软件工程导论 2018

Hox Zheng

2018-01-17 11:55:14

目录

1 软件工程的介绍

1.1 软件？

- 软件是一个产品。显示了由计算机硬件体现的计算能力。
- 软件是产品生产的载体，软件提供了计算机的控制，信息通信以及应用程序开发和控制的基础平台。
- 软件是指令的集合。通过这些指令可以满足预期的特征，功能和性能需求。
- 软件是数据结构，使程序能够合理的利用信息。
- 软件描述信息，它以硬拷贝和虚拟的形式存在，用来描述程序的操作和使用。

1.2 软件的特性，与硬件的区别？

- 软件是设计开发的，而不是传统意义上生产制造的。
- 软件不会磨损，但是长期使用和修改会缓慢退化。
- 软件是根据需求定制，并在不断的变动。
- 软件相对复杂。
- 而硬件是生产制造的，可以磨损和损坏，使用组件建造，与生产人员数量成正相关（软件不是），不会像软件一样的 “变”，相对简单。

1.3 软件退化

- 软件不会磨损，但是在使用过程中，不断的需求变更是软件退化的根本原因。
- 因为每次变更都会引入新的错误，使得失效率上升。不断的变更使得失效率整体逐渐上升。

1.4 软件危机

- 第一次软件危机。
  - 表现：效率低小。软件成本高。难以维护。
  - 原因：人员知识结构单一，code&fix 的模式。理论体系不成熟。开发管理不完善。
  - 解决：理论体系的形成，它指导了过程和项目管理。
- 第二次软件危机
  - 表现: 错误低质量的产品。项目延时，违约。工作时间太长。
  - 原因: 需求理解不充分。黑盒模型，与客户交流只是在开始和结束。
  - 解决：白盒模型的引入。
- 第三次软件危机
  - 需求变更，软件质量不能保证。软件演化，开发效率低下。软件重用和成本的问题。知识产权的问题。安全的问题。
  - 原因：遗产系统的使用。开源软件的使用。不确定的需求。
  - 解决：自动化，智能化。自适应，自修复，自测试。自主软件。

2 过程综述

2.1 软件工程的概念 [重要]

- 软件工程是建立和使用一套合理的工程原则。以便于经济的获得可靠的，可以在实际机器上运行的软件。
- 将系统化的，严格约束的，可量化的方法运用于软件开发，运行和维护，即将工程化方法应用于软件开发。
- 对上一条所定义的过程中的方法的研究。

2.2 软件的层次

- 从上到下: 工具, 方法，过程，质量关注点。
- 软件工程的基础：过程
- 软件工程的根基：质量关注点。

2.3 软件过程是工作产品构建时所进行的一系列活动，动作，任务的集合。

2.4 过程框架 [重要] ：沟通，策划，建模，构建，部署。

2.5 普适性活动：软件项目的跟踪和控制。风险管理。软件质量保证。正式技术评审。测量。软件配置管理。可复用管理。工作产品的准备和生产。

2.6 过程模型分为：惯用过程模型和敏捷过程模型。

3 过程模型 [惯例模型]

3.1 软件的生命周期。

- 同任何事物一样，一个软件产品或软件系统也要经历孕育、诞生、成长、成熟、衰亡等阶段，一般称为软件生存周期（软件生命周期）。
- 分析设计编码测试

3.2 软件过程模型 [重要]

3.2.1 瀑布模型（经典的生命周期）

- 标准的软件生命周期：分析设计编码测试
- 特点：
  - 从上一项活动中接受该项活动的活动成果作为输入
  - 利用这一输入实施该项活动应完成的内容
  - 给出该项活动的工作成果，作为输出传给下一项活动
  - 对该项活动实施的工作进行评审。若其工作得到确认，则继续下一项活动。
- 优点：简单，容易使用。为项目提供了按阶段划分的检查点，项目管理比较规范。每个阶段必须提供文档，而且每个阶段必须进行正式严格的技术审查。
- 缺点：
  - 瀑布模型过于依赖早期进行的唯一一次需求调查，不能适应需求的变化。
  - 瀑布模型是单一流程，开发中的经验教训不能反馈应用于本产品的过程。
  - 造成软件错误的积累和放大效应
- 适用：
  - 在实现之前清楚了解问题需求
  - 需求不会变更太大
  - 需求没有无法解决的高风险因素。
  - 需求对利益相关方来说是兼容的。
  - 实现需求的正确架构已经被很好的理解。

