软件测试

需求、设计代码评审

需求、设计代码评审---静态测试

- 主要目标及内容：
  - 评审的方法与技术
  - 需求规格说明书的检查
  - 系统结构与业务逻辑检查
  - 代码风格和规则检查。

本章内容

2.1 软件评审的方法与技术

2.2 产品需求评审

2.3 设计审查

2.4 代码评审

2.1 软件评审的方法与技术

- 评审的方法
- 评审会议
- 评审的技术

什么是软件评审

软件评审是对软件元素或者项目状态的一种评估手段，以确定其是否与计划的结果保持一致，并使其得到改进。

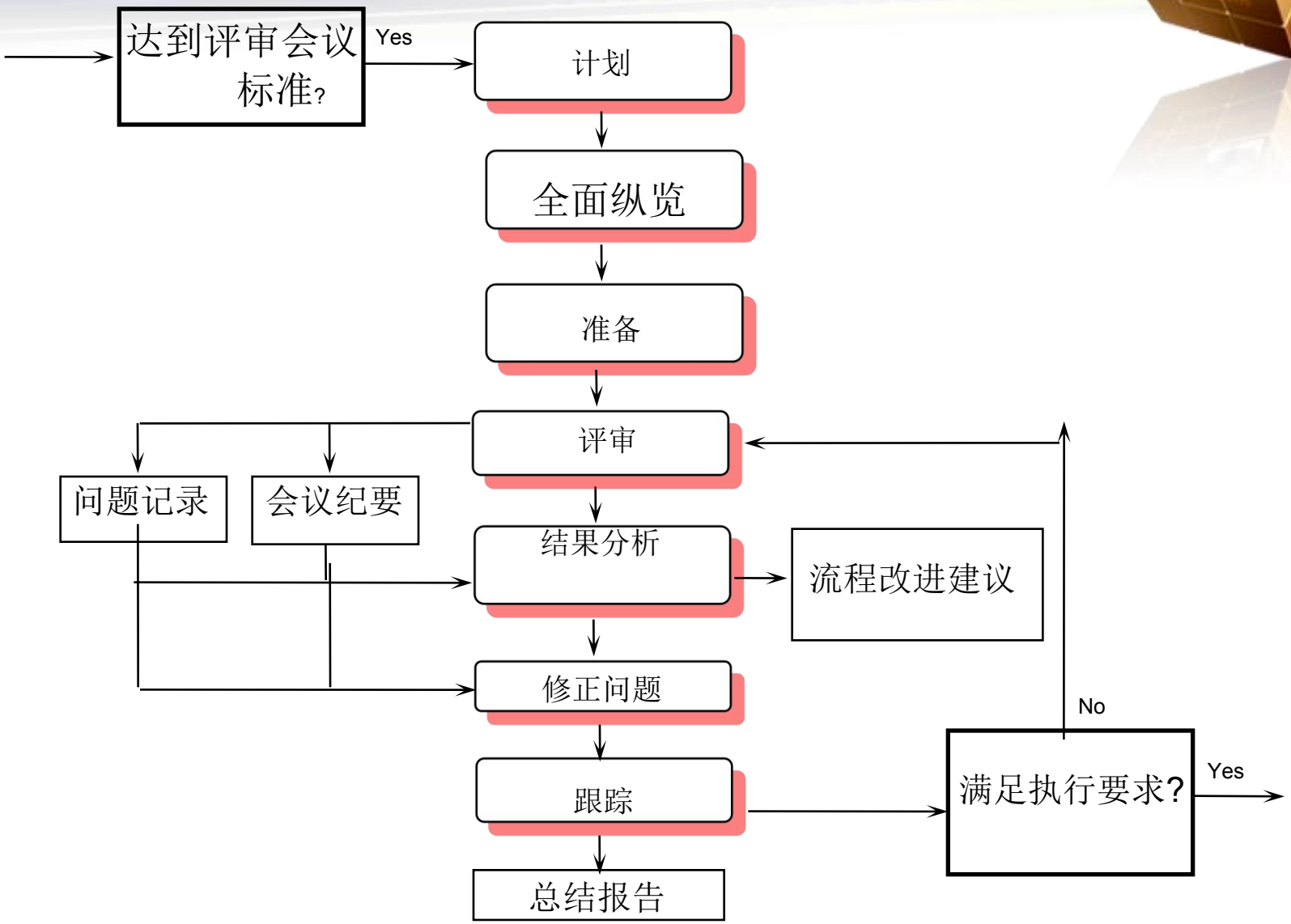
评审方法

- 评审方法包括：
  - 走读（Walkthrough）
  - 小组评审(Team Review)
  - 审查(Inspection)
- 走读、小组评审和审查都有正式评审和非正式评审两种方式

