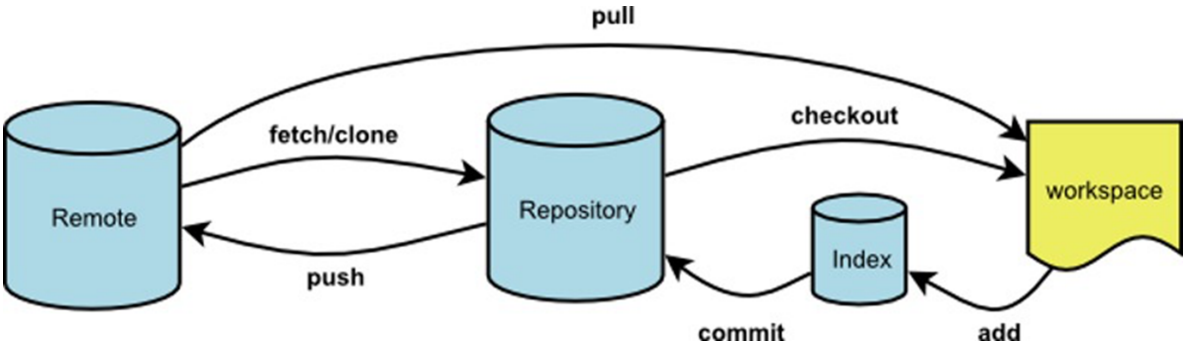
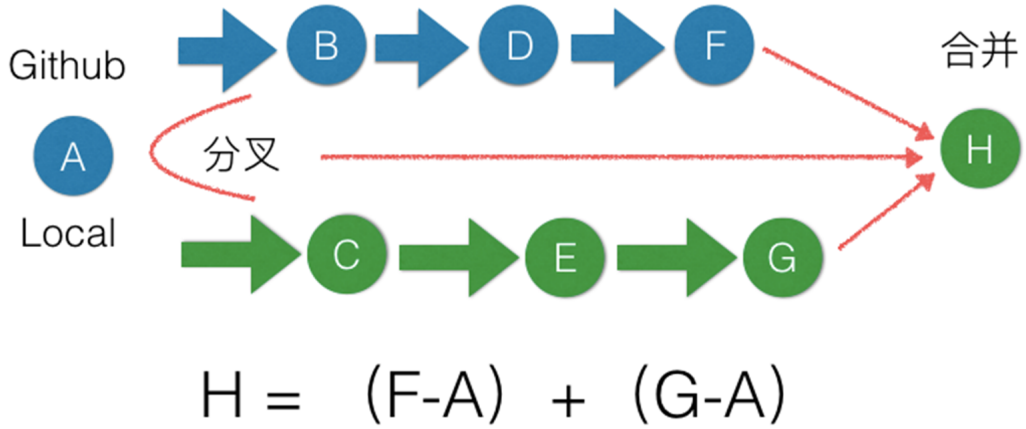
vscode命令

- 打开文件夹（ Ctrl/⌘+O）和关闭文件夹工作区（ Ctrl/⌘+K F）
- 新建文件（Ctrl/⌘+N）、关闭文件（Ctrl/⌘+W）、编辑文件和保存文件（Ctrl/⌘+S）
- 文件内搜索（Ctrl/⌘+F）
- 关闭所有文件（Ctrl/⌘+K W）
- 关闭已保存的文件（Ctrl/⌘+K U）
- Ctrl+/用于单行代码注释和取消注释，Alt+Shift+A用于代码块注释和取消注释。
- Ctrl/⌘+Shift+E 文件资源管理器
- Ctrl+Shift+G 源代码管理
- Ctrl/⌘+Shift+F 跨文件搜索
- Ctrl/⌘+Shift+D 启动和调试
- Ctrl/⌘+Shift+P查找并运行所有命令
- Ctrl/⌘+Shift+M查看错误和警告
- Ctrl/⌘+Shift+X 管理扩展插件
- Ctrl+`切换集成终端

