一、 软件开发模型

1. 瀑布模型：结构化方法。开发阶段性、需求明确、文档齐全、风险控制弱。前一步的错误会延申到后一步；
2. 原型开发模型：迭代方法。有两种开发方式，分别是原先开发和目标软件开发；需求不明确；
3. 螺旋模型：迭代方法。在原型开发模型和瀑布模型基础上产生的，适合大型的、复杂的、有风险的项目；
4. 喷泉模型：面向对象方法；复用性好；开发过程无间隙、节省时间；
5. V 模型：开发与测试结合；
6. 智能模型：基于规则系统的专家系统；
7. 变换模型：适用于形式化开发；
8. 快速开发模型（RAD）：基于构件的开发方法，用户参与、开发或复用构件、模块化要求高，不适合新技术；
9. 统一开发方法（RUP）：用例驱动、架构为中心、迭代、增量；
10. 可重用构建模型：基于构件的开发方法。开发或复用构件；
11. 敏捷开发：以人为本，与用户紧密协作，面对面沟通，尽早发布增量，小而自主的开发团队。适用于规模小的项目。

XP	高效、低风险、测试先行
Cockburn 水晶	强调不同的项目，采用不同策略
SCRUM 并列争球	迭代。30 天为一个迭代周期，按照需求优先级实现
FDD 功能驱动	将开发人员分类。分为指挥者（首席程序员）、类程序员
开放式源码	采用虚拟团队，开发成员分布各地
ASD 自适应	采用预测-协作-学习的方式进行开发

二、 软件设计模式

1. 创建型：单抽原件厂；用来创建对象。
2. 结构型：外侨组员戴配饰；处理类或对象的组合。
3. 行为型：观摩对策、责令解放、戒忘台；描述类与对象之间怎样交互、怎样分配职责。

英文

Singleton、Abstract Factory、Builder、Factory Method、Prototype、Adapter、Bridge、Composite、Decorator、Façade、Flyweight、Proxy、Chain of Responsi bility、Iterator、Mediator、Memento、Observer、State、Strategy、Command、Interpreter、Template Method、Visitor

(一) 创建型

单例模式	Singleton	每次只实例化一个对象（唯一实例）
抽象工厂模式	Abstract Factory	抽象接口
原型模式	Prototype	复用、拷贝原型
建造者模式	Builder	类与构造分离
工厂模式	Factory Method	子类实例化对象

(二) 结构型

外观模式	Façade	对外统一接口
桥接模式	Bridge	抽象实现分离
组合模式	Composite	部分与整体的关系、树形结构
享元模式	Flyweight	细粒度复用、共享
代理模式	Proxy	代理控制
适配器模式	Adapter	接口转换、兼容
装饰器模式	Decorator	附加职责

(三) 行为型

观察者模式	Observer	通知、自动更新
模板方法	Template Method	算法结构不变，子类定义步骤
迭代模式	Iterator	顺序访问
策略模式	Strategy	算法替换
责任链模式	Chain of Responsi bility	传递请求、职责、链接
命令模式	Command	参数化、日志记录
解释器模式	Interpreter	文法、解释
访问者模式	Visitor	类不变、新操作
中介者模式	Mediator	不直接引用
备忘录模式	Memento	保存、恢复
状态模式	State	状态变成类

三、 软件架构风格

定义：软件体系结构风格是描述某一特定应用领域中系统组织方式的惯用模式。

常用的软件架构风格有：数据流风格、调用返回风格、独立构件风格、虚拟机、仓库风格、过程控制、C2 风格。

数据流风格	批处理序列	构件读入-内部处理-输出。以整体为单位。
	管道/过滤器	构件读入-内部处理-输出。前一个输出是后一个的输入。传统编译器。
调用返回风格	主程序/子程序	主程序直接调用子程序。显示调用。
	面向对象	构件是对象，通过函数和过程的调用来交互。
	层次结构风格	分层，层次越多效率越差。
独立构件风格	进程通信	独立消息传递有点对点、异步或同步方式，以及远程过程（方法）调用。
	事件驱动程序	隐式调用，通过事件驱动。程序的语法高亮、语法错误提示。
虚拟机风格	解释器	自定义、灵活。解释引擎、被解释的代码的存储区、记录工作状态、进度的数据结构。
	基于规则的系统	自定义、灵活。规则集、规则解释器、规则/数据选择器和工作状态、进度的数据结构。
仓库风格	数据库系统	中央共享数据源、独立处理单元。构件控制中央共享数据。
	黑板系统	知识源、黑板和控制。语音识别、信号处理。中央数据结构状态触发进程。
	超文本系统	网状链接、互联网领域。
过程（闭环）控制		发出命令并接收反馈，设定参数不断调整。空调调温、汽车巡航定速。
C2		连接件与构件都有顶部、底部。绑定连接件按规则运作。

