[重难点总结（简版）]

1、软件危机的典型表现

（1）对软件开发成本和进度的估计常常很不准确

（2）经常出现用户对“已完成的”软件产品不满意的情况

（3）软件产品的质量往往达不到要求

（4）软件通常是很难维护的

（5）软件往往没有适当的文档资料

（6）软件成本在计算机系统总成本种所占比例逐年上升

（7）软件开发生产率提高的速度远远不能满足社会对软件产品日益增长的需求

2、消除软件危机的途径

（1）对计算机软件有一个正确的认识，软件≠程序

（2）必须充分认识到软件开发不是某种个体劳动的神秘技巧，而应该是一种组织良好、管理严密、各类人员协同配合、共同完成的工程项目。

（3）推广使用在实践中总结出来的开发软件的成功的技术和方法

（4）应该开发和使用更好的软件工具

即技术措施，组织管理措施

3、软件工程的本质特性

（1）软件工程专注于大型程序的构造

（2）软件过程的中心课题是控制复杂性

（3）软件产品交付使用后仍然需要经常修改

（4）开发软件的效率非常重要

（5）开发人员和谐的合作是成功开发软件的关键

（6）软件必须有效的支持它的用户

（7）在软件工程领域中通常由具有一种文化背景的人替具有另一种文化背景的人开发产品

4、软件工程的基本原理

（1）用分阶段的生命周期计划严格管理

（2）坚持进行阶段评审

（3）实行严格的产品控制

（4）采用现代程序设计技术

（5）结果应能清楚的审查

（6）开发小组人员应该少而精

（7）承认不断改进软件工程实践的必要性

5、瀑布模型

适用范围：需求明确，小规模软件开发

传统软件工程方法学的软件过程基本上可以用瀑布模型来描述

优点：强迫开发人员采用规范的技术方法；严格地规定了每个阶段必须提交的文档；每个阶段结束前必须正式进行严格的技术审查和管理复查。

缺点：开发人员和用户之间缺乏有效的沟通，很可能导致最终开发出来的软件产品不能真正满足用户的需求。

6、快速原型模型

适用范围：需求不明确的

所谓快速原型模型是快速建立起来的、可在计算机上运行得程序，它所能完成的功能往往是最终的软件产品所能完成功能的子集。原型是软件开发人员与用户沟通的强有力工具，因此有助于所开发出的软件产品满足用户的真实需求。

优点：使用这种软件过程开发出的软件产品通常能满足用户的真实需求；软件产品的开发过程基本上是线性顺序过程。

缺点：这样开发出来的系统，往往可能因为折中而采取不合适的操作系统或程序设计语言。

7、增量模型

适用范围：进行已有产品升级或新版本开发；对完成期限严格要求的产品；对所开发的领域比较熟悉而且已有原型系统。

增量模型也称为渐增模型，使用增量模型开发软件时，把软件产品作为一系列增量构件来设计、编码、集成和测试。每个构件由若干个相互协作的模块构成，并且能够完成相对独立的功能。

优点：能在较短时间内向用户提交可完成部分工作的产品；逐步增加产品功能，从而使用户有较充裕的时间学习和适应新产品，减少一个全新的软件给用户所带来的冲击。

缺点：集成困难，软件结构开放不好设计。

8、螺旋模型

适用范围：内部开发的大型软件项目

把它看作在每个阶段之前都增加了风险分析过程的快速原型模型，是一种风险驱动的迭代式开发过程。具体为不断重复生命周期模型，并在每个生命周期结构后，向用户提交一个可运行的软件产品版本。

优点：有利于已有软件得重用；有助于把软件质量作为软件开发的一个重要目标；减少了过多测试或测试不足所带来的风险；软件维护与软件开发没有本质区别。

缺点：需要相当的风险分析评估的专门技术，且成功依赖于这种技术。很明显一个大的没有被发现的风险问题，将会导致问题的发生，可能导致烟花的方向失去控制。这种模型相对比较新，应用不广泛，其功效需要进一步验证。

9、喷泉模型

喷泉模型是典型的面向对象的软件过程模型之一。

10、为什么说喷泉模型较好的体现了面向对象软件开发过程无缝和迭代的特性？

由于各阶段都使用统一的概念和表示符号，因此，整个开发过程都是吻合一致的，或者说是“无缝”连接的，这自然就很容易实现各个开发步骤的多次反复迭代，达到认识的逐步深化。而喷泉模型则很好的体现了面向对象软件开发过程迭代和无缝的特性。

11、Rational 统一过程

适用范围：大型的需求不断变化的复杂软件系统项目

优点：提高了团队生产力，在迭代的开发过程、需求管理、基于组建的体系结构、可视化软件建模、验证软件质量及控制软件变更等方面、针对所有关键的开发活动为每个开发成员提供了必要的准则、模板和工具指导，并确保全体成员共享相同的知识基础。它建立了简洁和清晰的过程结构，为开发过程提供较大的通用性。

缺点：RUP 只是一个开发过程，并没有涵盖软件过程的全部内容，例如它缺少关于软件运行和支持等方面的内容，此外，他没有支持多项目的开发结构，这在一定程度上降低了在开发组织内大范围实现重用的可能性。

RUP 得最佳实践：迭代式开发；管理需求；使用基于构件的体系结构；可视化建模；验证软件质量；控制软件变更。

RUP 软件开发生命周期：是一个二维的生命周期；9 个核心工作流和 4 个工作阶段。其中前 6 个是核心过程工作流程，后 3 个是核心支持工作流程。

9 个核心工作流：业务建模，需求，分析与设计，实现，测试，部署，配置与变更管理，项目管理，环境。

4 个工作阶段：初始阶段（建立业务模型，定义最终产品视图，并且确定项目的范围）；精化阶段（设计并确定系统的体系结构，制定项目计划，确定资源需求）；构建阶段（开发出所有构件和应用程序，把它们集成为客户所需得产品，并且详尽地测试所有功能）；移交阶段（把开发出的产品提交给用户使用）。

12、敏捷过程

适用范围：商业竞争环境下对小型项目提出的有限资源和有限开发时间的约束/开发可用资源及开发时间都有较苛刻约束得小型项目

敏捷软件开发宣言的 4 个简单的价值观：个体和交互胜过过程和工具；可以工作的软件胜过面面俱到的文档；客户合作胜过合同谈判；响应变化胜过遵循计划。

13、极限编程 XP

适用范围：需求模糊且经常改变的场合。

特点：对变化和不确定性的更快速、更敏捷的反应；在快速的同时仍然能够保持可持续的开发速度。

有效实践：客户作为开发团队的成员；使用用户素材；短交付周期；验收测试；结对编程；测试驱动开发；集体所有；持续集成；可持续的开发速度；开放的工作空间；及时调整计划；简单的设计；重构；使用隐喻。

