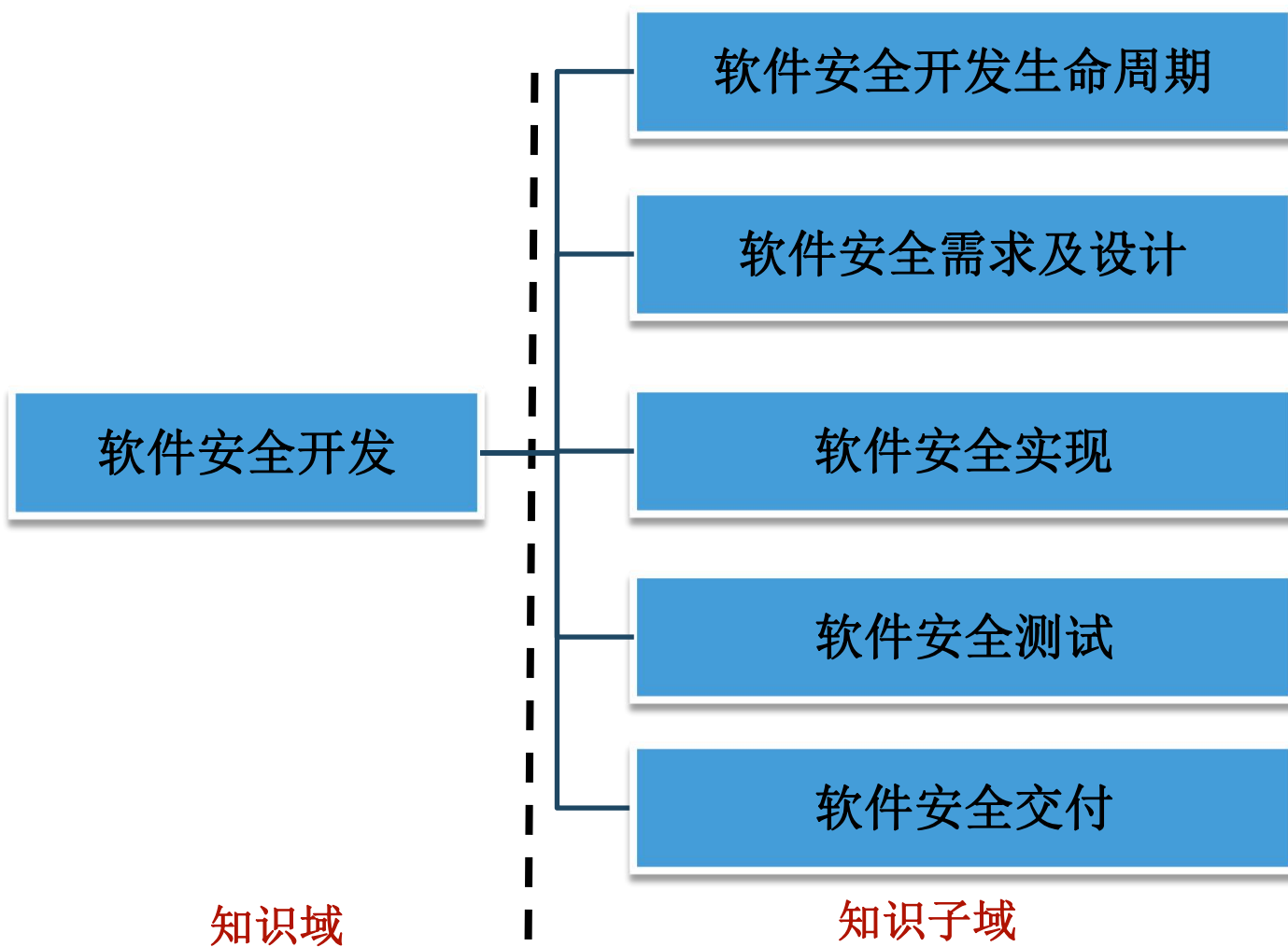
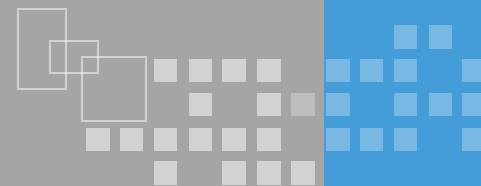




# 软件安全开发

版本：4.2

河南信安世纪 齐文振



## ❖ 软件生命周期模型

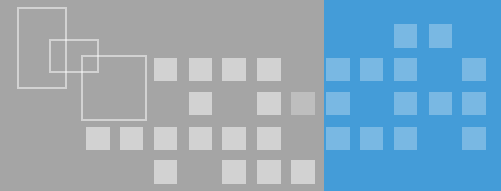
- 了解软件生命周期的概念及瀑布模型、迭代模型、增量模型、快速原型模型、螺旋模型、净室模型等典型软件开发生命周期模型。

## ❖ 软件危机与安全问题

- 了解三次软件危机产生的原因、特点和解决方案；
- 了解软件安全和软件安全保障的基本概念。

## ❖ 软件安全生命周期模型

- 了解SDL、CLASP、CMMI、SAMM、BSIMM等典型的软件安全开发生命周期模型。



## ❖ 软件的定义

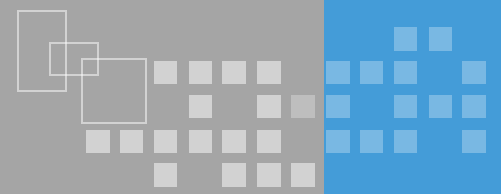
- 软件是与计算机系统操作有关的计算机程序、规程、规则，以及可能有的文件、文档及数据

## ❖ 软件全生命周期（SDLC），是软件从生产直到报废的生命周期。

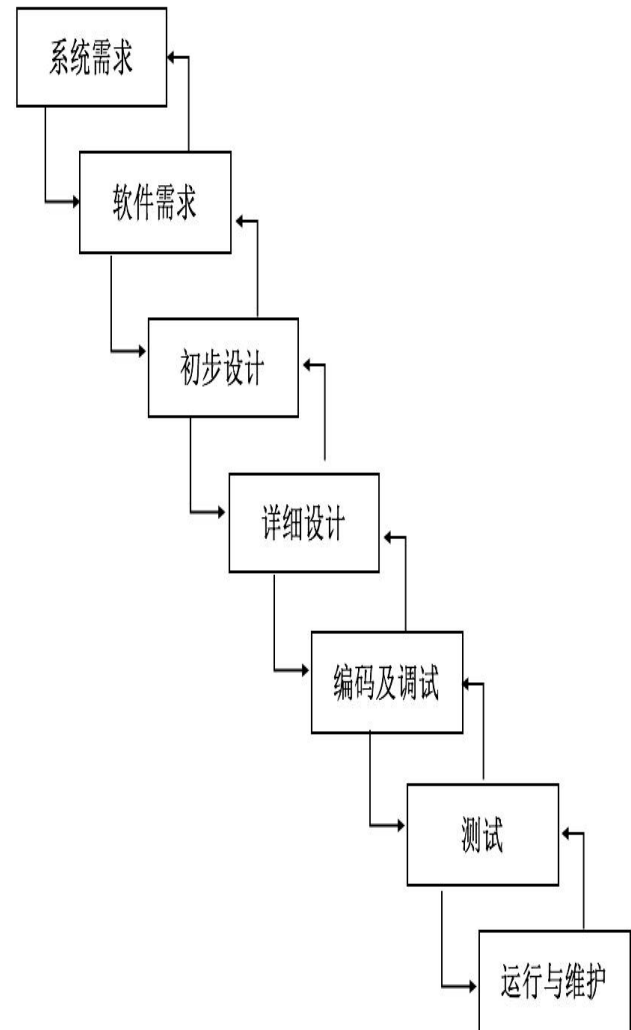
## ❖ 软件生命周期模型

- 瀑布模型
- 迭代模型
- 增量模型
- 快速原型模型
- 螺旋模型
- 净室模型

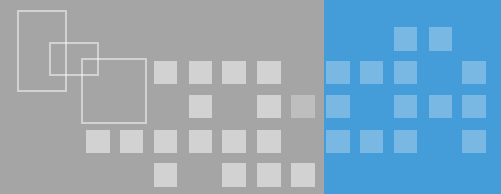
# 软件生命周期模型-瀑布模型



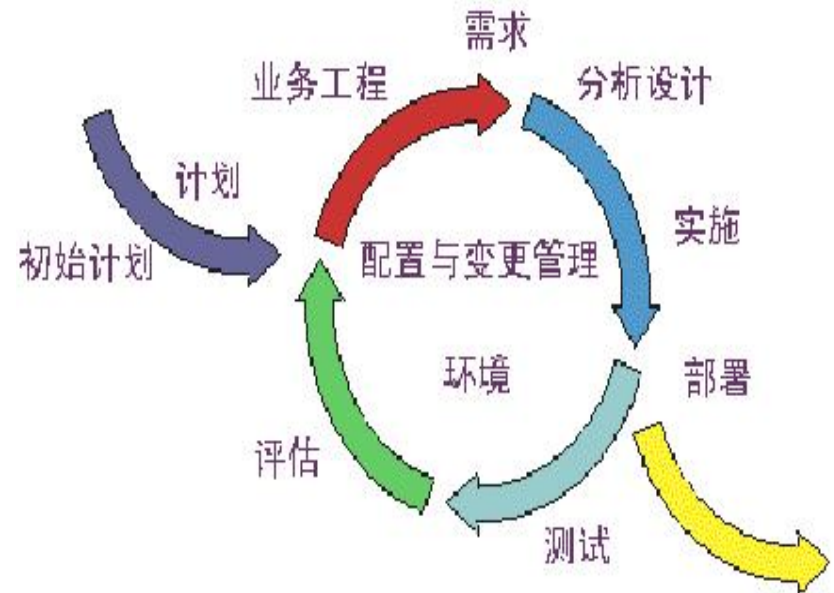
- ❖ 最早出现的软件开发模型
- ❖ 核心思想
  - 按工序将问题简化
  - 将功能的实现与设计分开
- ❖ 不足
  - 没有对开发周期后期发现错误做出相应的规定



# 软件生命周期模型-迭代模型



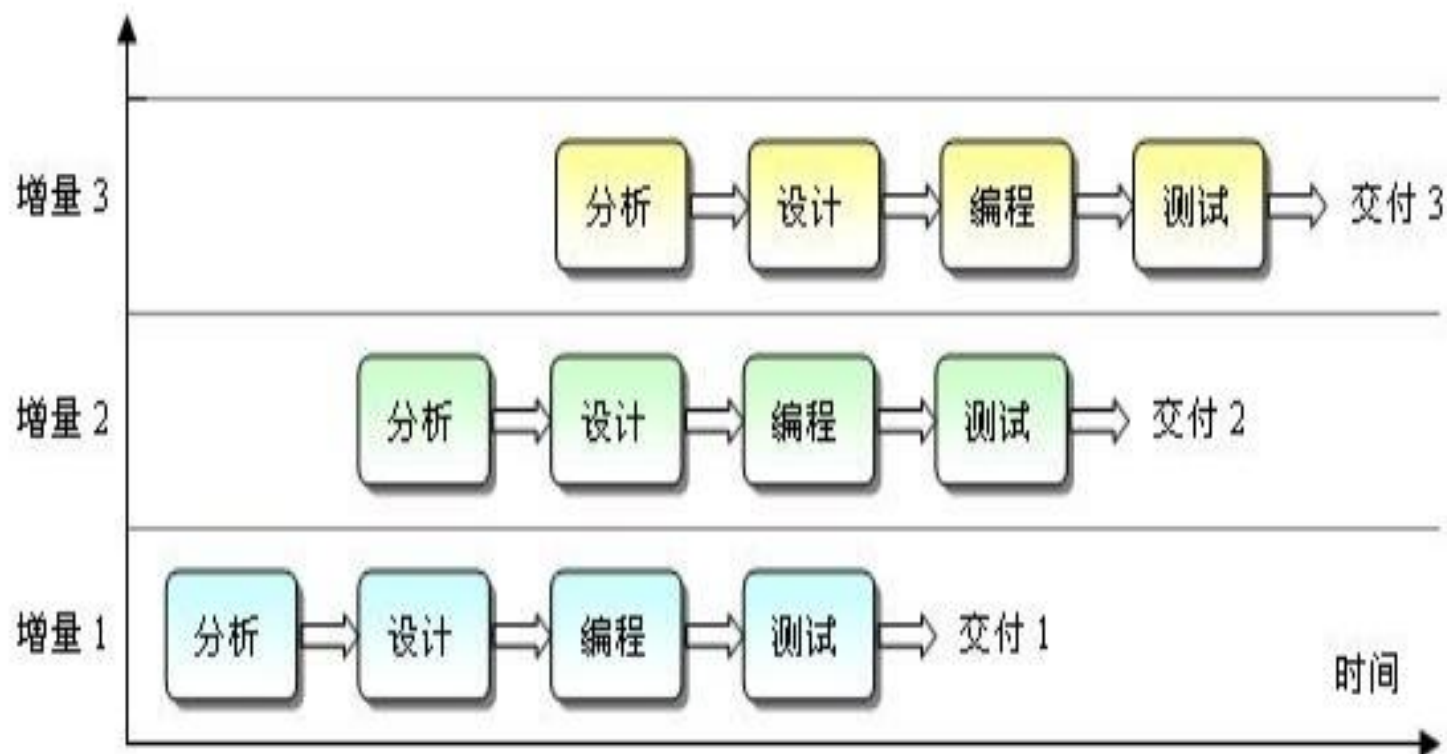
- ❖ 瀑布模型的小型化应用
- ❖ 完整的工作流程
- ❖ 降低风险
  - 增量开支的风险
  - 产品无法按期进入市场的风险
- ❖ 加快开发进度
  - 任务清晰
  - 需求更容易随需而变