四、 软件架构评估

定义：

(四) 质量属性

常用的质量属性有：性能、可用性、可修改性、安全性

性能	**单位时间内能处理的工作量。（处理任务所需时间或单位时间内的处理量）**
可用性	**两次故障之间间隔的时间比。（正常运行的时间比例）**
可修改性	**能够快速的以较高的性价比对系统作出修改的能力。**
安全性	**系统给合法用户提供服务并阻止非法用户的能力。**
可靠性	容错：出现错误后仍能保证系统正常运行，且自行修正错误； 健壮性：错误不对系统产生影响，按既定程序忽略错误。
功能性	需求的满足程度
可变性	原体系结构变更为新体系结构
互操作性	与其他系统互操作

主要质量属性的设计策略

性能	增加计算资源、资源调度、优先级队列、减少计算开销、引入并发机制
可用性	Ping/echo、心跳、冗余、进程监视器
可修改性	抽象、信息隐藏、限制通信路径、运行时注册
安全性	追踪/审计、用户授权、用户认证、限制访问

(五) 敏感点、风险点、权衡点、非风险点

风险点	某些做法可能有些隐患会导致一些问题。
敏感点	为了实现某种特定的质量属性，一个或多个系统组件所具有的特性。
权衡点	是多个质量属性的敏感点。
非风险点	某些做法是可行的、可接受的。

(六) 特定领域软件体系结构 DSSA

特定领域软件架构是在一个特定应用领域中，为一组应用提供组织结构参考的标准软件体系结构。DSSA 通常是一个具有三个层次的系统模型，包括领域开发环境（领域架构师）、领域特定应用开发环境（应用工程师）和应用执行环境（操作员）。DSSA 的活动如下：

领域分析	获得领域模型。领域模型描述领域中系统之间共同的需求，即领域需求。
领域设计	获得特定领域软件架构。DSSA 描述领域模型中表示需求的解决方案。
领域实现	依据领域模型和 DSSA 开发和组织可重用信息，并对基础软件架构进行实现。

1. 架构评估方法

基于技术的手段来看，分为三类：基于调查问卷或检查表的方式、基于场景的方式和基于度量的方式。

(七) 基于调查问卷或检查表的方式

该方式的关键是要设计好问卷或检查表，它充分利用系统相关人员的经验和知识，获得对架构的评估。其缺点是在很大程度上依赖于评估人员的主观推断。

(八) 基于场景的方式

它是通过分析软件架构对场景的支持程度，从而判断该架构对这一场景所代表的质量需求的满足程度。

a) ATAM（架构权衡分析法）

是一种系统架构评估方法，主要在系统开发之前，针对性能、可用性、安全性和可修改性等质量属性进行评价和折中。ATAM 可以分为 4 个主要的活动阶段，包括需求收集、架构视图描述、属性模型构造和分析、架构决策与折中，整个评估过程强调以属性作为架构评估的核心概念。

b)SAAM（软件架构分析方法）

SAAM 是最早形成文档并得到广泛应用的软件架构分析方法。SAAM 的主要输入是问题描述、需求说明和架构描述，其分析过程主要包括场景开发、架构描述、单个场景评估、场景交互和总体评估。

(九) 基于度量的方式

制定一些定量值来度量架构，如代码行数等。要制定质量属性和度量结果之间的映射。

五、 微服务

(一) 微服务优点

1. 每个微服务都很小，这样能聚焦一个指定的业务功能或业务需求。
2. 微服务能够被小团队单独开发，这个小团队是 2 到 5 人的开发人员组成。
3. 微服务是松耦合的，是有功能意义的服务，无论是在开发阶段或部署阶段都是独立的。
4. 微服务能使用不同的语言开发。
5. 去中心化。每个微服务都有自己的存储能力，可以有自己的数据库。也可以有统一数据库。

(二) 微服务缺点

1. 很难在不采用分布式事务的情况下跨服务实现功能
2. 测试工作更加困难
3. 跨服务实现要求功能要求团队之间的紧密协作
4. 部署复杂

(三) 在微服务中，应用网关 API 的作用

1. 提供统一入口
2. 可以进行权限身份认证等安全管理
3. 可以根据流量进行限流
4. 数据缓存
5. 性能监控等
6. 异常重试
7. 服务降级