14、微软过程

适用范围：商业环境下具有有限资源和有限开发时间约束的项目的软件过程模式。

优点：综合了 RUP 和敏捷过程的许多优点，是对众多成功项目的开发经验的正确总结

缺点：对方法、工具和产品等方面的论述不如 RUP 和敏捷过程全面，人们对它的某些准则本身也有不同意见。

生命周期：5 个阶段，规划阶段，设计阶段，开发阶段，稳定阶段，发布阶段。

15、为什么说分阶段的生命周期模型有助于软件项目管理？

软件是计算机系统的逻辑部件而不是物理部件，其固有的特点是缺乏可见性，因此，管理和控制软件开发过程相当困难。分阶段的生命周期模型提高了软件项目的可见性。管理者可以把各个阶段任务的完成作为里程碑来对软件开发过程进行管理。把阶段划分得更细就能够更密切地监控软件项目的进展情况。

16、可行性研究的必要性

来判断原有的系统规模和目标是否现实，系统完成后所能带来的效益是否大到值得投资开发这个系统的程度。可行性研究实质上是要进行一次大大压缩简化了的系统分析和设计的过程，也就是在较高层层次上以较抽象的方式进行的系统分析和设计的过程。

+目的+

从 3 个方面研究可行性：技术可行性、经济可行性、操作可行性，必要时还应该从法律、社会效益等更广泛的方面研究每种解法的可行性。

17、情景分析的用处表现在：它能在某种程度上演示目标系统的行为，从而便于用户理解，而且还可能进一步展示出一些分析员目前还不知道的需求；由于情景分析较易为用户所理解，使用这种技术能保证用户在需求分析过程中始终扮演一个积极主动的角色。

18、用于需求分析的结构化分析方法需遵循得准则

必须理解并描述问题的信息域，根据这条准则应该建立数据模型；必须定义软件应该完成的功能，这条准则要求建立功能模型；必须对描述目标系统信息、功能和行为的模型进行分解，永层次的方式展示细节。

19、情景与描述所有可能的动作序列的状态图之间有什么关系？

情景仅仅是通过部分或全部状态图的一条路径。也就是说，情景仅仅描述了系统的某个典型行为，而状态图则描述了系统所有行为。

20、在程序流程图中每个结点都必须有一条从开始结点到该结点本身的路径，以及一条从该结点到结束结点的路径。为什么数据流图没有关于结点之间可达性的类似规则？

数据流图不描述控制，因此，在一个数据流图中两个“处理”之间可能没有通路。如果每个处理都使用不同的输入数据，并生成不同得输出数据，而且一个处理的输出不用作另一个处理的输入，那么，在它们之间就没有弧。

21、在进行详细设计之前先进行总体设计的必要性：

可以站在全局高度上，花较少成本，从较抽象的层次上分析对比多种可能的系统实现方案和软件结构，从中选出最佳方案和最合理的软件结构，从而用较低成本开发出较高质量的软件系统。

22、抽象与求精是一对互补的概念。抽象使得设计者能够说明过程和数据，同时缺忽略了低层细节。事实上，可以把抽象看作是一种通过忽略多余细节同时强调有关细节，而实现逐步求精的方法。求精则帮助设计者在设计过程中逐步揭示出低层细节。这两个概念都有助于设计者在设计演化过程中创造出完整得设计模型。

23、耦合

设计原则：尽量使用数据耦合，少用控制耦合和特征耦合，限制公共环境耦合的范围，完全不用内容耦合。

数据耦合：模块间通过参数交换信息，参数仅仅是数据。

例子：计算机网络属于松耦合系统。

控制耦合：模块见传递的信息中有控制信息（尽管有时这种控制信息以数据的形式出现）。

例子：遥控器与电器。传递功能代码。

特征耦合：当整个数据作为参数传递而被调用的模块只需要使用其中一部分数据元素。

公共环境耦合：两个或多个模块通过一个公共数据环境相互作用时。

例子：多机系统。

内容耦合：一个模块访问另一个模块的内部数据；一个模块不通过正常入口而转到另一个模块的内部；两个模块有一部分程序代码重叠（只可能出现在汇编程序中）；一个模块有多个入口（这意味着一个模块有几种功能）。

例子：汇编程序模块。

24、内聚

偶然内聚：如果一个模块完成一组任务，这些任务彼此间即使有关系，关系也是很松散的。

逻辑内聚：一个模块完成的任务在逻辑上属于相同或相似的一类（例如一个模块产生各种类型的全部输出）。

例子：一个子程序将打印季度开支报告、月份开支报告和日开支报告，具体打印哪一个，将由传入的控制标志决定，这个子程序具有逻辑内聚性，因为它的内部逻辑是输进去的外部控制标志决定的。

时间内聚：一个模块包含的任务必须在用一段时间内执行（例如，模块完成各种初始化工作）。需是同“类”操作，如初始化变量和打开文件不是一“类”操作，因为初始化变量是程序特有的操作，打开文件是硬件要求的操作，是任何使用文件的程序都包含的一个操作，并非本程序特有的操作。

例子：将多个变量的初始化放在同一个模块中实现，或将需要同时使用的多个库文件打开操作放在同一个模块中，都会产生时间内聚的模块。

过程内聚：如果一个模块内的处理元素是相关的，而且必须以特定次序执行。例子：一个子程序，它产生读取雇员的名字，然后是地址，最后是它的电话号码。这种顺序之所

以重要，仅仅是因为它符合用户的要求，用户希望按这种顺序进行屏幕输入。另一个子程序将读取关于雇员的其他信息。这个子程序是过程内聚性，因为是由一个特定顺序而不是其他任何原因，把这些操作组合在一起的。

通信内聚：模块中所有元素都使用同一个输入数据和（或）产生同一个输出数据。

顺序内聚：一个模块内得处理元素和同一个功能密切相关，而且这些处理必须顺序执行（通常一个处理元素的输出数据作为下一个处理元素的输入数据）。例子：一个按给出的生日计算雇员年龄、退休时间的子程序，如果它是利用所计算的年龄来确定雇员将要退休的时间，name 它就具有顺序内聚性。而如果它是分别计算年龄和退休时间的，但使用相同生日数据，那它就只具有通讯内聚性。

