名词解释、填空、选择题知识点

1. 软件体系结构是软件在设计构成上的基本、可供设计选择的形态和总体结构。
2. 软件体系结构定义了软件的局部和总体计算部件的构成，以及这些部件之间的相互作用关系。
3. 体系结构是由结构和功能各异、相互作用的部件集合，按照层次构成的。
4. 体系结构 = 组件 + 连接件 + 约束
5. 体系结构的类别：概要型、需求型、设计型
6. 软件设计的层次：结构级、代码级、执行级。
7. 实现体系结构指导下的软件工程设计，关键就是要建立软件结构的分析和构造方法。
8. 软件复用就是具有特定性能的模式或部件在不同应用中的多次频繁使用。
9. 软件的体系结构应该是一个关于软件构成方面的具有层次性的知识体系。

   宏观的软件层次（高、中、低）

   软件的硬件层、基础控制描述层、资源及管理调度层、系统结构模式层、领域应用层
10. 软件是存储、通信、UI 和业务逻辑的紧密结合体。
11. 软件即抽象，抽象是软件的本质。抽象的本质在于对人的简单性，远离具体问题。
12. 部件描述有 3 个方面：计算功能、结构特性、其他特性。
13. 体系结构描述语言 ADL: Architeture Description Language 为体系结构提供概念性框架和描述的具体语法，也提供解析、显示、编译、分析或者仿真体系结构描述使用的工具。
14. 软件产品线是一个工程问题，它由一组软件密集型家族系统共享公共的、可管理的特征。
15. 软件体系结构风格是指一组设计词典、有关词典如何运用的限制条件、及词典语义的假设。
16. 体系结构文档化的过程就是用一个或者多个视图来描述一个系统结构的过程。
17. 软件体系结构的基础：计算机硬件结构、软件的基本组成、构成软件的可用组块三个方面。

18. 体系结构需要基础、层次、模式、清晰的角色划分。

19. 软件是对一组数据进行处理的一串指令。

20. 任何具有固定组成形式的数据、代码、数据集合、代码序列、数据和代码的结合体都可以称作结构。

21. 部件：数据、外部设备、程序段

22. 实现部件连接的四种方式：过程调用、远程过程调用、事件触发、服务连接

（请求部件 → 接口 → 分析器 → 执行器 → 请求部件）

23. 结构化连接模式：建立在基本控制流之上的高层次抽象，属于控制模式。

(条件连接、循环连接、查询连接、终端/事件方式、共享信息方式)

24. 常见的数据结构：线性结构、树形结构、复杂结构、文件结构

25. 抽象数据类型的定义，四元组（D，R，P，S）表示

26. 层次性是软件体系结构的不变性质，是软件构成的共同规律。

27. 部件和连接器被公认为体系结构的两大类构成部分

部件是软件功能设计和实现的载体

连接器是专门承担连接作用的特殊部件

区别与联系：

- 简单的连接器从结构上退化为部件之间的直接连接，复杂的连接器需要专门的机构来完成，所以连接器也是部件。
- 一般部件是软件功能设计和实现的载体，而连接器是负责完成部件之间信息交换和行为联系的专用部件

28. 部件的分类：

按照层次划分：基础部件、中层部件、高层部件

按照应用范围分：专用部件、通用部件

按照功能分：数据服务部件、功能服务部件、逻辑处理部件、界面部件 etc.

29. 部件的接口特性：完备性、最小化、正交性、方便性、效率

30. 部件的运行特性：中断处理、并行调度、多用户服务

31. 连接是部件间建立和维持行为关联和信息传递的途径。

简单连接和复杂连接

机制：连接得以发生和维持的机制

协议：连接能够正确、无二义性、无冲突地进行

连接的种类：操作/过程调用、控制/事件/消息发送、数据传送

32. 数据抽象和面向对象设计是在主程序和子过程设计基础上建立和发展起来的重要的软件描述方法。

33. 类是数据抽象的载体，类由数据成员和操作方法构成。

多态性是（1）同一个行为名，作用在不同的对象上，操作细节不同的性质；（2）同名方法对不同参数的处理过程不一样。

34. 层次结构的应用实例：虚拟机；API 接口；IS（信息系统）

35. 体系结构风格是根据结构组织模式构成的软件系统，表达了部件以及他们之间的关系.

