

第5章 可信赖性

质量延伸和增强

目录

- 5.1 可信赖性概念
- 5.2 可信赖性方法和技术
- 5.3 可靠性和可用性
- 5.4 安全性原则
- 5.5 密安性原则
- 5.6 生存性
- 5.7 总结

5.1 可信赖性概念

- 5.1.1 可信赖性的起因
- 5.1.2 可信赖性的定义
- 5.1.3 可信赖性的属性

5.1.1 可信赖性的起因

- Laprie分析了1985~1995年期间计算机故障导致灾难的案例，并对其发生故障的原因进行了总结。
- 他将系统发生故障的原因归结为物理的和设计方面的因素。发生故障的范围区分为局部的和分布的。
- 同时指出在系统的建造过程中，哪些质量因素要求考虑的不够充分。这些质量因素，如可靠性、可用性、安全性和保密性等，其重要程度已经远远超出了第4.4节论述的ISO9126一般意义上质量因素的要求。

典型的系统故障所案例的原因分解

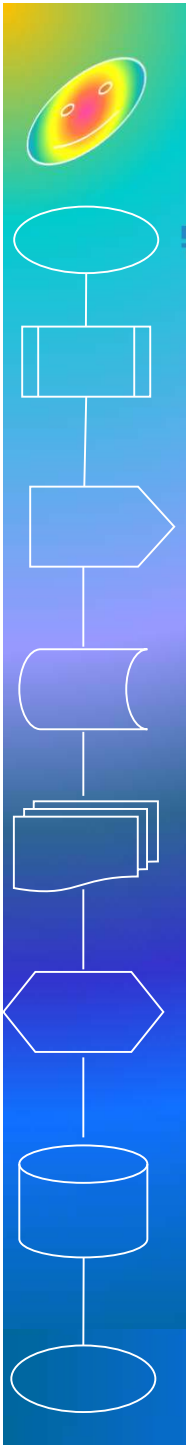
事故	原因			故障范围		可靠性	可用性	安全性	保密性
	物理	设计	交互	局部	分布式				
1980 年 6 月, 北美航空防御(NORAD)假警报	√			√			√		
1981 年: 航天飞机首次推迟发射		√		√			√		
1985 年 6 月~1987 年, X 光机超剂量放射治疗		√		√				√	
1986 年 8 月~1987 年: 黑客对数十台敏感的计算机设施进行攻击		√	√	√					√
1988 年 11 月: 互联网蠕虫病毒发作		√	√		√		√		
1990 年 1 月 15 日: 美国长途电话中断 9 小时		√			√		√		
1991 年 2 月, 海湾战争中爱国者导弹失误		√	√	√			√	√	
1992 年 11 月, 伦敦救护车服务系统崩溃		√	√		√		√	√	
1993 年 6 月 26-27 日, 法国信用卡拒绝使用	√	√					√	√	

5.1.2 可信赖性的定义

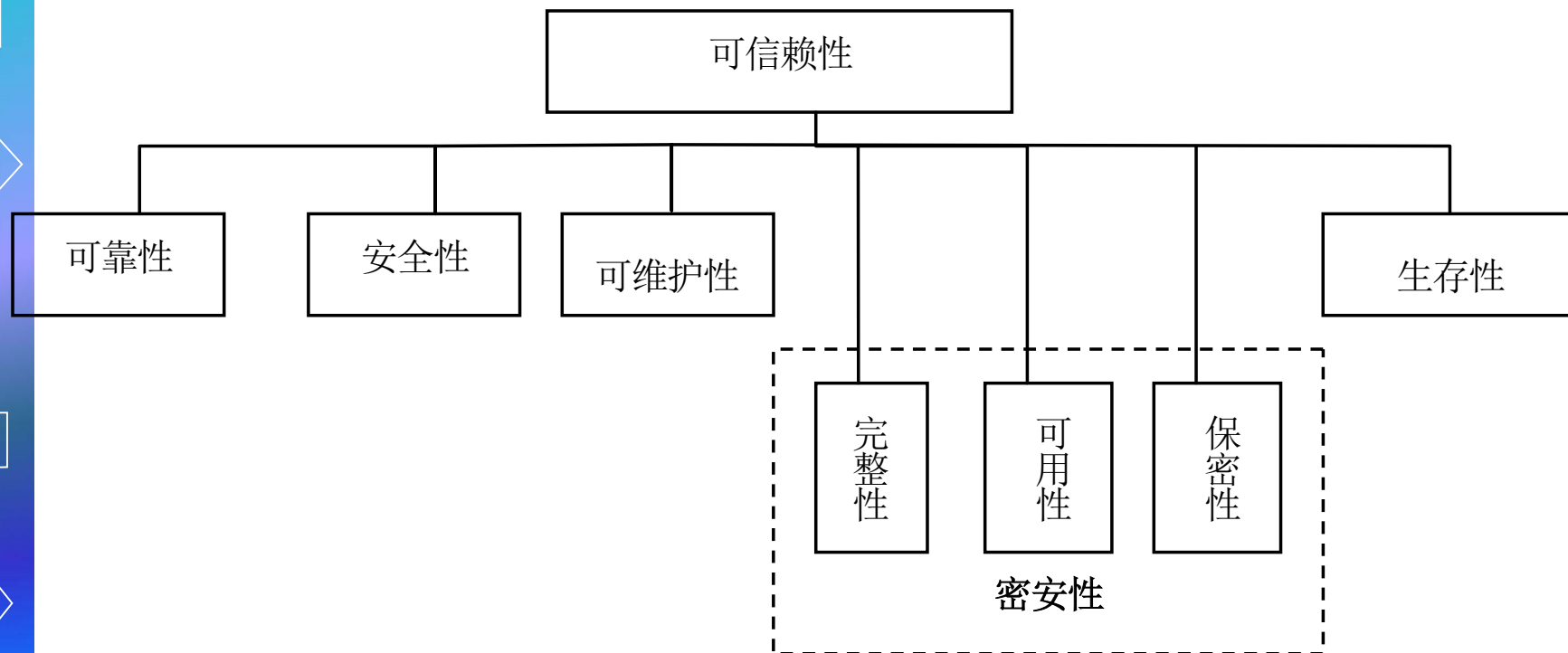
- 可信赖性(dependability)就是指：
 - “人们对基于计算机系统所提供服务的信任程度。”
- 1991年，ISO组织将可信赖性定义为：
 - “用来描述可使用性能和与其关联的因素(可靠性能、可维护性能、和维护支持性能)的集合。”
- CEI将其定义为：
 - “系统在确定的运行和环境条件下，在所定义的时间段内，排他地和正确地执行系统功能的可靠程度。”