功能内聚：模块内所有的处理元素属于一个整体，完成单一的功能。

例子：计算雇员年龄并给出生日的子程序就是功能内聚性，因为它只完成一项工作，而且完成得很好。

25、启发式规则（改进软件设计提高软件质量的途径）

（1）改进软件结构提高模块独立性。设计出软件的初步结构以后，应该仔细审查分析这个结构，通过模块分解或合并，力求降低耦合、提高内聚；

（2）模块规模应该适中。模块规模过大，则可理解程度很低；模块规模过小则开销大于有效操作。通过模块分解或合并并调整模块规模时，不可降低模块独立性。

（3）深度、宽度、扇出和扇入都应适应；

（4）模块的作用域应该在控制域之内；（模块的作用域：受该模块内一个判定影响得所有模块的集合。模块的控制域：模块本身以及所有直接或间接从属于它的模块得集合）

（5）力争降低模块接口的复杂程度；接口复杂或与模块功能不一致，是紧耦合或低内聚的征兆，应该重新分析这个模块得独立性。

（6）设计单入口单出口模块；这条启发式规则警告软件工程师不要使模块间出现内容耦合。

（7）模块功能应该可以预测。不要使模块功能过分局限。

26、交互式系统给出的出错信息或警告信息所具有的属性

（1）信息应该用用户可以理解的属于描述问题。

（2）信息应该提供有助于从错误中恢复的建设性意见。

（3）信息应该指出错误可能导致哪些负面后果（例如：破坏文件），以便用户检查是否出现了这些问题，并在确实出现问题时及时解决。

（4）信息应该伴随着听觉或视觉上的提示。

（5）信息不能带有指责色彩，也就是说，不能责怪用户。

27、对设计进行评估复审时的评估标准

（1）系统及界面的规格说明书的长度和复杂程度，预示了用户学习使用该系统所需要的工作量；

（2）命令或动作的数量、命令的平均参数个数或动作中单个操作的个数，预示了系统的交互时间和总体效率；

（3）设计模型中包含的动作、命令和系统状态的数量，预示了用户学习使用该系统时需要记忆的内容的多少；

（4）界面风格、帮助设施和出错处理协议，预示了界面的复杂程度及用户接受该界面的程度。

28、软件测试的准则

（1）所有的测试都应该能追溯用户需求；

（2）应该远在测试开始之前就制定出测试计划；

（3）应该把 Pareto 原理（测试发现的错误中的80%很可能是由程序中 20%的 模块造成的）应用到软件测试中；

（4）应该从“小规模”测试开始，并逐步过渡到“大规模”测试；

（5）穷举测试是不可能的，因此，测试只能证明程序中有错误，而不能证明程序中没有错误；

（6）为了达到最佳测试结果，应该由独立的第三方从事测试工作。

29、软件测试的方法

包括黑盒测试和白盒测试。

黑盒测试/功能测试：把程序看作一个黑盒子，完全不考虑程序的内部结构和处理过程。也就是说，黑盒测试是在程序接口进行的测试，它只检查程序功能是否按照规格说明书的规定争产使用，程序是否能适当地接受输入数据并产生正确的输出信息，程序运行过程中能否保持外部信息的完整性。

白盒测试/结构测试：把程序看成装在一个透明的白盒子里，测试者完全知道程序的结构和处理算法。这种方法按照程序内部的逻辑测试程序，检测程序中的主要执行通路是否都能按照预定要求正确工作。

30、测试步骤 ：

（1）模块测试（单元测试）：发现并改正程序模块中的错误，保证每个模块作为一个单元能正确地运行；特点：主要应用白盒测试，对多个模块的测试可以并发的进行。

（2）子系统测试（集成测试/组装测试）：把经过单元测试的模块组装成一个子系统，在组装的过程中同时进行测试；

（3）系统测试（集成测试/组装测试）：把经过测试的子系统组装成一个完整的系统并同时进行测试；

（4）验收测试（确认测试）：把软件系统作为单一的实体进行测试，验证系统确实满足用户的需要，主要使用实际数据进行测试；

（5）平行运行：同时运行新开发出的系统和被它取代的旧系统，通过比较新旧两个系统的运行结果来测试新系统。

其中，集成测试是把模块装配在一起形成完整的软件包，在装配的同时进行测试。集成测试的特点：在测试过程中可能发生接口问题。

31、非渐增式测试方法和渐增式测试方法

非渐增式测试方法：先分别测试每个模块，再把所有模块按设计要求放在一起结合成所要的程序；

渐增式测试方法：把下一个要测试的模块同已经测试好的那些模块结合起来进行测试，测试完以后再把下一个应该测试的模块结合起来测试；

两者优缺点对比：非渐增式测试一下子把所有模块放在一起，并把庞大的程序作为一个整体来测试，而测试中会遇到很多错误，改正错误更是极端困难。而一旦改正一个错误又会遇到新错误，这个过程将继续下去，看起来好像永远没有尽头；相反，渐增式测试在这个过程比较容易定位和改正错误，对接口可以进行更彻底的测试，可以使用系统化的测试方法。

32、自顶向下集成和自底向上集成

自顶向下集成的步骤：（1）对主控制模块进行测试，测试时用存根程序代替所有直接附属于主控制模块的模块；（2）根据选定的结合策略（深度优先或宽度优先），每次用一个实际模块代换一个存根程序（新结合进来的模块往往又需要新的存根程序）；（4）在结合进一个模块同时进行测试；（5）为了保证加入模块没有引进新的错误，可能需要进行回归测试（即全部或部分地重复以前做过的测试）。从第 2 步开始不断重复上述过程，直到构造起完整的软件为止。

自底向上集成的步骤：（1）把底层模块组合成实现某个特定的软件子功能的族；（2）写一个驱动程序（用于测试的控制程序），协调测试数据的输入和输出；（3）对模块组成的子功能族进行测试；（4）去掉驱动程序，沿软件结构自下向上移动，把子功能族结合起来形成更大的子功能族。第 2 步到第 4 步实质上构成了一个循环。

两个集成策略的比较：（一种方法的优先正好对应于另一种方法的缺点）

（1）自顶向下测试方法的优点：不需要测试驱动，能够在测试阶段的早期实现并验证系统的主要功能，且能在早期发现上层模块的接口错误；

（2）自顶向下测试方法的缺点：需要存根程序，可能遇到与此相联系的测试困难，底层关键模块中的错误发现较晚，且用这种方法在早期不能充分展开人力。

混合策略：改进的自顶向下测试方法（基本上使用自顶向下，但早期使用自底向上的少数关键模块，缺点也比自顶向下方法多一条，即测试关键模块时需要驱动程序）；混合法（当被测试的软件中关键模块比较多时，这种方法是最好的折衷方法）。

