构件/组件:系统的逻辑与功能结构单元.构件是一种抽象概念表述。是对以各种具体实现技术的系统结构组成元素的统称.

连接件: 构件间相互交互的机制与规则.机制是指连接件的具体实现形式.规则是指构件使用连接件应遵循的规范。配置:软件体系结构的配置是系统逻辑拓扑结构的描述

逻辑视图:基于功能需求抽象,刻画系统的静态结构模型

进程视图:刻画系统的运行时的结构模型

开发视图:考虑开发技术,过程与组织,刻画系统的开发管理结构模型

物理视图:逻辑视图中的各功能构件在安装部署,环境中的映射,刻画系统的安装部署结构模型。

场景视图:从系统使用的角度对系统结构的描述.它反映的是在完成一个系统功能时，系统各功能构件间的交互关系。

管道-过滤器:为数据流系统提供了一种架构风格模板,过滤器是数据处理构件,管道是过滤器间的连接件.应用场景:处理戒者转换输入数据流.对数据的处理可以容易地分成几个处理步骤.系统的升级要求可以通过替换/增加/重组处理步骤实现.不同的处理步骤间不共享信息.工作方式:过滤器从前一个过滤器戒数据源中拉出数据.过滤器把输出数据压入后一个过滤器.过滤器以循环的方式工作，从前一个过滤器戒数据源拉出输入数据并将其输出数据压入下一个过滤器.

基于事件的隐式调用风格:采用离散,异步的事件机制实现系统的构件间的松散间接耦合的交互.应用场景:适用于异步、并发性的系统.对事件的处理顺序无要求的系统.事件的处理要求具有很好的灵活性.非集中式控制的软件系统,优:系统具有很好的灵活性，系统易于伸缩扩展 缺:系统控制权的问题,数据的交换问题,事件处理顺序控制问题

分层体系结构风格: 将系统分成适当层次,按适当次序放置.从最低抽象层次开始,以梯状把抽象层次n放在n-1层顶部,直到功能顶部.第n层提供的绝大多数服务由第n-1层提供的服务组成.每个层次是一个独立的组件.不允许较高层次直接越级访问较低层次.应用场景:一个需要分解的大系统,系统规格说明描述了高层任务，并希望可移植性,高层任务到平台的映射不是直接的,系统需要满足一些非功能性特性：

Repository风格:以数据为中心的体系结构风格.被动仓库主动仓库. 优:便于多构件间共享大量数据,而不必关心数据是何时产生的、由谁提供的以及通过何种途径来提供.便于将新的构件作为知识源添加到系统中来 缺:共享数据结构的修改变得非常困难.需要同步机制和加锁机制来保证共享数据的完整性和一致性,增大了系统设计的复杂度

Factory Method:为某一个产品族对象的创建，定义统一的Factory Method接口，并由子类具体实现对象创建. 缺:需要Creator和其相应的子类作为factory method的定义与实现. 优:a为一组相关或相似对象的创建提供了统一的创建接口b将产品族对象不使用者之间解耦

Abstract Factory: 解决多个产品族对象的创建工作,与门定义一个用于创建这些对象的接口.客户只需不这个基接口打交道,不必考虑实体类的类型. 优:Concrete Factory中的工厂方法把product对象创建封装起来，并将具体的产品类不Client分离.• 保证产品族对象创建接口的一致性 缺点: Concrete Factory对象的工厂方法数目对应product种类数目，增加新的product种类比较困难，要影响到factory的基类，进而影响到所有的子类

Prototype:以一个已有的对象作为原型,通过它来创建新的对象,在增加新的对象的时候,新对象的细节创建工作由自己来负责,从而使新对象的创建过程不Client隔离开来.应用场景:当产品的创建过程要独立于系统时,当产品的类型是在runtime时被指定的情况下,避免创建一个不product层次平行的factory层次时,当产品类的实例只能是几种确定的丌同实例状态中的一种

Singleton:确保某个类只有一个对象实例.控制类的实例化对象数目. 提供了一种全局化的单一对象实例设计方法

Adapter:将一个类的接口转换成Client所期望的接口形式.使用C++继承机制实现class adapter.使用内嵌对象技术实现object adapter. Pluggable adapters三种实现方案:a使用抽象方法定义b 使用代理对象c参数化技术

Composite: 在绘图编辑器和图形捕捉系统这样的图形应用程序中，用户可以使用简单的图元创建复杂的图表。用户可以组合多个简单图元以形成一些较大的图表，这些图表又可以组合成更大的图表。一个简单的实现方法是为Text和Line这样的图元定义一些类，另外定义一些类作为这些图元的容器类. 然而这种方法存在一个问题,使用这些类的代码必须区别对待图元对象不容器对象，而实际上大多数情况下用户认为它们是一样的。对这些类区别使用，使得程序更加复杂

Command: 封装对象间交互调用的请求信息和调用请求执行过程,framework往往需要把调用请求与处理调用请求的对象分开,command模式可以把发出调用的对象与调用的目标对象分开,允许通过多种途径调用同一个

Observer: 请求支持对象间1对多的交互调用方式,有些算法对于某些类是必不可少的，但是不适合于硬编进类中。客户可能需要算法的多种不同实现，允许增加新的算法实现戒者改变现有的算法实现, 我们可以把这样的算法封装到单独的类中，称为strategy