5.1.3 可信赖性的属性

- 可信赖性(Dependability)表示计算机及其软件在服务期间的可信赖程度，具体为：
 - 随时可用的特征称为系统的可用性(availability);
 - 系统服务的连续程度形成可靠性(reliability);
 - 不会给环境和人员带来灾难性的后果称为安全性(safety);
 - 不会发生非法的信息泄露称为保密性(confidentiality);
 - 不会发生不适当的修改信息的情况称为完整性(integrity);
 - 修复和进入正常工作状态的能力称为可维护性(maintainability);
 - 而，密安性(Security)，或称为信息安全性(Information Security)是可用性，以及具有授权的完整性和保密性所构成。

- 
- 一个系统是否可信任，取决于在上述基本特性出现的频度，以及系统发生系统故障(failure)时，导致系统功能出现错误(error)后所导致对环境破坏的严重程度，或信息泄露(Security)导致的损失的严重程度。
 - 而系统发生故障的运营通常是系统的硬件、软件、或人为的错失(fault)或潜在的缺陷(defect)所造成的。
 - 生存性(Survivable)，并将其定义为：系统在受到攻击、故障、意外等存在的情况下，及时完成任务的能力。

可信赖性属性分解

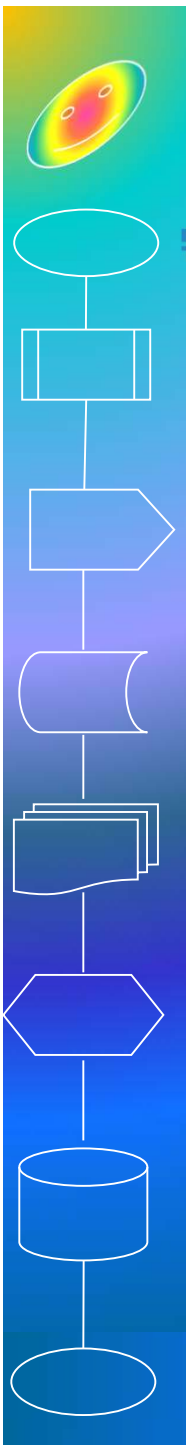


5.2 可信赖性方法和技术

- 5.2.1 开发可信赖系统的基本方法
- 5.2.2 可信赖性的属性讨论

5.2.1 开发可信赖系统的基本方法

- 开发一个可信赖的基于计算机的系统需要综合运用：
 - 防错(fault prevention): 如何防止错误的发生和引入;
 - 容错(fault tolerate): 在系统发生故障的情况下, 仍能保证系统能完成其功能;
 - 排错(fault removal): 如何减少错误的数量和严重错误的存在;
 - 错误预报(fault forecasting): 如何评价错误的现有的、潜在发生的数量, 并评估未来的影响和后果。



- 1) 尽可能降低影响可信赖性的因素：降低系统和软件中的故障、错误和失效几率。
- 2) 采用保证可信赖性的方法：采用防错、容错、排错、错误预报等技术与方法，使系统提供可信赖的服务的能力，并使系统达到性能等方面的质量要求。
- 3) 建立可信任的过程：依据待建立的系统，分析和确定系统的可信赖性的要求，采用能够达到可信赖性的开发过程和度量的要求。将可信赖性要求分解到过程的每个活动，并验证各种活动的正确性。