33、编程时使用的程序设计语言对软件开发与维护有何影响？

（1）程序设计语言是人们用计算机解决问题的基本工具，因此，它将影响软件开发人员的思维方式和解题方式。

（2）程序设计语言是表达具体的解题方法的工具，它的语法是否清晰易懂，阅读程序时是否容易产生二义性，都对程序的可读性和可理解性有较大影响。

（3）程序设计语言所提供的模板化机制是否完善，编译程序查错能力的强弱等，都对程序的可靠性有明显影响。

（4）程序设计语言实现设计结果的难易程度，是否提供了良好的独立编译机制，可利用的软件开发工具是否丰富而且有效等，都对软件的开发效率有影响。

（5）编译程序优化能力的强弱，程序设计语言直接操纵硬件设施的能力等，都将明显地影响程序的运行效率。

（6）程序设计语言的标准化程度，所提供的模板封装机制，源程序的可读性和可理解性等，都将影响软件的可维护性。

34、软件维护的活动

改正性维护（诊断和改正用户使用软件时所发现的软件爱你错误的过程）；适应性维护（为了使软件和变化了的环境适当地配合而进行的修改软件的活动）；完善性维护（用户在使用软件的过程中，往往提出增加新功能或改变某些已有功能的要求，还可能要求进一步的提高程序的性能，而为了满足这类要求而修改软件的活动）；预防性维护（为

了提高未来的可维护性或可靠性而主动地修改软件的活动，即指把今天的方法学应用到昨天的系统上，以支持明天的需求）。

预防性维护的方法：（1）反复多次地做修改程序的尝试，以实现所要求的修改。（2）通过仔细分析程序尽可能多地掌握程序的内部工作细节，以便更有效地修改它。（3）在深入理解原有设计的基础上，用软件工程方法重新设计、重新编码和测试那些需要变更的软件部分，即局部软件再工程。（4）以软件工程方法学为知道，对程序全部重新设计、重新编码和测试，为此可以使用 CASE 工具来帮助理解原有的设计，即软件再工程。

预防性维护的必要性：（1）维护一行源代码的代价可能是最初开发该行源代码代价的 14~40 倍。（2）重新设计软件体系结构时使用了现代设计概念，对将来的维护可能有很大的帮助。（3）由于现有的程序版本可作为软件原型使用，开发生产率可大大高于平均水平。（4）用户具有较多使用该软件的经验，能够很容易地搞清新的变更需求和变更的范围。（5）利用逆向工程和再工程的工具，可以使一部分工作自动化。（6）在完成预防性维护的过程中可以建立起完整的软件配置。

35、维护代价高昂——软件维护中无形的代价有：（1）因为可用的资源必须供维护任务使用，以致耽误甚至丧失了开发的良机。（2）当看来合理的有关改错或修改的要求不能及时满足时将引起用户不满。（3）由于维护时的改动，在软件中引入了潜伏的错误，从而降低了软件的质量。（4）当必须把软件工程师调去从事维护工作时，将在开发过程中造成混乱。（5）生产率的大幅度下降。

维护存在的问题：（1）理解别人写的程序非常困难，而且困难程度随着软件配置成分的减少而迅速增加。（2）需要维护的软件往往没有合格的文档，或者文档资料显著不足。（3）当要求对软件进行维护时，不能指望由开发人员给人们仔细说明软件。（4）绝大多数软件在设计时没有考虑将来的修改。（5）软件维护不是一项吸引人的工作。

36、软件维护的过程

维护过程本质上是修改和压缩了的软件定义和开发过程。软件维护过程可以描述为：（1）建立一个维护组织；（2）确定报告和评价的过程；（3）为每一个维护要求规定一个标准化的事件序列；（4）建立一个适用于维护活动的记录保管过程；（5）规定复审标准。

具体过程：维护组织；维护报告；维护的事件流；保存维护记录；评价维护活动。

36、可维护性复审的必要性：在软件再次交付使用之前，对软件配置进行严格的复审，则可大大减少文档的问题。事实上，某些维护要求可能并不需要修改软件设计或源程序代码，只是表明用户文档不清楚或不准确，因此只需要对文档做必要的维护。

37、代码重构与正向工程有何相同之处？有何不同之处？

相同之处：都需要重新设计数据结构和算法，并且需要重新编写程序代码。

不同之处：代码重构并不修改程序的体系结构，它只修改某些模块的设计细节和模块中使用的局部数据结构，并重新编写这些模块的代码。如果修改的范围扩展到模块边界之外并涉及程序的体系结构，则代码重构变成了正向工程。

38、用面向对象范型开发软件时与用结构化范型开发软件时相比较，软件的生命周期有何不同？这种差异带来了什么后果？

用结构化范型开发软件时，软件的生命周期：陈述需求阶段；规格说明（分析）阶段；设计阶段；实现阶段；维护阶段。

用面向对象范型开发软件时，软件的生命周期：陈述需求阶段；面向对象分析阶段；面向对象设计阶段；面向对象实现阶段；维护阶段。

有何不同：

（1） 用结构化范型开发软件时，规格说明（分析）阶段的主要任务是确定软件产品应该“作什么”；而设计阶段通常划分成结构设计（即概要设计）和详细设计这样两个子阶段。在结构设计子阶段，软件工程师把产品分解成若干个模块，在详细设计子阶段再依次设计每个模块的数据结构和实现算法。

（2） 如果使用面向对象范型开发软件，则面向对象分析阶段的主要工作是确定对象。因为对象就是面向对象软件的模块，因此，在面向对象分析阶段就开始了结构设计的工作。

由此可见，面向对象分析阶段比它在结构化范型中的对应阶段（规格说明（分析）阶段）走得更远，工作更深入。

后果：

（1） 使用结构化范型开发软件时，在分析阶段和设计阶段之间有一个很大的转变：分析阶段的目的是确定产品应该“作什么”，而设计阶段的目的是确定“怎么样”，这两个阶段的工作性质明显不同。

（2） 相反，使用面向对象范型开发软件时，“对象”从一开始就进入了软件生命周期，软件工程师在分析阶段把对象提取出来，在设计阶段对其进行设计，在实现阶段对其进行编码和测试。