体系结构风格有：数据流系统、调用和返回系统、独立构件、虚拟机、数据中心系统

数据流系统	调用和返回系统	独立构件	虚拟机	数据中心系统
批处理序列	主子程序	通信进程	解释器	数据库
管道过滤器	面向对象系统	时间系统	基于规则系统	超文本系统
	多级分层			黑板

36. MVC 的主要关系还是由 Observer、Composite(组合)和 Strategy 三个设计模式给出的

37. **CSP**：Communication sequential process：是一种形式化的描述语言，主要表现了基于顺序的通信处理过程。通过 CSP，可以深入刻画系统的功能行为和规划说明。

38. **CHAM**：Chemical abstract mechanism 化学抽象机制，通过将化学反应和抽象概念有机结合，用以描述系统状态变化。

39. **MDA**：Model Driven Architecture 模型驱动架构，为应对业务和技术的变化提供的一种开放的、中立的开发方法

40. **IOC**：Inversion of control 控制反转，是一个重要的面向对象编程的法则，用于降低耦合，是实现依赖倒置的一种方法

41. **DIP**：Dependency inversion principle 依赖倒置原理，强调系统的高层组件不应当依赖于底层组件；并且不论是高层组件还是底层组件都应当依赖于抽象。

42. **ADT（去年考到）**：抽象数据类型（Abstract Data Type）一个数学模型以及定义在该模型上的一组操作。 ADT 包括数据元素，数据关系以及相关的操作。

简答题

1. SA 对 SE 的贡献：6 点

开发团队的组织结构

捕获需求

设计方案的选择

分析和描述复杂系统的高层属性

人员交流

技术进步

2. SE 的目标是什么？

软件质量（运行时质量、非运行时质量、商业质量、体系结构质量）

3. 影响软件质量的因素：15 点

Correctness 正确 定义：做该做的事情，并且做得对

Functionality 功能

Performance 性能 定义：系统的响应时间，硬件资源的占用率

Security 安全 定义：在对合法用户提供服务的同时，阻止未授权用户的使用企图。

Robustness 鲁棒性 定义：能长时间正确运行并快速从错误状态恢复到正确状态

Availability 可信性、健壮性

Usability 可用性 定义：最终用户容易使用和学习

Ease of use　易用性

Modifiability 适应性　定义：系统很容易被修改从而适应新的需求或采用新的算法、数据结构的能力

Portability 移植性　定义：软件可以很简单地在平台间移植

Reusability 重用性　定义：在新系统中应用已有的组件

Integrability 集成性　定义：让分别开发的组件在一起正确的工作

Testability 可测性　定义：让软件容易被证明是错的

Compatibility 兼容性　定义：易于把软件元素和其他软件结合

Economy 经济效益　定义：开发成本、开发时间和对市场的适应能力

4. 提高质量即使体系结构：5 点

易于理解、可度量、可复用、可文档化、易于交流和执行

5. 软件设计的目标　5 点

便于维护和升级，因而应该是模块化的

设计应该是便于移植的（移植比重新设计花费要小的多）

设计应该具有适用性

设计过程应该受到理性的控制 Intellectual Control

设计应该表现出概念的完整性 Conceptual Integrity　(内在结构、外在表现)