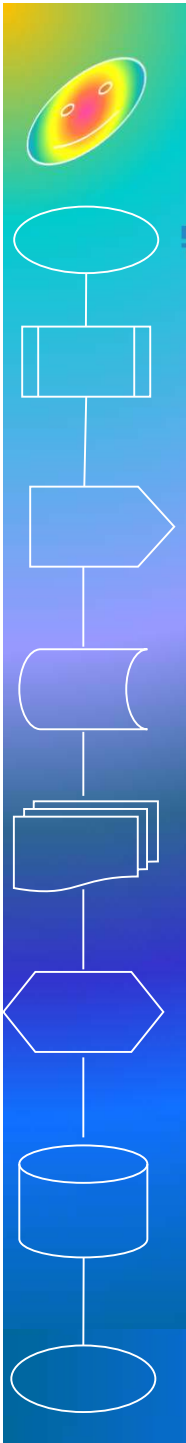
5.2.2可信赖性的属性讨论

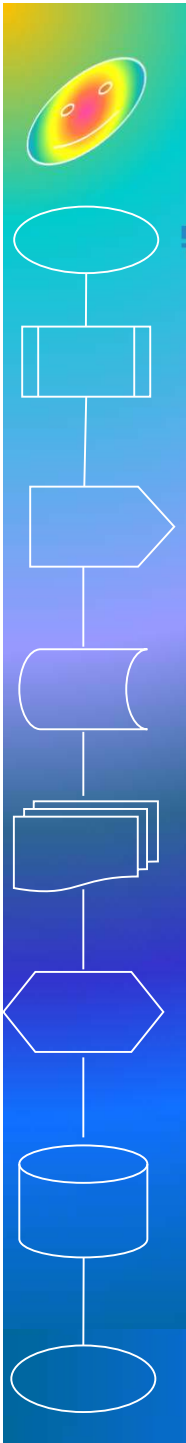
- 系统的完整性

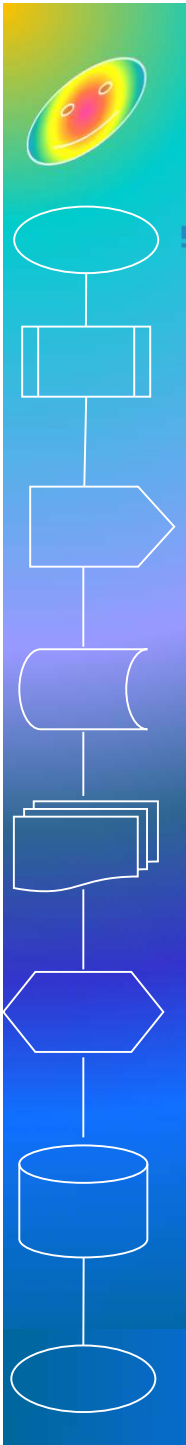
- 即，不允许通过任何手段对系统中的“信息”进行修改。常用的定义如：防止越权修改和删除信息，保证确认的修改。
- “信息完整”既包括了系统的数据和信息的完整性，也包括系统使用规程的完整性，以及软件程序和硬件的完整性。对于偶发性的错误，错误恢复是指“完整地恢复整个系统”。

- 密安性(**security**):

- 是保密性、可用性和完整性的组合。
- 包括：防止非授权用户的存取或访问系统中的信息，以及防止意外事故(如物理错误)等所引起的不希望的信息泄露。

- 
- 可维护性是指系统可以修理和进化(升级)的能力。可维护性不仅仅指保证系统正常运行所进行的日常正确性维护(corrective maintenance), 也包括系统进化的要求:
 - (1)适应性维护(adaptive maintenance)----对系统进行调整, 使之适应环境的变化;
 - (2)完善性维护(perfective maintenance)----根据用户和设计者的要求, 有规律地改进系统的功能。

- 
- 可靠性与可用性是相关联的。
 - 系统的可靠性差，就意味着系统的停机时间长，可用性就差。
 - 密安性与安全性：与密安性不同，安全性是指避免系统发生重大灾难的可能程度。
 - 密安性差的系统不一定导致重大的灾难，但是却是安全性的重要隐患。
 - 例如，恐怖分子和敌人可以篡改未授权的信息，引发(潜在)的安全失效，从而导致灾难性后果的而发生。特别是在网络互连的系统中，密安性成为导致安全性的重大因素。

- 
- **生存性和密安性：**密安性包括保护信息系统完整性、可用性和保密性的要求，指明系统要完整并受到保护。于此相反，生存性表达了在受攻击时，尽管系统的一部分受到损害，系统还能组织可运行部件，继续完成所期望的任务。
 - **生存性和安全性：**传统上，获得安全性的办法是通过设计容错机制，避免和预防系统发生故障时带来的灾难性损失。生存性则要求系统在遭到入侵、失效、或故障时，系统要具有鲁棒性。显然，系统的生存性包括容错，但是不仅仅是容错。

容错设计

- 容错是从发生事故和错误组合对系统统计概率角度出发的，而没有考虑系统被攻击的情况。
 - 容错是指一个系统有多个独立的部件(如三个f1、f2、f3)同时完成一个任务，因此，f1、f2、f3同时出错的概率要远远低于只有一个部件f1的出错概率。
- 但是，当敌方的有意识的进攻时，这种可能性就会增大。敌方会不断地通过各种手段了解系统的内容部知识，有组织地让三个独立软件部件的同时发生失效，从而导致系统发生故障。
- 在传统的容错系统设计和开发中，仅仅靠考虑三种组合的概率事件。但是，如果系统要求较高的生存性，就必须考虑更复杂可能组合情况，而不是简单的容错机制。