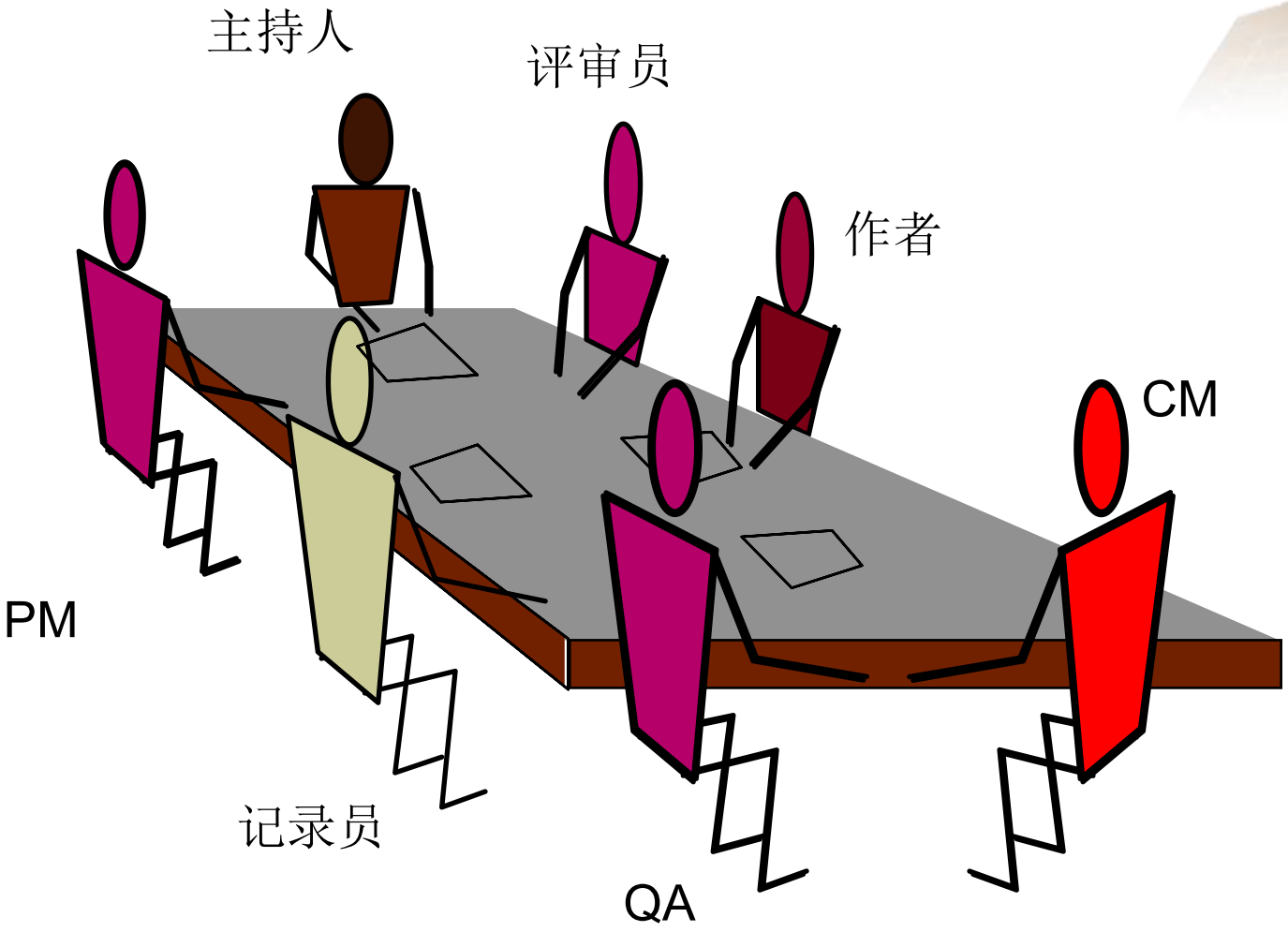
评审的技术

- 检查表
- 场景分析
- 头脑风暴
- 辅助工具（统计、分析工具）

评审会议—会议流程


达到评审会议标准?
Yes
计划
全面纵览
准备
评审
问题记录
会议纪要
结果分析
流程改进建议
修正问题
跟踪
No
满足执行要求?
Yes
总结报告

2.1.4评审会议—*会议角色*



评审会议—角色及职责

- 作者：解释疑问、解决识别的缺陷
- 主持人：从头到尾管理评审过程
- 评审员：识别缺陷和问题
- 记录员：记录识别的缺陷和问题并分发结果
- PM：确定评审计划、确认评审对象资格
- QA：评审过程支持、审计评审过程
- CM：工作产品配置管理