  - 有足够的时间顺序的进行项目。

3.2.2 增量模型

- 特点
  - 相当于瀑布模型的迭代
  - 随着日程时间的进展而交错的线性序列
  - 与原型不同，强调每个增量均发布一个可操作产品
  - 典型应用是同一产品的不同项目版本
- 优点
  - 提高对用户需求的响应
  - 有计划的管理技术风险，如早期增量版本中避免采用尚未成熟的技术
  - 若开发前期找不到足够的人员，早期的增量可以由少量人员完成，若核心产品口碑不错，可以成为下个增量投入更多人员
- 缺点
  - 不能破坏原来构造好的东西
  - 加入一个新的增量不是那么简单。调节各个增量之间的关系也很大.
  - 至始至终开发者和客户纠缠在一起，直到完成版本出来
  - 尽管其灵活性能适应需求变化，但也容易退化成边做边改模型，使软件过程控制缺乏整体性
- 适用：需求经常变化的软件开发，市场急需而开发人员和资金不能再设定的市场期限之前实现一个完善的产品的软件开发

3.2.3 演化过程模型

- 包括原型模型和螺旋模型。是迭代的过程模型：一系列活动的重复应用。

3.2.4 原型模型

- 特点快速开发可以演示的系统原型。有两种使用方法：1 作为最终产品的一部分。2 作为需求分析的工具。
- 优点
  - 快速的开发出可以演示的系统，便于与客户沟通交流, 及时获得用户的反馈。
  - 使用迭代技术能够逐步弄清用户的需求。
  - 可在项目早期就获取项目的相关数据，尽早进行风险管理和配置管理
  - 可使销售工作提前进行
  - 心理：对开发人员鼓舞
- 缺点
  - 软件的质量和可维护性差。结构不好。

  - 用户可能混淆原型系统和最终系统。
  - 不加控制让用户接触未稳定功能，对开发人员和用户不好
- 适用：需求模糊的项目，质量不能保证，是为定义需求而服务

3.2.5 螺旋模型 [风险驱动的过程模型]

- 特点：结合了原型的迭代性质和瀑布模型的系统性和可控性特点，在每个演进过程 (四个：制定计划、风险分析、工程实施、客户评估) 要标记里程碑，增加了风险分析
- 优点：
  - 结合了原型模型的跌打和瀑布模型的系统性，时空性。
  - 采用循环逐步加深系统的定义和实现深度。同时更好的理解，应对降低风险。
  - 确定一系列里程碑，确保各方都得到可行的解决方案。强调各开发阶段的质量
  - 可操作性好，风险驱动，支持现有软件复用。
- 缺点
  - 必须引入非常严格的风险识别，风险分析和风险控制，这对风险管理的技能水平提出高要求
  - 需要人员、资金和时间的大量投入
- 适用：大型开发系统和软件

3.2.6 喷泉模型

- 特点：是软件生命周期的各个阶段是相互重叠和多次反复的，各阶段没有明显的界线。是一种线性开发模型，与瀑布模型类似，只是从串行改并行
- 优点：提高软件项目开发效率，节省开发时间
- 缺点：
  - 在各个开发阶段是重叠的，因此在开发过程中需要大量的开发人员，不利于项目的管理
  - 要求严格管理文档，使得审核的难度加大，尤其是面对可能随时加入各种信息、需求与资料的情况
- 适用：用于面向对象方法，面向对象的分析和设计重叠，交叉、并行进行

3.2.7 统一过程

- 特点：用例驱动，以架构为核心，迭代并且增量，一种利用 UML 进行面向对象软件工程的框架，是一种增量模型。四个阶段：初始 inception，细化 elaboration，构造 construction，交付 transition
- 优点：
  - 提高了团队生产力，在迭代的开发过程、需求管理、基于组件的体系结构、可视化软件建模、验证软件质量及软件变更等方面，针对所有关键的开发活动为每个开发成员提供了必要的准则、模板和工具指导，并确保全体成员共享相同的知识基础。
  - 建立了简洁和清晰的过程结构，为开发过程提供较大的通用性。

- 缺点：在实际应用上，需要更多的工具的支持和普及推广工作
- 适用: 现代大型面向对象工程。

4 敏捷视角下的过程

4.1 敏捷开发

4.1.1 开发宣言：

- 个体交互胜过过程和工具
- 可工作软件胜过宽泛的文档
- 客户合作胜过合同谈判
- 响应变化胜过遵循计划

4.1.2 敏捷开发：为了克服传统的软件过程中认识和实践的弱点而设计。

- 特点：
  - 适应变更
  - 交流顺畅
  - 客户参与
  - 有效控制
  - 拉平了变更成本曲线。

4.2 极限编程

- 四个框架活动：策划，设计，编码，设计。
- 适用面向对象的方法作为推荐的开发泛型。
- 适用：小团队，高风险，快速变更的需求，强调可测试性。

5 系统工程

5.1 系统工程中的概念

- 系统：开发软件之前必须了解软件所处的外界环境。看树也要看森林。
- 系统中的元素：软件，硬件，人员，数据库，文档，规程，其他。
- 系统工程：关注目标系统各种相关要素的设计，并将其组织成有机的系统。