信息冗余和系统备份

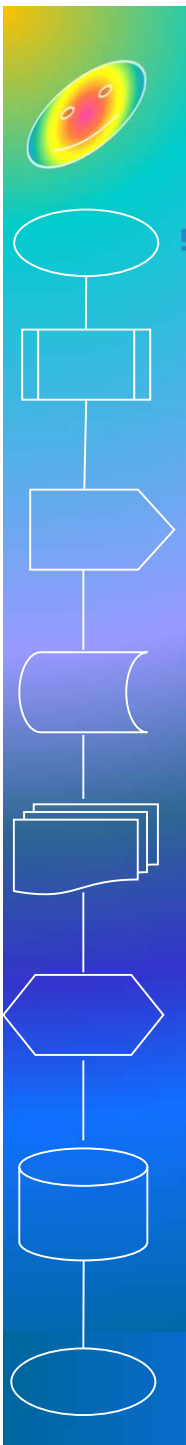
- 信息冗余和系统备份是设计师要考虑的问题，避免重要的信息丢失，从而提升系统生存能力的重要因素。
- 与容错机制一样，仅仅有冗余和备份是不够的。
 - 因为多个同样的系统都会具有同样的脆弱性。
 - 具有高生存性的系统则要求每个备份系统具有同样的功能，但是他们在实现上总是具有一定差异。
 - 可以使用这种差异来阻挡对系统进攻的企图，同时，所有的备份系统据也要具有各自对抗单个进攻的策略。

5.3 可靠性和可用性

- 当我们考虑一个基于计算机的系统时，可靠性的测量元是：
 - 平均失效间隔时间(MTBF -- Mean-Time-Between-Failure)。
 - $MTBF = MTTF + MTTR$
 - 其中，MTTF表示平均失效时间（Mean-Time-To-Failure），MTTR表示平均维修时间（Mean-Time-To-Repair）。
- 可用性 = $(MTTF / (MTTF + MTTR)) \times 100\%$

软硬件可靠性对比

- 提高硬件可靠性的方法是实施冗余和容错策略，例如，多个部件并行工作。其基础是假定每个零部件发生故障的概率是独立的。
- 对于软件来讲，不能把程序模块硬件模块系统做类比。因为，软件模块的故障发生概率不是独立的。用同一个软件(代码)版本做冗余是没有意义的，因为都会一定会出现同样的错误。
- 如果假定一个软件有多个不同的队伍、采用不同的算法、不同的编程语言等，开发出不同的版本，并将这些版本并行和冗余运行，软件的可靠性可以得到提高，因为不同软件版本同时发生一个同样的故障的概率得到了降低。这就是所谓的“软件多版本冗余”的容错体制。



- 软件多版本容错不是很容易能做到的，因为虽然将软件交给不同的开发队伍，他们仍然会采用同样的算法、编程语言、开发步骤，因为大家都会采用相对比较好的开发方法和过程，从而导致不同版本的软件出现同样的错误。
- 另一方面，软件的可靠程度来源于软件开发过程引入和遗留的错误，而不是运行中的随机故障概率。如果能够控制住软件开发中的错误，就能提高软件的可靠性，并预测出软件的可靠程度。
 - 本书的第15章讨论软件故障预测方法，第16章的过程质量控制和20章的过程改进都是基于过程提高软件可靠性的有效途径。