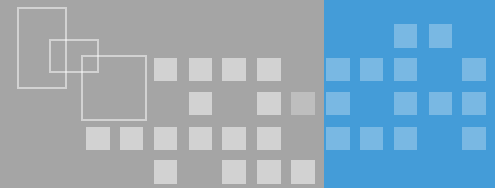
# 软件生命周期模型-增量模型



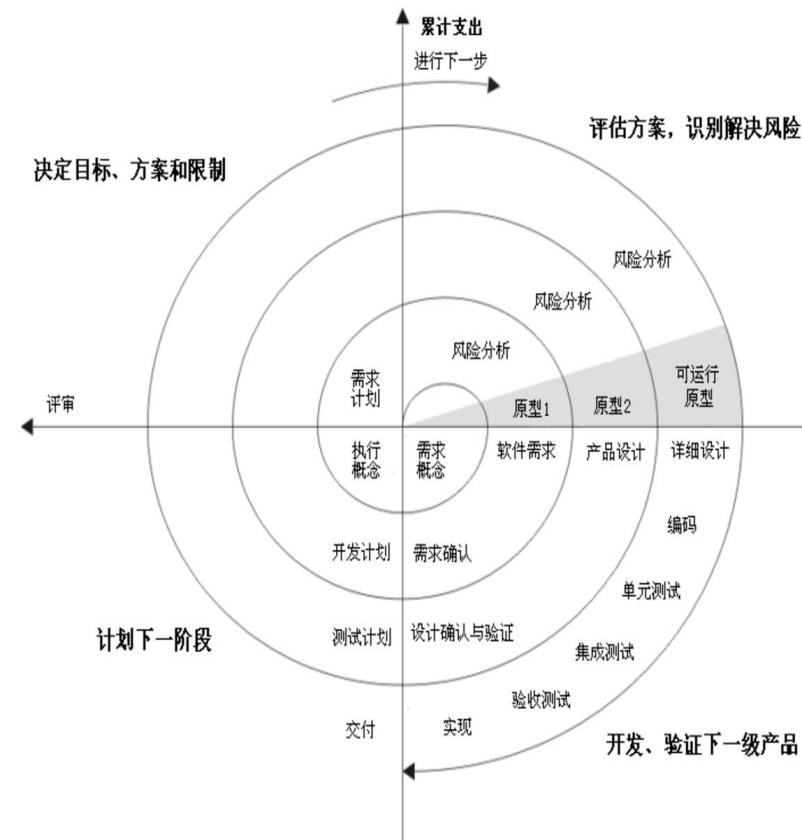
- ❖ 融合了瀑布模型和迭代模型的特征
- ❖ 本质上是迭代，每个增量发布一个可操作产品



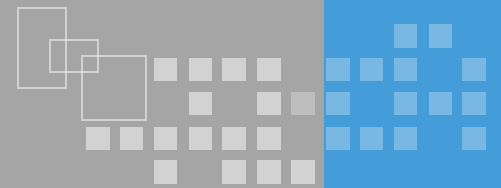
# 软件生命周期模型-螺旋模型



- ❖ 兼顾快速原型的迭代的特征以及瀑布模型的系统化与严格监控
- ❖ 引入了其他模型不具备的**风险分析**，使软件在无法排除重大风险时有机会停止，以减小损失
- ❖ 构建原型是螺旋模型用以减小风险的途径







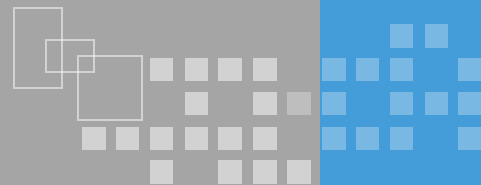
## ❖ 快速原型模型

- 快速原型模型又称原型模型，它是增量模型的另一种形式；它是在开发真实系统之前，构造一个原型，在该原型的基础上，逐渐完成整个系统的开发工作。

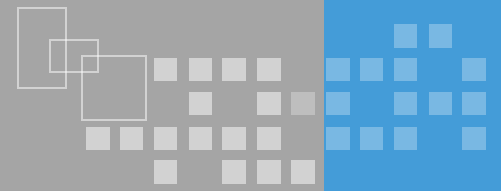
## ❖ 净室模型

- 净室是一种应用数学与统计学理论以经济的方式生产高质量软件的工程技术。力图通过严格的工程化的软件过程达到开发中的零缺陷或接近零缺陷。

# 各个软件过程模型的特点



模型名称	技术特点	使用范围
瀑布模型	简单，分阶段，阶段间存在因果关系，各个阶段完成后都有评审，允许反馈，不支持用户参与，要求预先确定需求	需求易于完善定义且不宜变更的软件系统
迭代模型	不要求一次性地开发出完整的软件系统，将软件开发视为一个逐步获取用户需求、完善软件产品的过程	需求难以确定、不断变更的软件系统
增量模型	软件产品是被增量式地一块块开发的，允许开发活动并行和重叠	技术风险较大，用户需求较为稳定的软件系统
快速原型模型	不要求需求预先完备定义，支持用户参与，支持需求的渐进式完善和确认，能够适应用户需求的变化	需求复杂、难以确定、动态变化的软件系统
螺旋模型	综合瀑布模型、快速原型和迭代模型的思想，并引进了风险分析活动	需求难以获取和确定、软件开发风险较大的软件系统
净室模型	高质量和高可靠性软件的方法，统计过程控制下的增量开发，基于函数的规范、设计和验证、以及统计测试和认证。	质量要求比较高的项目



## ❖ 第一次“软件危机” - 20世纪60年代

- 根源：日益庞大和复杂的程序对开发管理的要求越来越高
- 解决：软件工程

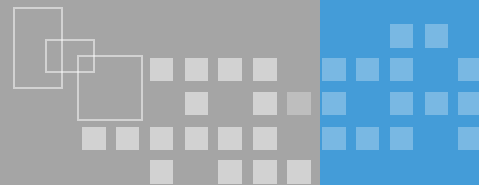
## ❖ 第二次“软件危机” - 20世纪80年代

- 根源：软件规模继续扩大，程序数百万行，数百人同时开发，可维护性难
- 解决：面向对象语言-C++/java/c#

## ❖ 第三次“软件危机” - 21世纪头十年

- 根源：软件安全
- 解决：软件安全开发生命周期管理

# 软件缺陷普遍存在



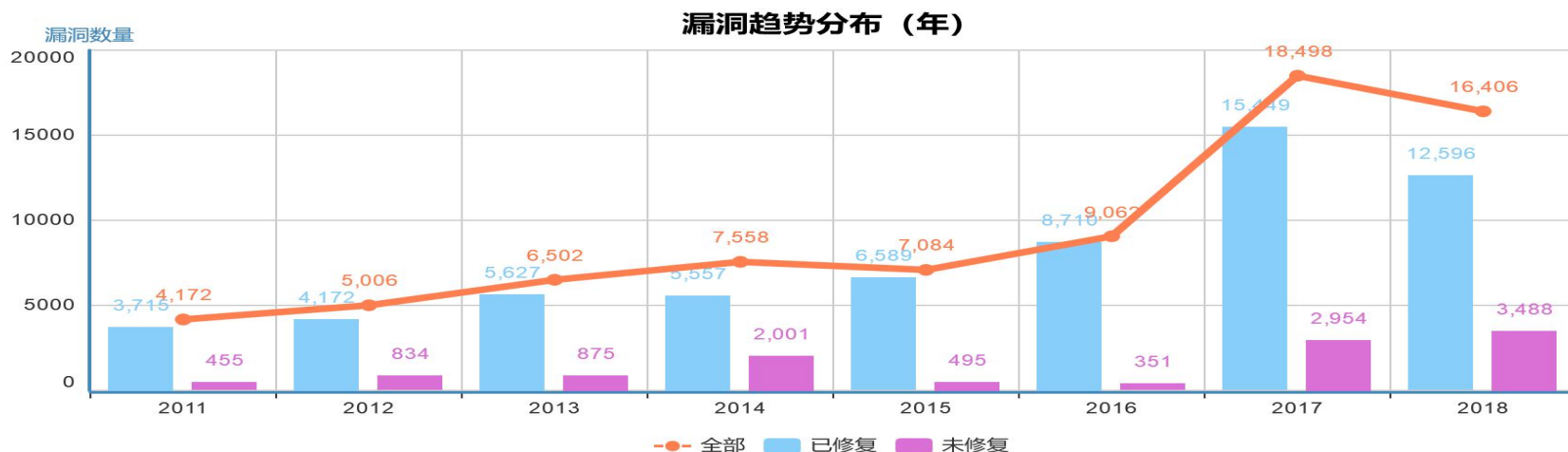
## ❖ 千行代码缺陷数量

- 普通软件公司：4~40
- 高管理软件公司：2~4
- 美国NASA软件：0.1

CMMI 分级标准

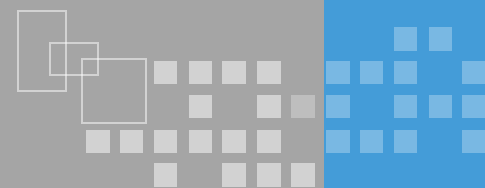
Thousands of lines of code defects	
CMMI1	11.95‰
CMMI2	5.52‰
CMMI3	2.39‰
CMMI4	0.92‰
CMMI5	0.32‰

## ❖ 漏洞数量

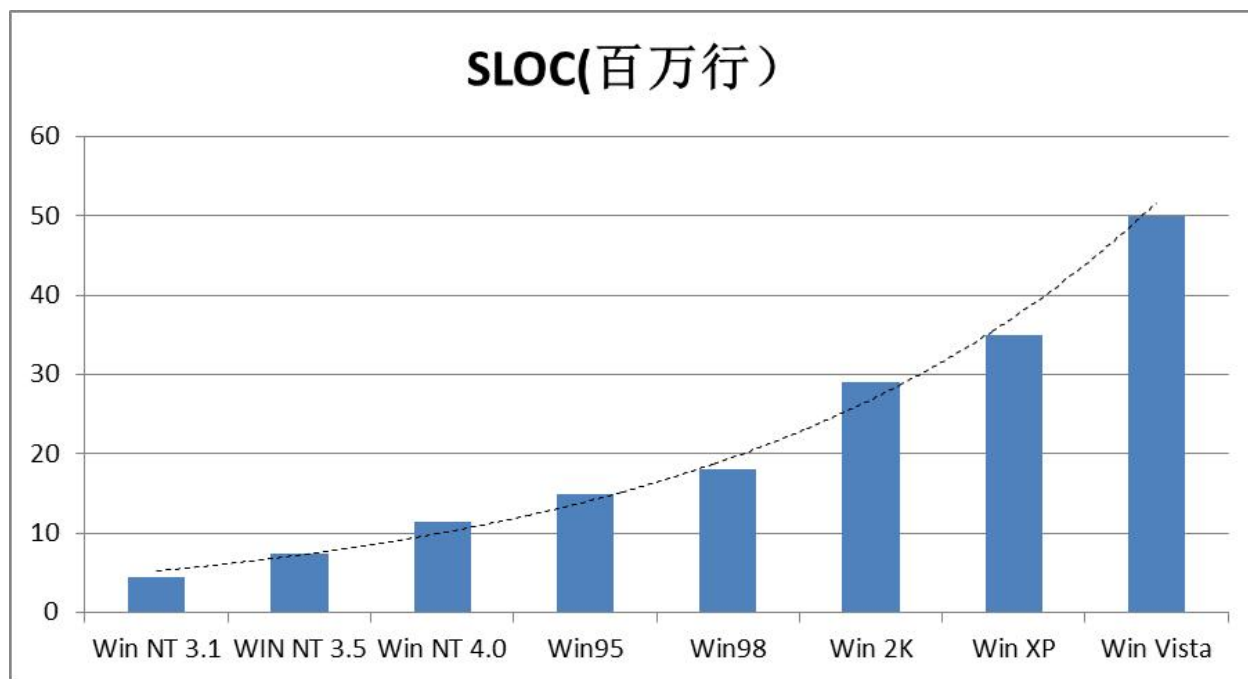


国家漏洞库漏洞数量趋势

# 软件安全问题产生-内因



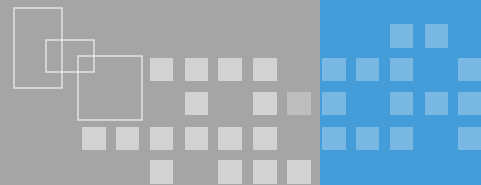
- ❖ 软件规模增大, 功能越来越多, 越来越复杂
- ❖ 软件模块复用, 导致安全漏洞延续
- ❖ 软件扩展模块带来的安全问题



Windows操作系统不同版本源代码数量



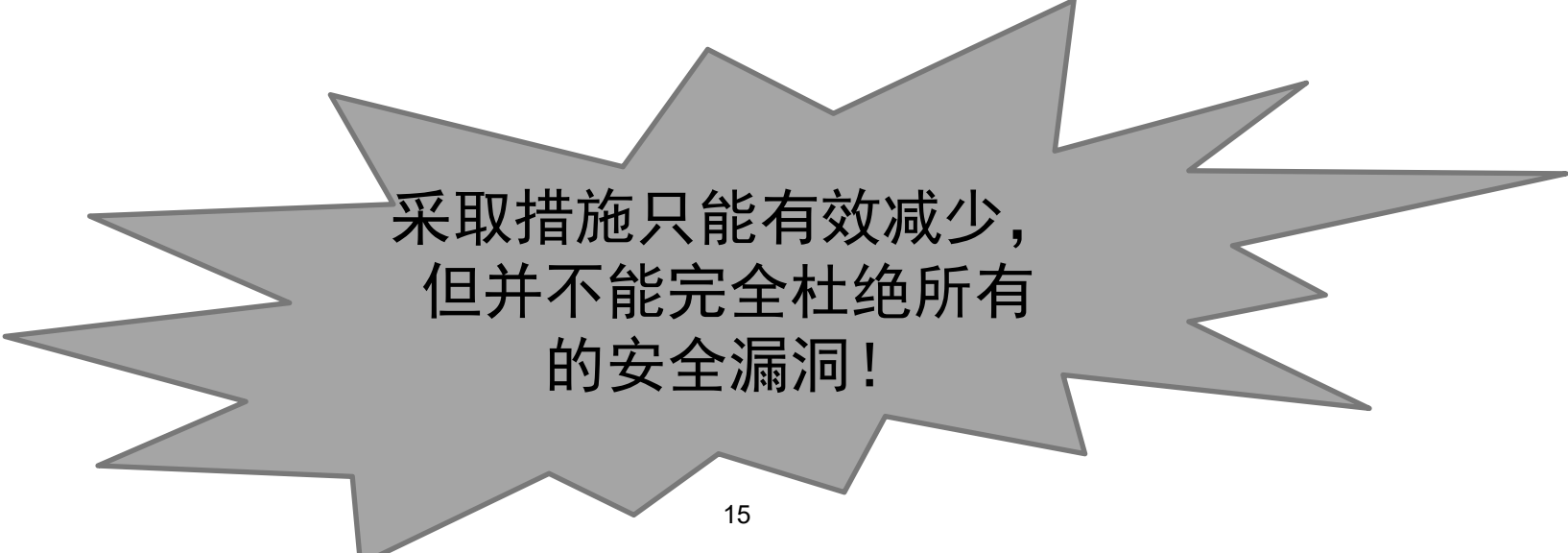
- ❖ 互联网发展对软件安全的挑战
- ❖ 开发环境和开发人员对软件安全的挑战
  - 开发者缺乏安全开发的动机
    - 市场和业务要求将交付期和软件功能做主要因素
    - 用户方没有提供安全方面的压力
  - 开发者缺乏相关知识
    - 软件复杂性加大，开发者需要学习更多东西
    - 传统软件开发不进行安全教育
  - 缺乏安全开发工具
    - 缺乏安全开发配套管理、测试等工具