5.1.1 系统全局视图由多个领域组成，每个领域由多个要素组成，每个要素由完成特定功能的构建组成。

5.2 系统建模

- 建立模型
  - 定义在多考虑视图中满足需要的过程
  - 描述过程行为和该行为所依据的假设
  - 明确定义模型外在和内在输入
  - 描述有助于工程师理解视图的全部联系（包括输出）
- 制约因素
  - 假设，简化，限制，约束，客户偏好。
- 本质上倾向于分层分级。

5.3 系统模型的分类

5.3.1 Hatley-Prabhai 建模：

- 利于用户界面，输入，系统功能和控制，输出，维护和自检的表达方式，可以建立一个系统构建模型，为后面每个工程规范步骤建立良好的基础

5.3.2 利用 UML 系统模型

- 部署图：部署图描述的是系统运行时的结构，展示了硬件的配置及其软件如何部署到网络结构中。一个系统模型只有一个部署图，部署图通常用来帮助理解分布式系统。
- 活动图：实现各种功能时的具体步骤。
- 类图和用例图。

6 需求工程

6.1 需求工程的任务

- 提供一种良好的机制：理解客户需要什么，分析需求，评估可行性，协商合理方案，无歧义的详细说明方案，确认规格说明，管理需求以至将这些需求转化为可执行的系统。
- 活动：起始，导出，精化（是一个分析建模动作，最终结果是形成一个分析模型，定义问题提供的信息，功能和行为域），协商，规格说明，确认和需求管理。

6.2 需求工程的工作产品

- 必要性和可行性的陈述
- 系统或产品范围的界限说明。
- 参与需求导出的客户，用户和其他共同利益者的列表
- 系统技术环境的说明。
- 需求列表（按照功能加以组织）。
- 一系列适用场景，有助于深入了解系统或者产品在不同运行环境下的使用。
- 任何能够更好定义需求的原型。

6.3 需求开发的方法 [重要]

- 需求获取：1 开始过程：与客户建立初步交流。2 导出过程：通过访问和调查，获得需求的描述
- 需求分析：精化过程，通过分析建模，建立精确的技术模型，说明软件功能，特征和约束。
- 需求处理：1 协商过程 2 形成规格说明 3 需求确认

7 构建分析模型

7.1 分析模型的作用

- 分析建模使用文字和图表的综合形式，以相对容易理解的方式描绘需求的数据，功能和行为，更重要的是，可以更直接地评审他们的正确性，完整性和一致性。

7.2 分析模型的构建原则

- 模型应关注在问题域可见的需求，抽象级别应该高一些。
- 分析模型的每一个元素应该增加对软件需求的整体理解。并提供对信息，功能，系统行为的深入理解。
- 基于结构和其他非功能的模型应推延到设计阶段在考虑
- 最小化整个系统内的关联
- 确认分析模型为所有共同利益者带来好处。
- 尽可能保持模型简洁。

7.3 方法 [重要]

7.3.1 场景建模：从用户角度描述软件需求。[重要]

- 用例 [重要]：参与者和原件之间的某个交互的叙述——是主要的建模元素。定义了特定的功能或者交互的关键步骤。为其他建模活动提供必须的输入。
- 活动图：描述在特定场景中的处理流。
- 泳道图：显示了处理流如何分配给不同的用户和类.
- 用例图: 从用户的角度描述系统的功能, 并指出各个功能操作者.
- 部署图: 是用来显示系统中软件和硬件的物理架构.

7.3.2 类建模 [重要]

- 使用基于场景和面向流的建模元素中提取信息, 确定分析类, 可以使用语法分析从文本叙述中提取候选类, 属性和操作, 并制定用于定义类的标准,CRC[类, 职责, 协作] 索引卡可以用于定义类之间的联系. 使用 UML 建模符号定义类之间的层次. 联系, 关联. 聚合和依赖. 使用分析包将类分组. 为大型系统提供更好的管理.
- 类图: 描述系统中类的静态结构. 不仅定义了系统和分组, 表示类之间的联系, 如关联依赖聚合组合. 也包括类的内部结构.
- 协作图: 描述对象间的协作关系. 和顺序图类似. 显示对相间的动态合作关系.

