SOFTWARE TESTING

Chapter 7

System Testing

School of Data & Computer Science
Sun Yat-sen University

Approaches & Technologies

中山大學
SUN YAT-SEN UNIVERSITY

OUTLINE

软件故障模型的概念

- 软件故障模型 (Software Fault Model) 指明在软件执行时，某种类型的软件故障引起软件失效的现象和原因，是一种由测试人员的经验和直觉归纳而来、可以重复使用的固定模式 (Pattern)。
- 软件故障模型具有统一的框架，可以针对具体的软件类型、应用环境甚至开发工具建立故障模型的实例。
- 基于软件故障模型的测试方法
  - 软件故障模型决定了测试策略以及测试用例。
  - 应用软件故障模型迫使软件暴露错误。
    - 针对每一种软件故障类型，测试人员需要详细了解故障原因以及利用故障模型的测试方法，按照故障模型所提供的缺陷类型和寻找该类缺陷的方法找到尽量多的软件缺陷。

软件故障模型的概念

- 软件故障模型是一个二维模型，其设计考虑两个因素：
  - 软件功能和技术特点
    - 包括：输入、输出、数据以及处理等。
    - 只接受正确的输入并正确处理，只输出用户接受并且正确的输出；保持数据结构的完整性 (数值、精度和位置) 和自我保护的合法计算、功能交互和数据共享。
  - 软件操作环境
    - 包括：用户界面、文件系统、操作系统环境、其它软件环境、操作系统 API 等。
    - 主要关注用户界面、文件系统接口和数据库系统接口、资源调配和管理，以及操作系统 API 调用。

软件故障模型的概念

21种软件故障模型

- 输入非法数据
- 输入默认值
- 输入特殊字符集
- 输入使缓冲区溢出的数据
- 输入产生错误的合法数据组合
- 产生同一个输入的各种可能输出
- 输出不符合业务规则的无效输出
- 输出属性修改后的结果
- 屏幕刷新显示
- 数据结构溢出
- 数据结构不符合约束
- 操作数与操作符不符
- 递归调用自身
- 计算结果溢出
- 数据共享或关联功能计算错误
- 文件系统超载
- 介质忙或不可用
- 介质损坏
- 文件名不合法
- 更改文件访问权限
- 文件内容受损

软件故障模型的概念

- 21种软件故障模型
  - 例：输入非法数据。
    - 缺陷原因
      - 开发人员处理非法输入的一般方法：拒绝不正确的输入数据；接收到不正确的数据后，提示错误信息并拒绝处理；允许不正确的输入并进行处理，软件失效时调用异常处理程序，显示错误信息。上述检查非法输入的代码可能被忽视，导致处理非法输入出错。
    - 测试方法
      - 从输入值的属性出发设计测试用例：