6. 软件体系结构的研究范畴有：4 点

体系描述语言和工具：概念性地描述了软件体系结构的框架概念，包括定义解析，显示，辨析以及系统仿真各个方面的工具，工具的建模，是通过符号的设计方式进行的。

产品线与标准：产品线是一个工程问题，它由一组密集型家族系统共享的

软件体系结构风格及风格应用：指一组设计词典，由关词典如何运用的限制条件及词典语义的假设。

体系结构文档化：文档化的过程就是用一个或多个视图来描述一个系统结构的过程。

7. 软件设计中出现的问题

设计对于需求的变化缺乏配合

过程控制对于维持设计的正确性缺乏保障

软件产品通常缺乏概念完整性

8. 软件体系结构在软件开发中的意义

软件体系结构是软件开发过程初期的产品，对于开发进度和软件质量的一切资金和劳务投入，可以获得最好的回报。

体系结构设计是形成的投资高回报的重要因素。

正确有效的体系结构设计会给软件开发带来极大的便利。

9. 软件体系结构的目标：

外向目标：建立满足终端用户要求的系统需求。

内向目标：建立满足后期设计者需要以及易于系统实现、维护和扩展的系统部件构成。

10. 软件结构基础的思想和概念，包括四个方面：

结构化控制流、结构化连接模式、数据结构、抽象数据类型

11. 软件体系结构层次模型：

层次系统（Layered Systems）是一种体系结构风格(6 层)

计算机硬件 ：软件运行的物质基础

软化的硬件层：在硬件结构和性能抽象的基础上，实现硬件的操作和控制描述（处理器：状态和指令集合 中断：状态和中断服务）

基础控制描述层：建立在高级程序语言描述上的纯粹软件描述层，包括了高级语言所支持的所有程序控制和数据描述概念（程序设计语言、结构化分析、面向对象分析设计）

资源和管理调度层：基础控制描述层建立的一切数据对象和操作，都需要在操作系统的协调和控制下才能实际的实现其设计的作用和功能（进程控制、分时系统、消息机制 etc ）

系统结构模式层 ：最高层次的软件结构概念 、属于体系结构风格或系统级别的设计模式、最高的抽象描述层（解释器、编译器、编辑器、管道/过滤器、黑板、C/S 、B/S、框架 etc.）

应用层 ：从纯粹应用领域出发所建立的系统结构概念（企业管理、公文处理、控制系统、CAD 系统、ERP 系统 ）

12. 层次模型对软件体系结构的认识 3 点

体系结构是关于软件的构成部件及其连接的分层的结构框架

体系结构包括软件的内在概念和外在操作结构

体系结构分析与设计涵盖并指导着从逻辑结构设计到运行实现的软件工程的全部过程

13. 体系结构设计中遵循的原理：11 点

抽象

抽象是人们用来处理复杂性问题的基本原理之一，包括：数据抽象、实体抽象、行为抽象、过程抽象、对象抽象、虚拟机抽象

封装

将抽象的属性和行为结合在一起，为不同的抽象提供了明确的界限，有利于非功能特性（可变性和可复用性）的实现。

数据隐藏（信息隐藏）

对用户隐藏部件的实现细节，接口与实现相分离，用来更好地处理系统的复杂性和减少各部件之间的耦合。

模块化

良好定义的分界，将构成应用的逻辑结构物理地分割成代码实体，是一个应用的功能和责任的物理容器。缺点：额外复杂的系统资源管理。

注意点分离

不同和无关联的责任应该在软件系统中分离开，让他们出现在不同的部件中。相互协作完成某一特定任务的部件应该和在其他任务中执行计算的部件分离开来。避免过多暴露所造成的对应用设计的负担和混乱，保证了组件运行的可靠和安全。

耦合和内聚

高内聚（模块内的功能联系）低耦合（模块间联系的紧密程度）

充分性、完备性和原始性

充分性是指部件应该把握住与其进行有意义和高效交互抽象的所有特性；

完整性是指一个部件应该把握住所有与其抽象相关的特性；

原始性是指部件应该完成的操作都可以容易地得到实现

策略和实现的分离（决策机构与实现部门）

策略部件负责处理上下文相关的决策、信息的语义和解释的知识、把不相关计算组合形成结果、对参数值进行选择等问题。策略部件因为与特定的应用相关，通常随时间的变化而改变。