7.3.3 行为建模

- 描述软件的动态行为. 使用基于场景, 面向流和基于类的元素作为输入, 从整体上表现分析类和系统的状态.
- 状态图: 描述类的所有可能的状态以及事件发生时状态的转移条件.
- 活动图: 通过提供特定场景内交互流的图形化表示来补充用例.
- 顺序图: 显示对象间的动态合作关系, 强调对象间消息发送的顺序, 同时显示对象间的交互.

8 设计工程

8.1 概念 [重要]

8.1.1 抽象

- 数据抽象：定义数据类型和施加于该类型对象的操作。限定对象的取值范围，只能通过这些操作来修改个观察数据。

- 过程抽象 (功能角度的抽象)：将功能体作为单个功能看待。该功能体实际上是由一系列更低级的功能或者代码来实现的。

- eg: 数据抽象：门过程抽象：开门

8.1.2 体系结构

- 软件体系结构关注系统的一个或者多个结构，包含软件部件，部件对外可见的属性以及部件之间的关系。
- 体系结构的作用
  - 方便利益相关人员的交流。
  - 有利于系统设计的前期决策
  - 可传递，易于理解的系统级抽象
- 体系结构反映了软件系统实现的高层方案
  - 软件系统越来越复杂，需要将其划分为若干部分分而治之—模块化。
  - 不同的小组或开发者负责不同的部分。最后在系统层面上集成。

  - 负责不同部分的开发者对于其他模块需要理解的信息越少越好 [抽象与信息隐藏]
  - 这些部分之间要定义清晰明确的接口。

8.1.3 模式 [便于复用软件]

- 每个模式都描述了一个在我们所处环境内反复发生的问题。描述了该问题的核心解决方案。
- 模式的类型
  - 结构模式 [如：张翔老师讲的 MVC，软件体系结构风格]
    * 表达了软甲系统的基本结构组织形式。包含一组预定义的子系统。规定这些系统的责任。同时提供了用于组织和管理这些子系统的规则和向导。
  - 设计模式 [如：适配器模式]
    * 为软件系统的子系统，组件，或者组件之间的关系提供一个精炼后的解决方案。
    * 它描述了在特定环境下，用于解决通用软件设计问题的组件以及这些组件相互通信时的可重现结构。
    * 如：适配器模式。适配器模式使得原本由于接口不兼容而不能一起工作的那些东西可以一起工作。
  - 编码模式
    * 是特定于编程语言的低级模式，描述如何用给定的编程语言实现特定的组件或者组件之间的关系
    * 例如：约定代码缩进风格，变量命名规范等代码层面上的问题。

8.1.4 模块化（面向过程编程思想）

- 背景：问题越复杂，解决问题花费的时间越多。
- 解决：将复杂问题分解成多个子问题，解决起来会更加的容易。（模块化思想的依据）
- 注意：不要无限制的划分软件。虽然划分的越多，子模块的工作量减少。但是工作量还取决于模块间接口的集成和负责不同模块的人员的沟通。如果划分的太多，集成和沟通的开销将会很大。
- 结论：寻找最佳模块化成都的平衡点。（每个模块成本和联接成本的权衡问题）

8.1.5 信息隐藏

- 每个模块隐藏自己的内部实现细节。模块内部的数据和过程不允许其他不需要这些信息的模块使用。
- 定义和实施对模块的过程细节和局部数据结构的存去限制。
- [地位] 信息隐藏是实现抽象/模块化机制的基本支撑

8.1.6 功能独立

- [地位] 功能独立是模块化的根本要求
- 模块各自完成独立的功能，明确可辨识。
- 模块独立性的指标：内聚度，耦合度。
- 高内聚：模块内部结构紧密
- 低耦合：模块间的关联依赖程度尽可能小。
- 符合信息隐藏和信息局部化的原则。
- [意义]：
  - 模块的开发者只需要注重一个相对独立的部分。
  - 不用过多的关心其他模块。
  - 修改和 Bug 所影响的范围被局部化。
  - 单个模块特别容易复用。
  - 独立的模块更易于维护和测试。
- 内聚度 [由强到弱]
  - 功能内聚 [最高程度的内聚]：单个模块内部的各个部分都是为完成一项具体功能而协同工作紧密联系不可分割的单个功能。
  - 分层内聚：高层能访问底层的服务，反之不能。
  - 顺序内聚：每个模块内部的各个组成部分必须顺序执行。前一个模块的动作的输出是后一个模块动作的输入。
  - 通信内聚：模块内各个组成部分的处理使用相同的输入数据或者具有相同的输出数据。
  - 过程内聚：模块内多个任务，即使动作不相同，不相联系，但是他们都受一个控制流支配，必须按照指定的过程执行。
  - 时间内聚：模块内个各个部分的处理与时间相关。如：启动时要进行的全部操作。
  - 实用内聚：每一个模块中，某一方面相关，其他方面都不相关的构建操作被分为一个组。