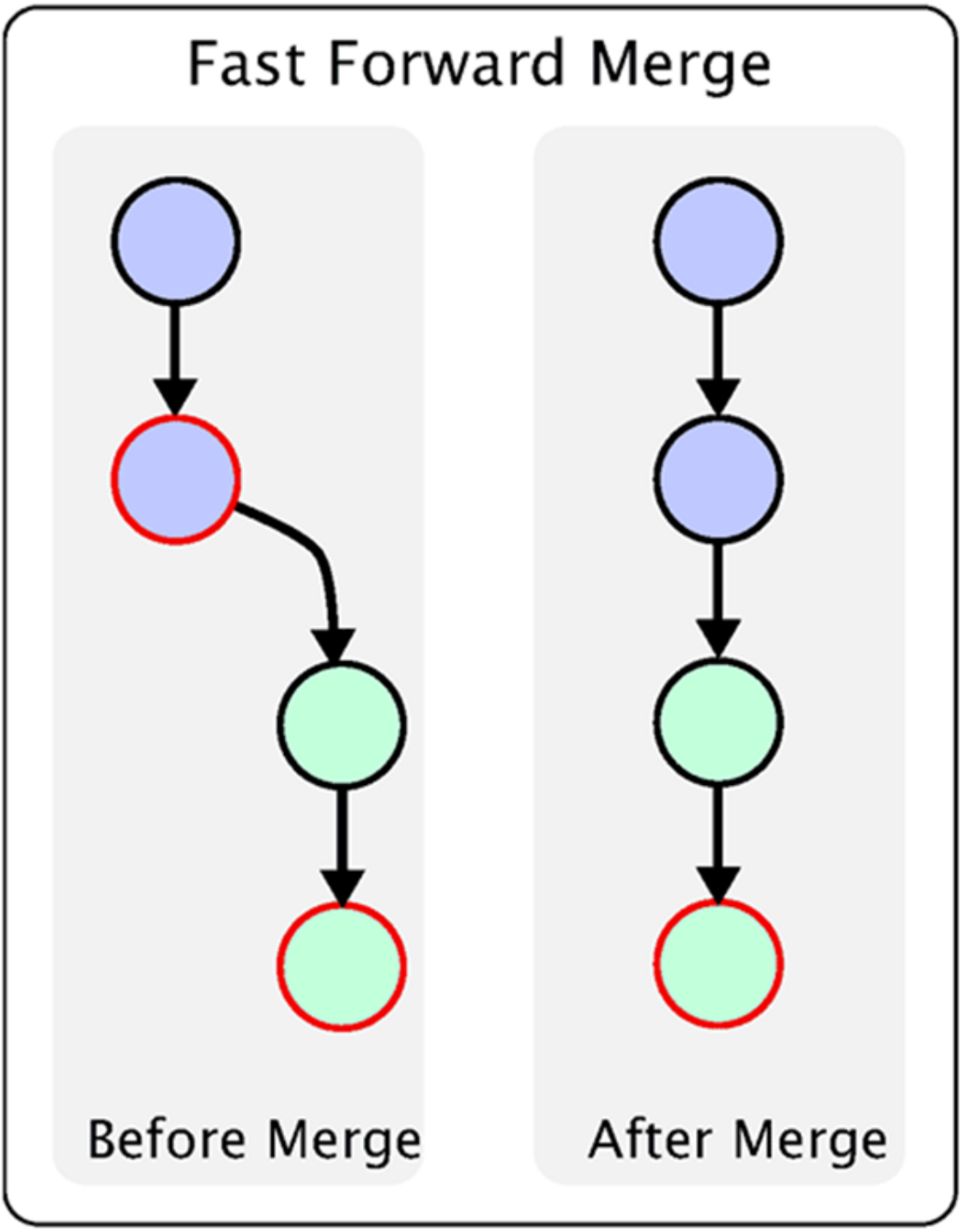
git命令

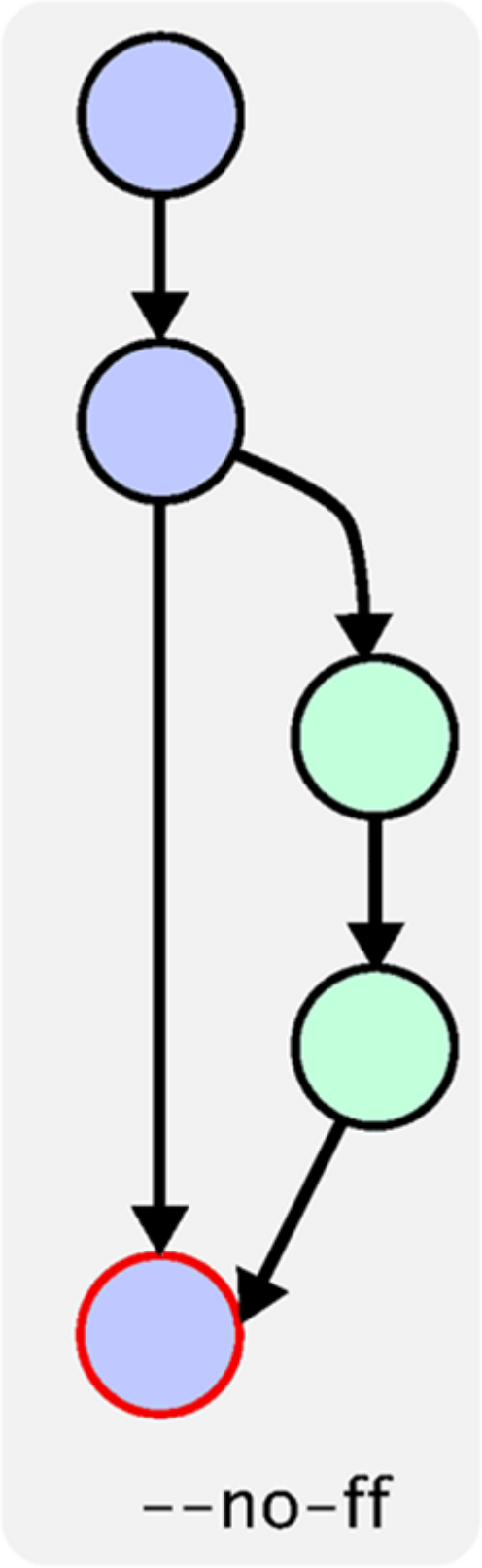
- git init # 在一个新建的目录下创建版本库
- git clone https://github.com/YOUR_NAME/REPO_NAME.git # 通过clone远端的版本库从而在本地创建一个版本库
- 