由此可见，使用面向对象范型开发软件时，在整个开发过程都是用统一的概念——对象，围绕对象进行工作，因此，阶段与阶段之间的转变比较平缓，从而减少了在开发软件过程中所犯的错误。

39、为什么在开发大型软件时，采用面向对象范型比采用结构化范型较易取得成功？

（1）结构化技术要么面向处理（如面向数据流的设计方法），要么面向数据（如面向数据结构的设计方法），但没有既面向处理又面向数据的结构化技术。用结构化技术开发出的软件产品的基本成分是产品的行为（即处理）和这些行为所操作的数据。由于数据和对数据的处理是分离的，因此，使用结构化范型开发出的软件产品本质上是一个完整的单元，由此带来的后果是软件规模越大，用结构化范型开发软件的技术难度和管理难度就越大。

（2）与结构化技术相反，面向对象技术是一种以数据为主线，把数据和处理相结合的方法。面向对象范型把对象作为由数据及可以施加在这些数据上的操作所构成的统一体。用面向对象范型开发软件时，构成软件系统的每个对象就好像一个微型程序，有自己的数据、操作、功能和用途，因此，可以把一个大型软件产品分解成一系列本质上相互独立的小产品来处理，不仅降低了软件开发的技术难度，而且也使得对软件开发工作的管理变得相对容易了。

40、为什么说夏利牌汽车是小汽车的特化，而发动机不是小汽车类的特化？

（1）夏利牌汽车具有小汽车的全部属性和行为，它只不过是一种特定品牌的小汽车，因此，夏利牌洗车可以从基类（小汽车）派生出来，也就是说，夏利牌汽车是小汽车类的特化。

（2）发动机是组成小汽车的一种零件。小汽车还有很多其他多种零件，小汽车所具有的许多属性和行为发动机都不具有，因此，发动机不能从小汽车类派出来，它不是小汽车类的特化。

41、为什么说面向对象方法与人类习惯的思维解题方法比较一致？

面向对象方法学的基本原则是按照人们习惯的思维方式建立问题域的模型，开发出尽可能直观、自然地表现求解方法的软件系统。面向对象的软件系统中广泛使用的对象是对客观世界中实体的抽象，对象实际上是抽象数据类型的实例，提供了立项的数据抽象机

制，同时又具有良好的过程抽象机制（通过发消息使用公有成员函数）。因此，面向对象的环境提供了强有力的抽象机制，便于人们在利用计算机软件系统解决复杂问题时使用习惯的抽象思维工具。此外吧 ，面向对象方法学中皮鞭进行的对象分类过程支持从特殊到一般的归纳思维过程；面向对象方法-学中通过建立类等级而获得的继承特性支持从一般到特殊的演绎思维过程。

42、确定问题域中对象类之间可能存在的继承关系的常用的两种方法

（1）自顶向下方法通过研究已知对象类在问题域中的行为和作用，找出在某些情况下需要特殊处理并且具有特殊属性的对象，从而把现有的对象类细化为更具体的子类。这种方法对类似的对象类进行分组，然后寻找它们之间的共性，所有相似的类的交集构成基类。也就是说，

（2）自底向上方法与自顶向下方法相反，这种方法对类似的对象类进行分组，然后寻找它们之间的共性，所有相似的类交际构成基类。也就是说，自底向上方法抽象出某些相似的类的共同特性（属性和操作），泛化出父类。

43、面向对象设计遵循的准则，简述每条准则的内容，并说明遵循这条准则的必要性

（1）模块化：对象是面向对象软件系统中的模块，它是把数据结构和操作这些数据的方法紧密地结合在一起所构成的模块。

（2）抽象：面向对象方法不仅支持过程抽象，而且支持数据抽象。对象类实际上是一种具有继承机制的抽象数据类型。

（3）信息隐藏：在面向对象方法中，信息隐藏通过对象的封装性实现：类结构分离了接口与实现。从而支持了信息隐藏。

（4）弱耦合：包括交互耦合（对象之间的耦合通过消息连接实现）和继承耦合（继承是一般化类与特殊类之间的耦合的一种形式。通过继承关系结合起来的基类和派生类，构成了系统中粒度更大的模块）。

为使交互耦合尽可能松散所遵守的准则：尽量降低消息连接的复杂程度，应该尽量减少消息中包含的参数个数，降低参数的复杂程度；减少对象发送（或接收）的消息数。

（5）强内聚：服务内聚（一个服务应该完成一个且仅完成一个功能）、类内聚（一个类应该只有一个用途，它的属性和服务应该是高内聚的）、一般—特殊内聚（应该是对相应领域知识的正确抽取）。

（6）可重用：软件重用是提高软件开发生产率和目标系统质量的重要途径。重用基本上从设计阶段开始。重用有两方面的含义：一是尽量使用已有的类，二是如果确实需要创建新类，则在设计这些新类的协议时，应该考虑将来的可重复使用性。

44、启发规则

必要性：人们使用面向对象方法学开发软件的历史虽然不长，但也积累了一些经验。总结这些经验得出了几条启发式规则，它们往往能帮助软件开发人员提高面向对象设计的质量。

内容：设计结果应该清晰易懂（因素/内容：用词一致；使用已有的协议；减少消息模式的数目；避免模糊的定义），一般-特殊结构的深度应适当，设计简单的类（注意/内容：避免包含过多的属性；有明确的定义；尽量简化对象之间的合作；不要提供太多服务），使用简单的协议，使用简单的服务；把设计变动减至最小。

45、设计类中的服务包括：确定类中应用的服务，设计实现服务的方法（设计实现服务的算法（应考虑因素：算法复杂度、容易理解与容易实现、易修改），选择数据结构，算法与数据结构的关系，定义内部类和内部操作）。

确定算法与数据结构所需要考虑的因素：（1）分析问题寻找数据特点，提炼出所有可行有效的算法；（2）定义与所提炼算法相关联的数据结构；（3）依据此数据结构进行算法的详细设计；（4）进行一定规模的实验与评测；（5）确定最佳设计。

46、请比较功能内聚和信息性内聚

（1）内聚是衡量组成模块的各个元素彼此结合的紧密程度，它是信息隐藏和局部化原理的自然扩展。设计软件时应该力求做到模块高内聚。

（2）当采用结构化范型设计软件系统时，使用功能分解方法划分模块。组成这类模块的元素最主要是完成模块功能的可执行语句。如果模块内所有处理元素属于一个整体，完成单一完整的功能，则该模块的内聚称为功能内聚。采用结构化范型开发软件时，功能内聚是最高级的内聚。