- 耦合度 [由高到底]
  - 内容耦合：一个模块可以直接访问另一个模块的内容或者内部功能
  - 共用耦合：多个模块共同访问某些公共数据环境。
  - 外部耦合：一组模块都访问同一全局简单变量，而不是同一全局数据结构，也不是通过参数表传递该全局变量的信息。
  - 控制耦合：模块间的交互参数包含控制信息，可影响另一个模块的执行逻辑。
  - 标记耦合：模块间传递特定的数据结构
  - 数据耦合：模块间只通过参数列表传递简单的数据。
  - 非直接耦合：两个模块可以相对独立工作。
    * 例程调用耦合：一个操作调用另一操作。[需要减少]
    * 类型使用耦合：A 使用 B 定义的一个数据类型.
    * 包含或者导入耦合
- 精化
  - 把问题的求解过程分解成若干步骤，每一步都比上一步更加精化，更接近实际问题的解法。
  - 常与分层抽象的思想结合：
    * 抽象使得设计者能够描述过程和数据而忽略底层的细节。
    * 求精有助于设计者在设计过程中届时底层的细节。
    * 高层抽象将在下层不断精化，最周得到软件的实现。

8.1.7 重构

- 重构是不改变外部功能的情况下改善代码的内部结构。
- 重构的目的是：发现个修改冗余，未使用的元素，抵消的算法，垃圾的数据结构，不好的设计。

9 软件体系结构

9.1 为什么进行体系结构设计 [重要]

- 可以让软件工程师在满足软件需求的情况下分析设计的有效性
- 工程师可以在设计变更相对容易的阶段考虑架构的替代方案
- 减少与软件架构相关的风险。
- 软件系统设计是开发者之间分工和合作的基础，方便了人员之间的交流。
- 有利于系统的亲啊其决策，设计方案是决定系统质量的主要因素。
- 体系建构构建了一个可传递，易于理解的系统级抽象。

9.2 软件体系结构风格 [重要]

9.2.1 定义

- 体系结构风格定义了一系列系统的结构组织模式，它是一类具有相似结构的系统体系结构的抽象。
- 换句话说，一个问题有多种解决问题的模式，但我们根据经验，通常采取特定的模式，这就是风格。
- 体系结构描述：构件，负责构件之间通信协作的连接器，定义构建之间怎样集成的系统约束，使设计者能够理解整个系统的语义模型。

9.2.2 软件体系结构风格分类

- 数据为中心的体系结构：一些数据保存在整个结构的中心，并且被其他部件频繁的使用，添加，删除，或者修改。
- 数据流风格的体系结构：适用于输入数据被一系列计算或者处理构建变换成输出数据。
- 调用和返回的体系结构：这种风格使一个软件设计者设计出非常容易修改和扩充的体系结构
  - 主程序/子程序分割：传统的程序就够，功能分解为控制层次
  - 远程过程调用风格的体系结构：该体系结构的构建分布在网络的多个计算机上。
  - 程序结构的深度和宽度：程序结构的层数/同一层的的最大模块数
  - 模块的扇入和扇出：扇出表示一个模块直接调用或控制其他模块的数目。扇入表示调用一个给定模块的模块的数目。
- 面向对象风格的体系结构：系统部件封装数据和操作数据的方法。部件之间的交互和协调通过消息来传递。
- 层次式风格的体系结构：在这种结构中，定义了不同的层次，每层都完成了相对外层更靠近机器指令的操作。缺陷：降低了系统的性能，有时会导致级联式修改。

9.2.3 软件体系结构的设计方法

- 结构化设计：主要解决如何将数据流图映射为软件体系结构
  - 复审和精华数据流图
  - 确定数据流图的类型 1 事务型 [数据流沿着输入路径到达一个事务中心] 2 变换型 [数据流图分为输入变化和输出]
  - 将数据流图映射为初始结构图
  - 改善初始的结构图
- 面向对象设计：
  - 子系统设计。OOD 方法也采取了问题分解的方法。定义若干个一致的类与对象，关系，行为，功能的集合，这样一个集合就是一个子系统。
  - 对象设计：对每一个对象的数据进行设计，基本上属于构建级别的设计阶段。
  - 消息设计：描述每个对象可以接收和发送的消息的接口。具体的消息设计在构建级设计阶段完成。