2.2 产品需求评审

2.2.1需求评审的重要性

2.2.2 需求评审的标准

2.2.3需求评审案例

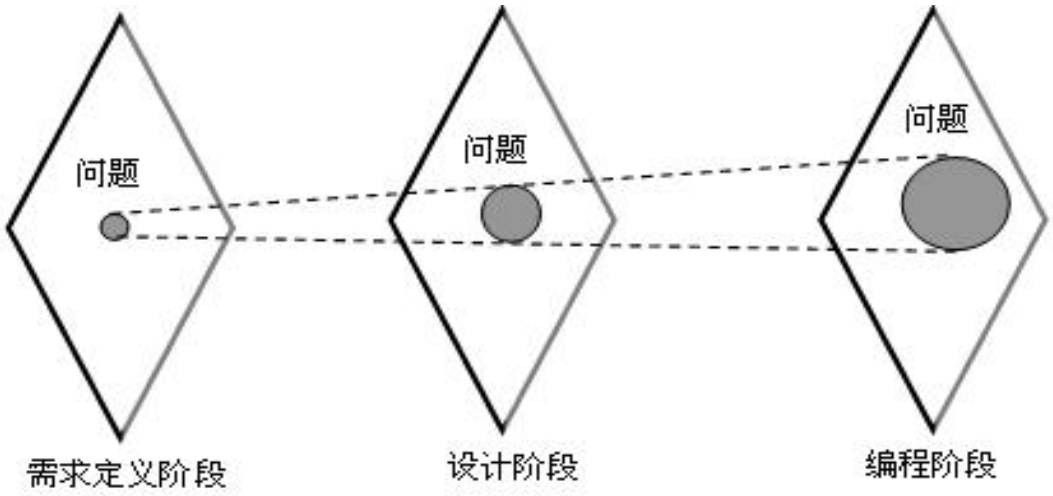
需求模型

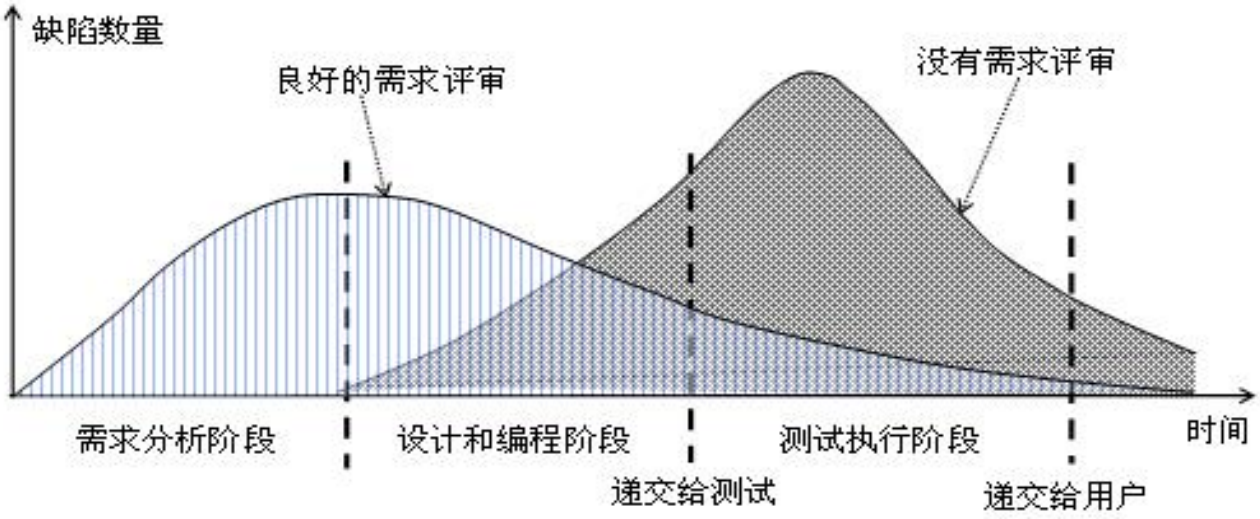
- 需求模型它关注软件的需求，刻画了需求的特征，如用例、行为、功能等。针对一个研究对象可以建立起其多个模型，以从不同的角度和层次来认识对象，如软件需求可以从用例上认识它，也可以从行为上认识它

2.2.1需求评审重要性

- 发现需求定义中的问题，尽早发现缺陷，降低劣质成本。
- 保证软件需求的可测试性。
- 与市场、产品、开发等相关人员在需求理解上认识一致，以免后期的争吵。
- 更好的理解产品的功能性与非功能性需求，为制定测试计划打下基础。
- 确定测试目标与范围。虽然此后需求会发生变更，但能得到有效控制，降低测试风险。

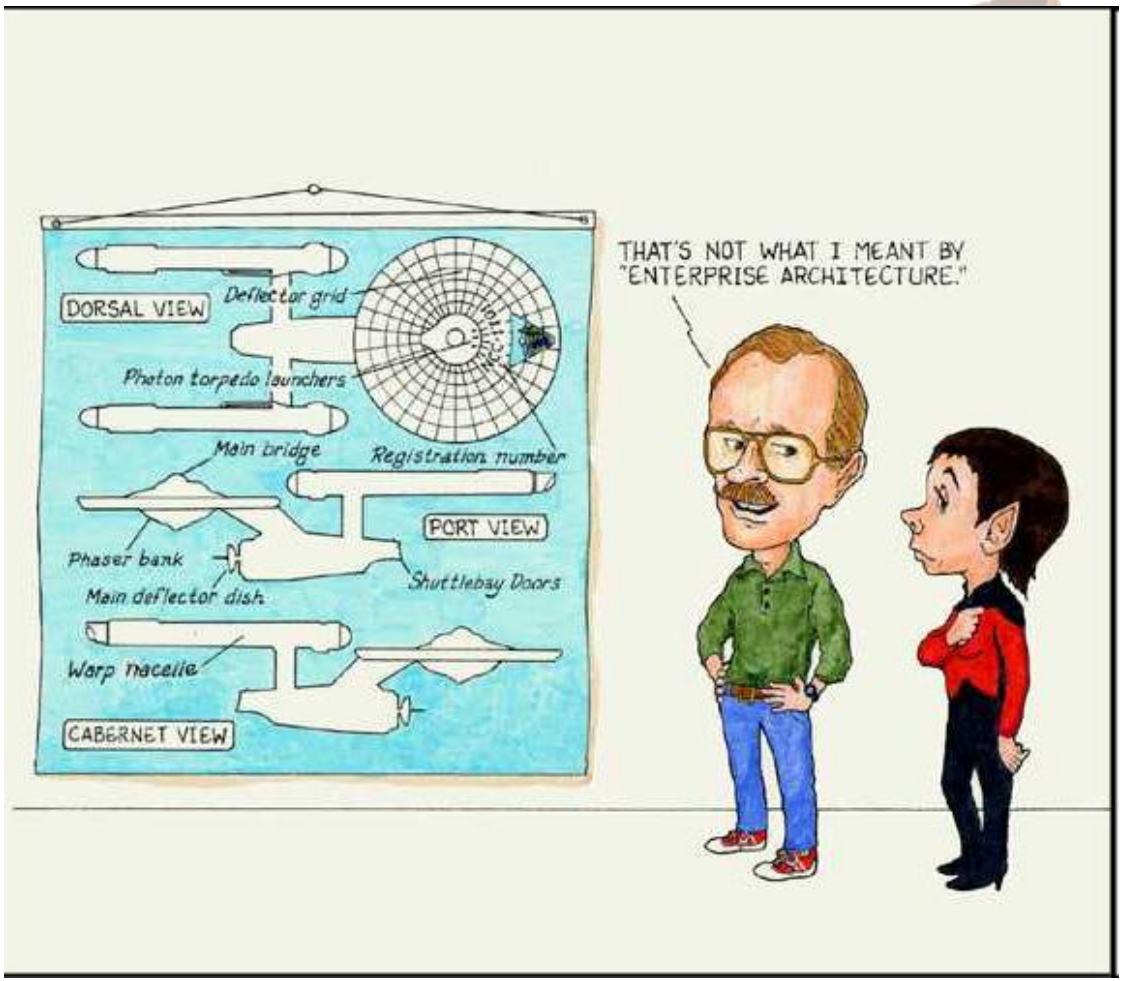
2.2.1 需求评审重要性直观描述





2.2.2需求评审的标准

- 正确性
- 完备性
- 易理解性
- 一致性
- 可行性
- 易修改性
- 易测试性
- 易追溯性



实例

需求1：系统应在不少于每10秒的正常周期内提供状态信息；

- 系统应该以误差上下不超过1秒的10秒间隔，在用户界面的指定位置显示状态信息
- 显示的状态信息应包括如下内容：。。。
- 如果显示状态信息有错误，应提示如下的错误信息：。。。