（3）采用面向对象范型设计软件时，使用对象分解方法划分模块。对象是面向对象软件的基本模块，它是由描述该对象属性的数据及可以对这些数据施加的操作封装在一起构成的统一体。因此，组成对象的主要元素既有数据又有操作，这两类元素是同等重要的。如果一个对象可以完成许多相关的操作，每个操作都有自己的入口点，它们的代码相对独立，而且所有操作都在相同的数据结构上完成，也就是说，操作围绕对其数据所

需要做的处理来设置，不设置与这些数据无关的操作，则该对象具有信息性内聚。实际上，信息性内聚的对象所包含的操作本身应该是功能内聚的。

47、多态重用与继承重用有何关系

（1）当已有的类构件不能通过实例重用方式满足当前系统的需求时，利用继承机制从已有类诞生出符合当前需要的子类，从而获得可在当前系统中使用的类构件，这种重用方式称为继承重用。

（2）如果在设计类构件时，把可能妨碍重用的、与应用环境密切相关的操作从一般操作中分离出来，作为适配接口，并且把这类操作说明为多态操作，类中其他操作通过调用适当的多台操作来完成自己的功能，则为了在当前系统中重用已有的类构件，在从已有类诞生出的子类中只需要重新定义某些多态操作即可满足当前系统的需求，这种重用方式称为多态重用。

（3）通过上面的叙述可以知道，多态重用实际上是一种特殊的继承重用，是充分利用多态性机制支持的继承重用。一般说来，使用多态重用方式重用已有的类构件时，在子类中需要重新定义的操作比较少，因此，这种重用方式的成本比继承重用方式的成本低。

48、面向对象语言的优点

（1）一致的表示方法。面向对象开发基于不随时间变化的、一致的表示方法。既有利于在软件开发过程中始终使用统一的概念，也有利于维护人员理解软件的各种配置成分。

（2）可重用性。既可重用面向对象分析结果，也可重用相应的面向对象设计和面向对象程序设计结果。

（3）可维护性。程序显式地表达问题域语义，对维护人员理解待维护的软件有很大帮助。在选择编程语言时，应该考虑的首要因素是哪个语言能恰当地表达问题域语义。

49、程序设计风格

新规则：提高可重用性（准则：提高方法的内聚；减小方法的规模；保持方法的一致性；把策略和实现分开；全面覆盖输入条件的各种可能组合；尽量不使用全局信息；尽量利用继承机制），提高可扩充性（准则：封装类的实现细节；避免使用多分支语句；精心选择和定义公有方法），提高健壮性（准则：预防用户的错误操作；检查参数的合法性；不要预先设定数据结构的限制条件；先测试后优化）。

50、面向对象实现应该选用哪种程序设计语言？为什么？

（1）面向对象实现应该尽量选用面向对象语言来实现面向对象分析、设计的结果。

（2）原因……[答面向对象语言的优点]

51、测试面向对象软件时，单元测试、集成测试和确认测试各有哪些新特点？

（1）单元测试，是在类层面上的测试，由于继承和复合，类（或对象）在很多情况下已不再是单纯意义上的单个操作。因此，具体的测试将在多有与操作有关的每个子类语境中进行。

（2）集成测试，由于面向对象软件中类的成分直接或间接交互，使得传统测试方法已经失去意义。因此有两种策略可供选择，分别是基于线程的测试和基于使用的测试。

（3）确认测试，关注与用户可见的动作和用户识别的系统输出，但基于场景的测试总是主宰面向对象系统的确认测试。

52、测试面向对象软件时，主要有哪些设计单元测试用例的方法？

设计单元测试用例的方法主要有随机测试、划分测试、基于故障的测试。

（1）随机测试：通过执行一些随机产生的测试用例，来对类和对象进行测试的过程。

（2）划分测试：通过把输入和输出分类，设计测试用例以测试划分出的每个类别的过程。主要分为以下几种方法：

①基于状态的划分：根据类操作改变类状态的能力来划分类操作。

②基于属性的划分：根据类操作使用的属性来划分类操作。

③基于功能的划分：根据类操作所完成的功能来划分类操作。

（3）基于故障的测试：首先推测软件中可能有的错误，然后设计出最可能发现这些错误的测试用例。

53、测试面向对象软件时，主要有哪些设计集成测试用例的方法？

设计集成测试用例的方法主要有多类测试、从动态模型中导出测试用例。

（1） 多类测试：多类测试可分为随机测试和划分测试两种。

① 随机测试

a. 对每个客户类，使用类操作符列表来生成一系列随机测试序列。

b. 对所生成的每个消息，确定协作类和在服务器对象中的对应操作符。

c. 对服务器对象中的每个操作符，确定传递的消息。

d. 对每个消息，确定下一层被调用的操作符，并把这些操作符结合进测试序列中。

② 划分测试

a. 应该扩充测试序列以包括那些通过发送给协作类的消息而被调用的操作。

b. 根据与特定类的接口来划分类操作。

（2）从动态模型中导出测试用例：类的状态图可以帮助人们导出测试该类的动态行为的测试用例。通过导出大量的测试用例，保证该类的所有行为都被适当地测试了。在类的行为导致与一个或多个类协作的情况下，应该使用多个状态图去跟踪系统的行为流。

54、测试面向对象软件时，主要有哪些设计确认测试用例的方法？

设计确认测试用例的方法主要有传统的黑盒方法、基于情景的方法。

（1）黑盒测试：黑盒测试也称功能测试，它是通过测试来检测每个功能是否都能正常使用。在测试中把程序看作一个不能打开的黑盒子，在完全不考虑程序内部结构和内部特性的情况下，在程序接口进行测试，它只检查程序功能是否按照需求规格说明书的规定正常使用，程序是否能适当地就收输入数据而产生正确的输出信息。黑盒测试着眼于程序外部结构，不考虑内部逻辑结构，主要针对软甲那界面和软件功能进行测试。

（2）基于情景的方法：场景，是一种有假设条件的故事，可以辅助测试人员把一个复杂的问题或系统通过电影那样地过一遍。测试人员通过把整个场景都设想出来，在设想中的场景中进行的测试就是基于场景的测试。