  - 方法设计：进一步对对象的方法进行求精。主要在构建级设计阶段完成。

10 构建级别建模

10.1 什么是构建

- 面向对象的观点：构建就是一些相互协作的类的集合。
- 传统的观点：构建是程序的一个基本元素。结合了过程逻辑，以及实现过程逻辑的数据结构。提供接口让组件调用它。

10.2 构件的设计原则 [重要：原谅我在 UML 就背会了哈哈哈]

- 开闭原则：构建应该对扩展是开放的，而对修改时封闭的。
- 替换原则：所有引用基类的地方必须能透明地使用其子类的对象。通俗的讲就是：只要父类能出现的地方子类就可以出现，而且替换为子类也不会产生任何错误或者异常，使用者可能根本就不需要知道是父类还是子类，但是，反过来就不行了，有子类出现的地方，父类未必就能适应。因为子类有的东西父类不一定有。
- 依赖导致原则：更多的去依赖抽象的类和接口，而不是依赖具体的类。
- 接口分离原则：将通用的接口分离成多个功能独立的子接口。

10.3 内聚性和耦合性 [重要: 参考前面]

10.4 构建设计的方法

- 画软件流程图
- 决策表 [四个部分]
  - 左上部列出所有条件
  - 右上部是可能组合的条件，每一列表示一种可能
  - 左下部是所有可能做的动作
  - 右下部是每一种条件组合对应的应作的工作
- PDL ： 是一种描述功能部件算法设计和处理细节的语言，陈伟设计性语言，是一种伪代码

11 测试

- 测试时验证和确认的一部分。验证：我们做的程序正确吗？确认：我门在做正确的产品吗 [需求]?

11.1 单元测试 [重要]

- 单元测试时针对程序中的模块或者部件的测试，主要揭露编码阶段产生的错误。
  - 单元测试侧重于对软件设计的最小单元进行测试
  - 单元测试的目的是在开发环境中检查单元程序模块内部的逻辑，算法和数据处理结果的正确性
  - 单元测试的内容是算法逻辑，数据定义的理解和使用，接口，各种控制路径，边界条件，错误处理。
- 面向对象的单元测试，最小的单元是方法。[由于继承，多态等特性，对方法测试往往是无效的]
- 面向对象环境中的单元测试主要是对类的测试。

11.2 集成测试 [重要]

- 集成测试是构造软件体系结构的系统化技术，同时也是进行一些旨在发现与结构相关错误的测试。
- 为什么进行集成测试?
  - 数据可能在通过接口时丢失
  - 一个模块可能对另一个模块产生非故意的有害影响
  - 当子功能结合起来时，可能达不到预期的主功能。
  - 单个模块的可以接受的不精确性，结合起来时可能扩大到不可接受
  - 全局数据结构可能存在问题
- 传统的集成测试策略
  - 一步到位的集成
  - 增量式集成
    * 自顶而下的集成：从主控模块开始，按照程序结构图的个控制层次，采用深度优先或者广度优先的方式逐个集成到整个结构中，并对其测试。
    * 自底而上的集成：从程序的结构的最底层木块开始，或者原生的子模块。按照程序结构图的控制层次将上层的模块集成到整个结构中，并对其进行测试。
- 面向对象的集成测试
  - 面向对象没有明显的控制层次，不能使用自顶而下，什么什么的方法。类之间的相互作用，也不能每次只集成一个操作。
  - 方法
    * 基于线程的测试：集成响应系统的一个输入时间所需的一组类，每个线程单独的集成和测试
    * 基于使用的测试：先测试独立类，然和测试依赖类，直到整个系统集成成功
- 回归测试：对以及测试过的子集重新执行，确保对程序的修改没有传播非故意的副作用
- 冒烟测试：尽可能早的发现可能造成项目延迟的业务阻塞

11.3 确认测试：发现软件与需求不一致的地方

11.4 系统测试：对集成后的产品和解决方案进行测试，评价系统对需求规格说明的功能和非功能的符合性的测试。

11.5 测试技术

11.5.1 白盒测试 [重要]

- 也称结构测试，或者逻辑驱动测试。
- 基于内部逻辑结构，针对程序语句，路径，变量状态的测试。
- 确认程序中各个分支条件是否有效，正确。

11.5.2 黑盒测试 [重要]

- 也称行为测试
- 把程序看出不能打开的盒子，不考虑程序内部结构和内部特性。只考察数据的输入，条件限制，和数据输出。

11.5.3 静态分析：对模块的源代码进行研读。不需要对代码编译运行。动态分析：运行程序，发现错误。

11.5.4 手工测试 [重要]

- 静态分析只能进行手工测试，虽然有辅助工具，但是以人为主体。
- 特点：
  - 发现缺陷率高
  - 容易实施
  - 拥有创造性和灵活性
  - 覆盖率量化困难
  - 效率低，依赖人力