- git init # 初始化一个本地版本库
- git status # 查看当前工作区（workspace）的状态
- git add [FILES] # 把文件添加到暂存区（Index）
- git commit -m “wrote a commit log infro” # 把暂存区里的文件提交到仓库
- git log # 查看当前HEAD之前的提交记录，便于回到过去
- git reset —hard HEAD^^/HEAD~100/commit-id/commit-id的头几个字符 # 回退
- git reflog # 可以查看当前HEAD之后的提交记录，便于回到未来
- git reset —hard commit-id/commit-id的头几个字符 # 回退
- git clone命令官方的解释是“Clone a repository into a new directory”，即克隆一个存储库到一个新的目录下。
- git fetch命令官方的解释是“Download objects and refs from another repository”，即下载一个远程存储库数据对象等信息到本地存储库。
- git push命令官方的解释是“Update remote refs along with associated objects”，即将本地存储库的相关数据对象更新到远程存储库。

- git merge命令官方的解释是“Join two or more development histories together”，即合并两个或多个开发历史记录。
- git pull命令官方的解释是“Fetch from and integrate with another repository or a local branch”，即从其他存储库或分支抓取并合并到当前存储库的当前分支。
- 合并的概念:
- 

- 默认的合并方式为”快进式合并”（fast-farward merge）：


Fast Forward Merge
Before Merge
After Merge



- git checkout -b mybranch 创建新的分支
- git branch 切换分支
- git rebase -i [startpoint] [endpoint]

vim

我选择gg

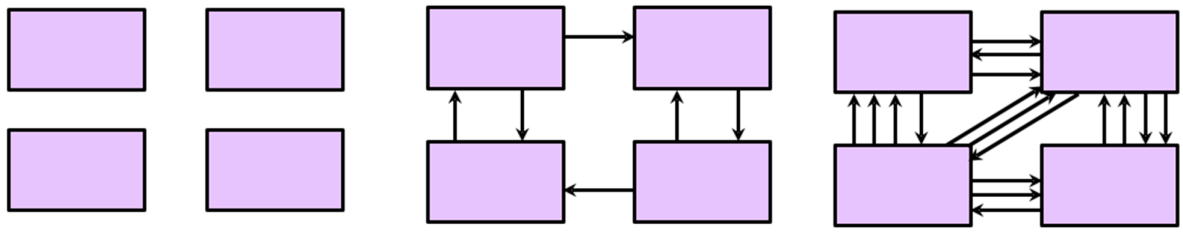
编写高质量代码的基本方法

1. 通过控制结构简化代码(if else/while/switch)
2. 通过数据结构简化代码
3. 一定要有错误处理
4. 注意性能优先的代价
5. 拒绝修修补补要不断重构代码

模块化

基本原理

模块化（Modularity）是在软件系统设计时保持系统内各部分相对独立，以便每一个部分可以被独立地进行设计和开发。这个做法背后的基本原理是关注点的分离 (SoC, Separation of Concerns) 分解成易解决的小问题,降低思考负担。
每个模块只有一个功能，易于开发，并且bug会集中在少数几个模块内，容易定位软件缺陷，也更加容易维护。
软件设计中的模块化程度便成为了软件设计有多好的一个重要指标，一般我们使用耦合度（Coupling）和内聚度（Cohesion）来衡量软件模块化的程度。

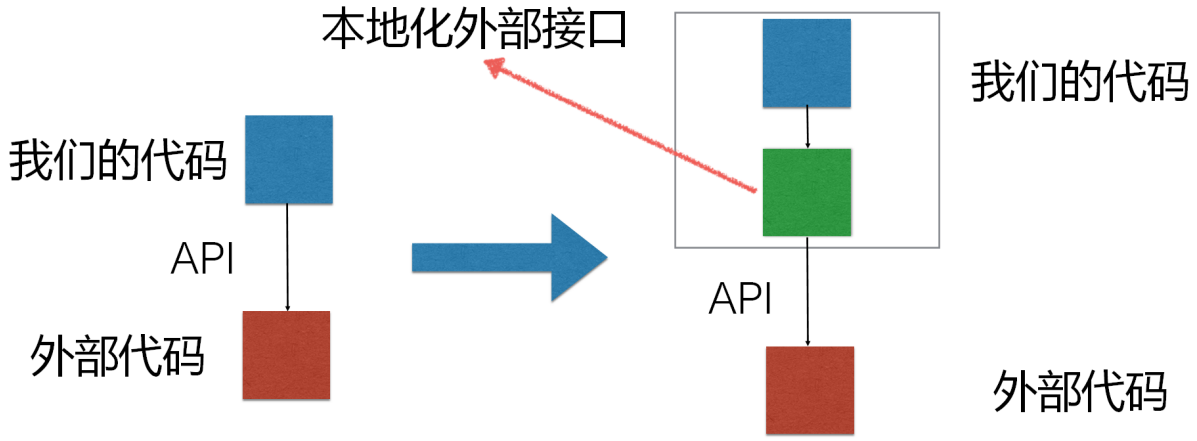


一般追求松散耦合。
内聚度是指一个软件模块内部各种元素之间互相依赖的紧密程度。
理想的内聚是功能内聚，也就是一个软件模块只做一件事，只完成一个主要功能点或者一个软件特性（Feather）。

基本方法

KISS（keep it simple&stupid）原则

- 一行代码只做一件事
- 一个块代码只做一件事
- 一个函数只做一件事
- 一个软件模块只做一件事
  使用本地化外部接口，本质上是一种设计模式，代理模式,即客户端通过代理间接地访问该对象，从而限制、增强或修改该对象的一些特性。能够有效降低模块与外部的耦合度。