55、为什么应该尽量使用面向对象语言来实现面向对象分析和设计的结果？

面向对象语言充分支持对象、类、封装、继承、多态、重载等面向对象的概念，编译程序能够自动地在目标程序中实现上述概念，因此，把面向对象的设计结果翻译成面向对象程序设计比较容易，这就降低了编程工作量而且减少了编程错误。更重要的是，从面向对象分析到面向对象设计再到面向对象程序设计，始终使用统一的概念，既可以保证在软件开发过程的各个阶段之间平滑过渡，又有助于提高软件的可重用性和可维护性。

56、什么是强类型语言？这类语言有哪些优点？

（1）按照编译时对程序中使用的数据进行类型检查的严格程度，可以把程序设计语言划分为两类。如果语言仅要求每个变量或属性隶属于一个对象，则是弱类型的；如果语法规定每个变量或属性必须准确地属于某个特定的类，则这样的语言是强类型的。

（2）强类型的语言主要有两个优点：一是有利于在编译时发现程序错误；二是增加了优化的可能性。因此，强类型语言提高程序的可靠性和运行效率。

57、用动态联编实现多态性是否会显著降低程序的运行效率？

（1）绝大多数面向对象语言都优化了动态联编时查找多态操作入口点的过程，由于实现了高效率查找，因此并不会显著降低程序的运行效率。

（2）以 C++语言为例，该语言的动态联编时通过使用“虚函数表”实现的。所谓虚函数表就是编译程序替每个使用虚函数的类构造的一个函数指针数组。类中每个虚函数在

表中都有一个表项（即数组元素），它是一个函数指针。注意，如果在派生类中没有重新定义基类的虚函数，又没有通过一般的函数重载屏蔽类的虚函数，则派生类的虚函数表中有指针项指向其继承的基类虚函数。

（3）每个类的实例都有一个隐含的指向该类虚函数表的指针。当执行调用虚函数的语句时，系统首先用调用虚函数的对象的虚函数表指针找到相应类的虚函数表，再由虚函数表中与虚函数名对应的表项找到该虚函数的入口点。为了进一步提高效率，根据虚函数名查找虚函数表中对应表项的过程，可以使用哈希表技术。

（4）从 C++语言实现动态联编的方法可以知道，调用虚函数确实比调用普通函数的开销大一倍，主要是多了读虚函数表指针的操作，实际上开销增加地并不多，虚函数调用只比普通函数调用慢一点点。

58、为什么说参数化类有助于提高可重用性？

（1）在实际的应用程序中，往往有这样一些软件元素（即函数、类等软件成分），从它们的逻辑功能看，彼此是相同的，所不同的主要是处理的对象类型不同。

（2）所谓参数化类，就是使用一个或多个类型去参数化一个类的机制，有了这种机制，程序员就可以先定义一个参数化的类模板（即在类定义中包含以参数形式出现的一个或多个类型），然后在使用时把数据类型作为参数传递进来，从而把这个类模板在不同的应用程序中重复使用，或在同一程序的不同部分重复使用。

59、把策略方法与实现方法分开后，为什么能提高可重用性？

（1）从所完成的功能看，有两类不同的方法。一类方法负责作出决策，提供变元，并且管理全局资源，可称为策略方法。另一类方法负责完成具体操作，但并不作出是否执行这个操作的决定，可成为实现方法。

（2）策略方法通常紧密依赖于具体应用，应用系统不同，策略方法往往也不同。实现方法是自含式算法，相对独立于具体应用，因此，在其他应用系统中也可能重用它们。

（3）为提高可重用性，编程时不要把策略和实现放在同一个方法中，应该把算法的核心部分放在一个单独的具体实现方法中。当开发不同的应用系统时，可以从已有类派生出新的子类，子类从基类直接继承不需修改的实现方法，并且根据需要重新定义策略方法。

60、面向对象软件的哪些特点使得测试和维护变得比较容易？哪些特点使得测试和维护变得比较困难？

（1）封装性使得对象称为独立性很强的模块，理解一个对象所需要了解的元素，大部分都在该对象内部，因此，测试和维护比较容易。对象彼此之间仅能通过发送消息相互作用，不能从外界直接修改对象的私有数据，进一步使得测试和维护变得更容易。信息隐藏确保了对对象本身的修改不会在该对象以外产生影响，从而大大减少了回归错误的数量，因此，这个特点也使得测试和维护变得比较容易。

（2）与封装性和信息隐藏相反，继承性和多态性加大了测试（含调试）和维护的难度。

a.由于派生类继承了它的全部基类的属性和方法，为了理解和修改派生类，必须研究整个继承结构；

b.如果继承结构的上层结点（即基类）发生了某种变化，则这种变化将传递给下层结点（即派生类）。

c.由于多态性和动态联编的存在，如果程序中有调用多态的语句，调试人员或维护人员将不得不研究运行时可能发生的各种绑定，并且对代码运行情况进行跟踪，才能判断出大量方法中的哪一个方法会在代码的这一点被调用。

61、工作量估算

软件估算模型使用由经验导出的公式来预测软件开发工作量，工作量是软件规模的函数，工作量的单位通常是人月（pm）。没有一个估算模型适用于所有类型的软件和开发环境。有：静态单变量模型，动态多变量模型，COCOMO2 模型

COCOMO2 模型给出了 3 个层次的估算模型：应用系统组成模型（主要用于估算构建原型的工作量，模型名字暗示在构建原型时大量使用已有的构件）；早期设计模型（适用于体系结构设计阶段）；后体系结构模型（适用于完成体系结构设计之后的软件开发阶段）。

COCOMO2 使用的 5 个分级因素：项目先例性（指出对于开发组织来说该项目的新奇程度）；开发灵活性（反映出为实现预先确定的外部接口和为了及早开发出产品而增加的工作量）；风险排除度（反映了重大风险已被消除的比例）；项目组凝聚力（表明了开发人员相互协作时可能存在的困难）；过程成熟度（反映了按照能力成熟度模型度量出的项目组织的过程成熟度）。

62、软件质量

特点：（1）软件需求是度量软件质量的基础，与需求不一致就是质量不高。（2）指定的开发标准定义了指导软件开发标准，没有遵守这些准则，会导致软件质量不高。（3）软件满足明确描述的需求，但不满足隐含的需求，那么软件的质量是值得怀疑的。

与软件可靠性的关系：软件质量是软件与明确地叙述的功能和性能需求、文档中明确描述的开发标准以及任何专业开发的软件产品都应该具有的隐含特征相一致的程度。软件可靠性是程序在给定的时间间隔内按照规格说明书的规定成功地运行的概率。

