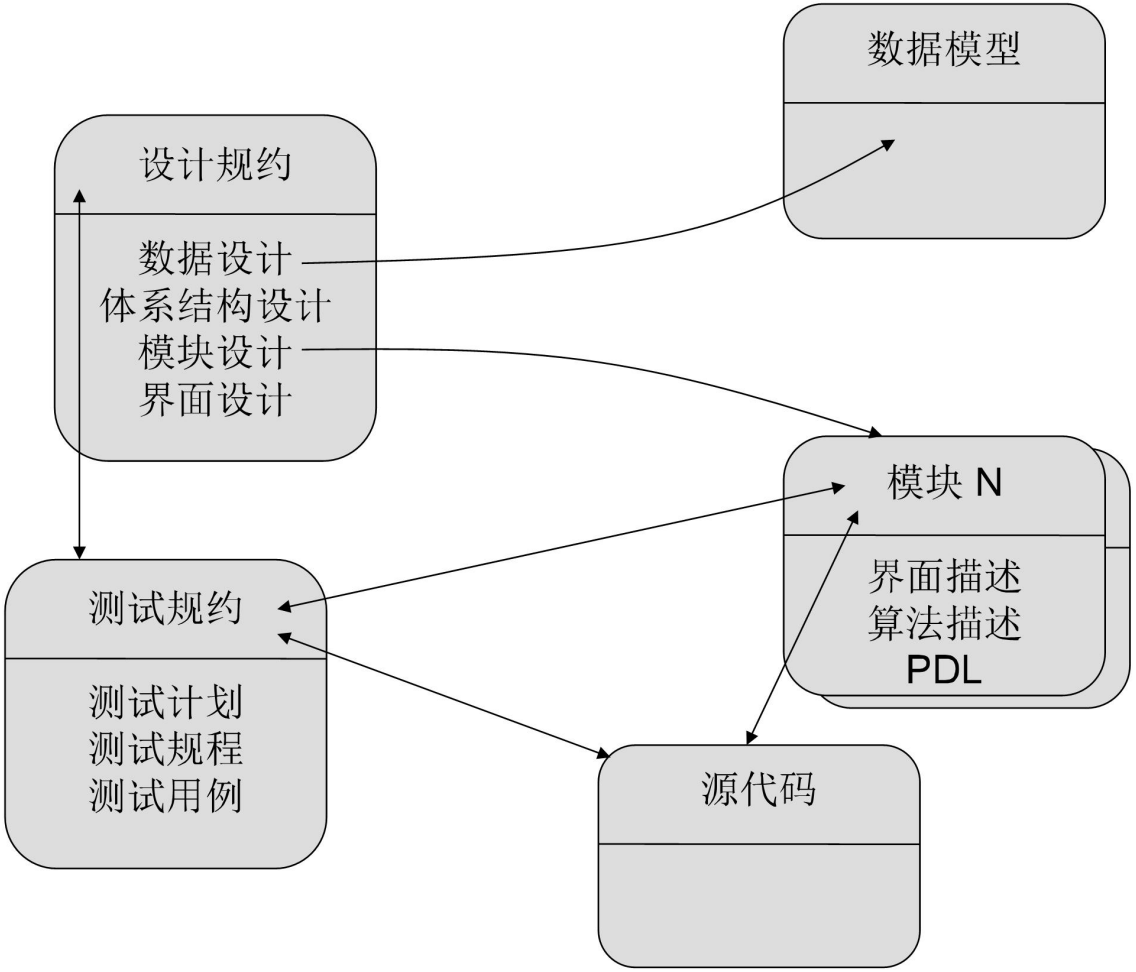


依赖关系	配置项1	配置项2	…	配置项N
配置项1		X		
配置项2				X
…				
配置项N				

配置项之间的关系

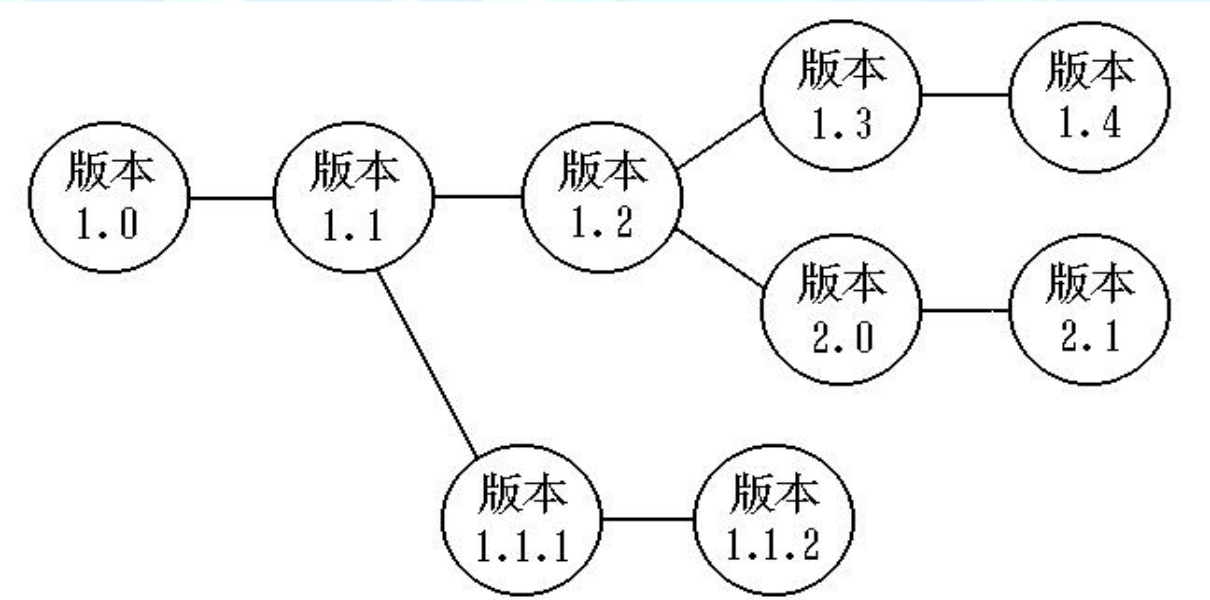
- 配置项之间都可能有哪些类型的依赖关系？
  - 整体-部分关系
    - **Data flow diagram (DFD) <part-of> analysis model**
    - **Analysis model <part-of> requirement specification**
  - 关联关系
    - **Data model <interrelated> data flow diagram (DFD)**
    - **Data model <interrelated> test case class m**
  - 还有哪些？