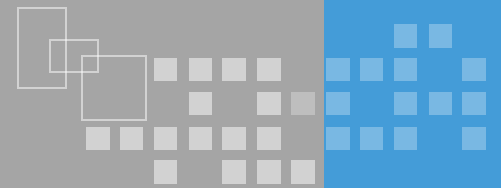
## ❖ 贯彻风险管理思想

- 安全不必是完美无缺的，但风险必须是可管理的
- 树立对软件安全控制的信心，该信心是通过保障活动来获取的

## ❖ 通过在软件开发生命周期各阶段采取必要的、相适应的安全措施来避免绝大多数的安全漏洞

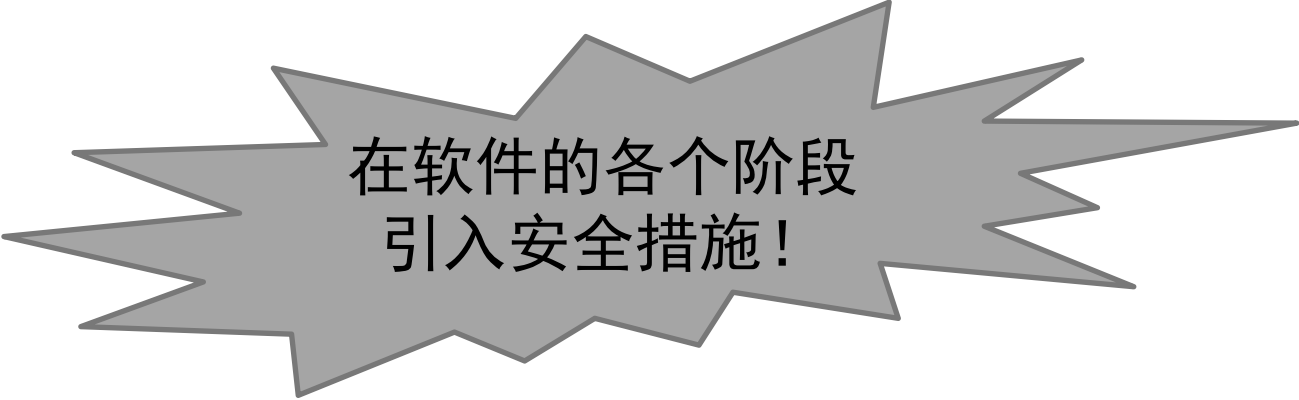


采取措施只能有效减少，  
但并不能完全杜绝所有的  
安全漏洞！



## ❖ 软件安全开发覆盖软件整个生命周期

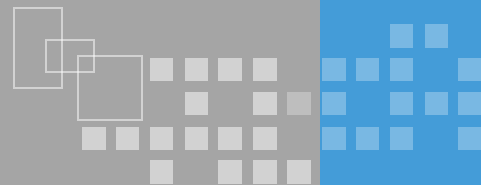
- 需求分析阶段考虑软件的安全需求
- 在设计阶段设计符合安全准则的功能
- 编码阶段保证开发的代码符合安全编码规范
- 安全测试和运行维护确保安全需求、安全设计、安全编码各个环节得以正确有效的实施



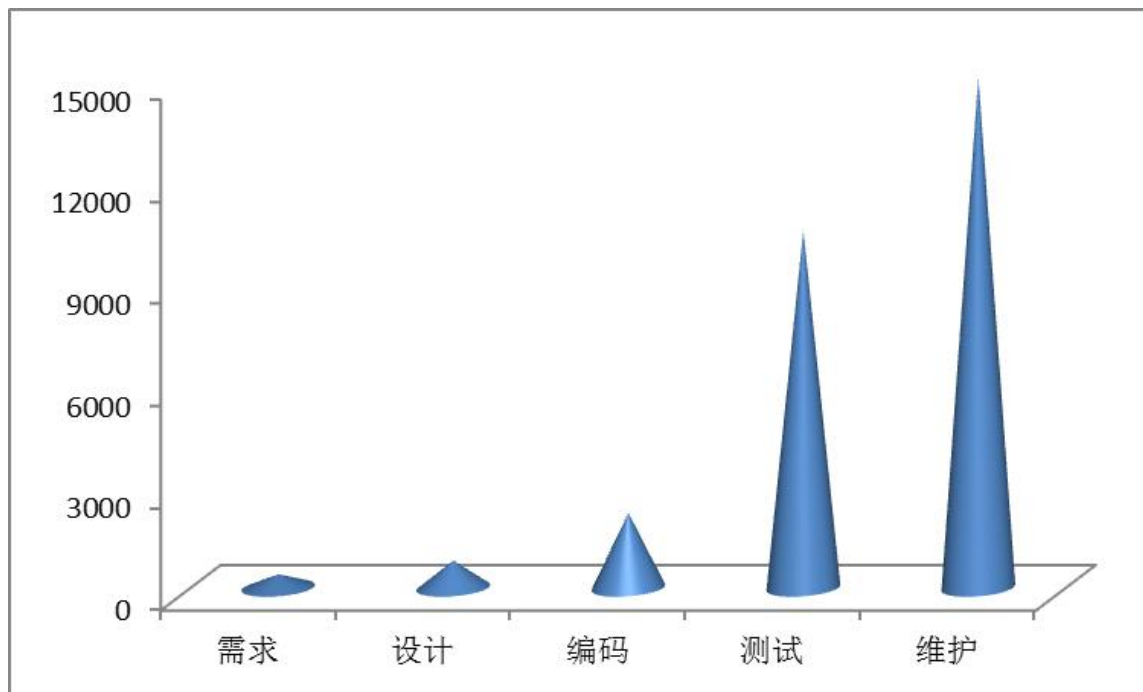
在软件的各个阶段  
引入安全措施！

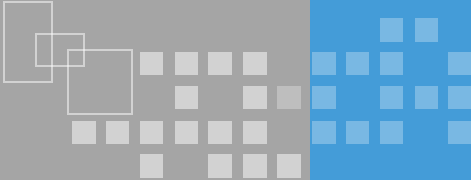


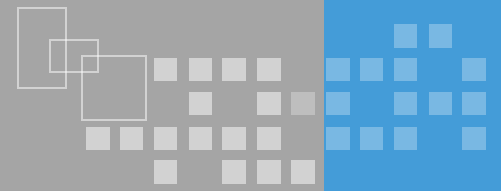
# 软件安全问题越早解决成本越低



- ❖ 在软件开发生命周期中，后面的阶段改正错误开销比前面的阶段要高出数倍
- ❖ NIST：在软件发布以后进行修复的代价是在软件设计和编码阶段即进行修复所花代价的30倍



- 
- ❖ 软件存在漏洞和缺陷是不可避免的，实践中尝试用软件缺陷密度（Defects/KLOC）来衡量软件的安全性。假设某个软件共有 29.6 万行源代码，总共被检测出 145 个缺陷，则可以计算出其软件缺陷密度值是（）
- ❖ A. 0.00049
  - ❖ B. 0.049
  - ❖ C. 0.49
  - ❖ D. 49

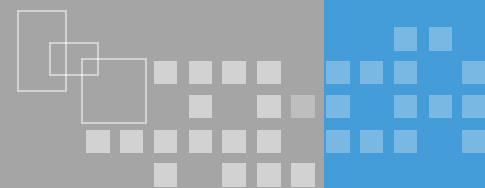


## ❖ 安全软件开发生命周期

- 安全设计原则
- 安全开发方法
- 最佳实践
- 安全专家经验

## ❖ 多种模型被提出和研究

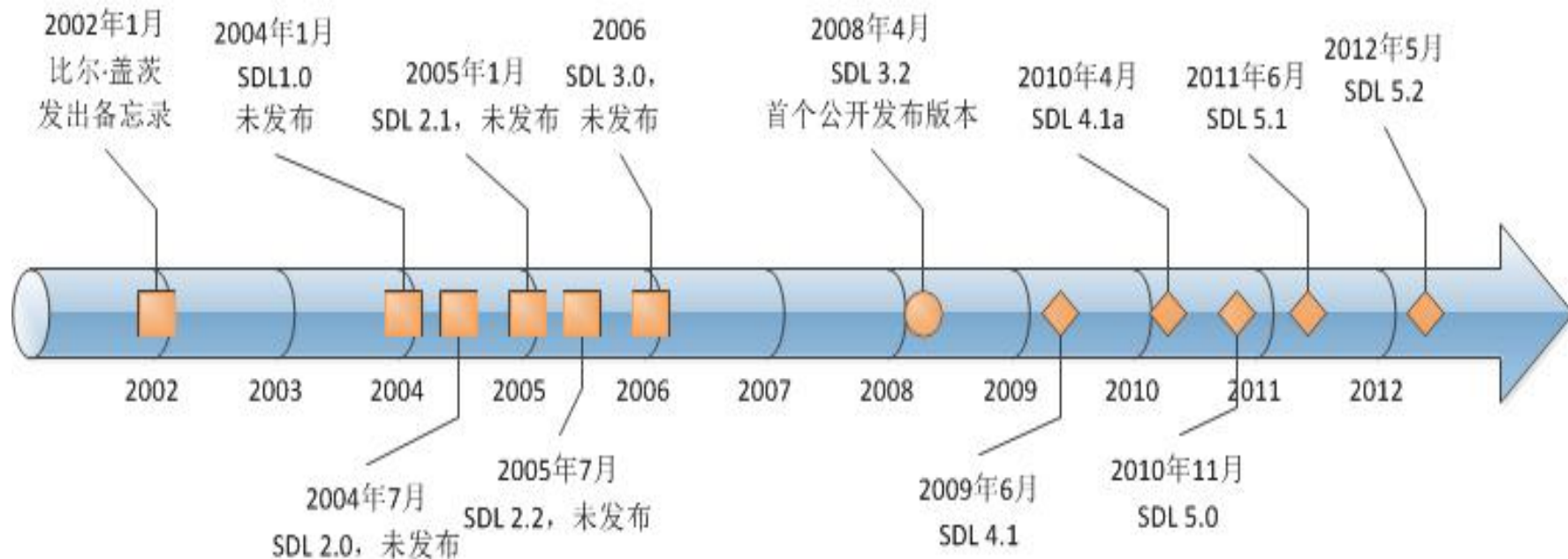
- 可信计算安全开发生命周期（微软）
- CLASP（OWASP）综合的轻量应用安全过程
- BSI系列模型（Gary McGraw等）
- SAMM（OWASP）软件保证成熟度模型



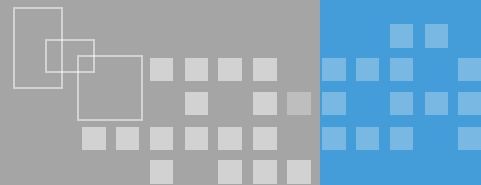
## ❖ 什么是SDL

- 安全开发生命周期 (Security Development Lifecycle, SDL)