5.4 安全性原则

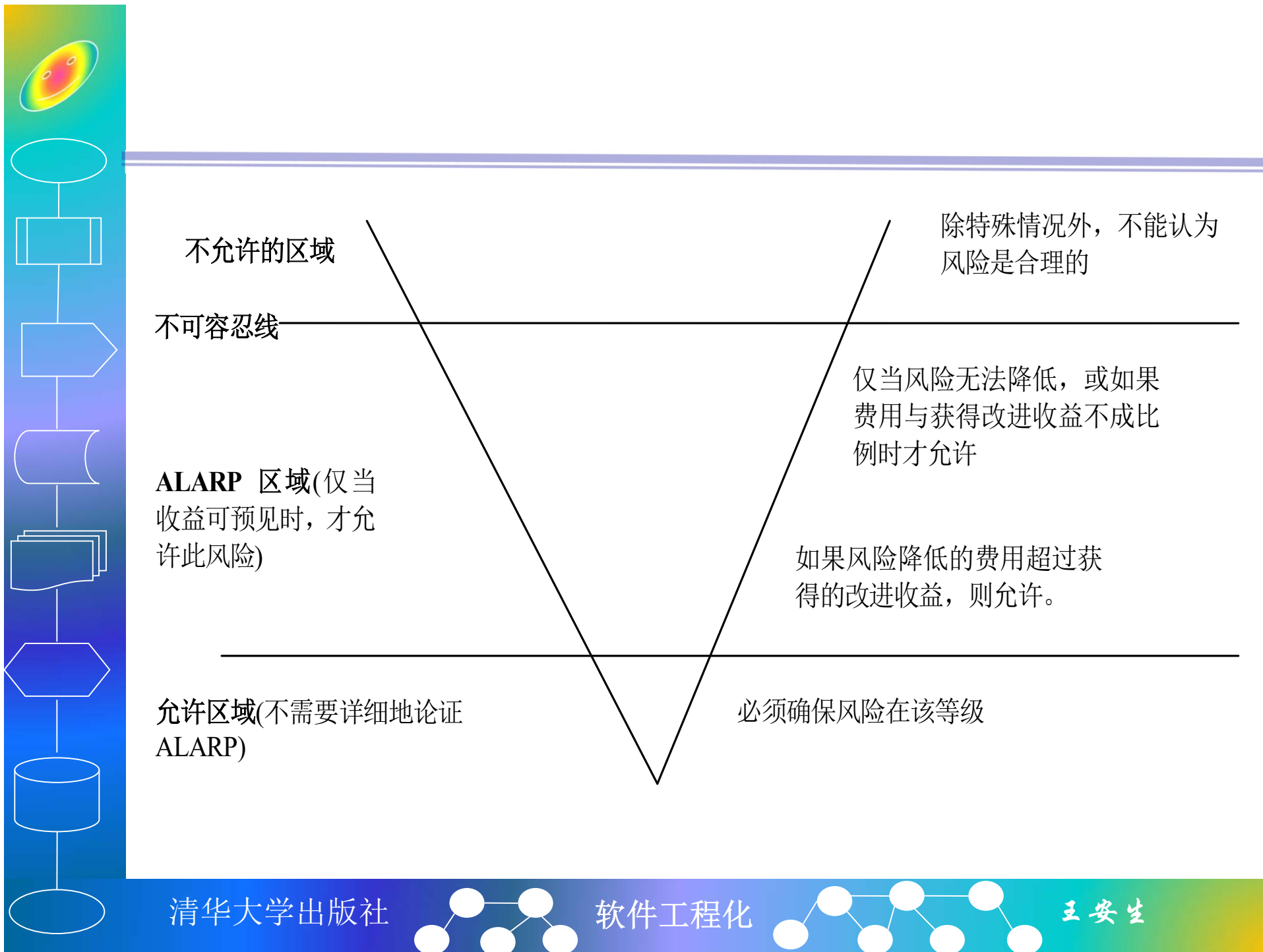
- 5.4.1 ALARP安全原则
- 5.4.2 软件安全证据考虑
- 5.4.3 基于开发过程的证据

5.4 安全性原则

- Sommerville把“安全关键软件”分为两类，
 - 一类是软件嵌入在计算中的，对设备和系统进行控制和处理的情况。这种情况下，软件的错误会导致硬件的错误，并最终导致对人和环境的破坏。
 - 第二类是软件错误直接导致对人和环境的损害。例如，在医院的病人和医药信息管理系统中，软件的错误可能会导致病人用错药，从而造成伤害。
- 第一类的软件问题是通常所指的安全性问题。
 - 这里主要集中讨论第一类系统的安全性原则和证据，或者说，如何表明一个系统是安全的。
- 第二类问题可以纳入到软件完整性和信息的安全性。
 - 第二类问题归结于信息系统安全，在26章26.8节做讨论。

5.4.1 ALARP安全原则

- 英国人从法律的层面提出了“将风险尽可能降到可行(ALARP--as low as reasonably practicable)”，或称为“最低合理可行”原则。
- 依据ALARP原则，要根据风险的严重程度将项目可能出现的风险进行分级。
- 项目风险由不可容忍线和可忽略线为界分为严重风险区(不允许区)、ALARP区和允许区。



- 
- 严重风险区和ALARP区是项目风险辨识的重点所在，风险辨识必须尽可能地找出该区所有的风险。同时要提供确定项目风险的判据标准。

ALARP适用于软件？

- McDermid 提出了“软件安全：证据在哪里 (Software Safety: Where's the Evidence?)”，认为将ALARP运用到软件安全时，必须考虑三个问题：
 - 大部分的技术，例如，非常严格的测试也只是提供风险信息，并没有减少风险(在此意义上，ALARP不适用)。
 - 即使我们假设可以通过分析消除错误，但是我们不能完全预测出错误。因此人们不知道所运用的技术是否真的有意义。无法判断是否存在某种技术可以达到ALARP的要求，即，“将风险降到了合理可行”。
 - ALARP原则隐含着：判断出风险是便宜的，但是降低风险是昂贵的。对于软件情况更是如此：很多软件缺陷是无法检测到的。

5.4.2 软件安全证据考虑

- 一个开发队伍按照要求的开发过程和技术方法进行安全软件的开发，并不能保证软件是绝对安全的。
 - 仅仅说明开发队伍在尽力用“合理可行”方法最大程度地降低安全风险，这个软件开发队伍满足了法律上的责任要求，虽然不能保证最终的软件没有安全风险。
 - 总比根本就没有做到“合理可行”的开发队伍出现错误的可能性要小。
- 必须认识到：绝对软件安全是不存在的，软件安全的目标和责任是“采用最佳的软件工程过程，将风险尽可能降到合理可行(ALARP)。”