2.2.2 需求评审标准---用语说明

禁忌词语 ➔ 术语 ➔ 前后置条件匹配 ➔ 上下文

- 禁忌词语：这类词语，不能出现在需求说明书中；否则，优秀需求的特征就不可能得到满足
- 禁忌词语举例
  - 某些、有时、常常、通常、贯常、经常、许多、大多、几乎、用户友好的、容易的、简单的、复杂的、健壮的、无缝的、透明的、优雅的、 最新技术等。这些词语太过模糊，所描述的功能无法验证
  - 等等、诸如此类、依此类推、包括但不限于等。不完整，无法验证
  - 良好、迅速、廉价、高效、灵活、稳定、显著、醒目等。这些是不确定的说法，不可验证
  - 可接受的、足够的、差不多的、可选择的、合理的、充分的、必要的、相关的等
  - 一般情况下、理想情况下、必要时、合适时

示例

- 需求陈述
  - XML解析器应该生成标记出错的报告，这样就可以使XML初学者使用它来迅速找错
- 分析
  - 禁忌词：迅速
  - 何时生成报告？
- 改正
  - 在XML解析器完全解析完一个文件后，该解析器将生成一个出错报告，其内容包括解析文件过程中所发现的所有XML错误的行号及其文本内容，还包括对每个错误的描述
  - 如果在解析过程中未发现任何错误，则不必生成出错报告

术语工具

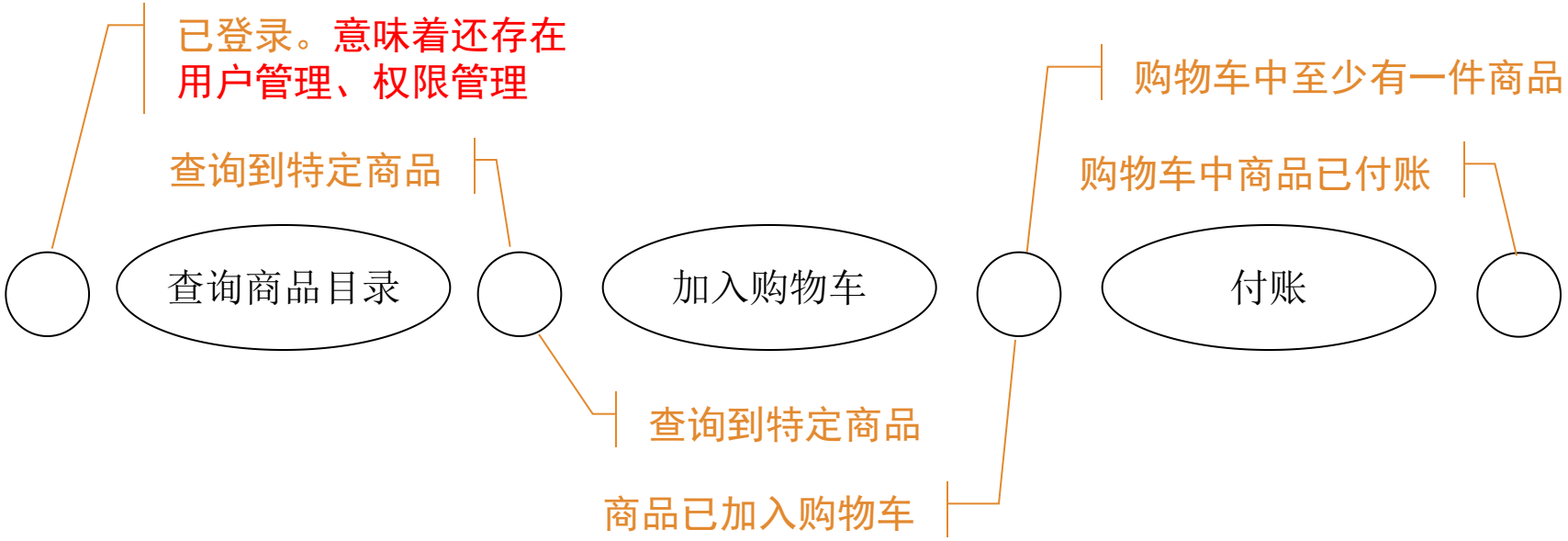
- 术语：需求说明书中出现的术语、名词，其含义如果不包含在以下三种情况中，则意味着需求描述存在问题
  - 约定俗成的
  - 已定义在“术语与缩写”、“数据字典”或“外部接口说明书”中
  - 本文中有解释

示例

- 需求陈述
  - 系统应对登录到其上的无人值守的工作站提供超时安全保护措施
- 分析
  - 系统如何判断“无人值守”？
  - “超时”中的“时”究竟是多长时间？
  - 何谓“安全保护措施”？
- 改正
  - 登录到系统上的工作站，如果超过（300 ±5）秒没有发生任何鼠标或键盘操作，则系统应使得工作站进入屏幕保护状态

前后置条件匹配工具

- 前后置条件匹配：某个用例的前置条件，往往是由另一个（些）用例来设置的，即后者的后置条件就是前者的前置条件



2.2.3 需求评审案例

- 1、某产品经理在主持需求评审会，评审开始时间不长，就被一位主管打断，明确指出此方案与企业业务发展方向不符，不能实施。紧接着其他与会人员纷纷发言表示同意，结果评审会无法继续进行，需求最终被否决。
- 原因何在？

- 问题所在：

  缺乏初步沟通，目标性需求尚未明确，功能性需求和操作性需求无从谈起。

2.2.3 需求评审案例

- 2、某次需求评审会，主要是公司内部相关领域的专家参加，在评审会开始后不久，某专家就对需求中的某个具体问题提出了自己的不同意见，于是，与会人员纷纷就该问题发表自己的意见，大家争执不下，结果，致使会议出现了混乱状况，主持人无法控制局面，会议大大超出了计划评审时间。
- 原因何在？