实现部件负责全面规范算法的执行，执行中不需要上下文相关信息进行决策。实现部件因为独立于上下文环境，因而更容易重用和维护。

接口与实现的分离

接口定义了部件所提供的功能并规范了功能的使用方法

实现部分包括了部件所提供功能的实际代码

强调一个客户只应该知道他需要知道的东西

是一种信息隐藏技术

接口和实现的分离可以很好的支持可变性

分而治之

自上而下的分析，自下而上的实现

横向的阶段分解

层次化

层次化在软件构造中是一个基本思想。类和对象是层次化的，操作系统构造、编译器构造、应用系统构造，无一不是层次化的。同样，软件体系结构也是层次化的。

14. 软件的非功能特性 6 点

功能特性主要直接针对用户的功能需求，多数是容易感知和判断的。

不成熟的客户、投资者、设计者往往片面追求表面功能而忽略了内在结构和非功能性。

软件系统越大越复杂、生命周期越长，非功能性就越重要。

可变性（可维护性、可扩充性、可重构性、可移植性）

互操作性（指不同的计算机系统、网络、操作系统和应用程序一起工作并共享信息的能力）

效率（软件运行过程中对资源的使用情况，以及对系统的响应时间、存储消耗和 I/O 吞吐量的影响；不仅仅是设计精良的算法，而且是部件操作责任合理的

分配、部件之间的耦合关系等体系结构问题，良好结构、丰富功能和高效率等方面要权衡利弊）

可靠性（是软件系统在各种情况下维持其功能的能力，分为容错性（当错误时间发生时确保正确的系统响应，必要时采取内部补救措施）和健壮性（不要求软件在出错后还要继续执行，只需要保证软件能以明确和可接收的方式终止））

可测试性（支持可测试性的软件体系结构可以为错误探测和改正以及代码调试和部件的临时集成给予支持）

可重用性（重用软件系统或其中一部分能力的难易程度）

15. 部件及其作用

部件是软件系统的结构块单元，是软件功能设计和实现的承载体。

系统是部件及其关联的集合。

使用的时候：一个部件至少有一个接口，每一个接口代表对外联系的一种角色，这是部件与外界发生联系的窗口。

设计系统时候：需要根据对部件的功能、与其他部件的关联、对部件的特殊性要求，建立内部处理和控制结构。

16. 部件的表达形式

任何具有独立结构和行为特性的软件体都可以成为部件

不同软件设计环境下服务于不同目的，部件具有不同的类型或名称

17. 连接的实现机制

计算机硬件提供了实现一切连接的基础

高层次的连接建立在低层次的连接之上，实现连接在不同的层次上有不同的概念或方法

无论多么复杂的连接关系，其实现都是基于以下基本连接机制：过程调用、中断、I/O、DMA、进程、线程、共享、同步、并/串行、事件、并发 etc.