5.4.3 基于开发过程的证据

- 软件的失效是由于开发过程所引起的。人们可以通过制定严格开发过程来评价和保证所开发出的软件达到一定的安全性要求。
- 例如，IEC 61508共分为4级，对SIL3级的要求是在连续工作的条件下，每小时发生故障的概率在 10^{-7} 到 10^{-8} 之间。DAL分为5级，其中要求A级软件的飞行一小时出现严重故障的概率要小于 10^{-9} 。基于开发过程的安全证据的基本依据是：
 - 对过程的SIL和DAL等级要求越高，生产出的软件“越好”；
 - SIL和DAL的等级越高，过程的费用越昂贵。因此，在定义安全等级时，不是将等级定义的越高越好。
- 那么，啥样的软件开发队伍才能做到所要求的开发保证等级要求哪？见本书的第四部分。

5.5 密安性原则

- 5.5.1 软件密安性的威胁
- 5.5.2 密安性的误区
- 5.5.3 密安性的公开原则
- 5.5.4 系统密安性的模型
- 5.5.5 产品的密安性需求与认证

5.5.1 软件密安性的威胁

- 1996年，一个瑞典黑客阻塞了佛罗里达州911紧急电话系统，采用的方法是用呼叫----“拒绝服务（denial-of-service）”攻击。
- 1997年，美国国家安全署（National Security Agency）模拟了安全攻击的情况，在4天内成功攻击和接管了芝加哥、洛杉矶、纽约和华盛顿的电力网控制。这个实验唤醒了对信息安全的重视。
- 4年后的2001年，一个业余黑客寻找到了加利福尼亚州的独立系统运行者（California Independent System Operator）的缺陷，该系统控制着该州75%的电力配送。通过攻击接管了两个由防火墙保护的服务器。

被攻击迟早的事

- 美国总统的关键设施保护委员会主任Roger Cressey在华盛顿邮报上发文指出：民用设施，特别是关键设施和服务，一定会受到攻击。只是“时间”问题，而不是“是否”的问题。

5.5.2 密安性的误区

- 在考虑一个系统保密安全时，最直观的想法可能是“通过模糊化得到安全(Security through obscurity)”。
 - 例如许多软件开发者会把一些重要信息放到他所认为的安全地方，并认为“黑客”不会发现这样的秘密，而管理者要求“开发者发誓保守秘密”。
- 这些安全方法在理论上和实际中都是十分脆弱的。
 - 就像在日常生活中，许多人外出时，将钥匙放到门前的鞋垫下面，并相信别人不会猜想到。
 - 这只是“防君子，不防小人”的做法。许多小偷可以轻易地会找到鞋垫下和门框上的钥匙。
 - 但是房子的主人们却会相信“只要钥匙不放到公共场所，小偷就不会知道”。

通过模糊化得到安全？

• 当然“通过模糊化得到安全”总比没有安全意识要好。但是具有明显的缺点：

- 1) 秘密是很难保住的，特别是在计算机系统中，敌人很容易通过对代码的跟踪(例如，反汇编等)方法获得密钥，只是时间的和工作量问题；
- 2) 你所相信的“发誓保守秘密”的人极可能会发起对系统的攻击；
- 3) 保密设计总可以被逆向分析的。法律上严禁对系统进行逆向工程是没有用的，因为敌人和恐怖分子不会遵守法律；
- 4) 一旦系统中许多软件瑕疵被公开，许多模糊的东西很快会被曝光。例如，黑客们不断公布Windows XP中的设计漏洞；
- 5) 多数情况下，设计的不公开性导致没法进行“同行评审”。就像一个数学家宣布证明了某个定理，而却没有经过其它数学家的评审一样，这样的定理证明是不可信的。

• 不公开的密安设计只会降低系统安全的可信度。

5.5.3 密安性的公开原则

- Auguste Kerckhoffs于19世纪确立的原则是：“一个密码系统的设计是安全的，仅当除了密钥信息外，其它都是公开的”。
- 一个密码系统的6个基本原则：
 - 1、系统必须是实用的，除非通过数学手段，否则不可能被解开。
 - 2、系统本身不能是秘密的，即使落入敌人的手中，也不会带来麻烦。
 - 3、密钥必须容易记忆，不需要写下来，并且更改方便。
 - 4、应该支持通过电报传递加密文件。
 - 5、应该便于携带，其使用和功能不需要多人操作。
 - 6、系统使用起来应该非常简单，不需要记忆大量规则或者给使用者造成心理压力。

Shannon的看法

- 香农(Claude Shannon) 将该原则转译为：“（假设）敌人知道系统。”
- 因此在建设一个软件系统时，建设者和使用人员必须假设敌方能够知道我们的系统中的软件代码。
- 这样就可以在采用开源代码和公开加密算法进行软件加密开发的同时，充分考虑系统密安性的程度，并由此确认系统在密安性方面的可信程度。

5.5.4 系统密安性的模型

- 1974年, Lampson提出形式化的访问控制方法。该模型建立一个主体(Subject)集合 S 、一个客体(Object)集合 O 、以及访问模式 A 。从密安性的角度出发, 形成一个矩阵, 其行表示 S , 列表示 O , 矩阵的元素是 A 的子集, 表示是否允许主体访问客体。
- 用这种方法表示“笔和纸”世界密安性可解释为:
 - 分密级的文件是被保护的客体, 阅读和改写文件的人必须是具有相应权限的人(主体)。