- 问题所在：
- 　　前期沟通和准备不够，缺乏应对不同意见的准备，难以化解争执。
- 　　主持者不能有效把握会议目标，会议讨论过于关注细节，导致评审失控。

2.2.3 需求评审案例

- 3、某产品经理主持需求评审会，在讲解需求说明书时，与会人员似懂非懂，没有提出任何有价值的问题，致使会议没有得到预期效果，不得不改日重新进行。
- 问题何在？

- 问题所在：
- 　　前期准备和沟通不够，评审变成了培训。
- 　　没有选择合适的评审人员，无法获得有价值的信息。

2.2.3需求评审案例

- 4、某需求评审会，与会人员各抒己见，气氛热烈，产品经理忙于收集意见，结果散会时发现对需求有价值的并不多，并且遗漏了许多要评审的问题，评审效果不佳。与会人员在离开会议室后，私下也认为评审没有多少实际效果，完全是在走过场。
- 问题何在？

- 问题所在：
- 评审缺乏有效依据和规范，不能保证评审的覆盖率和有效性。
- 产品经理没有把握好会议主题，评审变成了头脑风暴。

需求评审常见问题汇总

- 目标性需求没有沟通好，后面的需求变成空中楼阁。
- 缺乏评审的可操作依据，遗漏评审内容。
- 没有作好前期准备工作，导致评审时间长，效率低。
- 没有选择合适的评审人员，无法获得有价值的反馈。
- 参加人员过多，容易陷入细枝末节的讨论，会议演变成一场人人自由的混战。

2.3 设计评审

2.3.1 系统架构设计的评审

2.3.2 组件设计的审查

2.3.3 界面设计的评审

2.3.4 系统部署审查

2.3.1系统设计的评审标准

- 设计技术评审标准。稳定、清晰、合理
- 非功能性质量特性的设计评审要求。安全、性能、稳定、扩展、可靠。
- 评审的输入：体系结构文档、设计规范与指南、风险列表
- 评审的输出：经认可的软件体系结构文档、变更需求、评审记录
- 评审的检查点：软件体系结构、设计模式、部署视图、进程视图、封装体、协议。

2.3.1系统架构设计的审查

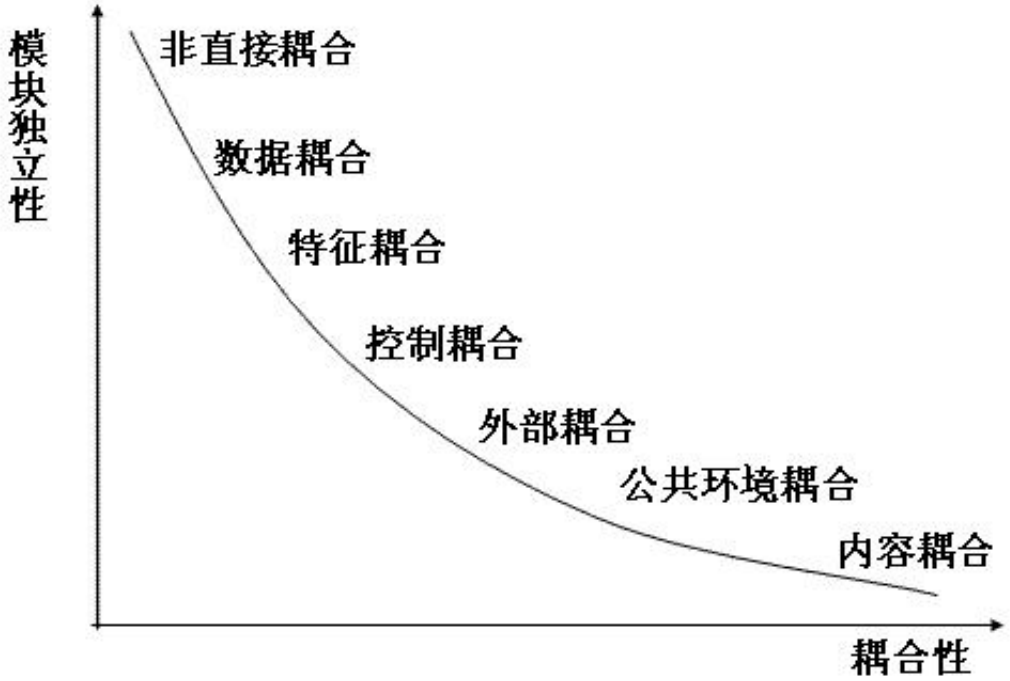
系统架构设计的基本要求就是保证系统具有高性能、高可靠性、高安全性、高扩展性和可管理性 。系统架构设计评审就是保证这些特性在设计中得到充分考虑。

采用分层评审和整体评审相结合，经过整体评审到分层评审、再从分层评审到整体评审的过程，这样既能确保评审的深度，又能确保评审的一致性

- 整个系统不应该存在单一故障点
- 系统是否建立了故障转移机制
- 是否建立了良好的负载平衡机制
- 关键业务 或关键任务 ？

2.3.2组件设计的审查

- 功能和接口定义
- 算法的有效性和优化
- 合理的数据结构、数据流和控制流
- 可测试性 等



2.3.3界面设计的审查

(1) 易懂性、易用性

(2) 一致性和规范性

(3) 美观与协调性

(4) 遵守惯例和通用法则

(5) 独特性

(6) 快捷方式的组合

(7) 自助功能

(8) 错误保护

2.3.4系统部署设计的审查

系统部署设计的审查是基于软件服务的质量目标，用来审查软件部署的目标、策略是否合理，是否得到彻底的执行。

- 着重是否服从和遵守部署设计的技术规范
- 逻辑设计的审查
- 物理设计的审查
- 可用性设计的审查
- 可伸缩型设计的验证
- 安全性设计的验证

设计的评审条目解释

检查要素	检查内容
清晰性	是否所有的单元和进程的设计目的都已文档化
	单元设计，包括数据流、接口描述是否清楚
	单元的整体功能是否描述清楚
完整性	是否描述了所采用的标准
	是否确定了每个单元应用的算法
	是否列出了单元的所有调用
	是否记录了设计继承的历史和已知的风险
规范性	文档是否遵从了公司的标准
	单元设计是否使用了要求的方法和工具
一致性	在程序单元和单元的接口中数据成员的名称是否保持一致
	所有接口之间，接口和接口规格书之间是否保持一致
	详细设计是否同概要设计一致