## ❖ SDL发展



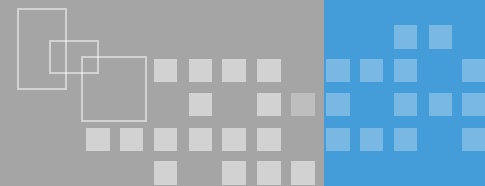
# SDL的阶段和安全活动



## ❖ 七个阶段

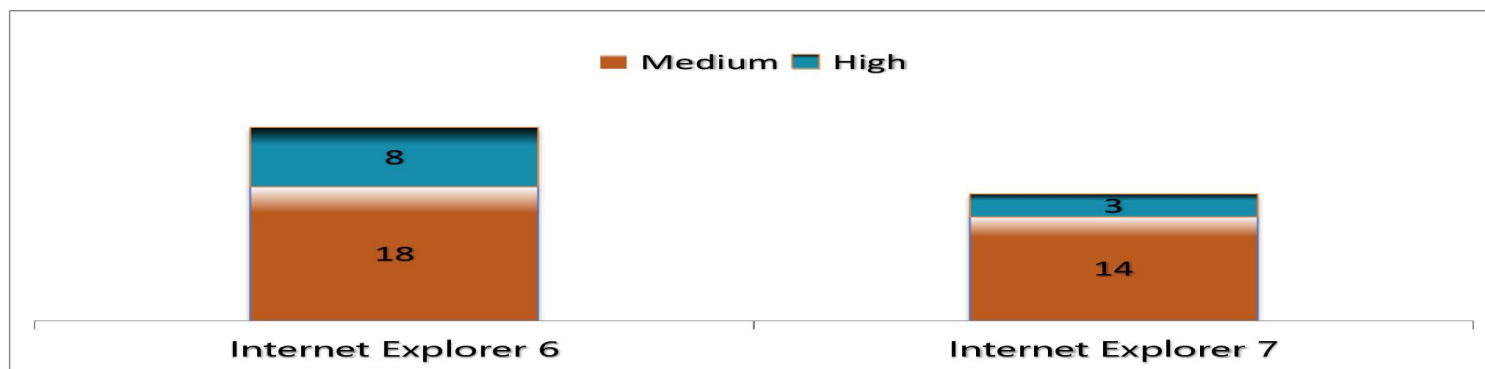
## ❖ 十七项必需的安全活动



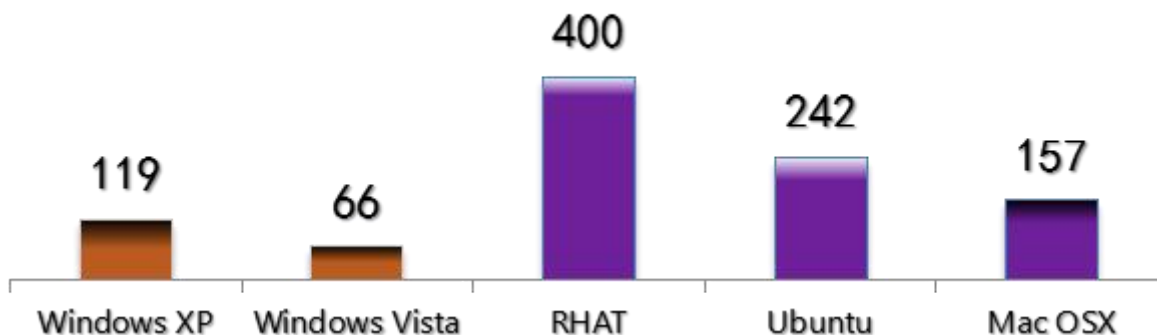


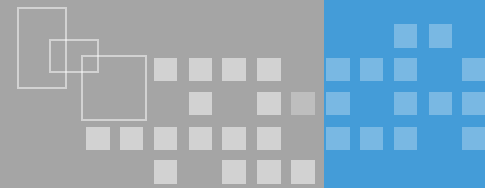
## ❖ 正式发布软件后12个月内的漏洞对比

- IE：漏洞总数下降35%，高危漏洞数下降63%



- 操作系统：漏洞总数降低45%





## ❖ 什么是CLASP

- 综合的轻量应用安全过程（Comprehensive, Lightweight Application Security Process , CLASP）
- 用于构建安全软件的轻量级过程，由30个特定的活动(activities)和辅助资源构成的集合
- 针对这些活动给出了相应的指南、导则和检查列表

## ❖ 特点

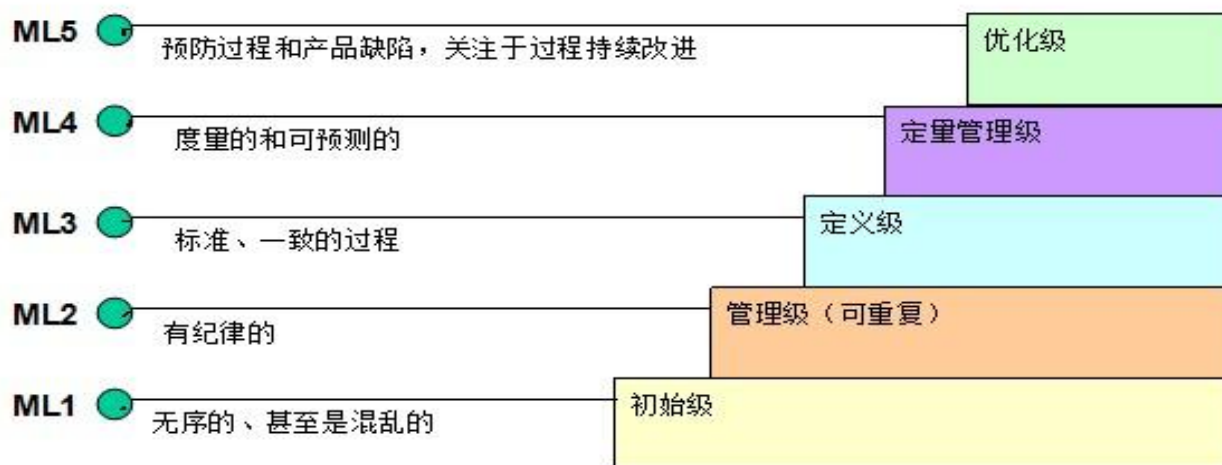
- 基于角色的安排



## ❖ 什么是CMMI

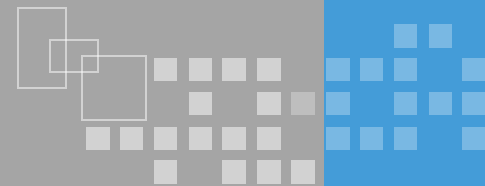
- 软件能力成熟度集成模型 (Capability Maturity Model Integration)

## ❖ 五级



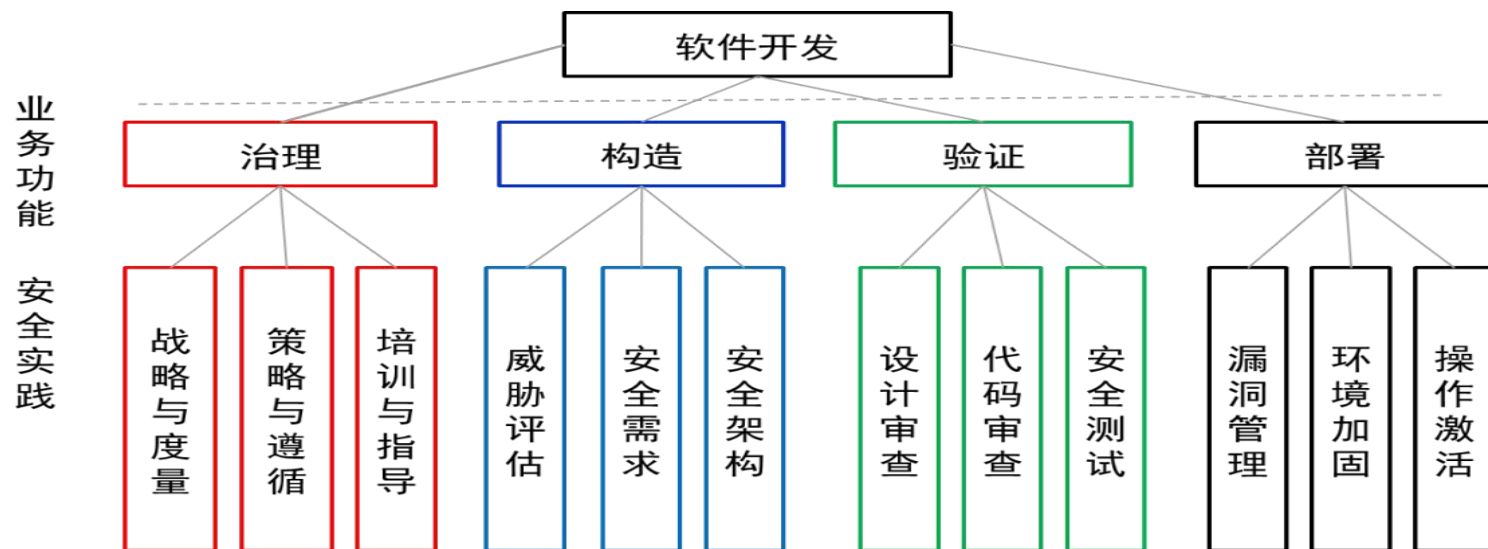
## ❖ 过程区域

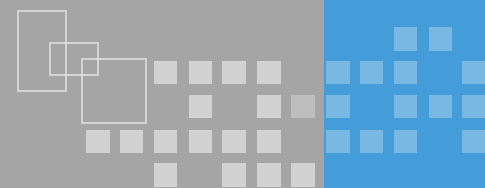




## ❖ 什么是SAMM

- 软件保证成熟度模型（Software Assurance Maturity Mode, SAMM）
- 提供了一个开放的框架，用以帮助软件公司制定并实施所面临来自软件安全的特定风险的策略，





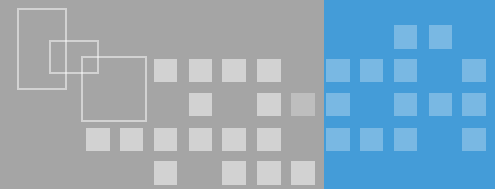
## ❖ BSI (Building Security IN)

- 使安全成为软件开发必须的部分
- 强调应该使用工程化的方法来保证软件安全

## ❖ 软件安全的三根支柱

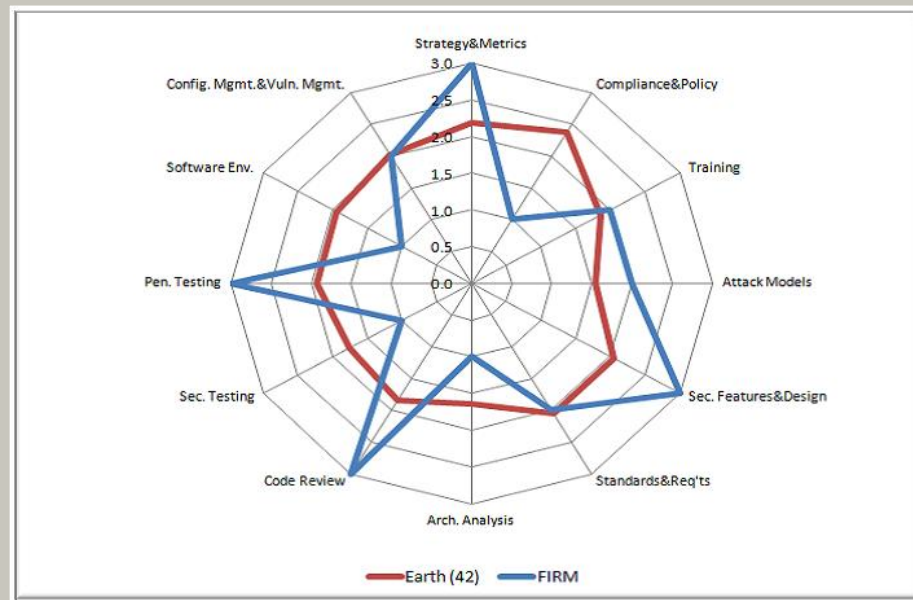
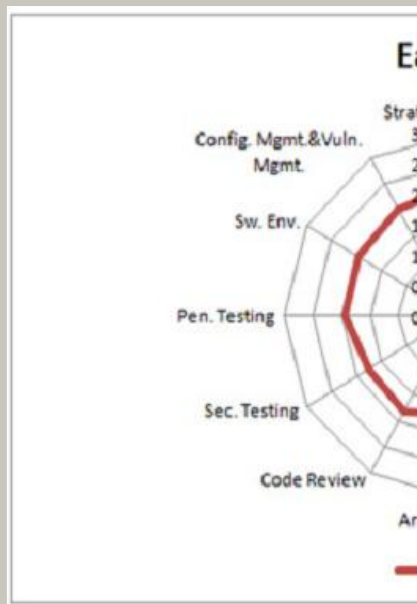
- 风险管理：策略性方法
- 接触点：一套轻量级最优工程化方法，攻击与防御综合考虑
- 安全知识：强调对安全经验和专业技术进行收集汇总，对软件开发人员进行培训，并通过安全接触点实际运用



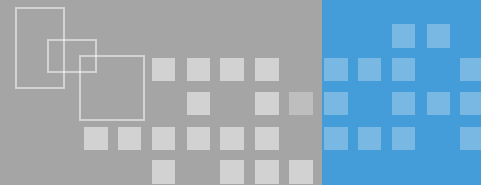


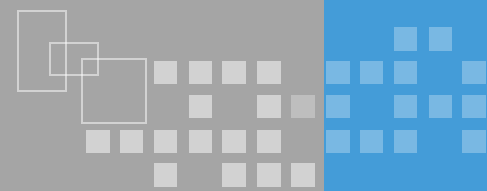
## ❖ BSI 成熟度模型

- 对真实的软件安全项目所开展的活动进行量化
- 构建和不断发展软件安全行动的指南



# 各模型比较





## ❖ 威胁建模

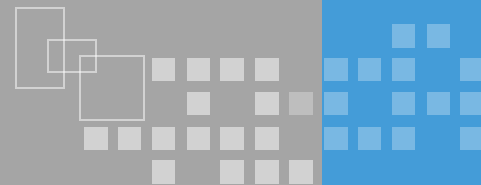
- 理解威胁建模的作用及每个阶段的工作内容；
- 掌握STRIDE模型用于进行威胁建模实践。

