27-29

- **chp27 程序规模对构建的影响**
  - 27.1 交流和规模
    - 和成员数量的平方成正比
  - 27.2 项目规模的范围
  - 27.3 规模对错误的影响
  - 27.4 规模对生产率的影响
  - 27.5 规模对开发活动的影响
- **chp28 管理构建**
  - 28.1 鼓励良好的编码实践
    - 受人尊敬的架构师制定共同开发标准
    - 设定标准的前提
      - 团队不排斥
      - 排斥：提供知道原则、建议、最佳实现也可以
    - 具体采用的手段
      - 结对编程
      - 代码review
      - 要求代码签名
      - 提供好的代码示例参考
      - 强调代码的公共财产属性，确认可公开
      - 奖励好代码
      - 一份简单标准
        - 管理者不是技术尖子反而有利于阻止产生‘聪明的’或难易理解的代码（管理者要理解所有代码）
    - 本书的角色
  - 28.2 配置管理
    - 变更控制：系统定义项目工件和处理变化，以使项目一直保持其完整性的的实践活动
    - 需求变更和设计变更
      - 遵循某种系统化的变更控制手续
      - 成组的处理变更请求

- 成组的处理变更请求
  - 开发过程中收集变更点，最后一起处理。从中可以选出优先级高的
  - 评估每项的变更成本
  - 提防大量的变更请求
  - 成立变更控制委员会或类似机构
  - 警惕官僚主义，但也不能为此不做变更控制
- 软件代码变更
  - 引入代码版本控制软件
- 工具版本|pom等
- 机器配置
  - 保持相同的机器配置，代码开发环境等
- 备份计划

28.3 评估构建进度表

- 评估的方法
  - 建立评估的目标
  - 为评估预留专门的时间，并做出计划
  - 清楚的说明软件需求
  - 在底层细节层面进行评估
  - 使用若干不同的评估方法，并且比较其结果
  - 定期做重新评估
- 评估构建的工作量
- 对进度的影响
- 评估与控制
- 如果你落后了该怎么办
  - 希望自己能赶上
  - 扩充团队
    - 项目可细分成不同的任务时才可以
  - 缩减项目范围
    - 必须有
    - 有了更好
    - 可选择

- 有关软件评估的额外资源

- 28.4 度量
  - 任何一种项目特征都是可以用某种方法被度量的，而且总会比根本不度量好的多
  - 留心度量的副作用
    - 谨慎选择在哪些环节要被度量
    - 人们会倾向做那些被度量的工作，而忽视未被度量的工作
- 28.5 把程序员当人看
  - 程序员的日常时间分配
  - 性能差异和质量差异
    - 最顶尖的20%占全部产出的50%
    - 个体差异，好的坏的存在数量级的差异
    - 团队差异，个人产出满足2 8定律。接受高薪聘请顶尖人员
  - 信仰问题
  - 物理环境
    - 环境好坏会影响效率
- 28.6 管理你的管理者
  - 要诀在于要使你的管理者认为他任然在管理你
  - 把希望做什么的念头先藏起来，等着你的管理者组织一场有关你希望什么的管理风暴
  - 把做事情的的正确方法传授给你的管理者，这是一项需要持之以恒的工作，因为管理人员经常会提升、调迁或者解聘，
  - 关注你管理者的兴趣，按照他的（真正）意图来做。而不要用一些不必要的实现细节来分散其注意力（请把它设想成是对你工作的一种封装）
  - 拒绝按照你的管理者所说的去做，坚持用正确的方法做自己的事
  - 换工作

chp29 集成

- 29.1 集成方式的重要性
  - 方便诊断、缺陷少
  - 花费更少的时间获得第一个能工作的产品
  - 增加项目完成的机会
  - 更可靠的评估进度表

- 29.2 集成频率
  - 阶段式集成
    - 1 设计编码测试、调试各个类（单元开发）
    - 2 将这些类组合成一个庞大的系统（系统集成）
    - 3 测试并调整整个系统。（系统瓦解）
    - 缺点：必须到项目后期，所有类测试完成之后才进行集成，容易导致大量错误的同时产生。不利于程序员有条理的处理错误
  - 增量集成
    - 一小块一小块的编写并测试你的程序，然后一次一块的将他们拼接起来。
    - 1 开发一个小的系统功能部件，测试后。作为骨架，后续逐渐丰富。
    - 2 设计、编码、测试、调试某个类
    - 3 将新类集成到骨架上，测试调试使其能正常工作，在这之后重复步骤2
  - 增量集成的益处
    - 易于定位错误
    - 及早取得系统级成果
      - 提高士气
    - 改善对进度的监控
    - 改善客户的关系
    - 更加充分的测试系统中的每个单元
    - 缩短工期
      - 一部分设计和开发工作可以并行
- 29.3 增量集成的策略
  - 1 自顶向下集成
    - 确保类接口之间的定义交互明确
    - 可以提前暴露整体的设计上的问题
  - 2 自底向上集成
    - 先注重底层实现类，最后集成上层类
  - 3（推荐方式）三明治集成
    - 1 首先集成顶层类和广泛使用的底层类
    - 2 其次集成抽象中间类
  - 4 风险导向性的集成

- 5 功能导向的集成
- 6 T形集成

- 29.4 daily build和冒烟测试
  - 及时发现问题，形成良性循环
  - 工作成果可见，提高士气
  - 操作方法
    - 每日构建
    - 检查失败的build
    - 每天进行冒烟测试
    - 及时丰富冒烟测试case
    - 将每日构建和冒烟用例自动化
    - 即使有压力也好进行dailybuild和冒烟测试
  - daily
    - 哪些项目能用daily bulid
      - 项目越大，增量集成越重要
    - 持续集成
      - 频率适合是每天