设计的评审条目解释

检查要素	检查内容
正确性	是否有逻辑错误
	需要使用常量名称的地方是否有错误
	是否所有的条件都被处理（>,=<0，switch case）
	分支所处的状态是否正确
数据	是否所有声明的数据都已经被使用
	定位于单元的数据结构是否已经描述
	如果有对共享数据、文件的修改，对数据的访问是否按照正确的共享协议进行（例如：信号灯）
可靠性	对所有错误情况都做安排了有意义的消息反馈
	特殊情况下的返回码是否和文档中定义的全局返回码一致
	是否考虑了异常情况（如磁盘空间不足、网络掉线）
可追溯性	是否每个详细设计单元都可以追溯到概要设计

设计的评审条目解释

检查要素	检查内容
接口	参数表是否在数量、类型和顺序上保持一致
	是否所有的输入输出都已经正确定义并检查过
	所传递参数的顺序是否描述清楚
	参数传递的机制是否确定
	通过接口传递的常量和变量是否与单元设计的相同（例如：函数中定义的常量不能在所调用的子过程中被修改）
	是否以度量单位描述了参数的值区间、准确性和精度

内容

2.1 软件评审的方法与技术

2.2 产品需求评审

2.3 设计审查

2.4 代码评审

2.4 代码评审

- 2.4.1代码风格与规则检查
- 2.4.2程序设计和结构的检查
- 2.4.3业务逻辑的检查

2.4.1代码风格和规格检查

- 代码风格和规则

  代码风格和规则的检查是在每个程序员完成一个模块或类的时候要进行编码规范的检查。做一个检查表，以表的内容为检查依据，检查表的内容主要是检查的要点。

  - Camel标记法 -- 首字母是小写的，接下来的单词都已大写字母开头。
  - Pascal标记法 -- 首字母是大写的，接下来的单词都已大写字母开头。
  - 匈牙利类型标记法 -- 在以Pascal标记法命名的变量前附加一个小写字母开头(或小写字母序列)，说明该变量的类型，例如，i表示整数，s表示字符串。

2. 4. 1代码风格和规格检查

- Java 命名规范
  - 包(Packages): 包名命名全部小写字母：com.hisense.hiatmp
  - 类(Classes)：类名命名采用Pascal标记法：public class PassImpl
  - 接口(Interfaces): 接口类名以大写“I”开头，命名采用Pascal标记法；示例：Ipass
  - 方法(Methods): 方法名命名采用camel标记法；public List getDirect()
  - 变量(Variables): 变量名命名采用camel标记法；尽量避免单个字符的变量名，除非是一次性的临时变量。临时变量通常被取名为i，j，k，m和n，它们一般用于整型；c，d，e，它们一般用于字符型；
  - 常量(Constants): 常量命名全部大写，单词间用下划线隔开；常量必须是静态、final类型。

2.4.1代码风格和规格检查

- 程序风格检查
  - 程序风格的一致性、规范性；代码必须能保证符合企业规范，保证所有成员的代码风格一致、规范、工整。例如对数组做循环，不要一会儿采用下标变量从下到上的方式（如:for(I=0;I++；I<10)），一会儿又采用从上到下的方式(如:for(I=10;I--;I>0))；应该尽量采用统一的方式，或则统一从下到上，或则统一从上到下。
  - 表示符定义的规范一致性；保证变量命名能够见名知意，并且简洁但不宜过长或过短、规范、容易记忆、最好能够拼读。并尽量保证用相同的表示符代表相同功能，不要将不同的功能用相同的表示符表示；更不要用相同的表示符代表不同的功能意义。

2.4.1代码风格和规格检查

- 程序风格检查
  - 检查方法内部注释是否完整；是否清晰简洁；是否正确的反映了代码的功能，错误的注释比没有注释更糟；是否做了多余的注释；对于简单的一看就懂的代码没有必要注释。
  - 检查注释文档是否完整；对包、类、属性、方法功能、参数、返回值的注释是否正确且容易理解；是否会落了或多了某个参数的注释，参数类型是否正确，参数的限定值是否正确。特别是对于形式参数与返回值中关于神秘数值的注释，如：类型参数 应该指出 1.代表什么，2.代表什么等。对于返回结果集(Result Set)的注释，应该注释结果集中包含那些字段及字段类型、字段顺序等。

2.4.2 程序设计和结构的检查

- 程序设计和结构的检查
  - 检查算法的逻辑正确性；确定所编写的代码算法、数据结构定义（如：队列、堆栈等）是否实现了模块或方法所要求的功能。
  - 模块接口的正确性检查；确定形式参数个数、数据类型、顺序是否正确；确定返回值类型及返回值的正确性。
  - 输入参数有没有作正确性检查；如果没有作正确性检查，确定该参数是否的确无需做参数正确性检查，否则请添加上参数的正确性检查。经验表明，缺少参数正确性检查的代码是造成软件系统不稳定的主要原因之一。

2.4.2 程序设计和结构的检查

- 程序设计和结构的检查（续）
  - 调用其他方法接口的正确性；检查实参类型正确与否、传入的参数值正确与否、个数正确与否，特别是具有多态的方法。返回值正确与否，有没有误解返回值所表示的意思。最好对每个被调用的方法的返回值用显示代码作正确性检查。
  - 出错处理；模块代码要求能预见出错的条件，并设置适当的出错处理，以便在一旦程序出错时，能对出错程序重做安排，保证其逻辑的正确性，这种出错处理应当是模块功能的一部分。若出现下列情况之一，则表明模块的错误处理功能包含有错误或缺陷：出错的描述难以理解；出错的描述不足以对错误定位，不足以确定出错的原因；显示的错误信息与实际的错误原因不符；对错误条件的处理不正确；在对错误进行处理之前，错误条件已经引起系统的干预等。