18. 连接器及其作用

连接器是实现部件与部件之间联系（调用、消息传递、数据转换传送、部件间实时并行的协调控制等）的特殊机制或特殊**部件**

简单的连接器从结构上退化为部件之间的直接连接，复杂的连接器需要专门

的机构来完成，所以连接器也是部件

连接器承担了实现部件间信息和行为关联的作用，是系统复杂性的来源，对系统的各种性能有着重要的影响

19. 连接的特性

连接的方向性（控制的渠道和信息的传送&&双向性）；

连接的角色（C/S；B/S；P2P；中断源、中断处理者），角色和地位的不同在连接实施表现为所进行的操作不同、期望获得的信息不同；

连接的激发；

连接的响应特性（从动方队连接请求处理的实时性、方式（同步 or 异步）、并发处理能力），响应特性大大增加了实现的复杂性；连接不匹配和解决办法

20. 连接器的特性

连接的关系 1：1、1：n、n：1、m：n

连接的角色和方向

连接的交互方式（信号、语言式）

连接的可扩展性

连接的互操作性

连接请求响应特性

连接请求的处理策略

连接的代价、处理速度或能力

21. 主程序和子程序的优缺点：

优点

是一切软件结构的最本质、最基础的形式

代码的效率可以得到最大限度的发挥和提高。

缺点

部件的连接关系不明显。 //简单的调用的背后实际上可能较为复杂。

代码维护性差。//简单的调用关系为维护带来了困难。

代码的复用性差。 //单纯的过程不能反映复杂的结构，难以成为复用的单元和基础。

结论

该方法存在的问题导致了各种软件设计方法的研究和提出，导致了对软件体系结构的研究和发展，OO 方法是最主要的突破和发展。

22. 面向对象的优缺点

优点

信息隐藏保证了对象行为的可靠性，Called By Methods。

受封装的独立运行对象把数据和操作捆绑在一起，提高了对象作为一种模块的内聚力，使系统更容易分解成相互作用又相互独立的对象集合。

将操作请求和实现细节的分离使得可能在不影响调用者情况下改变操作的实现，为系统的维护和升级提供了良好的条件。

继承性和封装性提高了复用的可能性和方便性。

缺点

对象之间方法的调用必须知道被调用者的标识，造成系统维护上的困难

如果系统分析初期就使用面向对象的方法则可能大大加剧系统分析的复杂性

23. 层次化设计的来源

事物总是从最简单的、最基础的层次开始发生的

来自众多复杂软件设计的实践，几乎所有的系统都是从层次化结构建立起来的

24. 层次结构的实现：

定义为合适的分层而采取的抽象标准，这种标准通常反映了与平台概念的差距；

抽象标准决定模型层次的数目，层次的过少或者过多都导致问题的发生；给各个层次命名和分配任务；

规范服务，层次之间要严格分开，确保没有部件会跨越两层以上。高层应该扩展以覆盖更大的可用性，底层应该设法保持瘦小（倒金字塔型式）；

为每个层次定义接口；

构建独立的层次，工作重点尽可能放在层次之间的接口上。

25. 层次结构的变种：

*松弛的层次系统：是分层模型的一种变种，每层可以使用其下所有层不仅仅是相邻层的服务；有些服务提供给上层，有些提供给所有的上层。

灵活性和性能的提高以牺牲可维护性为代价。经常用于系统软件，而不常用于应用软件。

*通过继承的层次结构：常见于面向对象的程序设计中：低层作为基类，较高层通过继承使用基类提供的服务。高层可根据需要改写底层所提供的服务，如果基层改变了，那么继承该基类的所有子类都要重新编译。这种由继承引入的依赖性称作脆弱的基类问题。

26. 举例说明对象的实现类型

对象的实现类型

1) 普通数据对象：
2) 持久对象
3) 界面对象
4) 组件：实现软件系统运行模块组装和连接的技术。
5) 二进制对象：
6) 对象库：MFC、JFC etc.

27. 层次结构的优缺点：

优点：

层次的复用性；

对标准化的支持，允许在不同层使用来自不同商家的产品；

便于支持系统的可移植性和可测试性；

可替换性，独立层次的实现能够被功能相同的模块替换。享用互操作性的好处的代价是增加了程序开销。

缺点：

改变行为的连锁效应（在维护升级的时候，可能需要在许多层次上做大量的工作）；

低效率；

包含许多不必要的工作；

28. 一些面向对象的设计法则 4 点

优先使用（对象）组合，而非（类）继承

针对接口编程，而非（接口的）实现

开放－封闭法则（OCP）

Liskov 替换法则（LSP）

29. MVC 模式：

MVC(Model/View/Controller)模式，包括三类对象。

Model 是应用对象，View 是它在屏幕上的表示，Controller 定义用户界面对用户输入的响应方式。

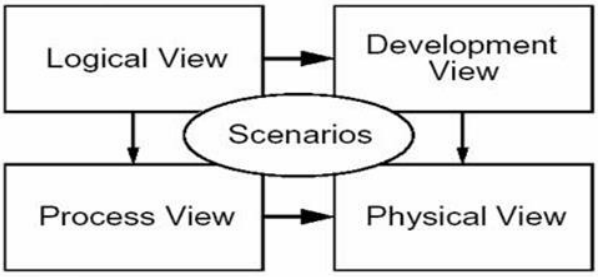
模型是应用程序的主体部分。模型表示业务数据，或者业务逻辑。视图是应用程序中用户界面相关的部分，是用户看到并与之交互的界面。控制器工作就是根据用户的输入，控制用户界面数据显示和更新 model 对象状态。

MVC 式的出现不仅实现了功能模块和显示模块的分离，同时它还提高了应用系统的可维护性、可扩展性、可移植性和组件的可复用性。

MVC 的缺点是由于它没有明确的定义，所以完全理解 MVC 并不是很容易。使用 MVC 需要精心的计划，由于它的内部原理比较复杂，所以需要花费一些时间去思考。

30. 画出 4+1 视图，并描述各个部分的作用

视图名称	过程视图	开发视图	逻辑视图	物理视图	场景
组件（元素）	任务	模块、子系统	类、类层次	节点	步骤、脚本
连接工具	消息、RPC、会话、广播等	编译依赖性、with 语句、include	关联、继承、约束	通信媒体、LAN、WAN、总线等	无