自动化测试 [重要] * 动态测试可以部分或全部自动化测试。* 特点 * 自动运行速度快 * 测试结果准确 * 测试脚本的高复用 * 永不疲劳 * 可靠

12 项目管理

12.1 4P 模型 [重要]

- 人员 [people]：如何有效的组织人员

- 产品 [product]： 确定产品的范围。获得开发者和客户的一致认可
- 过程 [process]：指定软件开发的综合计划
- 项目 [project]：采用科学的方法和工具对项目进行管理

12.2 W5HH[5 个问题]

- 为什么开发这个系统?
- 将要做什么?
- 什么时候做?
- 某功能由谁负责?
- 他们的机构组织位于何处?
- 如何完成技术和管理工作?
- 每种资源需要多少

13 度量 [对一个系统，构建，过程具有给定属性的量化测试程度]

13.1 McCall 的质量因素 [重要]

- 影响因素分为两大类：可以直接测量的因素 [例如测试时发现的缺陷]。只能间接测量的因素 [易用性，可维护性。。].
- 内容：正确性、可靠性、效率、完整性、易用性、可维护性、灵活性、可测试性、可移植性、可复用性、互操作性。

13.2 Measures, Metrics, Indicators [重要]

- 测度 (measure)：为产品或过程的某些属性的程度、数量、维数、容量或大小提供量化的指标，而”测量 (measurement)“是确定测度的动作
- 度量 (Metrics): 度量是一个系统、构件或过程具有给定属性量化测量程度。
- Indicators 指标：是一个度量或者多个度量的组合，它提供了对软件过程，软件项目或产品本身的深入理解

13.3 度量的作用 [重要]

- 过程度量作用：提供能够引导长期的软件过程改进的一组过程指标。
- 项目度量作用：使得软件管理者能够（1）评估正在进行中的项目的状态（2）跟踪潜在的分险（3）在问题造成不良影响前发现他们（4）调整工作流程或任务（5）评估项目团队控制软件工作产品质量的能力
- 产品度量作用：为分析、设计、编码和测试能更客观的执行和更定量的评估提供基础

14 估算

项目计划任务和内容 [重要]

14.0.1 项目计划任务

- 规定项目范围
- 确定可行性
- 分析风险
- 确定需要的资源（a、确定需要的人力资源；b、确定可重用的软件资源 c、标识环境资源）
- 估算成本和工作量（a、分解问题 b、使用规模，功能点，过程任务或用例等方法进行两种以上恩德估算 c、调和不同的估算）
- 制定项目进度计划（a、建立一组有意义的任务集合 b、定义任务网络 c、使用进度计划工具指定时间表 d、定义进度跟踪机制）

14.0.2 项目计划的内容

- 项目计划内容估算，进度安排，风险分析，质量管理计划和变更管理计划

14.1 LOC & FP

14.1.1 LOC[面向规模的度量]

- LOC 以代码行作为其他测量的规范化因子

14.1.2 FP[面向功能的度量]

- FP(功能点) 则是从信息域及对问题复杂度的主观评估中导出的

14.1.3 二者共性

- 均以问题分解为基础，LOP 更甚，问题分解越详细越好。
- 要求 WBS(work breakdown structure) 中每个工作包都是可以分别独立估算的
- 每个工作包都有基线生产率度量值可以参考，或者可以预估其生产率。
- 基线生产率度量：根据 LOC 和 FP 的测量，计算出软件开发组织对某类问题的生产率，LOC/pm 或 FP/pm

15 进度

15.1 项目工作量分配原则 40-20-40：总体工作量的 40% 分配给前期的分析和设计，20% 用于编码，40% 用于后期测试

15.2 任务网络 [数据结构？]：活动网络，是一个项目任务流程的图形表示。

- 优点：很好地展示优先关系；定义关键路径的能力；执行 “如果” 分析的能力
- 缺点：默认模型假定资源是无限的，我们需要加紧资源依赖关系当确定自己的 “真正” 的关键路径；难以跟随大的项目

15.3 关键路径 [李必信没讲]

- 机动时间：在不影响整个工期的情况下，当前任务允许延迟的最长时间
- 关键路径：由机动时间为 0 的任务组成的项目路径。

15.4 里程碑 [重要]

- 甘特图：项目进行的时间表，就是项目进度表
- 在甘特图中，没想任务的完成以必须交付的文档和通过评审为标准，因此他们往往作为项目的里程碑
- 作用：敏捷、详细、可度量、可分配、现实性、期间时限