文件密安性和访问权限矩阵

密安分类 \ 批准许可	绝密文件	机密文件	秘密文件	内部文件	公开文件
绝密权限人	√	√	√	√	√
机密级权限人		√	√	√	√
秘密权限人			√	√	√
内部员工				√	√
自然人					√

BLP 模型

- 是 Bell和Lapadula最早提出的密安性模型
 - 直观地模拟了“笔和纸”世界的多层安全(MLS-Multi-Level-Security)政策。简要地说，该模型中的主体 S 定义为主动的活动，客体 O 定义为被动的文件。并假定 L 是安全操作等级的晶格，由此，产生一个 $S \cup O$ 到 L 的映射 C 表示出每个主体/客体的许可和密安分类。

BLP 模型

- 更简化一步，假设只有两个操作模式：读和写。那么信息流的策略可以划分为：
 - 1) 简单安全特征：允许一个主体 s 对客体 o 进行读操作，当且仅当 $C(s)$ 可以支配 $C(o)$ 。
 - 即，可以允许被读，只有当密安性级别能够被主体的权限所支配时。一个主体不能读更高级别的客体，这种需求常常表示为：不能向上读(**no read up**)。
 - 2) *安全特征：允许一个主体 s 对客体 o 进行写操作，当且仅当 $C(o)$ 可以支配 $C(s)$ 。
 - 3) 主体不能对客体进行写操作，如果客体的密级比主体的权限低的话，即，不能向下写。

HRU模型

- Harrison, Ruzzo and Ullman(HRU)以Lampson的访问控制矩阵框架为基础提出的模型，丰富了对状态更改活动的原语。
- 特别是，主体可以在状态的基础上增加或删除矩阵中的项，并可以做“行和列”的增加和删除操作。这样，就可以有一个复杂的条件重写体系。
- 但是该模型产生的问题是不可判定性：是否具有确定的权限状态？即，是否能保证对数据有权访问？
- 而BLP模型能满足灵活性、简单性和可判定性。“特定的客体是否可以允许被特定的主体访问？”只需要简单地比较密级分类和权限就行。

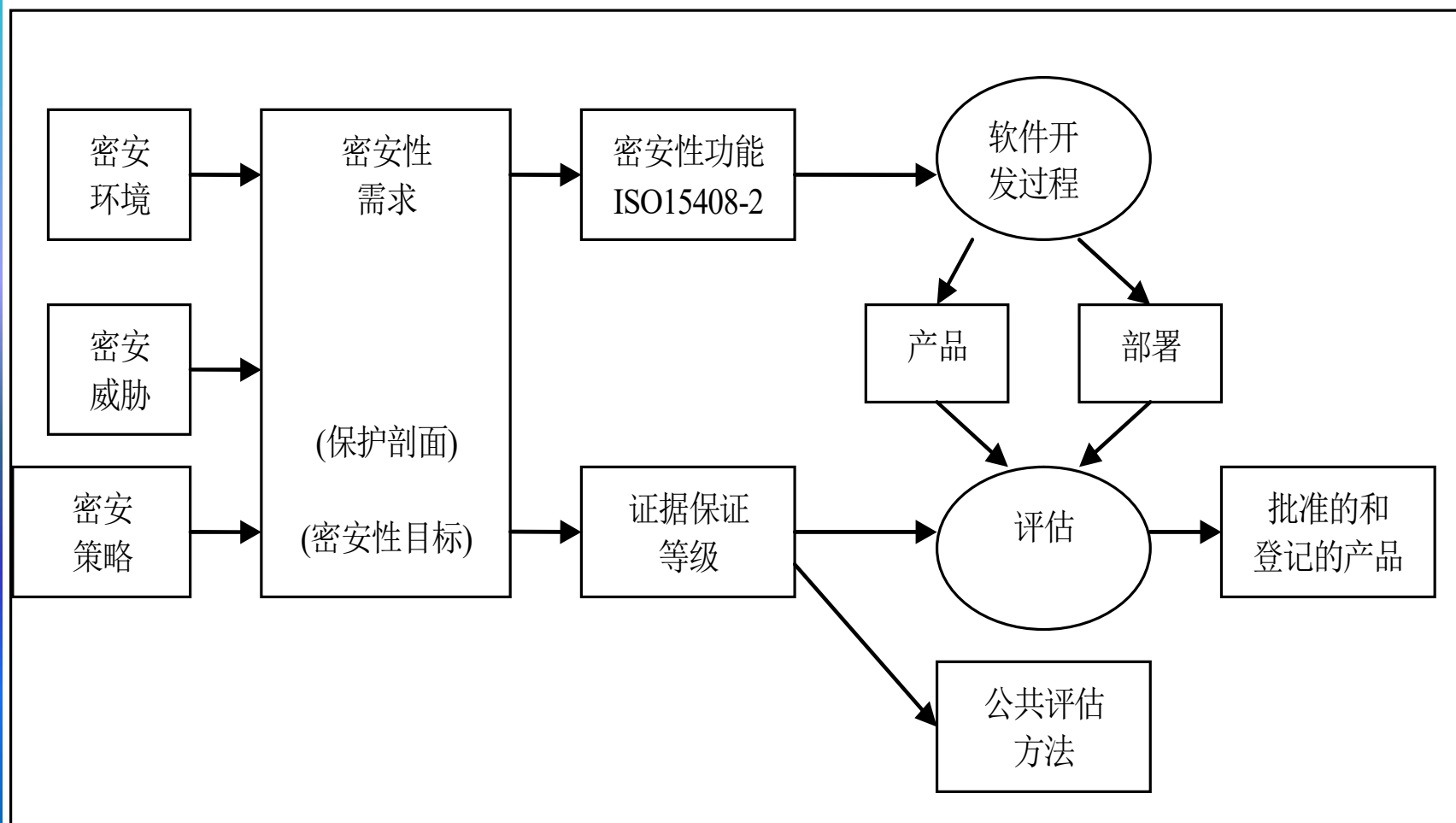
中国院墙(Chinese Walls)