## ❖ 软件安全需求分析

- 理解软件安全需求在软件安全开发过程中的重要性；
- 理解安全需求分析的方法和过程。

## ❖ 软件安全设计

- 理解软件安全设计的重要性及内容和主要活动；
- 理解最小特权、权限分离等安全设计的重要原则；
- 理解攻击面的概念并掌握降低攻击面的方式。

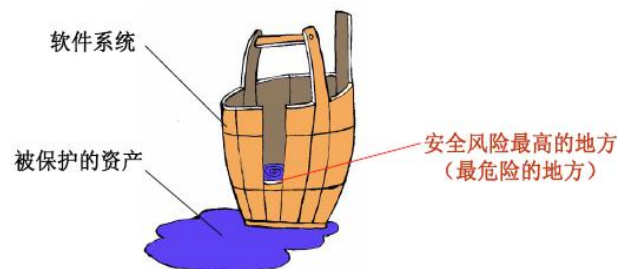


## ❖ 什么是威胁建模

- 威胁建模是了解系统面临的安全威胁，确定威胁风险并通过适当的缓解措施以降低风险，提高系统安全性的过程。

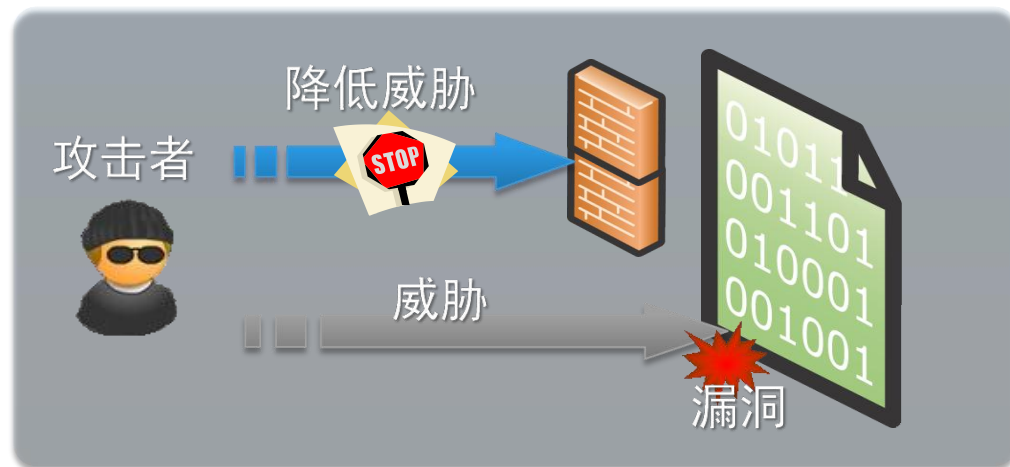
## ❖ 为什么要威胁建模

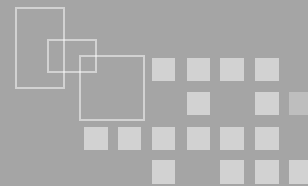
- 帮助在设计阶段充分了解各种安全威胁，并指导选择适当的应对措施
- 对可能的风险进行管理
- 可以重新验证其架构和设计
- 有助于软件的受攻击面降低



# 威胁建模流程

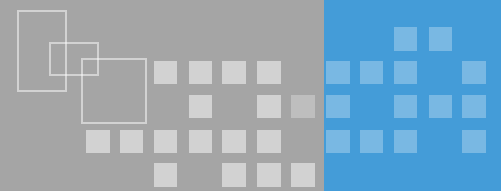
- ❖ 确定对象
- ❖ 识别威胁
- ❖ 评估威胁
- ❖ 消减威胁





- ❖ 确定要保护和评估的目标（资产）
- ❖ 在使用实例和应用场景中分析
  - 明确应用或系统的关键威胁场景
    - 部署方式、配置信息、用户使用方式等
  - 典型场景
    - 移动或小型设备物理失窃场景
    - Web应用匿名用户场景



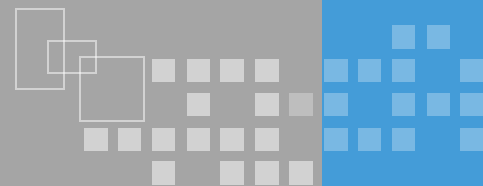


## ❖ 识别每一个可能面临的威胁

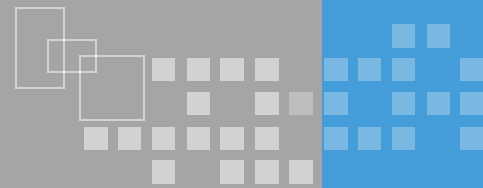
- 理解软件可能面临的威胁是安全开发的前提
- 威胁不等于漏洞
- 威胁永远存在

S	Spoolfing Identity	假冒身份/欺骗标识
T	Tampering with data	篡改数据
R	Repudiation	抵赖
I	Information Disclosure	信息泄漏
D	Denial of Service	拒绝服务
E	Elevation of Privilege	权限提升

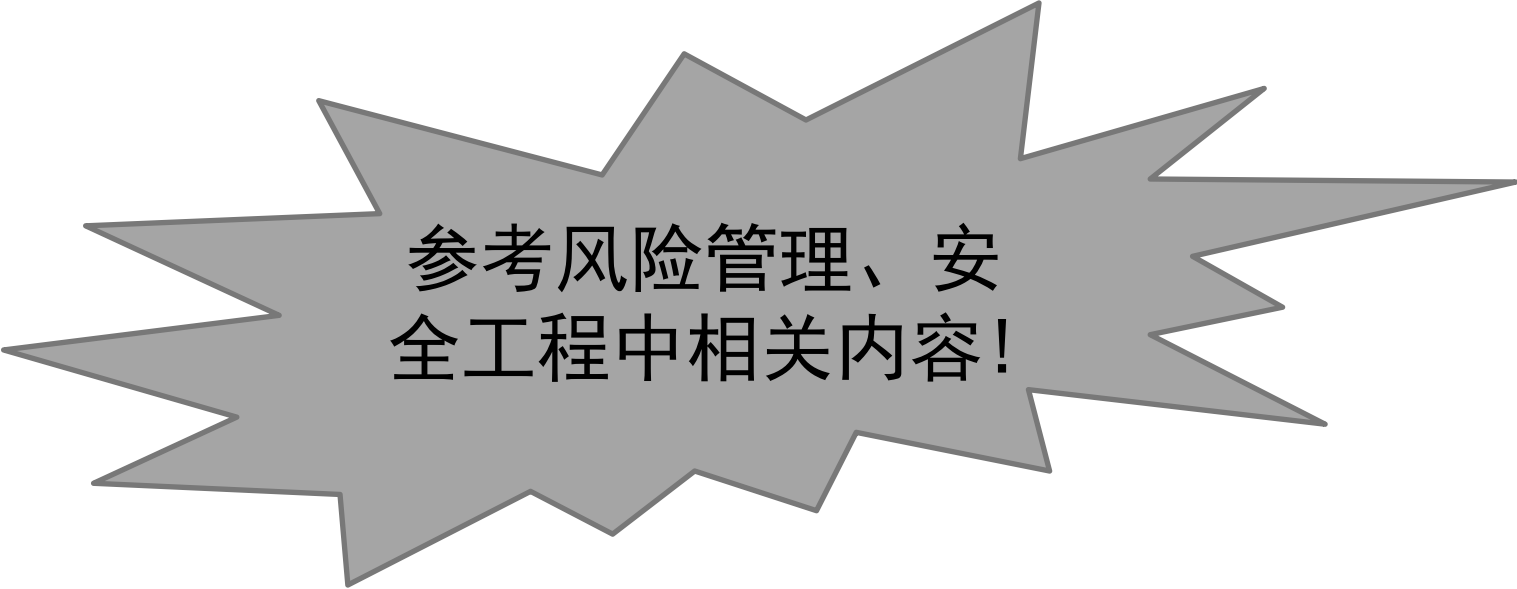
# 理解STRIDE六类威胁



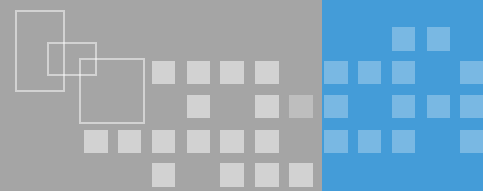
威胁	安全属性	定义	举例
<b>Spoofing</b> （欺骗）	可鉴别性	模仿其他人或实体	伪装成microsoft.com或ntdll.dll。
<b>Tampering</b> （篡改）	完整性	修改数据或代码	修改硬盘、DVD或网络数据包中的DLL
<b>Repudiation</b> （抵赖）	不可抵赖性	声称没有执行某个动作	“我没有发送过那封电子邮件”，“我没有修改过那个文件”，“亲爱的，我确实没有访问过那个网站！”
<b>Information Disclosure</b> （信息泄露）	机密性	把信息披露给那些无权知道的人	允许某人阅读Windows源代码；公布某个Web网站的用户清单。
<b>Denial of Service</b> （拒绝服务）	可用性	拒绝为用户提供服务	使得Windows或Web网站崩溃，发送数据包并耗尽CPU时间，将数据包路由到某黑洞中。
<b>Elevation of Privilege</b> （权限提升）	授权	获得非授权访问权	允许远程因特网用户执行命令，让受限用户获得管理员权限。



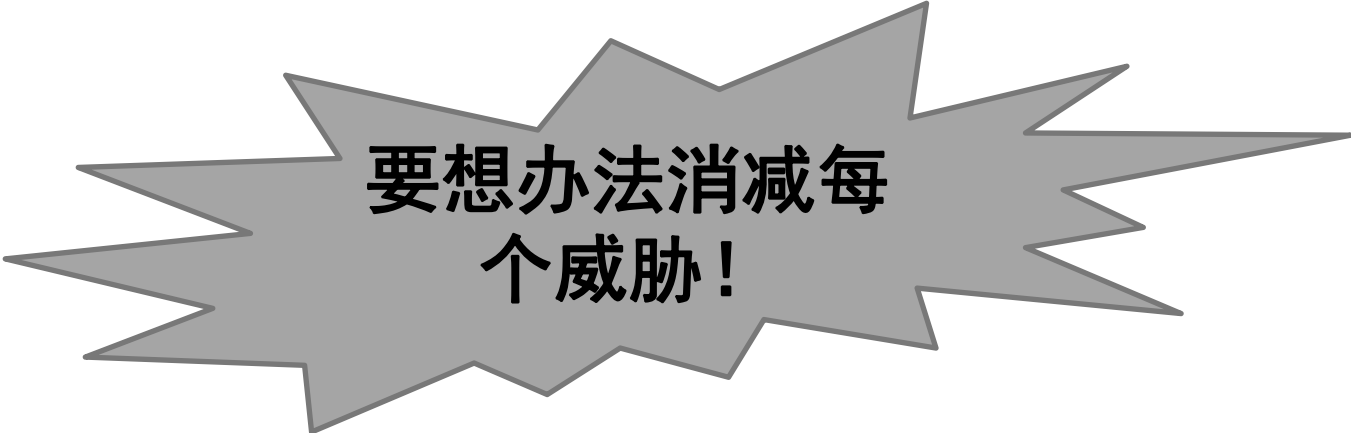
- ❖ 评估威胁风险值
- ❖ 评估被利用和攻击发生的概率
- ❖ 评估攻击后资产的受损后果，并计算风险



参考风险管理、安全工程中相关内容！



- ❖ 重新设计并排除这个威胁
- ❖ 使用标准的威胁消减技术
- ❖ 发明新的消减方法
- ❖ 根据安全Bug标准来确定是否可接受风险
- ❖ 把威胁作为漏洞记录下来，以后再想办法消减



**要想办法消减每个威胁！**

# 软件安全需求及安全设计的重要性

- ❖ 软件安全需求和设计是开发安全软件的基础
- ❖ 软件安全需求分析
  - 以风险管理为基础，建立“威胁”分析计划
  - 建立软件安全需求定义，确保软件安全需求定义正确
  - 安全需求应文档化
- ❖ 软件安全设计
  - 软件系统的每一项需求，都应该在软件安全设计阶段认真考虑



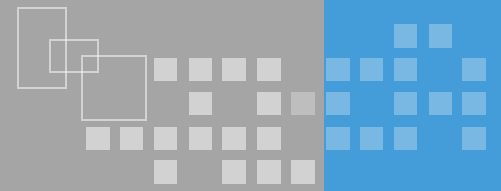
## ❖ 安全需求分类

- 安全功能需求
- 安全保障需求

## ❖ 需求分析的要点

- 安全需求进行有效定义
- 不仅考虑系统功能，还要考虑系统不应该做什么
- 功能需求、安全需求、安全目标要达到平衡

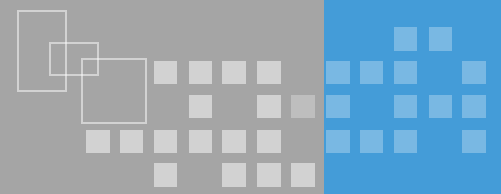
需求工程师不要仅仅从用户的角度出发考虑系统的功能，还应从攻击者的角度出发考虑系统的漏洞。



- ❖ 系统调查
- ❖ 定性分析系统的脆弱点和可能遭受的安全威胁
- ❖ 脆弱点和安全威胁的定量分析
- ❖ 需求的确定

建立在风险分析的基础上！





## ❖ 安全编码？安全测试？

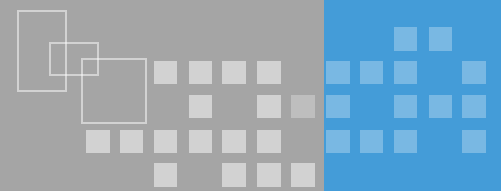
- 传统方法：软件发布后测试、等待修复Bug
- Gary McGraw：50%的安全问题由设计瑕疵引起
- 安全提前介入，效益高，成本低

**安全编码？安全测试？安全设计？**

## ❖ 设计缺陷——举例

- Microsoft Bob
- 明文存储口令，甚至将口令拿到客户端对比验证





## ❖ 安全概要设计阶段

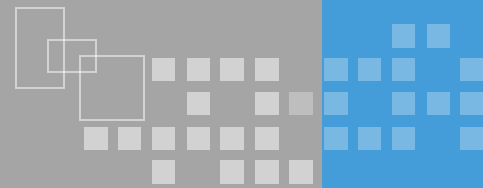
- 包括但不限于：安全体系结构设计、各功能块间的处理流程、与其他功能的关系、安全协议设计、安全接口设计等。

## ❖ 安全详细设计阶段

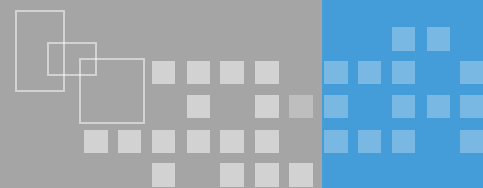
- 详细设计阶段作为安全功能的程序设计阶段，应当直接指导安全功能的编码工作。包括但不限于：模块设计、内部处理流程、数据结构、输入/输出项、算法、逻辑流程图等

根据安全需求方案确定的安全目标，对初步风险评估确定的控制措施的具体技术实现而进行安全设计

# 安全设计的主要活动

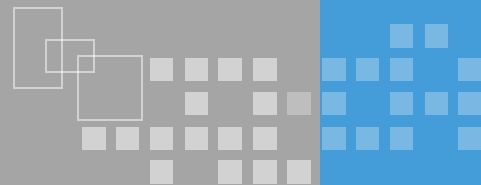


- ❖ 详细风险评估
- ❖ 控制措施选择
- ❖ 安全技术实现
- ❖ 安全设计评审



- ❖ 最小特权原则
- ❖ 权限分离原则
- ❖ 最少共享机制原则
- ❖ 完全中立原则
- ❖ 心理可接受度原则
- ❖ 默认故障处理保护原则
- ❖ 经济机制原则
- ❖ 不信任原则
- ❖ 纵深防御原则
- ❖ 保护最薄弱环节原则
- ❖ 公开设计原则
- ❖ 隐私保护原则
- ❖ 攻击面最小化原则