63、软件质量保证措施（SQA）

措施：

（1）基于非执行的测试（也称为复审或评审）：主要用来保证在编码前各阶段产生的文档的质量。

（2）基于执行的测试（即以前讲过的软件测试）：在程序编写完后进行，保证软件质量的最后一道防线。

（3）程序正确性证明：使用数学方法严格验证程序是否对它的说明完全一致。

技术复查（包括走查和审查）的必要性/优点：能够较早发现软件错误，从而可防止错误被传播到软件过程的后续阶段。

走查两种方式：参与者驱动法；文档驱动法。

审查过程的 5 个基本步骤：

（1）综述：由负责编写文档的成员向审查组综述该文档。

（2）准备：评审员仔细阅读文档。

（3）审查：评审组仔细走查整个文档。

（4）返工：文档的作者负责解决在审查报告中列出的所有错误及问题。

（5）跟踪：组长必须确保所提出的每个问题都得到了圆满的解决。

64、能力成熟度模型

CMM 在改进软件过程中所起的作用：指导软件机构通过确定当前的过程成熟度并识别出对过程改进起关键作用的问题，明缺过程改进的方向和策略；通过集中开展与过程改进的方向和策略相一致的一组过程改进活动，软件机构便能稳步而有效地改进其软件过程，使其软件过程能力得到循序渐进的提高。

对能力成熟度划分的原因：对软件过程的改造，是在完成一个又一个小的改进步骤基础上不断进行的渐进过程；这 5 个成熟度等级定义了一个有序的尺度，用以测量软件机构的软件过程成熟度和评价其软件过程能力，这些等级还帮助软件机构把应做的改进工作排出优先次序；成熟度等级是妥善定义的向成熟软件机构前进途中的平台，每个成熟度等级都为软件过程的继续改进提供了一个台阶。

65、为什么成本估算模型中的参数应该根据软件开发公司的历史数据来确定？

每个公司开发的软件类型都不完全相同，此外，每个公司都有不同的经验、习惯、标准和策略，也就是说，不同公司的能力成熟度并不相同，因此，不同公司开发软件的生产率也不相同，显然不能用同样的成本估算参数来估算工作量，而应该根据该公司开发软件的历史数据确定成本估算模型中的参数。

66、为什么推迟关键路径上的任务会延迟整个项目？

关键路径定义为一组任务（称为关键任务），这组任务决定了完成项目所需要的最短时间。如果位于关键路径上的一个关键任务的完成时间被推迟了，则关键路径上的下一个任务的开始时间和结束时间也要相应的延迟。这样依次传递，会波及关键路径上的最后一个任务，从而延迟整个项目。

67、机动时间有何重要？

虽然不在关键路径上的任务并不决定完成项目所需要的最短路径，可以适当延迟一些时间，但是，如果这些任务延迟过久，则整个项目的完成时间也会被延迟。机动时间给出了完成这类任务的时间范围。

此外，在制定进度进化时仔细研究并充分利用工程网络中的机动时间，往往能够安排出既节省资源又不影响最终竣工时间的进度表。

68、假设你被指定为项目负责人，你的任务是开发一个应用系统，该系统类似于你的小组以前做过的那些系统，只不过规模更大且更复杂一些。客户已经写出了完整的需求文档。你将选用哪种项目组结构？为什么？你打算采用哪种软件过程模型？为什么？

由于待开发的应用系统类似于以前做过的系统，开发人员已经积累了较丰富的经验，没有多少技术难题需要攻克。为了减少通信开销，充分发挥技术骨干的作用，统一意志统一行为，提高生产率，加快开发速度，项目组的组织结构以基于主程序员组的形式为宜。针对待开发的系统，客户已经写出了完整的需求文档，项目组又有开发类似系统的经验，因此，可以采用广大软件工程师熟悉的瀑布模型来开发本系统。

69、假设自己被指派为一个软件公司的项目负责人，任务是开发一个技术上具有创新性的产品，该产品把虚拟显示硬件和最先进的软件结合在一起。由于家庭娱乐市场的竞争非常激烈，这项工作的压力很大。应该选择哪种项目组结构？为什么？打算采用哪种（些）软件过程模型？为什么？

（1）由于是技术上具有创新性的产品，所以需要采用民主制程序员组，大家可以集思广益，共同攻关技术难题。

（2）要求把虚拟现实硬件和最先进的软件结合在一起，所以需要采用一种完整而且完美的模型进行开发，所以 RUP 最为合适。

70、假设自己被指派作为大型软件产品公司的项目负责人，工作是管理该公司已经被广泛应用的字处理软件的新版本开发。由于市场竞争激烈，公司规定了严格的完成期限并且对外公布了。应选择哪种项目组结构？为什么？打算采用哪种（些）软件过程模型？为什么？

（1）应该选择现代程序组，因为小组成员都能对发现程序错误持积极、主动的态度。能更好的适应竞争。

（2）大型软件应采用演化模型中的螺旋模型。

71、一个程序能既正确又不可靠吗？请解释你的答案。

所谓软件可靠性，是程序在给定的时间间隔内按照规格说明书的规定成功地运行概率。通常认为，软件可靠性既包含正确性又包含健壮性，也就是说，不仅在预定环境下程序应该能正确地完成预期功能，而且在硬件发生故障，输入的数据无效或用户操作错误等意外环境下，程序也应该能作出适当的响应。

如果一个程序在预定环境下能够正确地完成预期的功能，但是在意外环境下不能作出适当的响应，则该程序就是既正确又不可靠。

72、为什么在开发软件的过程中变化既是必要的又是不可避免的？为什么必须进行配置管理？

在开发软件的过程中，下述原因会导致软件配置项发生变化：新的市场条件导致产品需求或业务规则发生变化；客户提出了新需求，要求修改信息系统产生的数据或产品提供的功能；企业改组或业务缩减，引起项目优先级或软件工程队伍结构变化；预算或进度限制，导致对目标系统的重新定义；发现了在软件开发过程的前期阶段所犯的错误，必须加以改正。

但是，变化也很容易失去控制，如果不能适当地管理和控制变化，必然会造成混乱并产生许多严重的错误。软件配置管理就是在软件的整个生命期内管理和控制变化的一组活动。可以把软件配置管理看做是应用于整个软件过程的软件质量保证活动，是专门用来管理和控制变化的软件质量保证活动。软件配置管理的目标是，使变化更正确且更容易被适应，在必须变化时减少为此而花费的工作量。从上面的叙述可以知道，软件配置管理是十分必要的。