- 要求一个咨询员C不能访问两个竞争公司A和B的敏感信息，就要在法律上约定：如果C访问A的文件，就要阻止他访问B的文件，反之亦然。
 - 这种“进过张家院里，就不能再进李家”的要求可以建立排他性条款，避免客户间潜在的信息冲突。
- 这样的策略可以在HRU模型中用MLS框架建立模型，只需要用动态权限。C具有访问A和B的权限，但是，一旦C访问了A的文件，其访问B的权利就被删除，反之亦然。

5.5.5 产品的密安性需求与认证

- ISO-15408考虑了三个密安性的目标：
 - 对未授权接收者的信息泄露(可信性缺失);
 - 通过未授权的修改进行破坏(完整性缺失);
 - 通过未授权的资产访问的访问进行破坏 (可用性缺失)。

密安产品的评估过程



密安保护等级与开发过程要求

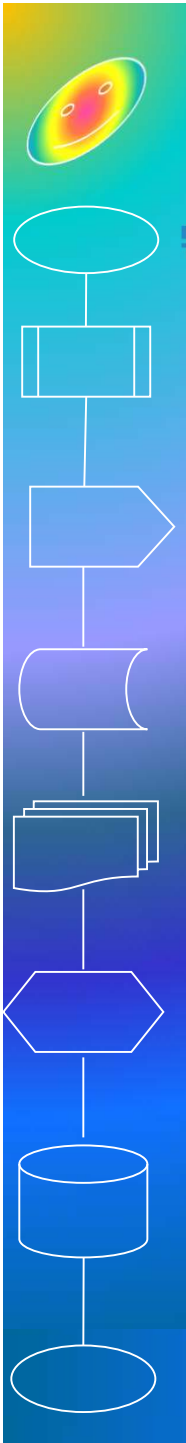
评估保证等级(EAL- Evaluation Assurance Levels)

EAL	安全威胁	对开发过程的要求
1	密安威胁不严重	进行功能化测试。
2	较低或中等的密安保证	必须进行了结构化测试
3	没有明显侵害情况	基于一定的方法进行测试和检查
4	与良好的商业实践匹配	基于一定方法进行了设计、测试、评审。
5	中等密安要求	基于半形式化进行了设计和测试
6	针对严重风险，保护高价值财富。	基于半形式化进行了设计和测试
7	针对极端的高风险和高价值财富	基于形式化进行了设计和测试，

5.6生存性

- CMU/SEI的报告中将生存性定义为：系统在遭受攻击、发生失效或灾难的情况下，能按时完成其任务的能力。

关键特性	含义	例子
1)系统阻挡进攻的能力	反击进攻的策略	用户授权和甄别，程序的多样化
2)系统识别被攻击和损坏程度的能力	判断进攻(包括入侵)和理解系统当前状态，包括评估损害的程度	入侵使用模式的识别
3)被攻击后，系统全部和基本服务恢复能力	恢复承诺的信息和功能的策略，限制损害的程度。 维护或必要时在规定的时间内，及时恢复基本服务。 条件允许时，恢复全部服务。	数据的复制和备份； 数据的重新初始化。
4)降低未来被攻击的的适应性和进化方法	从入侵中所学到的知识，并以此为基础，改进系统生存的策略。	引进识别入侵的新模式。

- 
- 现在的民用系统和军事系统一样，都会遭受恐怖分子和敌对势力主动攻击。
 - 因此在设计一个软件系统时，必须考虑系统被攻击的可能性，以及被攻击后的生存能力。
 - 而不能只关注软件系统的功能和一般意义上的质量(ISO-9126)。

5.7总结

- 任何一个基于计算机的或软件的系统，不仅仅需要一般意义上(ISO-9126定义的)质量属性，仍需要考虑可信赖性的要求。
- 这些属性要求决定了系统是否会泄露具有密级要求的信息和数据(密安性)，是否会发生或避免灾难(安全关键特征)、系统能否在敌方的攻击或自然等原因情况下能够生存下来继续提供服务(生存性)。