配置项之间的依赖关系


数据模型
设计规约
数据设计
体系结构设计
模块设计
界面设计
模块 N
界面描述
算法描述
PDL
测试规约
测试计划
测试规程
测试用例
源代码

配置项的演变图

- 在对象成为基线以前可能要做多次变更，在成为基线之后也可能需要频繁的变更
- 对于每一配置项都要建立一个演变图，以记录对象的变更历史

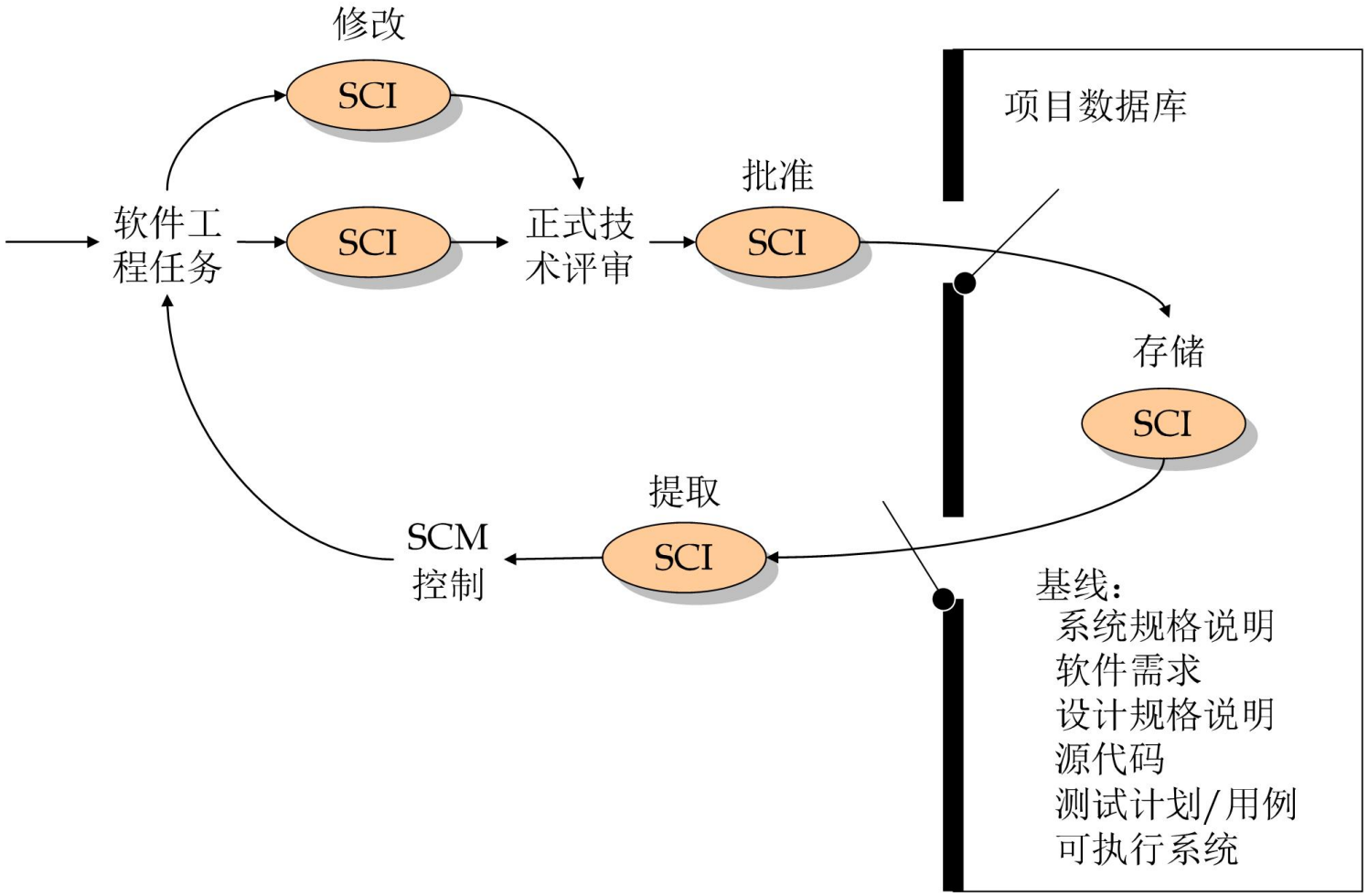


(2) 基线

- 基线：已经通过正式评审和批准的软件配置项（软件规格说明或代码等），它可以作为进一步开发的基础，并且只有通过正式的变更规程才能修改它
- 在软件配置项成为基线之前，可以迅速而随意的进行变更
- 一旦成为基线，变更时需要遵循正式的评审流程才可以变更
- 因此，基线可看作是软件开发过程中的重要“里程碑”