先写伪代码的代码结构

接口

基本概念

接口就是互相联系的双方共同遵守的一种协议规范，在我们软件系统内部一般的接口方式是通过定义一组API函数来约定软件模块之间的沟通方式。
在面向过程的编程中，接口一般定义了数据结构及操作这些数据结构的函数；而在面向对象的编程中，接口是对象对外开放（public）的一组属性和方法的集合。函数或方法具体包括名称、参数和返回值等。

基本要素

1. 接口的目的
2. 接口的前置条件
3. 接口的协议规范（如http协议，png图片格式，json数据格式定义etc..）
4. 接口的后置条件
5. 接口的质量属性（如响应时间）

RESTful API

GET用来获取资源；
POST用来新建资源（也可以用于更新资源）；
PUT用来更新资源；
DELETE用来删除资源。

耦合方式

1. 公共耦合
   当软件模块之间 `共享数据区` 或 `变量名` 的软件模块之间即是公共耦合，显然两个软件模块之间的接口定义不是通过显式的调用方式，而是 `隐式` 的共享了共享了数据区或变量名。
2. 数据耦合
   在软件模块之间仅通过显式的调用传递 `基本数据类型` 即为数据耦合。

3. 标记耦合
   在软件模块之间仅通过显式的调用传递复杂的数据结构(结构化数据）即为标记耦合，这时数据的结构成为调用双方软件模块隐含的规格约定，因此耦合度要比数据耦合高。但相比公共耦合没有经过显式的调用传递数据的方式耦合度要低。

通用接口定义的基本方法

1. 参数化上下文(使用参数传递信息,不依赖上下文环境,即不使用闭包函数)
2. 移除前置条件(sum函数中使用数组传递参数,不再限定参数个数)
3. 简化后置条件(移除参数之间的关系,使sum返回的是数组全部元素的和)

软件质量的三个角度

1. 产品的角度，也就是软件产品本身内在的质量特点；
2. 用户的角度，也就是软件产品从外部来看是不是对用户有帮助，是不是有良好的用户体验；
3. 商业的角度，也就是商业环境下软件产品的商业价值，比如投资回报或开发软件产品的其他驱动因素。

需求

需求的类型

1. 功能需求：根据所需的活动描述所需的行为
2. 质量需求或非功能需求：描述软件必须具备的一些质量特性
3. 设计约束(设计约束): 设计决策，例如选择平台或接口组件
4. 过程约束(过程约束): 对可用于构建系统的技术或资源的限制

高质量需求

1. Making Requirements Testable(需求可测试)

   > Fit criteria(标准) form objective standards for judging whether a proposed solution satisfies the requirements
   >
   > 1. It is easy to set fit criteria for quantifyable(可量化的) requirements
   > 2. It is hard for subjective quality(质量) requirements
   >
   > Three ways to help make requirements testable
   >
   > 1. Specify a quantitave(定量) description for each adverb(副词) and adjective(动词)
   > 2. Replace pronouns(代词) with specific names of entities
   > 3. Make sure that every noun is defined in exactly one place in the requirements documents

2. Resolving Conflicts(解决冲突)

   > Different stakeholder has different set of requirements

> 1. potential conflicting ideas
>
> Need to prioritize(优先次序) requirements
> Prioritization might separate requirements into three categories
>
> 1. essential: absolutely must be met
> 2. desirable: highly desirable but not necessary
> 3. optional: possible but could be eliminated

3. Characteristics of Requirements(需求很有特点)

> Correct
> Consistent
> Unambigious无二义性
> Complete
> Feasible可行
> Relevant无与主要目标不相关的需求
> Testable
> Traceable可追溯

需求分析的两种方法

1. 原型化方法
   原型化方法可以很好地整理出用户接口方式（UI，User Interface），比如界面布局和交互操作过程。
2. 建模的方法
   建模的方法可以快速给出有关事件发生顺序或活动同步约束的问题，能够在逻辑上形成模型来整顿繁杂的需求细节。

用例

基本概念

用例（Use Case）的核心概念中首先它是一个业务过程（business process），经过逻辑整理抽象出来的一个业务过程，这是用例的实质。什么是业务过程？在待开发软件所处的业务领域内完成特定业务任务（business task）的一系列活动就是业务过程。

四个必要条件

必要条件一 :它是不是一个业务过程?
必要条件二:它是不是由某个参与者触发开始?
必要条件三:它是不是显式地或隐式地终止于某个参与者?
必要条件四: 它是不是为某个参与者完成了有用的业务工作?