- Scenario (use case)
- 逻辑视图（Logical View），设计的对象模型（使用面向对象的设计方法时）。
- 过程视图（Process View），捕捉设计的并发和同步特征。
- 物理视图（Physical View），描述了软件到硬件的映射，反映了分布式特性。
- 开发视图（Development View），描述了在开发环境中软件的静态组织结构。

31. 什么是软件体系结构的形式化描述？常用的有哪些？

所谓的软件体系结构的形式化描述是采用形如数学符号的语句和语义来刻画软件整体系统的功能、行为以及软件系统的规划和说明。

常用的软件体系结构的形式化描述有：类属理论，Z notation, CSP, CHAM

分类：

描述软件体系结构配置的 ADL：CHAM

描述软件体系结构实例的 ADL：Rapide

描述软件体系结构风格的 ADL：Wright

目标：

精确的语义描述

支持分析推理

32. 模式与抽象在软件领域中的作用

模式的作用：

1) 用一种标准的方式描述设计经验。
2) 为设计者提供一种通用的语言。
3) 增加复用性，减少设计的多样性。

4) 增强设计变更的灵活性。
5) 提高设计文档的质量。
6) 增强设计的可理解性。

抽象的作用：

1) 抽象是认识复杂现象过程中使用的思维工具，即抽出事物本质的共同性而暂不考虑它的细节，不考虑其他因素。即把非本质的性质隐藏起来，只突出那些本质的性质，以减轻人们思考和注意的负担。软件工程过程中的每一步都可以看作是对软件解决方法的抽象层次的一次细化
2) 抽象过程使得设计阶段创建的对象模型仅仅用来描述系统应该做什么，但不必关心如何去做，从而清晰的划清软件设计与软件编码的界限。
3) 数据抽象是面向对象设计的理论基础。
4) 过程抽象忽略任务具体完成的过程，而只精确描述该任务所要完成的功能。

33. 根据你已经掌握的知识经验，总结你对软件体系结构以及它在软件设计开发中作用的认识

软件体系结构是一种对软件本身的抽象，它包含了软件的宏观特征，基本特征，以及概念抽象，软件体系结构在软件工程与软件过程的研究与发展中，起了重要的推进作用，软件体系结构是软件整体结构的基本的可供选择的主题和基本形态。

软件体系结构的重要性主要用于在软件设计与开发的周期的各个阶段中：

1. 软件项目的总体规划阶段：软件体系结构为项目可行性分析，投资规模，风险预测等具有指导性意义
2. 在项目的需求分析阶段：让开发商与客户更加深入理解软件的模式，以及软件的规范，让客户需求得到更清晰的阐述，让设计更加明确，更加针对需求。
3. 在项目设计分析阶段：要从项目的实现角度出发，通过体系结构进行深入研究。
4. 在项目实施阶段：体系结构有助于开发人员更好的协调分工，更好的进行协作。
5. 在评估阶段：体系结构有助于进行项目质量分析，评审。
6. 在系统维护与开发中，体系结构是升级与维护必须遵守的原则，有利于系统的实现升级。

34. 乱七八糟（ ▼-▼ ）

理论的形式化方法：

形式化的基本概念：用于开发计算机系统的形式化方法是描述系统性质的基于数学的技术。约束方法的目标：无二义性、一致性和完整性。ADL 语言具备的特点：严谨的语法、语义；描述能力强；有工具支持；简单易懂；不同的抽象级别；静态分析、动态分析

Z Notation：是牛津大学 Programming Research Group 研究的一种数学语言。使用标准的逻辑操作符和集合操作符以及他们标准的常规语义。一个 Z 规范是由文字和数学描述构成的。数学描述是一个类型的集合，每个类型带有为其取值为真的谓词，Z 中的类型是值的集合。