软件演化与配置管理

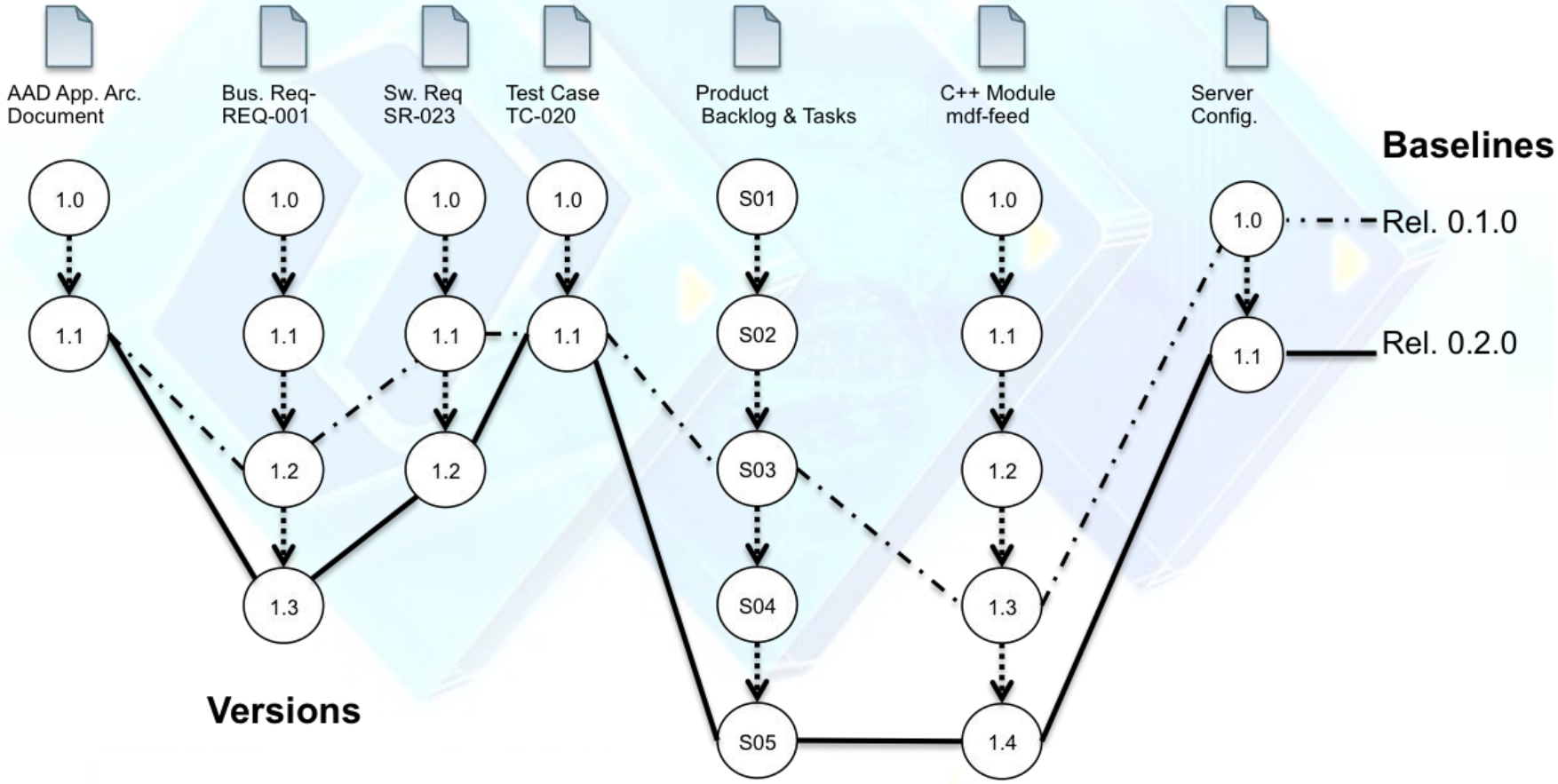
基线



软件演化与配置管理

配置项(SCI)与基线

- 基线是在某个时间点上对产品属性的一致描述，它是定义变化的基础



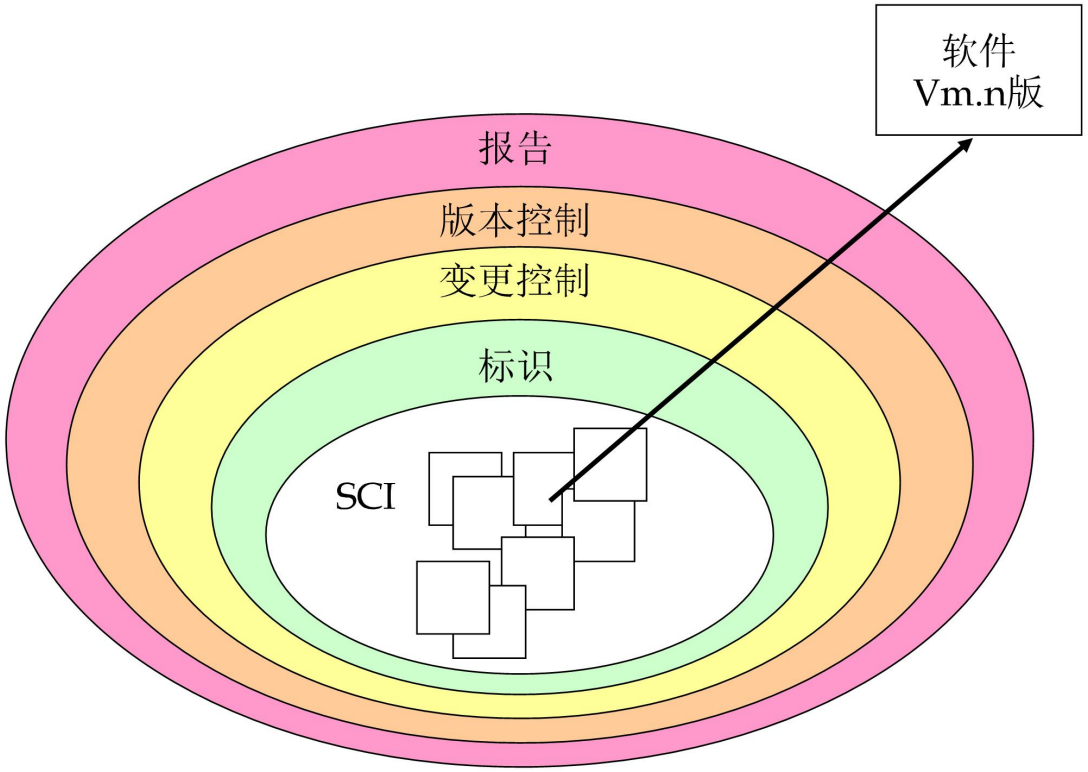
Version版本≥ Release发布≥ Baseline基线≥ Milestone里程碑≥ Checkpoiint检查点

(3) 配置管理数据库

- 配置管理数据库(CMDB)(也称“SCM中心存储库”)，用于保存与软件相关的所有配置项的信息以及配置项之间关系的数据库
  - 每个配置项及其版本号
  - 变更可能会影响到的配置项
  - 配置项的变更路线及轨迹
  - 与配置项有关的变更内容
  - 计划升级、替换或弃用的配置项
  - 不同配置项之间的关系