16 风险

16.1 被动和主动的风险管理

- 被动：项目团队在发生风险时对其做出反应；缓解 - 预计消防的额外资源计划；(救火模式) 修复失败-当风险发生，才找到资源并应用；危机管理 - 失败不响应应用资源，项目处于危险之中
- 主动：执行正式风险分析；组织纠正风险的根本原因　主动风险管理过程：风险识别，分析，规划，监控

16.2 RMMM[风险规划]

- 风险避免 [Risk Mitigation]
- 风险监测 [Risk Monitoring]：可以通过风险知识，评估风险是否真的发生了。保证正确实施了各项缓解步骤。收集相关有用的风险分析信息。
- 风险管理及应急计划 [Risk Management]

- 实施条件：风险避免和缓解工作已经失败。风险已经发生、

17 质量

17.1 软件质量的因素

- 软件要符合明确规定的功能和性能需求，符合已清晰文档化的开发标准，并且具有专业人员开发的软件所应有的隐含特征
- 当根据对象的可测量特征考察一个对象时，可以有两种不同质量：（1）设计质量：设计者为一个产品规定的特征，包括系统的需求，规格说明和设计（2）一致性质量：在制造产品过程中遵守设计规格说明的制度。
- 质量成本包括预防成本，失效成本，鉴定成本
- 用户满意度 = 合格的产品 + 好的质量 + 按预算和进度安排交付
- 内部质量/外部质量

17.2 质量保证活动

- 指定 SQA 计划
- 参与开发项目的软件过程描述
- 评审各项软件工程活动，以检验是否符合定义的软件过程
- 审核指定的软件工作产品，以验证其是否符合定义的软件过程中的相应部分
- 确保软件工作及工作产品中出现的偏差已文档化，并按文档化的规程进行了处理
- 记录所有不符合的部分，报告高层管理者

17.3 正式技术评审 [重要]: 是一项由软件工程师（以及其他人）进行的软件质量控制活动。

- 在软件工程过程的每个活动的后期进行
- 采用正式的会议评审
- 通过正式评审意味着里程碑和基线
- 目标:
  - 发现软件的任何一种表示形式中的功能、逻辑或实现上的错误;
  - 验证评审中的软件是否满足其需求。
  - 保证软件表示符合预先定义的标准
  - 得到以统一的方式开发的软件
- 作用:
  - 提供培训机会，使初级工程师能够了解软件设计和实现的不同方法
  - 使大量人员对软件系统中原本不熟悉的部分更了解
  - 培训后备人员和促进项目连续性

  - 评审会议条件：（1）通常 3-5 人（2）提前准备，占用每人工作时间不超过 2 小时（3）会议时间小于 2 小时
- 软件质量的成本
  - 预防成本：包括质量计划，正式技术评审，测试设备及培训
  - 鉴定成本：鉴定成本：包括为深入了解 “首次通过” 各个过程时的产品状态而开展的那些活动
  - 失效成本：如果在将产品交付给客户之前没有缺陷的话就不会存在的成本。分为内部失效成本和外部失效成本。
    * 内部失效成本是指在产品交付给客户之前发现了错误而引发的成本，包括返工、修复以及失效模式分析、
    * 外部失效成本是指与产品交付给客户之后所发现的缺陷相关的成本，如解决客户抱怨、退换产品、保修工作等。

18 变更

18.1 软件配置项

- 在软件工程过程中创建的信息。为配置管理设计的软件工作产品的集合，它在配置管理过程中作为单个实体对待。
- 包括：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（包含在程序内部的数据，或程序外部的数据）、开发环境

18.2 版本

- 与计算机软件配置项的完全按编纂或重编纂相关的计算机软件配置项的初始发布或再发布

18.3 基线

- 已经通过正式评审和批准的规格说明或产品，它可以作为进一步开发的基础，并且只有通过正式的变更控制规程才能修改它。

18.4 发布

- 一项配置管理行为，它说明某配置项的一个特定版本已准备好用于特定的目的（例如发布测试产品）

18.5 软件配置管理 SCM 流程

18.5.1 SCM 任务：SCM 任务：标识，版本控制，变更控制，配置审核

- 标识：为控制和管理软件配置项，必须对每个配置项单独命名，然后用面向对象方法进行组织。可以进行标识的对象：基本对象和聚合对象。
- 版本控制：结合规程和工具，用来管理在软件过程中所创建的配置对象的不同版本。
- 变更控制：
- 配置审核：通过正式技术评审或软件配置审核来保证变更的有效性。是正式技术评审的补充。
- 报告：配置状态报告 CSR（状态记录），解答了问题，发生什么事，是谁做的，什么时候发生的，会影响到别的什么。可使管理者和开发人员可以评估重要的变更。