2.4.2 程序设计和结构的检查

- 程序设计和结构的检查（续）
  - 保证表达式、SQL语句的正确性；检查所编写的SQL语句的语法、逻辑的正确性。对表达式应该保证不含二义性，对于容易产生歧义的表达式或运算符优先级（如：《 、=、 》、 &&、||、++、 --等）可以采用扩号“（）”运算符避免二义性，这样一方面能够保证代码的正确可靠，同时也能够提高代码的可读性。
  - 检查常量或全局变量使用的正确性；确定所使用的常量或全局变量的取值和数值、数据类型；保证常量每次引用同它的取值、数值和类型的一致性。

2.4.3 业务逻辑检查

- 业务逻辑的检查
  - 主要是检查单元功能设计是否满足概要设计要求
  - 模块功能
  - 模块层次结构
  - 模块调用关系

2.4 代码评审-Java

重要性	激活	级别	检查项
总计			
命名			
重要		20	命名规则是否与所采用的规范保持一致？
		20	是否遵循了最小长度最多信息原则？
重要		50	has/can/is前缀的函数是否返回布尔型？
注释			
重要		10	注释是否较清晰且必要？
重要	Y	10	复杂的分支流程是否已经被注释？
		10	距离较远的}是否已经被注释？
		10	非通用变量是否全部被注释？
重要	Y	50	函数是否已经有文档注释？（功能、输入、返回及**其他**可选）
		10	特殊用法是否被注释？

2.4 代码评审-Java

声明、空白、缩进			
		20	每行是否只声明了一个变量？（特别是那些可能出错的类型）
重要		40	变量是否已经在定义的同时初始化？
重要		40	类属性是否都执行了初始化？
		20	代码段落是否被合适地以空行分隔？
	Y	20	是否合理地使用了空格使程序更清晰？
		20	代码行长度是否在要求之内？
		20	折行是否恰当？
语句/功能分布/规模			
		20	包含复合语句的{}是否成对出现并符合规范？
		20	是否给单个的循环、条件语句也加了{}？
		20	if/if-else/if-else if-else/do-while/switch-case语句的格式是否符合规范？
		40	单个变量是否只做单个用途？
重要		20	单行是否只有单个功能？（不要使用；进行多行合并）
重要		40	单个函数是否执行了单个功能并与其命名相符？
	Y	20	操作符＋＋和— —操作符的应用是否复合规范？

2.4 代码评审-Java

规模			
重要		20	单个函数不超过规定行数？
重要		100	缩进层数是否不超过规定？
重要		100	是否已经消除了所有警告？
重要	Y	40	常数变量是否声明为final？
重要		80	对象使用前是否进行了检查？
重要		80	局部对象变量使用后是否被复位为NULL？
重要		70	对数组的访问是否是安全的？（合法的index取值为[0, MAX_SIZE-1]）。
重要		20	是否确认没有同名变量局部重复定义问题?
		20	程序中是否只使用了简单的表达式？

2.4 代码评审-Java

重要	Y	20	是否已经用（）使操作符优先级明确化？
重要	Y	20	所有判断是否都使用了（常量==变量）的形式？
		80	是否消除了流程悬挂？
重要		80	是否每个if-else if-else语句都有最后一个else以确保处理了全集？
重要		80	是否每个switch-case语句都有最后一个default以确保处理了全集？
		80	for循环是否都使用了包含下限不包含上限的形式？（k=0; k<MAX）
重要		40	XML标记书写是否完整，字符串的拼写是否正确?
		40	对于流操作代码的异常捕获是否有finally操作以关闭流对象?
		20	退出代码段时是否对临时对象做了释放处理?
重要		40	对浮点数值的相等判断是否是恰当的？（严禁使用==直接判断）

2.4 代码评审-Java

可靠性（函数）			
重要	Y	60	入口对象是否都被进行了判断不为空？
重要	Y	60	入口数据的合法范围是否都被进行了判断？（尤其是数组）
重要	Y	20	是否对有异常抛出的方法都执行了try...catch保护？
重要	Y	80	是否函数的所有分支都有返回值？
重要		50	int的返回值是否合理？（负值为失败，非负值成功）
		20	对于反复进行了int返回值判断是否定义了函数来处理？
		60	关键代码是否做了捕获异常处理?
重要		60	是否确保函数返回CORBA对象的任何一个属性都不能为null?
重要		60	是否对方法返回值对象做了null检查，该返回值定义时是否被初始化?
重要		60	是否对同步对象的遍历访问做了代码同步?
重要		80	是否确认在对Map对象使用迭代遍历过程中没有做增减元素操作？
重要		60	线程处理函数循环内部是否有异常捕获处理，防止线程抛出异常而退出?
		20	原子操作代码异常中断，使用的相关外部变量是否恢复先前状态?
重要		100	函数对错误的处理是恰当的？

2.4 代码评审-Java

可维护性			
重要		100	实现代码中是否消除了直接常量？（用于计数起点的简单常数例外）
		20	是否消除了导致结构模糊的连续赋值？（如a= (b=d+c)）
		20	是否每个return前都要有日志记录?
		20	是否有冗余判断语句？（如：if (b) return true; else return false;）
		20	是否把方法中的重复代码抽象成私有函数?