# 降低攻击面



## ❖ 作用

- 攻击面越小，安全风险越小

## ❖ 实现

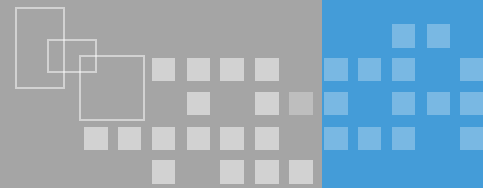
- 取消不需要的功能
- 增加对功能的安全防护

## ❖ 示例

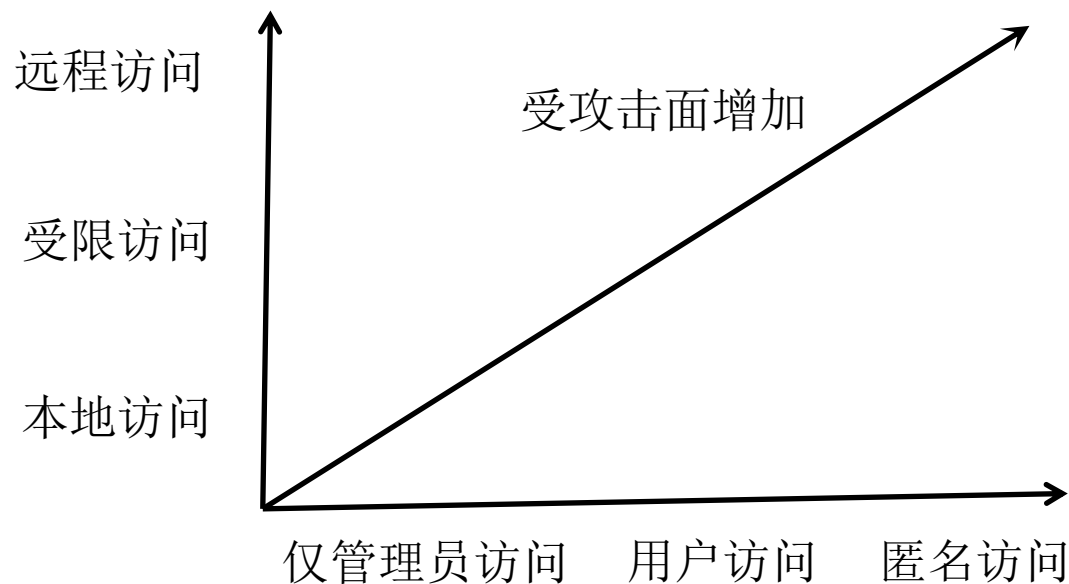
- SQL Server 2005 默认关闭 xp\_cmdshell 存储过程



# 分析软件攻击面



- ❖ 分析产品功能的重要性（是否必须）
- ❖ 分析从哪里访问这些功能（本地&远程）
- ❖ 分析访问权限（匿名&经过认证）



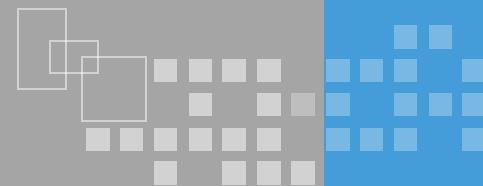
# 降低攻击面策略



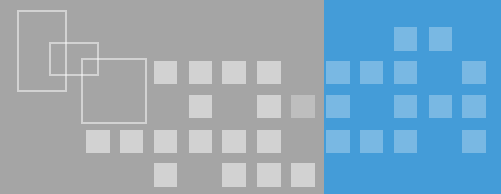
- ❖ 重要等级为低的功能：攻击面大，取消该功能
- ❖ 重要等级为中的功能：攻击面大，设置为非默认开启，需要用户配置后才予以开启
- ❖ 重要等级为高的功能：攻击面大，关闭或限制一些接口方式，增加一些安全的保证措施或技术

降低受攻击面  
对于提高软件源代码安全性  
至关重要！

# 降低软件攻击面通常做法

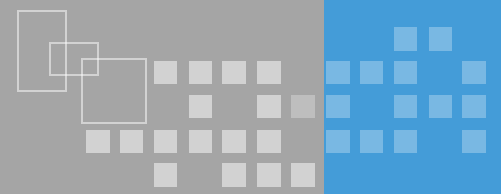


较高受攻击面	较低受攻击面
默认执行	默认关闭
打开网络连接	关闭网络连接
同时侦听UDP和TCP流量	仅侦听TCP流量
匿名访问	鉴别用户访问
弱ACLs	强ACLs
管理员访问	普通用户访问
因特网访问	本地子网访问
代码以管理员或root权限运行	代码以Network Services、Local Services 或自定义的低权限账户运行
统一缺省配置	用户可选的配置
ActiveX控件	.NET代码
标记有脚本安全的ActiveX控件	未标记有脚本安全的ActiveX控件
非SiteLocked ActiveX控件	SiteLocked ActiveX控件

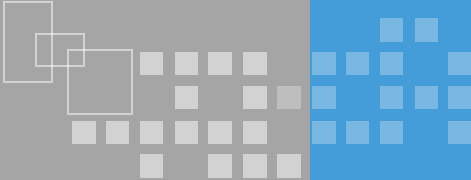


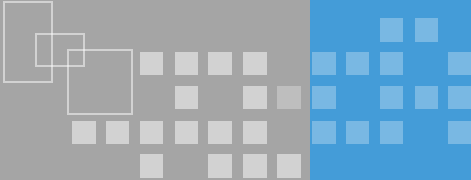
- ❖ 下面哪项属于软件开发安全方面的问题（）？
- ❖ A. 软件部署时对所选用的服务器性能不高，导致软件执行效率低
- ❖ B. 应用软件未考虑多线程技术，在对多用户服务时按序排队提供服务
- ❖ C. 应用软件存在 SQL 注入漏洞，若被黑客利用能窃取数据库所有数据
- ❖ D. 软件受许可证（LICENSE）限制，不能在多台电脑上安装

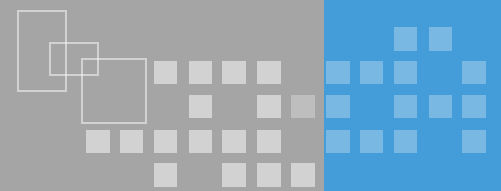




- ❖ 某电子商务网站在开发设计时，使用了威胁建模方法来分析电子商务网站所面临的威胁。STRIDE 是微软SDL 中提出的威胁建模方法，将威胁分为六类，为每一类威胁提供了标准的消减措施，Spoofing 是STRIDE 中欺骗类的威胁，以下威胁中哪个可以归入此类威胁？
- ❖ A. 网站竞争对手可能雇佣攻击者实施DDos 攻击，降低网站访问速度
- ❖ B. 网站使用http 协议进行浏览等操作，未对数据进行加密，可能导致用户传输信息泄漏，例如购买的商品金额等
- ❖ C. 网站使用http 协议进行浏览等操作，无法确认数据与用户发出的是否一致，可能数据被中途篡改
- ❖ D. 网站使用用户名、密码进行登录验证，攻击者可能会利用弱口令或其他方式获得用户密码，以该用户身份登录修改用户订单等信息

- 
- ❖ 微软提出了STRIDE 模型，其中R 是Repudiation(抵赖)的缩写，关于此项安全要求下面描述错误的是
  - ❖ A. 某用户在登录系统并下载数据后，却声称“我没有下载过数据” 软件系统中的这种威胁就属于R 威胁
  - ❖ B. 解决R 威胁，可以选择使用抗抵赖性服务技术来解决，如强认证、数字签名、安全审计等技术措施
  - ❖ C. R 威胁是STRIDE 六种威胁中第三严重的威胁，比D 威胁和E 威胁的严重程度更高
  - ❖ D. 解决 R 威胁，也应按照确定建模对象、识别威胁、评估威胁以及消减威胁等四个步骤来进行

- 
- ❖ 以下关于威胁建模流程步骤说法不正确的是
  - ❖ A. 威胁建模主要流程包括四步：确定建模对象、识别威胁、评估威胁和消减威胁
  - ❖ B. 评估威胁是对威胁进行分析，评估被利用和攻击发生的概率，了解被攻击后资产的受损后果，并计算风险
  - ❖ C. 消减威胁是根据威胁的评估结果，确定是否要消除该威胁以及消减的技术措施，可以通过重新设计直接消除威胁，或设计采用技术手段来消减威胁。
  - ❖ D. 识别威胁是发现组件或进程存在的威胁，它可能是恶意的，威胁就是漏洞。



## ❖ 安全编码原则

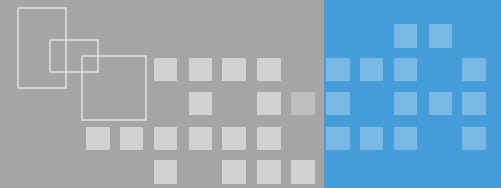
- 了解验证输入、避免缓冲区溢出、程序内部安全、安全调用组件、禁用有风险的函数等通用安全编程准则；
- 了解相关的安全编码标准及建议；
- 理解常见的代码安全问题及处置办法；

## ❖ 代码安全编译

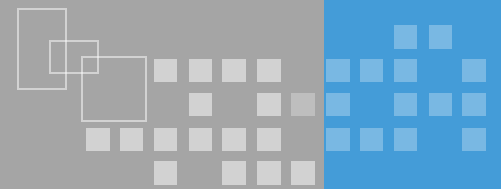
- 了解代码编译需要关注的安全因素；

## ❖ 代码安全审核

- 理解代码审查的目的；
- 了解常见源代码静态分析工具及方法



- ❖ 对**所有输入数据**进行检查、验证及过滤
  - 应用程序的“数据防火墙”，避免恶意数据进入
- ❖ 什么时候验证
  - 最初接收数据时
  - （第一次）使用数据时



## ❖ 命令行

- 参数数量、数据格式、内容

## ❖ 环境变量

- 环境变量可能超出期望
- 有的环境变量存储格式存在危险

## ❖ 文件

- 不信任可以被不可信用户控制的文件内容
- 不信任临时文件

## ❖ 网络

- 来自网络的数据是“高度不可信的”

## ❖ 其他来源

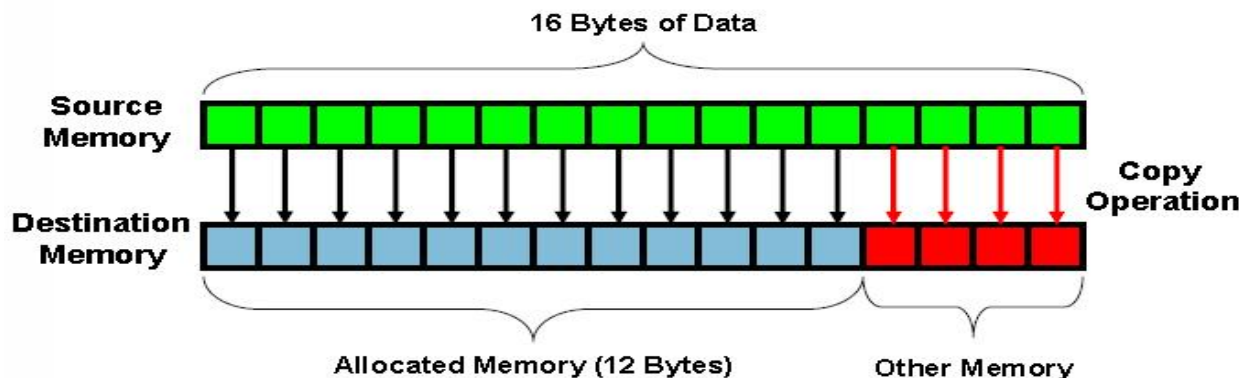
# 通用安全编码原则-避免缓冲区溢出

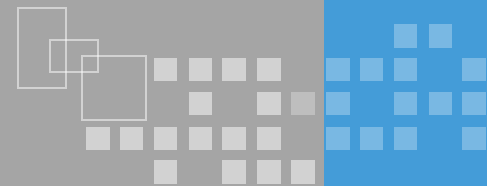
## ❖ 缓冲区溢出

- 缓冲区：包含相同数据类型的实例的一个连续计算机内存块
- 溢出：数据被添加到分配给该缓冲区的内存块之外

## ❖ 外部数据比目标空间大

## ❖ 是一个非常普遍而且严重的问题





## ❖ 溢出后果

- 攻击者可以使远程服务程序或者本地程序崩溃
- 攻击者可以设计溢出后执行的代码

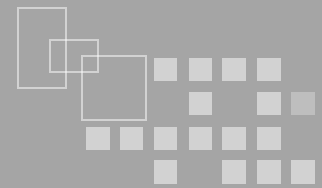
## ❖ C/C++语言

- 语言特性决定
- 大量的库函数存在溢出
  - strcpy、strcat、gets等

## ❖ 其他语言

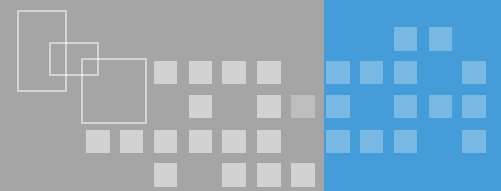
- 调用C语言库
- C#允许设置“不安全”例程





## ❖ 解决办法

- 填充数据时计算边界
  - 动态分配内存
  - 控制输入
- 使用没有缓冲区溢出问题的函数
  - strncpy、strncat、C++中std::string
- 使用替代库
  - Libmib、libsafe
- 基于探测方法的防御
  - StackGuard、ProPolice、/GS
  - 将一个“探测”值插入到返回地址的前面
- 非执行的堆栈防御
  - 不可在堆栈上执行代码