CSP：是由英国图灵奖获得者 C.A.R Hoare 于 1978 年提出的分布式程序设计原型“通信顺序进程 Communication Sequential Process”被广泛用于分布式设计的建模和分析中。采用 CSP 作为软件体系结构规格说明的语义基础，可以清晰地表达系统体系结构的交互行为。但，CSP 的语义是基于进程的静态交互描述，而且在进程通信时只允许传值。这很大程度上限制了 CSP 在描述结构时的广度和深度。

类属理论：是一种表达对象关系的数学语言。提供了概念划分的统一性，软件研究者把他看成是表达抽象和依赖关系的工具；在体系结构的部件和连接器描述中都被广泛采用。优点在于结构特性的自动维持。

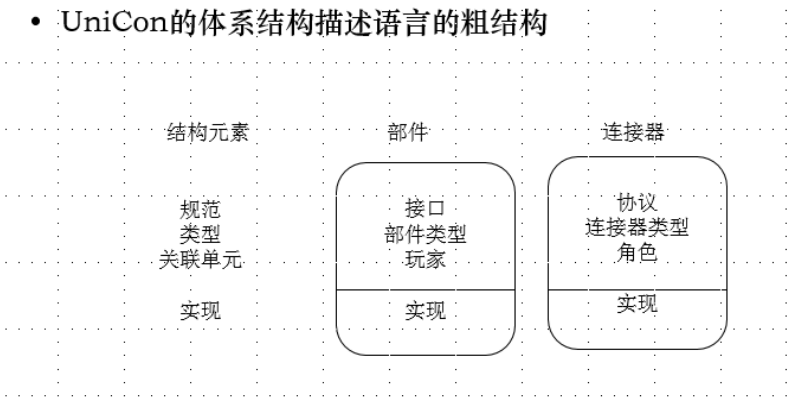
化学抽象机模型：是 G.Berry 和 G.Boudol 于 1992 年提出的。

软件体系结构集成环境

软件体系结构集成环境的目标：能够描述各种风格的软件系统结构；支持自顶向下（自下而上）的分层化体系结构元素；确保接口一致；提供一个体系结构知识库存放于体系结构设计相关的设计模式、设计要素、设计需求；具有一个提供数据和控制模型代码的模板生成器；提供体系结构的图形化描述工具，从不同的视角反应体系结构。

UniCon：是卡内基梅隆大学的 Mary Shaw 等人提出并研究的体系结构描述语言和环境。是基于部件和连接器的，即体系结构的描述模型是由基于可识别的称为部件和连接器的个体元素构成的。部件和连接器元素都具有类型、规范和实现。

- UniCon的体系结构描述语言的粗结构



Darwin：标识符、关键字、函数。与 IDL 的约定相同。用标识符定义和命名构件的类型、接口类型、参数、常量、端口、实例等。扩展了 IDL 以支持函数的调用。Darwin 的构件说明定义了可以创建一个或多个实例的构件类型。局部构件：构件的类型可以从其他构架类型完全或者局部导出。还有通用构件。Darwin 系统程序结构构造：支持自底向上；支持自顶向下；支持合并构件；支持分组构件；

Wright：用于描述软件系统的体系结构，可以描述体系结构的风格、系统族、体系结构实例和单个系统。提供对计算构件和连接件的描述。构件间的交互关系是指模块之间的通讯，Wright 体系结构描述语言基于交互。根据构件、连接件和配置等基本体系结构元素的抽象而构造系统，构件作为计算部件，连接件形式化为交互模式。

ACME：卡内基梅隆大学 1995 年提出并研究的体系结构描述语言和环境。ACME 将系统描述成通过连接器实现交互关系的部件的图。采用图形化编辑方式直接描述构件和连接件。

ACME 的核心概念：1、部件：是系统描述的基本组块，代表了系统计算的核心。可以用来表达硬件或者软件。

2、连接器：体系结构模型的一个重要特征是把交互看成是建模的概念。这与 OO 不同；通过显示的连接器的概念，为明确地表达系统的通信关系提供了机制。

体系结构的描述可以分为两类：

1、把现有的图形化方法应用到体系结构设计中

2、开发体系结构描述语言：每种体系结构描述语言都以独立的形式存在，描述语法不同而且互不兼容，这给体系结构语言的推广带来了一定的困难。