三个抽象层级

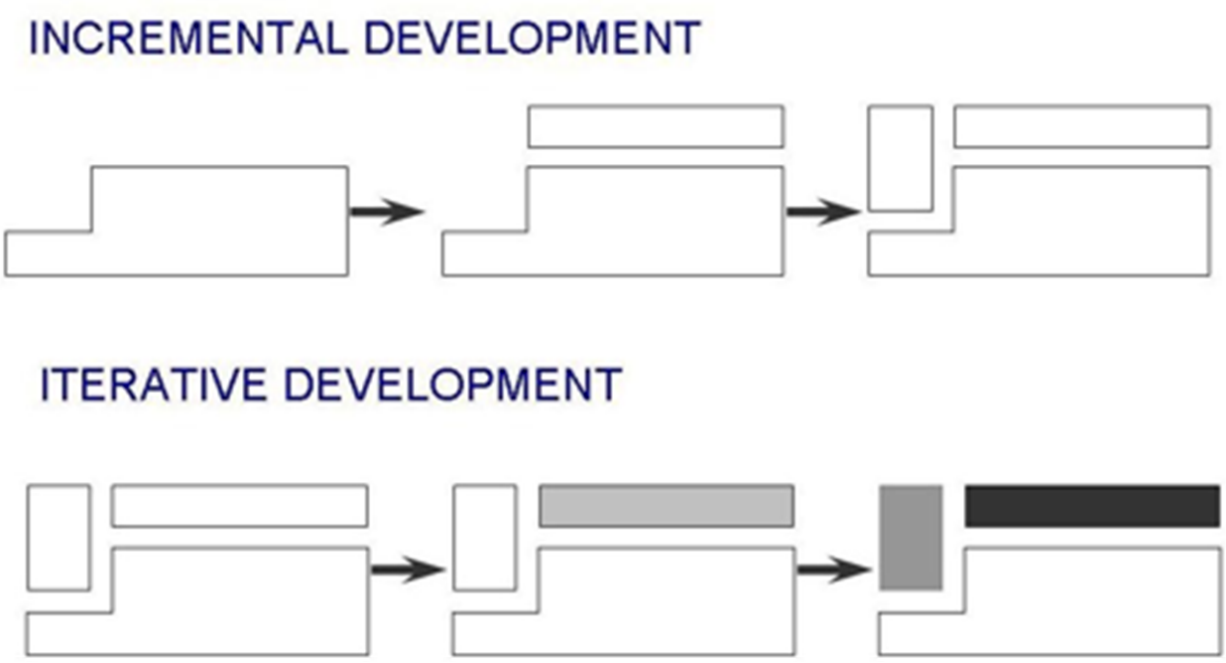
1. 抽象用例（Abstract use case）。只要用一个干什么、做什么或完成什么业务任务的动名词短语，就可以非常精简地指明一个用例。

2. 高层用例（High level use case）。需要给用例的范围划定一个边界，也就是用例在什么时候什么地方开始，以及在什么时候什么地方结束；
3. 扩展用例（Expanded use case）。需要将参与者和待开发软件系统为了完成用例所规定的业务任务的交互过程一步一步详细地描述出来，一般我们使用一个两列的表格将参与者和待开发软件系统之间从用例开始到用例结束的所有交互步骤都列举出来。
   扩展用例最后可以用两列表格描述。

统一过程

核心要义

统一过程（UP，Unified Process）的核心要义是`用例驱动`（Use case driven）、`以架构为中心`（Architecture centric）、`增量且迭代`（Incremental and Iterative）的过程。用例驱动就是我们前文中用例建模得到的用例作为驱动软件开发的目标；以架构为中心的架构是后续软件设计的结果，就是保持软件架构相对稳定，减小软件架构层面的重构造成的混乱；增量且迭代体现在下图中。



敏捷统一过程的四个关键步骤

第一，确定需求；
第二，通过用例的方式来满足这些需求；
第三，分配这些用例到各增量阶段；
第四，具体完成各增量阶段所计划的任务。
显然，第一到第三步主要是计划阶段的工作，第四步是接下来要进一步详述的增量阶段的工作。
在每一次增量阶段的迭代过程中，都要进行从需求分析到软件设计实现的过程，具体敏捷统一过程将增量阶段分为五个步骤：
• 用例建模（Use case modeling）；
• 业务领域建模（Domain modeling）；
• 对象交互建模（Object Interaction modeling）,使用剧情描述来建模,最后转换为剧情描述表；
• 形成设计类图（design class diagram）；
• 软件的编码实现和软件应用部署；

三种系统类型

S系统：有规范定义，可从规范派生

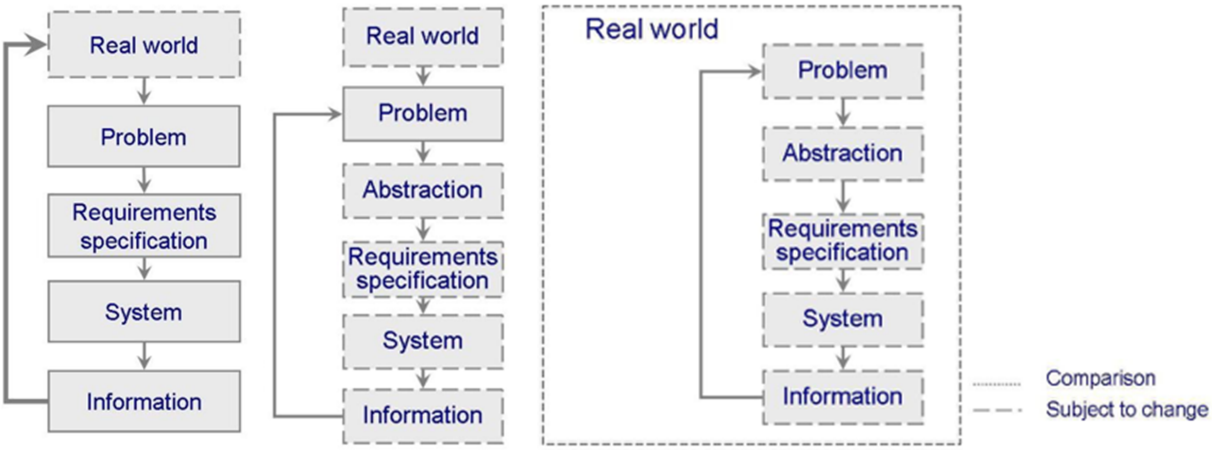
- 矩阵操纵矩阵运算

P系统：需求基于问题的近似解，但现实世界保持稳定

- 象棋程序

E系统：嵌入现实世界并随着世界的变化而变化(大多数软件都属于这个类型)