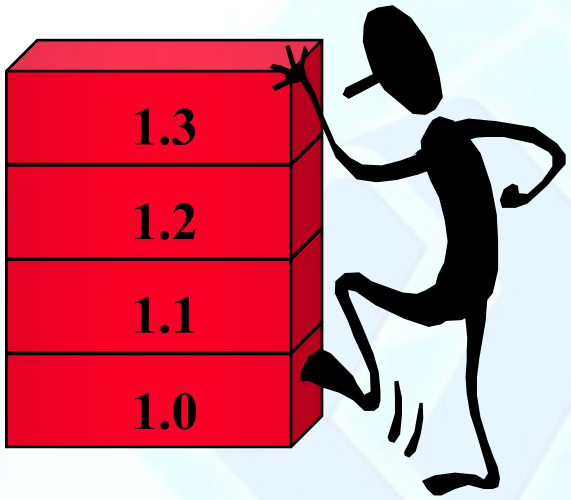
配置管理数据库

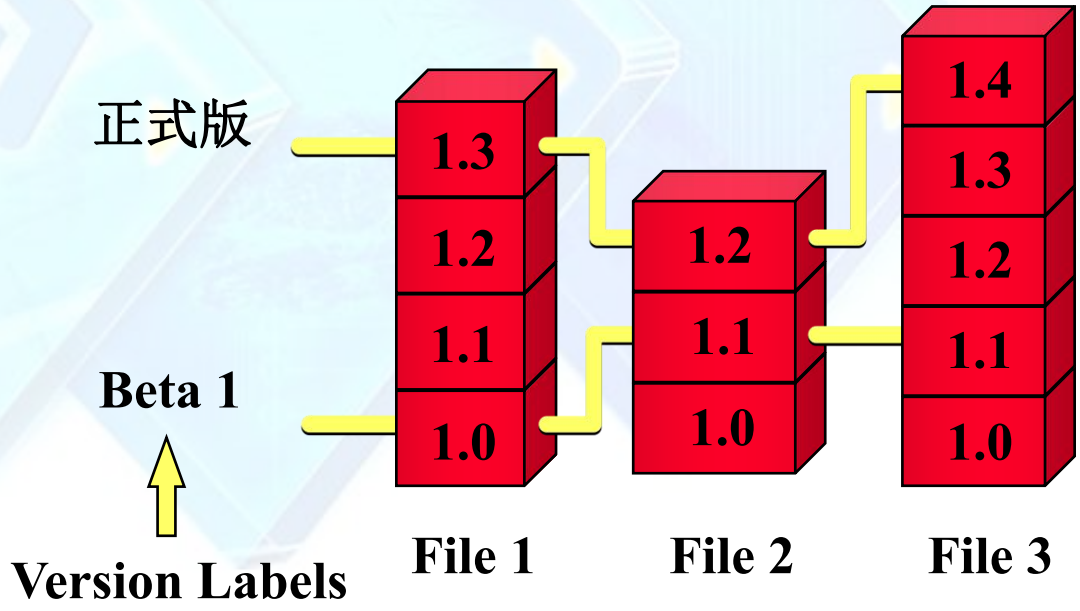
- **CMDB的功能：**
  - 存储配置项及其之间的关系
  - 版本控制
  - 相关性跟踪和变更管理
  - 需求跟踪
  - 配置管理
  - 审核跟踪



软件演化与配置管理

版本控制





SCM常用工具

- VSS (Microsoft Visual SourceSafe)
- CVS (Concurrent Version System)
- IBM Rational ClearCase
- SVN (Subversion)
- Git（GitHub、GitLab、GitEE …）

软件演化与配置管理

1 软件演化

2 软件维护

3 软件配置管理(SCM)

4 持续集成

持续集成

- 持续集成：敏捷开发的一项重要实践
  - Martin Fowler：团队开发成员经常集成他们的工作，每个成员每天至少集成一次，每天可能会发生多次集成；每次集成都通过自动化的构建(包括编译，发布，自动化测试)来验证，从而尽快地发现集成错误，大大减少集成的问题，让团队能够更快的开发内聚的软件
- 价值：
  - 减少风险：不是等到最后再做集成测试，而是每天都做
  - 减少重复过程：通过自动化来实现
  - 任何时间、任何地点生成可部署的软件
  - 增强项目的可见性
  - 建立团队对开发产品的信心

持续集成

- 所有的开发人员需要在本地机器上做本地构建，然后再提交到版本控制库中，从而确保他们的变更不会导致持续集成失败
- 开发人员每天至少向版本控制库中提交一次代码
- 开发人员每天至少需要从版本控制库中更新一次代码到本地机器
- 需要有专门的集成服务器来执行集成构建，每天要执行多次构建
- 每次构建都要100%通过
- 每次构建都可以生成可发布的产品
- 修复失败的构建是优先级最高的事情