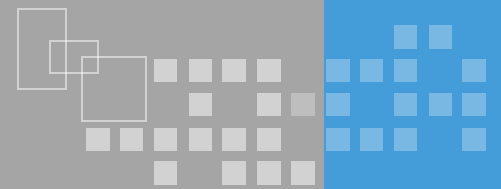


## ❖ 程序内部接口安全

- 程序内部接口数据的检查

## ❖ 异常的安全处理

- 检测异常，安全处理各种可能运行路径
- 检测到某些错误行为/数据，必须以合适的方式处理，保证程序运行安全
- 必要时立即拒绝服务，甚至不回送详细的错误代码



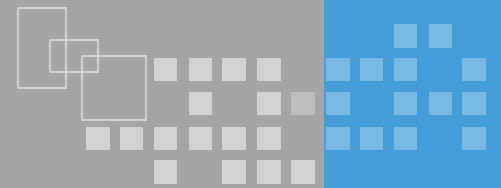
## ❖ 最小化反馈

- 避免给予不可靠用户过多的信息
  - 成功或失败
  - 作为跟踪检查的日志可以记录较为详细的信息
- 认证程序在认证前尽量少给信息
- 如果程序接受了密码，不要返回它

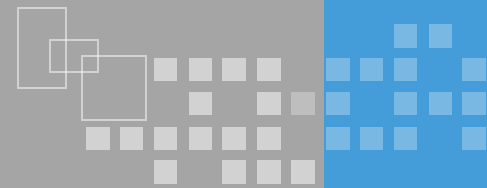
## ❖ 避免竞争条件

- 访问共享资源时（文件/变量）没有被适当地控制
- 使用原子操作
- 使用锁操作——避免死锁

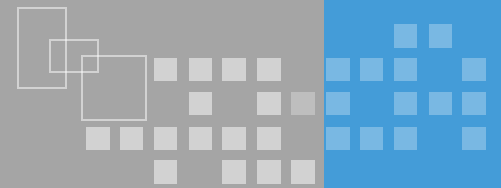
## ❖ 安全使用临时文件



- ❖ 应用程序实际上几乎都不会是自包含的，它们通常都会调用其他组件
  - 底层的操作系统
  - 数据库
  - 可重用的库
  - 网络服务（WEB、DNS）



- ❖ 使用安全组件，并且只采用安全的方式
  - 检查组件文档，搜索相关说明
    - gets
    - 随机数
  - 使用经过认可的组件
  - 尽可能不调用外部命令，如果不得已要调用，必须严格检查参数
    - system、open、exec、

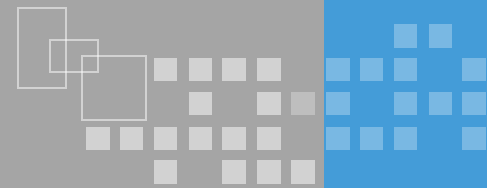


## ❖ 正确处理返回值

- 一定要检查返回值，调用是否成功
- 成功时检查
  - 返回值，是否按照期望值处理
  - 数据中可能含有 NUL 字符、无效字符或其他可能产生问题的东西
- 错误时检查
  - 错误码

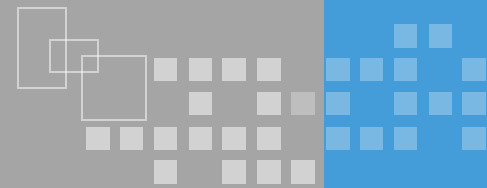
## ❖ 保护应用程序和组件之间传递的数据

- 视安全需求和安全环境
  - 考虑传输加密，包括密码算法和安全协议



## ❖ 编码中禁止使用的危险函数举例

禁止使用
strcpy, wcsncpy, strncpy, _tcscpy, _ftcsncpy, _mbncpy
strcat, wcscat, strcat, _tcscat, _ftcsnat, _mbnat
vsprintf, vswprintf, wvsprintf, wvnsprintf, _vstprintf
sprintf, swprintf, wsprintf, wnsprintf, _stprintf
gets, _getws, _getts



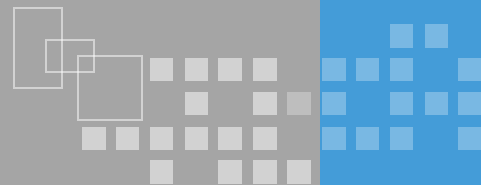
## ❖ 确保编译环境的安全

- 使用最新版本编译器与支持工具
- 可靠的编译工具
- 使用编译器内置防御特性

## ❖ 确保运行环境的安全

- 将软件运行环境基于较新版本的系统





## ❖ 什么是源代码审核

- 通过分析或检查源程序的语法、结构、过程、接口等来检查程序的正确性，报告源代码中可能隐藏的错误和缺陷

## ❖ 源代码审核方式

- 人工审核
  - 费时费力
  - 容易遗漏
- 工具审核
  - 速度快，自动
  - 可升级知识库

统计证明，在整个软件开发生命周期中，**30%至70%**的代码逻辑设计和编码缺陷是可以通过源代码审核来发现的。

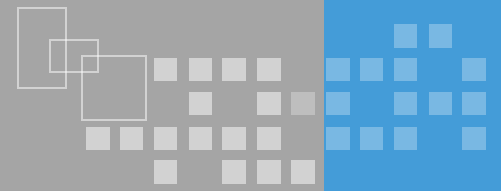


## ❖ 软件测试

- 了解软件测试的基本概念；
- 了解常见的软件测试方法及不同测试方法之间的区别和优缺点；

## ❖ 软件安全测试

- 了解软件安全测试的基本概念；
- 理解模糊测试、渗透测试等软件安全测试方法的原理、相互的区别以及各自的优势；
- 掌握安全测试的思路和方法。

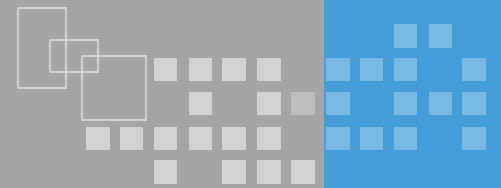


## ❖ 什么是软件测试

- 使用人工和自动化的手段来运行或测试某个系统的过程，其目的在于检验它是否满足规定的需求或是弄清预期结果与实际结果之间的差异

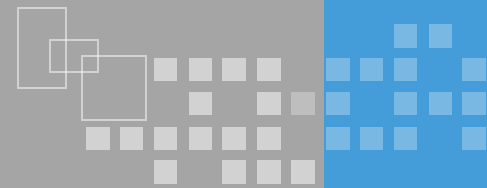
## ❖ 基本概念

- 测试用例
- 测试覆盖率度量指标

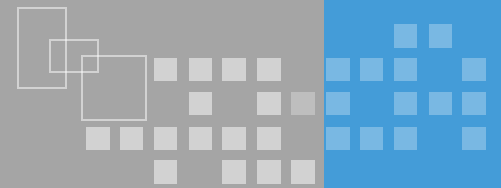


## ❖ 测试的信条

- 预期测试的测试结果是预先确定的
- 好的测试用例发现错误的概率高
- 成功的测试就是发现了错误的测试
- 测试独立于编码
- 需要具备应用（用户）及软件（编程）两方面的专业知识
- 测试人员使用不同于开发人员的工具
- 只检查常见的测试用例是不够的
- 测试文档要能够再利用



- ❖ 单元测试、集成测试、系统测试
- ❖ 黑盒测试、白盒测试、灰盒测试
- ❖ 静态测试、动态测试
  - 代码走查、代码审查、代码评审
- ❖ 回归测试
- ❖ 验收测试

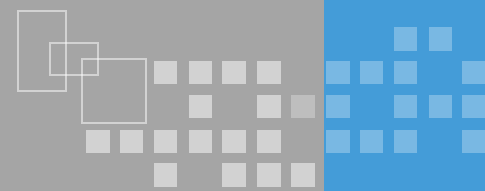


## ❖ 什么是软件安全测试

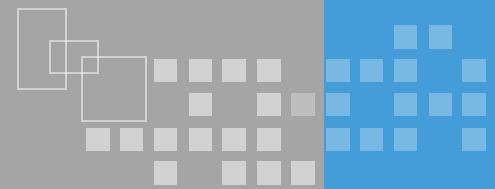
- 确定软件的安全特性实现是否与预期设计一致的过程
- 有关验证软件安全等级和识别潜在安全缺陷的过程
- 查找软件自身程序设计中存在的安全隐患，并检查应用程序对非法侵入的防范能力

## ❖ 为什么需要软件安全测试

- 传统测试仅考虑软件出错时的处理，没有考虑对软件的故意攻击



- ❖ 在应用投产前，应由独立的安全团队对应用的安全性进行综合评估
  - 功能性安全测试
  - 对抗性安全测试
- ❖ 安全测试方法
  - 模糊测试
  - 渗透测试
  - 静态源代码审核

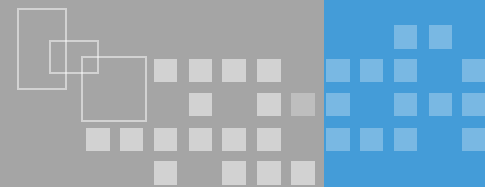


## ❖ 什么是模糊测试

- 也称Fuzzing测试，一种通过提供非预期的输入并监视异常结果来发现软件故障的方法
- 黑盒测试，不关心被测试目标的内部实现，而是利用构造畸形的输入数据引发被测试目标产生异常，从而发现相应的安全漏洞

**非常有效的漏洞挖掘技术，已知漏洞大部分都是通过这种技术发现的。**

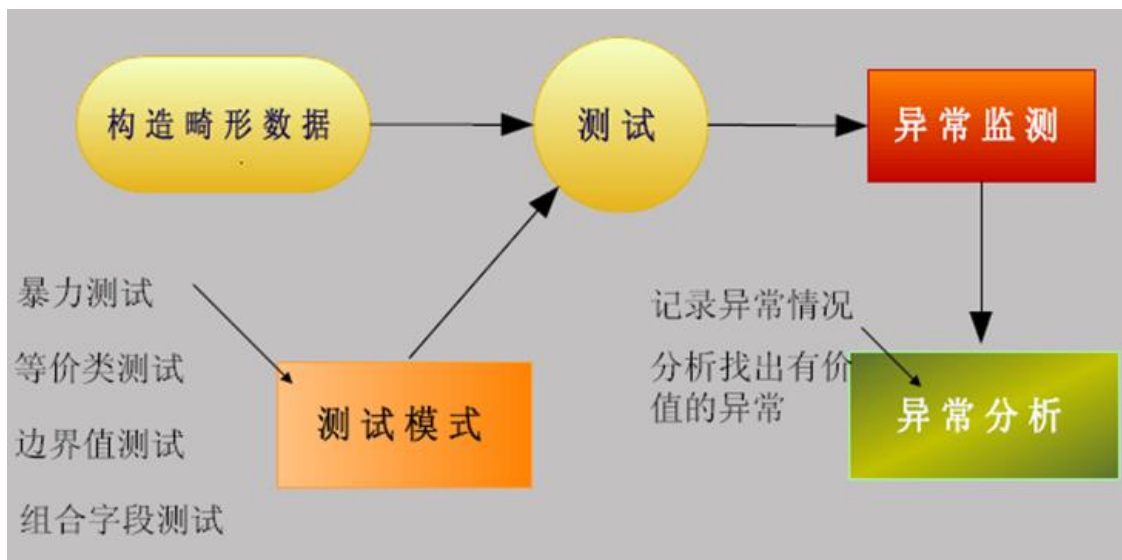




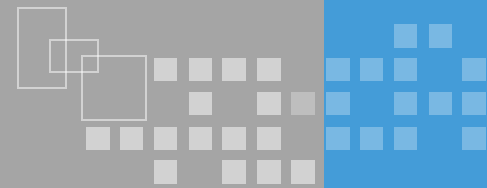
- ❖ 强制软件程序使用恶意/破坏性的数据并进行观察结果的一种测试方法
  - 不够强壮的程序会崩溃
  - 编码良好的程序正常运行
- ❖ 特性
  - 方法学
  - 随机值
  - 大量测试用例
  - 查找漏洞或可靠性错误

## ❖ 模糊测试过程

- 生成大量的畸形数据作为测试用例；
- 将这些测试用例作为输入应用于被测对象；
- 监测和记录由输入导致的任何崩溃或异常现象；
- 查看测试日志，深入分析产生崩溃或异常的原因



# 影响模糊测试效果的关键因素



## ❖ 测试点

- 数据通道入口、可信边界点

## ❖ 样本选择

- 选择覆盖面广、便于测试的多个样本

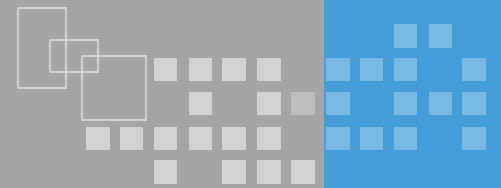
## ❖ 数据关联性

- 智能模糊测试

## ❖ 自动化框架

## ❖ 异常监控与异常恢复

## ❖ 分析评估



## ❖ 渗透测试

- 通过模拟恶意攻击者进行攻击，来评估系统安全的一种评估方法
- 从攻击的角度测试软件系统是否安全
- 使用自动化工具或者人工的方法模拟攻击者的输入，找出运行时刻目标系统所存在的安全漏洞