- 预测经济运行方式的软件（但经济尚未被完全理解）



软件具有复杂性和易变性,从而难以达成概念的完整性与一致性。(需求的达成永远赶不上需求的变化)

设计模式

设计模式的本质是面向对象设计原则的实际运用总结出的经验模型。目的是包容变化,即通过使用设计模式和多态等特殊机制,将变化的部分和不变的部分进行适当隔离。(高内聚,低耦合)
正确使用设计模式具有以下优点:

- 可以提高程序员的思维能力、编程能力和设计能力。
- 使程序设计更加标准化、代码编制更加工程化，使软件开发效率大大提高，从而缩短软件的开发周期。
- 使设计的代码可重用性高、可读性强、可靠性高、灵活性好、可维护性强。

设计模式由四个部分组成:

- 该设计模式的名称；
- 该设计模式的目的，即该设计模式要解决什么样的问题；
- 该设计模式的解决方案；
- 该设计模式的解决方案有哪些约束和限制条件。

根据作用对象分两类:

- 类模式:用于处理类与子类之间的关系，这些关系通过继承来建立，是静态的，在编译时刻便确定下来了。比如模板方法模式等属于类模式。
- 对象模式:用于处理对象之间的关系，这些关系可以通过组合或聚合来实现，在运行时刻是可以变化的，更具动态性。由于组合关系或聚合关系比继承关系耦合度低，因此多数设计模式都是对象模式。

七大原则

1. 单一职责原则 (Single Responsibility Principle)
2. 开放-关闭原则 (Open-Closed Principle)
3. 里氏替换原则 (Liskov Substitution Principle)
4. 依赖倒转原则 (Dependence Inversion Principle)

   > 高层模块不应该依赖低层模块，两者都应该依赖其抽象；抽象不应该依赖细节，细节应该依赖抽象
   > 由于在软件设计中，细节具有多变性，而抽象层则相对稳定，因此以抽象为基础搭建起来的架构要比以细节为基础搭建起来的架构要稳定得多。
   > 依赖倒置原则在模块化设计中降低模块之间的耦合度和加强模块的抽象封装提高模块的内聚度上具有普遍的指导意义

5. 接口隔离原则 (Interface Segregation Principle)不考
6. 迪米特法则（Law Of Demeter）
7. 组合/聚合复用原则 (Composite/Aggregate Reuse Principle)

MVC与MVVM

主要区别在于,MVC中,用户对于M的操作是通过C传递的,然后C将改变传给V,并且M将在发生变化时通知V,然后V通过C获取变化;在MVVM中,用户直接与V交互,通过VM将变化传递给M,然后M改变之后通过VM将数据传递给V,从而实现解耦。
另一个区别是,当M的数据需要进行解析后V才能使用时,C若承担解析的任务,就会变得很臃肿;在MVVM中,VM层承担了数据解析的工作,这时C就只需要持有VM,而不需要直接持有M了。从而完成了数据的解耦。

软件架构复用

1. 克隆,完整地借鉴相似项目的设计方案，甚至代码，只需要完成一些细枝末节处的修改适配工作。
2. 重构,构建软件架构模型的基本方法，通过指引我们如何进行系统分解，并在参考已有的软件架构模型的基础上逐步形成系统软件架构的一种基本建模方法。

几种架构的分解方法

- 面向功能的分解方法，用例建模即是一种面向功能的分解方法；
- 面向特征的分解方法，根据数量众多的某种系统显著特征在不同抽象层次上划分模块的方法；

- 面向数据的分解方法，在业务领域建模中形成概念业务数据模型即应用了面向数据的分解方法；
- 面向并发的分解方法，在一些系统中具有多种并发任务的特点，那么我们可以将系统分解到不同的并发任务中（进程或线程），并描述并发任务的时序交互过程；
- 面向事件的分解方法，当系统中需要处理大量的事件，而且往往事件会触发复杂的状态转换关系，这时系统就要考虑面向事件的分解方法，并内在状态转换关系进行清晰的描述；
- 面向对象的分解方法，是一种通用的分析设计范式，是基于系统中抽象的对象元素在不同抽象层次上分解的系统的方法。

架构的描述方法

- 分解视图 Decomposition View
- 依赖视图 Dependencies View
- 泛化视图 Generalization View
- 执行视图 Execution View
- 实现视图 Implementation View
- 部署视图 Deployment View
- 工作任务分配视图 Work-assignment View

没有银弹的含义

“在10年内无法找到解决软件危机的杀手锏（银弹）。
软件中的根本困难，即软件概念结构(conceptual structure)的复杂性，无法达成软件概念的完整性和一致性,自然无法从根本上解决软件危机带来的困境。

软件的生命周期

分析、设计、实现、交付和维护五个阶段

1. 分析阶段的任务是需求分析和定义,分析阶段一般会在深入理解业务的情况下，形成业务概念原型
2. 设计阶段分为软件架构设计和软件详细设计，前者一般和分析阶段联系紧密，一般合称为“分析与设计”；后者一般和实现阶段联系紧密，一般合称为“设计与实现”。
3. 实现阶段分为编码和测试，其中测试又涉及到单元测试、集成测试、系统测试等。
4. 交付阶段主要是部署、交付测试和用户培训等。
5. 维护阶段一般是软件生命周期中持续时间最长的一个阶段，而且在维护阶段很可能会形成单独的项目，从而经历分析、设计、实现、交付几个阶段，最终又合并进维护阶段。