2.4 代码评审--C/C++

文件结构		
重要性	审查项	结论
	头文件和定义文件的名称是否合理？	
	头文件和定义文件的目录结构是否合理？	
	版权和版本声明是否完整？	
重要	头文件是否使用了ifndef/define/endif 预处理块?	
	头文件中是否只存放“声明”而不存放“定义”	

2.4 代码评审--C/C++

程序的版式		
重要性	审查项	结论
	空行是否得体？	
	代码行内的空格是否得体？	
	长行拆分是否得体？	
	“{”和 “}”是否各占一行并且对齐于同一列？	
重要	一行代码是否只做一件事？如只定义一个变量，只写一条语句。	
重要	If、for、while、do等语句自占一行，不论执行语句多少都要加“{}”。	

2.4 代码评审--C/C++

程序的版式		
重要性	审查项	结论
重要	在定义变量（或参数）时，是否将修饰符 * 和 & 紧靠变量名？	
	注释是否清晰并且必要？	
重要	注释是否有错误或者可能导致误解？	
重要	类结构的public, protected, private顺序是否在所有的程序中保持一致？	
	……	

2.4 代码评审--C/C++

命名规则		
重要性	审查项	结论
重要	命名规则是否与所采用的操作系统或开发工具的风格保持一致？	
	标识符是否直观且可以拼读？	
	标识符的长度应当符合“min-length && max-information”原则？	
重要	程序中是否出现相同的局部变量和全部变量？	
	类名、函数名、变量和参数、常量的书写格式是否遵循一定的规则？	
	静态变量、全局变量、类的成员变量是否加前缀？	

2.4 代码评审--C/C++

常量		
重要性	审查项	结论
	是否使用含义直观的常量来表示那些将在程序中多次出现的数字或字符串？	
	在C++ 程序中，是否用const常量取代宏常量？	
重要	如果某一常量与其它常量密切相关，是否在定义中包含了这种关系？	
	是否误解了类中的const数据成员？因为const数据成员只在某个对象生存期内是常量，而对于整个类而言却是可变的。	

2.4 代码评审--C/C++

内存管理		
重要性	审查项	结论
重要	用malloc或new申请内存之后，是否立即检查指针值是否为NULL？（防止使用指针值为NULL的内存）	
重要	是否忘记为数组和动态内存赋初值？（防止将未被初始化的内存作为右值使用）	
重要	数组或指针的下标是否越界？	
重要	动态内存的申请与释放是否配对？（防止内存泄漏）	
重要	是否有效地处理了“内存耗尽”问题？	

2.4 代码评审--C/C++

内存管理		
重要性	审查项	结论
重要	是否修改“指向常量的指针”的内容？	
重要	是否出现野指针？例如 （1）指针变量没有被初始化。 （2）用free或delete释放了内存之后，忘记将指针设置为NULL。	
重要	是否将malloc/free 和 new/delete 混淆使用？	
重要	malloc语句是否正确无误？例如字节数是否正确？类型转换是否正确？	
重要	在创建与释放动态对象数组时，new/delete的语句是否正确无误？	

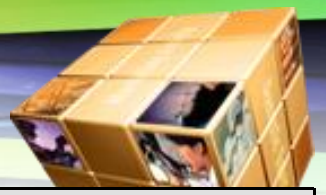
2.4 代码评审--C/C++

C++ 函数的高级特性		
重要性	审查项	结论
重要	是否混淆了成员函数的重载、覆盖与隐藏？	
	运算符的重载是否符合制定的编程规范？	
	是否滥用内联函数？例如函数体内的代码比较长，函数体内出现循环。	
重要	是否用内联函数取代了宏代码？	

2.4 代码评审--C/C++

类的构造函数、析构函数和赋值函数		
重要性	审查项	结论
重要	是否违背编程规范而让C++ 编译器自动为类产生四个缺省的函数：（1）缺省的无参数构造函数；（2）缺省的拷贝构造函数；（3）缺省的析构函数；（4）缺省的赋值函数。	
重要	构造函数中是否遗漏了某些初始化工作？	
重要	是否正确地使用构造函数的初始化表？	
重要	析构函数中是否遗漏了某些清除工作？	
	是否错写、错用了拷贝构造函数和赋值函数？	

2.4 代码评审--C/C++



类的构造函数、析构函数和赋值函数		
重要性	审查项	结论
重要	赋值函数一般分四个步骤：（1）检查自赋值；（2）释放原有内存资源；（3）分配新的内存资源，并复制内容；（4）返回 *this。是否遗漏了重要步骤？	
重要	是否正确地编写了派生类的构造函数、析构函数、赋值函数？注意事项： （1）派生类不可能继承基类的构造函数、析构函数、赋值函数。 （2）派生类的构造函数应在其初始化表里调用基类的构造函数。 （3）基类与派生类的析构函数应该为虚（即加virtual关键字）。 （4）在编写派生类的赋值函数时，注意不要忘记对基类的数据成员重新赋值。	

2.4 代码评审--C/C++

类的高级特性		
重要性	审查项	结论
重要	是否违背了继承和组合的规则？ （1）若在逻辑上B是A的“一种”，并且A的所有功能和属性对B而言都有意义，则允许B继承A的功能和属性。 （2）若在逻辑上A是B的“一部分”（a part of），则不允许B从A派生，而是要用A和其它东西组合出B。	

2.4 代码评审--C/C++

其它常见问题		
重要性	审查项	结论
重要	数据类型问题： （1）变量的数据类型有错误吗？ （2）存在不同数据类型的赋值吗？ （3）存在不同数据类型的比较吗？	
重要	变量值问题： （1）变量的初始化或缺省值有错误吗？ （2）变量发生上溢或下溢吗？ （3）变量的精度够吗？	
重要	逻辑判断问题： （1）由于精度原因导致比较无效吗？ （2）表达式中的优先级有误吗？ （3）逻辑判断结果颠倒吗？	

2.4 代码评审

其它常见问题		
重要性	审查项	结论
重要	循环问题： （1）循环终止条件不正确吗？ （2）无法正常终止（死循环）吗？ （3）错误地修改循环变量吗？ （4）存在误差累积吗？	
重要	错误处理问题： （1）忘记进行错误处理吗？ （2）错误处理程序块一直没有机会被运行？ （3）错误处理程序块本身就有毛病吗？如报告的错误与实际错误不一致，处理方式不正确等等。 （4）错误处理程序块是“马后炮”吗？如在被它被调用之前软件已经出错。	
重要	文件I/O问题： （1）对不存在的或者错误的文件进行操作吗？ （2）文件以不正确的方式打开吗？ （3）文件结束判断不正确吗？ （4）没有正确地关闭文件吗？	

Q & A