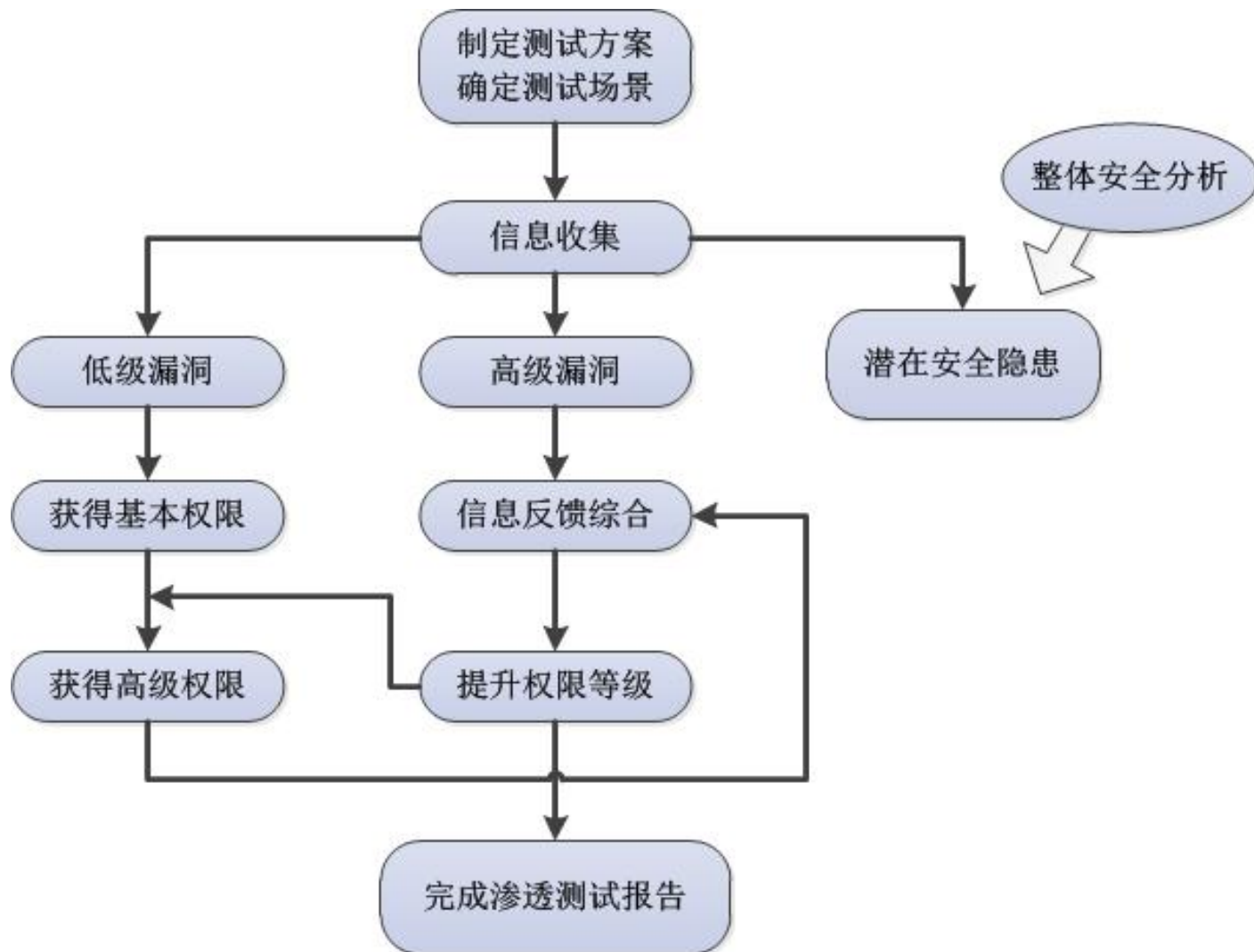
## ❖ 优点

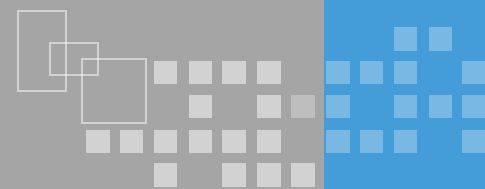
- 找出来的问题都是真实的，也是较为严重的

## ❖ 缺点

- 只能到达有限的测试点，覆盖率较低

# 渗透测试流程





## ❖ 测试目的

- 安全性的评估，不是摧毁或破坏

## ❖ 测试人员

- 技术、知识和经验很重要
- 像“坏人”一样思考问题

## ❖ 安全问题

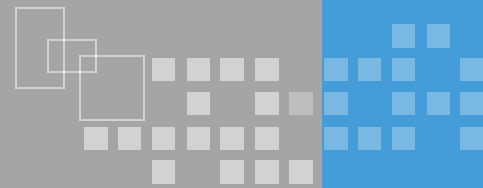
- 系统备份和恢复措施
- 风险规避

**\* 如果测试参数由哪些不想发现安全问题的人所确定，那么，渗透测试就很可能变成一种毫无用处的自我满足练习！**



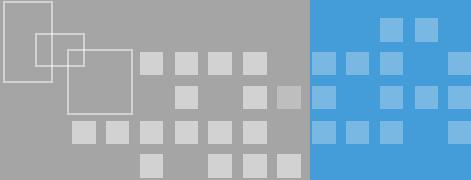
必须完整的遵循“安全开发”过程吗？

- ❖ 代码审核 + 体系结构风险评估
- ❖ 基于风险的安全测试 + 渗透测试
- ❖ 安全需求分析 + 滥用案例开发
- ❖ 代码审核 + 渗透测试
- ❖ 体系结构风险分析 + 基于风险的测试
- ❖ ...



- ❖ 充分了解软件安全漏洞
- ❖ 评估软件安全风险
- ❖ 拥有高效的软件安全测试技术和工具





❖ 某网站在设计时经过了威胁建模和攻击面分析，在开发时要求程序员编写安全的代码，但是在部署时由于管理员将备份存放在 Web 目录下导致了攻击者可以直接下载备份，为了发现系统中是否存在其他类似问题，以下哪种测试方法是最佳的测试方法：

- ❖ A. 模糊测试
- ❖ B. 源代码测试
- ❖ C. 渗透测试
- ❖ D. 软件功能测试



## ❖ 软件供应链安全

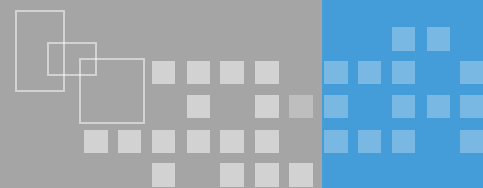
- 了解软件供应链安全的概念并理解软件供应链安全措施。

## ❖ 软件安全验收

- 了解软件安全验收的重要性及需要考虑的内容。

## ❖ 软件安全部署

- 了解软件安全部署的重要性及软件安全加固、软件安全配置的概念。

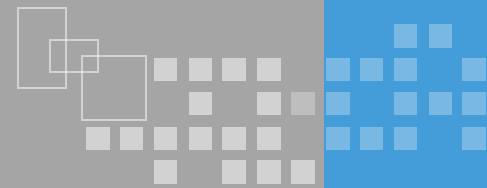


## ❖ 供应链安全概念

- 目前软件安全开发生命周期中新的威胁，涉及到软件的代码编写、代码编译、软件分发、软件更新
- 代码编写：共享库
- 代码编译：被污染的编译软件
- 软件分发/更新：污染源头

## ❖ 供应链安全应对策略

- 安全流程覆盖到引入的第三方代码中
- 可靠的编译软件获取方式
- 官方渠道、发布验证

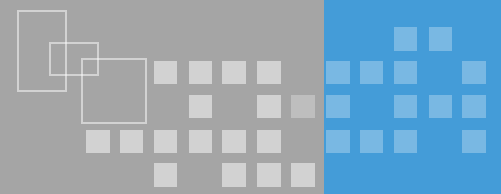


## ❖ 软件验收

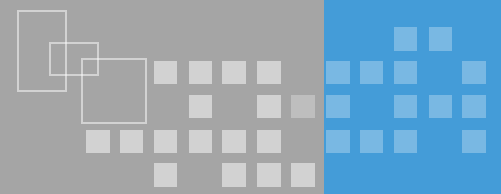
- 正式的验收流程
- 安全纳入到验收考虑中

## ❖ 安全部署

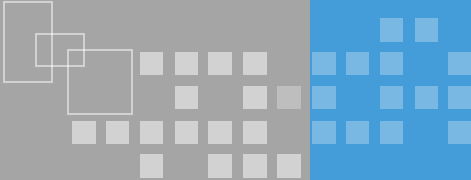
- 提供软件部署所需要的文档和工具
- 软件加固
- 软件安全配置

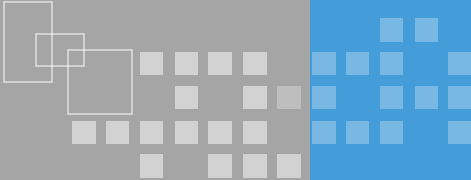


- ❖ 关于软件安全开发生命周期(SDL)，下面说法错误的是：
- ❖ A. 在软件开发的各个周期都要考虑安全因素
- ❖ B. 软件安全开发生命周期要综合采用技术、管理和工程等手段
- ❖ C. 测试阶段是发现并改正软件安全漏洞的最佳环节，过早或过晚检测修改漏洞都将增大软件开发成本
- ❖ D. 在设计阶段就尽可能发现并改正安全隐患，将极大减少整个软件开发成本



- ❖ 针对软件的拒绝服务攻击是通过消耗系统资源使软件无法响应正常请求的一种攻击方式，在软件开发时分析拒绝服务攻击的威胁，以下哪个不是需要考虑的攻击方式：
- ❖ A. 攻击者利用软件存在逻辑错误，通过发送某种类型数据导致运算进入死循环，CPU 资源占用始终100%
- ❖ B. 攻击者利用软件脚本使用多重嵌套查询，在数据量大时会导致查询效率低，通过发送大量的查询导致数据库响应缓慢
- ❖ C. 攻击者利用软件不自动释放连接的问题，通过发送大量连接消耗软件并发连接数，导致并发连接数耗尽而无法访问
- ❖ D. 攻击者买通了 IDC 人员，将某软件运行服务器的网线拔掉导致无法访问

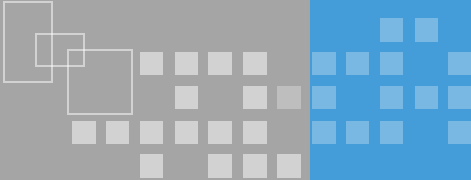
- 
- ❖ 最小特权是软件安全设计的基本原则，某应用程序在设计时，设计人员给出了以下四种策略，其中有一个违反了最小特权的原则，作为评审专家，请指出是哪一个？
  - ❖ A. 软件在Linux 下按照时，设定运行时使用nobody 用户运行实例
  - ❖ B. 软件的日志备份模块由于需要备份所有数据库数据，在备份模块运行时，以数据库备份操作员账号连接数据库
  - ❖ C. 软件的日志模块由于要向数据库中的日志表中写入日志信息，使用了一个日志用户账号连接数据库，该账号仅对日志表拥有权限
  - ❖ D. 为了保证软件在 Windows 下能稳定的运行，设定运行权限为 system，确保系统运行正常，不会因为权限不足产生运行错误



❖ 管理人员设计了网站, 以用户提交用户名和密码的方式验证身份, 为防止被攻击, 首先以用户名密码验证, 然后再对用户名密码加密处理存储, 然后再对存储数据进行备份, 以下原则正确的是

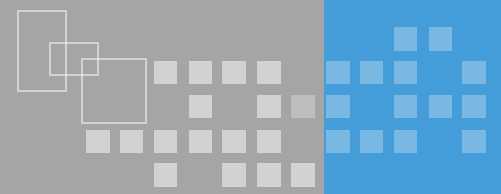
- ❖ A. 最小特权
- ❖ B. 权限分享
- ❖ C. 纵深防御
- ❖ D. 最少共享机制



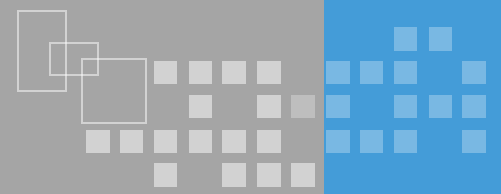


某电子商务网站最近发生了一起安全事件，出现了一个价值1000 元的商品用1 元被买走的情况，经分析是由于设计时出于性能考虑，在浏览时使用Http 协议，攻击者通过伪造数据包使得向购物车添加商品的价格被修改。利用此漏洞，攻击者将价值1000 元的商品以1 元添加到购物车中，而付款时又没有验证的环节，导致以上问题，对于网站的这个问题原因分析及解决措施。最正确的说法应该是？

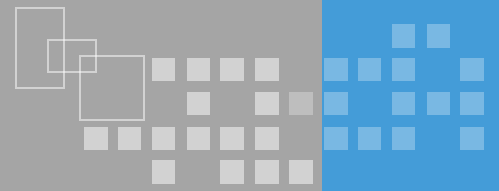
- ❖ A. 该问题的产生是由于使用了不安全的协议导致的，为了避免再发生类似的闯题，应对全网站进行安全改造，所有的访问都强制要求使用https
- ❖ B. 该问题的产生是由于网站开发前没有进行如威胁建模等相关工作或工作不到位，没有找到该威胁并采取相应的消减措施
- ❖ C. 该问题的产生是由于编码缺陷，通过对网站进行修改，在进行订单付款时进行商品价格验证就可以解决
- ❖ D. 该问题的产生不是网站的问题，应报警要求寻求警察介入，严惩攻击者即可



- ❖ 关于源代码审核，下列说法正确的是：
- ❖ A. 人工审核源代码审核的效率低，但采用多人并行分析可以完全弥补这个缺点
- ❖ B. 源代码审核通过提供非预期的输入并监视异常结果来发现软件故障，从而定位可能导致安全弱点的薄弱之处
- ❖ C. 使用工具进行源代码审核，速度快，准确率高，已经取代了传统的人工审核
- ❖ D. 源代码审核是对源代码检查分析，检测并报告源代码中可能导致安全弱点的薄弱之处



- ❖ 软件安全设计和开发中应考虑用户隐私保护，以下关于用户隐私保护的哪个说法是错误的？
- ❖ A. 告诉用户需要收集什么数据及搜集到的数据会如何被使用
- ❖ B. 当用户的数据由于某种原因要被使用时，给用户选择是否允许
- ❖ C. 用户提交的用户名和密码属于隐私数据，其它都不是
- ❖ D. 确保数据的使用符合国家、地方、行业的相关法律法规



- ❖ 软件安全开发生命周期模型
- ❖ 软件安全需求与设计
  - 威胁建模与攻击面
- ❖ 软件安全实现
- ❖ 软件安全测试
- ❖ 软件安全交付



**谢谢，请提问题！**