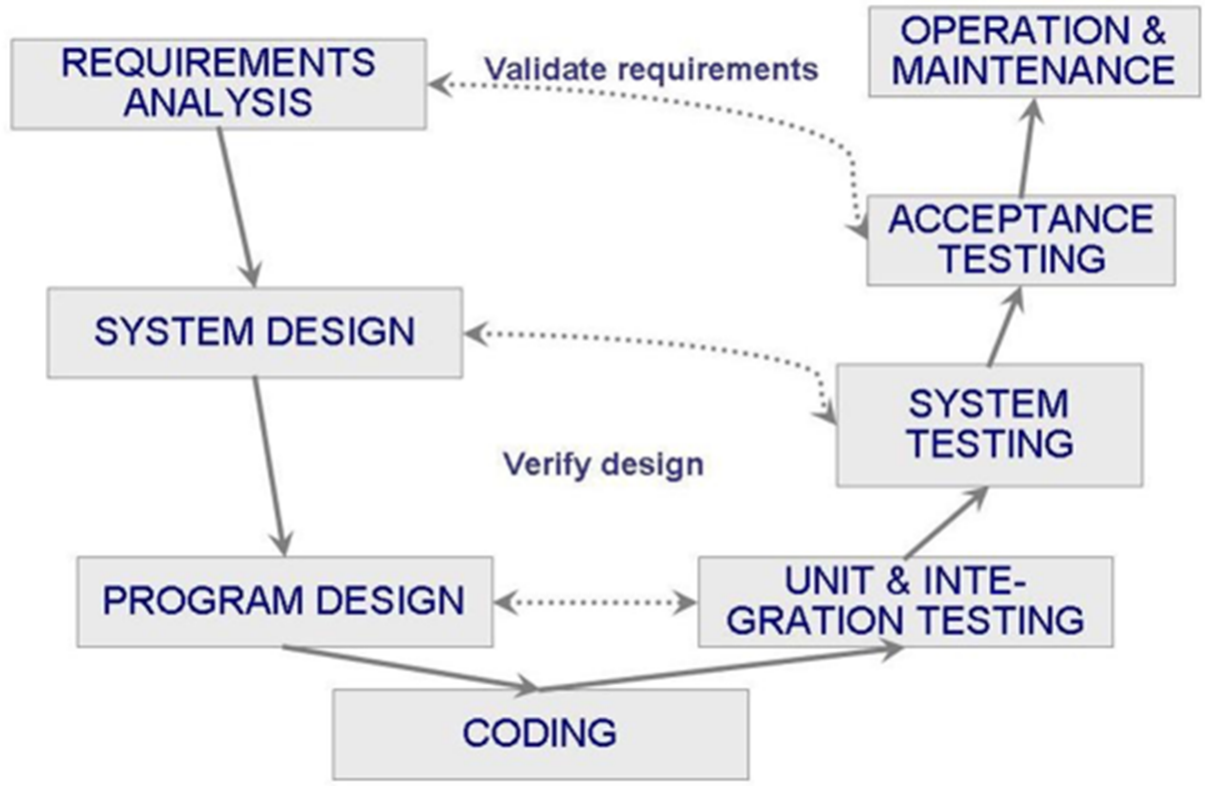
软件过程

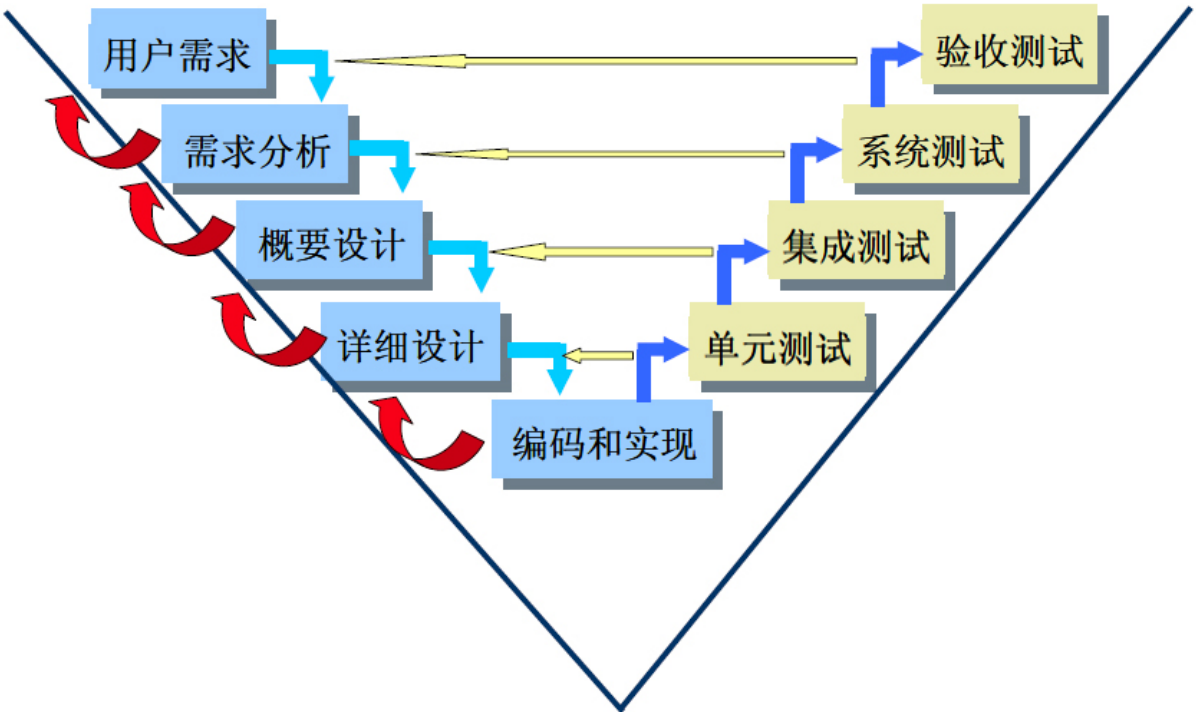
软件过程又分为描述性的（descriptive）过程和说明性的（prescriptive）过程。
描述性的过程试图客观陈述在软件开发过程中实际发生什么。
说明性的过程试图主观陈述在软件开发过程中应该会发生什么。
采用不同的过程模型时应该能反映出要达到的过程目标，比如构建高质量软件、早发现缺陷、满足

预算和日程约束等。不同的模型适用于不同的情况，我们常见的过程模型，比如瀑布模型、V模型、原型化模型等都有它们所能达到的过程目标和适用的情况。

V模型

V模型也是在瀑布模型基础上发展出来的，我们发现单元测试、集成测试和系统测试是为了在不同层面验证设计，而交付测试则是确认需求是否得到满足。也就是瀑布模型中前后两端的过程活动具有内在的紧密联系，如果将模块化设计的思想拿到软件开发过程活动的组织中来，可以发现通过将瀑布模型前后两端的过程活动结合起来，可以提高过程活动的内聚度，从而改善软件开发效率。这就是V模型。
V模型是开始一个特定过程活动和评估该特定过程的过程活动成对出现，从而便于软件开发过程的组织和管理。

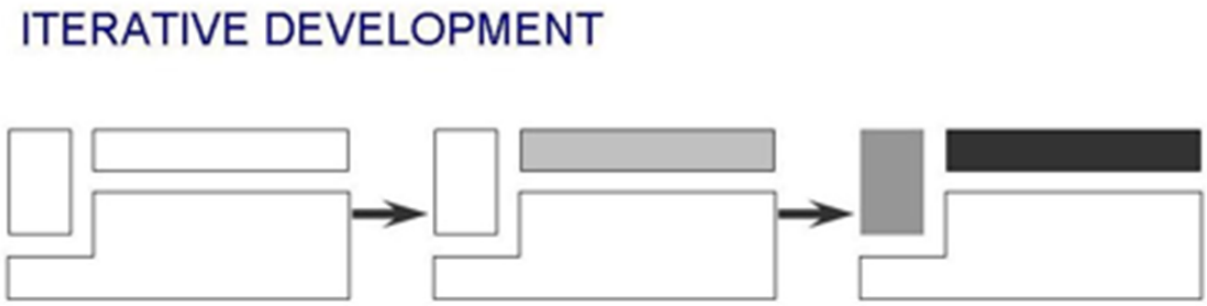




分阶段开发

分阶段开发的交付策略分为两种，一是增量开发（Incremental development），二是迭代开发（Iterative development）。

- 增量开发就是从一个功能子系统开始交付，每次交付会增加一些功能，这样逐步扩展功能最终完成整个系统功能的开发。
- 迭代开发是首先完成一个完整的系统或者完整系统的框架，然后每次交付会升级其中的某个功能子系统，这样反复迭代逐步细化最终完成系统开发。



持续集成持续交付。

团队

基本要素

1. 团队的规模
2. 团队的凝聚力
3. 团队协作的基本条件

评价团队的方法

CMM/CMMI用于评价软件生产能力并帮助其改善软件质量的方法，成为了评估软件能力与成熟度的一套标准，它侧重于软件开发过程的管理及工程能力的提高与评估，是国际软件业的质量管理标准。
CMMI共有5个级别，代表软件团队能力成熟度的5个等级，数字越大，成熟度越高，高成熟度等级表示有比较强的软件综合开发能力。

- 一级,初始级,软件组织对项目的目标与要做的努力很清晰，项目的目标可以实现。但主要取决于实施人员。
- 二级,管理级,软件组织在项目实施上能够遵守既定的计划与流程，有资源准备，权责到人，对项目相关的实施人员进行了相应的培训，对整个流程进行监测与控制，并联合上级单位对项目与流程进行审查。这级能保证项目的成功率。
- 三级,已定义级,软件组织能够根据自身的特殊情况及自己的标准流程，将这套管理体系与流程予以制度化。科学管理成为软件组织的文化与财富。
- 四级,量化管理级,软件组织的项目管理实现了数字化，降低了项目实施在质量上的波动。
- 五级,持续优化级，软件组织能够充分利用信息资料，对软件组织在项目实施的过程中可能出现的问题予以预防。能够主动地改善流程，运用新技术，实现流程的优化。

敏捷方法的敏捷宣言

- 个体和互动 高于 流程和工具
- 工作的软件 高于 详尽的文档
- 客户合作 高于 合同谈判
- 响应变化 高于 遵循计划
  尽管右项有其价值，我们更重视左项的价值。