      (1) 类型：输入无效的类型数据产生的信息。

      (2) 长度：对字符型输入太多的字符数据产生的信息。

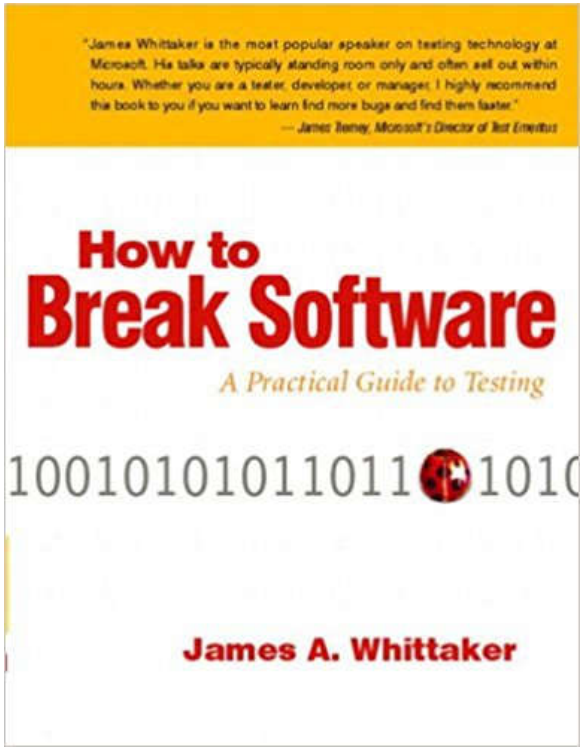
      (3) 边界值：输入等于边界值或超过边界值的数据产生的信息。

- 软件故障模型的概念
  - 21种软件故障模型
    - 例：同一个输入的各种可能输出。
      - 缺陷原因
        - 单个输入产生多种输出的情况与先前的输入和被测系统的状态都有关系。例如在文字处理程序中单击“关闭”按钮，如果文件被编辑且未被保存，程序应提示是否保存。如果文件已被保存过，则直接关闭。
        - 可能忽略某一个输出对某些系统状态的依赖。
      - 测试方法
        - 理解需求规格说明书中的内容，针对输出对输入和系统状态之间的依赖关系设计测试用例。

- 软件故障模型的概念
  - 21种软件故障模型
    - 例：介质损坏。
      - 缺陷原因
        - 损坏的介质使操作系统传回错误代码，这些错误代码没有在应用程序中得到处理。
        - 操作系统不能检测出所有的介质损坏错误。
      - 测试方法
        - 使用损坏的介质 (如刮伤、灰尘、磁干扰等)，检查应用程序对介质损坏错误的处理能力。应用程序可以对错误进行处理或者将问题告知用户，并确保用户数据文件不丢失或受人为损坏。

软件攻击测试概述

- 软件攻击测试针对软件系统中容易出现的故障进行测试。
  - 软件攻击测试的核心是基于软件故障模型的测试用例设计。
  - How to Break Software - *James Whittaker*



软件攻击测试概述

软件攻击测试的考虑因素

- 攻击目标
  - 一个攻击测试所针对的具体的软件功能需求
- 攻击方法
  - 攻击测试的具体方法的选择和操作步骤的设计
- 攻击成功的原因分析
  - 攻击测试的目的是迫使软件暴露其实现过程中的问题。一次攻击测试成功的背后必然蕴含着形成软件错误的原因。
- 攻击有效性的确认
  - 事先定义的攻击成功的标志

软件攻击测试的分类

- 典型软件攻击测试方法 (Patterns) 分类：
  - 用户接口输入攻击：6种
  - 用户接口输出攻击：4种
  - 用户接口数据攻击：3种
  - 用户接口计算攻击：4种
  - 文件系统介质攻击：3种
  - 文件系统文件攻击：3种
  - 操作系统和软件接口攻击：2种
- 软件攻击测试的核心是基于软件故障模型的测试用例设计，上述典型的软件攻击测试方法分类 (25种) 与软件故障模型 (21种) 对应。

软件攻击测试的分类

- 用户接口输入攻击：6种
  - 使用非法输入
    - 是否有适当 (正确) 的错误处理代码处理所有的非法输入。
  - 使用默认值的输入
    - 是否实施了变量初始化。
  - 使用特殊字符集和数据类型的合法输入
    - 是否正确处理了特殊字符和数据类型。
  - 使用使缓冲区溢出的合法输入
    - 是否检查了字符串/缓冲区的边界。
  - 使用可能产生错误的合法输入组合
    - 是否考虑了一些单个合法输入进行组合后的合法性。
  - 重复输入相同的合法输入序列
    - 是否考虑了循环处理的边界。

软件攻击测试的分类

- 用户接口输出攻击：4种
  - 产生同一个输入的各种可能输出
    - 一个正确的输入在不同情况下可能产生不同的输出，是否充分考虑了这些不同情况。
  - 强制产生不符合业务背景知识的无效的输出
    - 开发人员是否了解域/业务背景知识；
    - 是否会产生不符合业务背景的输出。
  - 强制通过输出修改一些属性
    - 是否已经同步了初始化代码和修改代码。
  - 检查屏幕刷新
    - 屏幕刷新时机是否正确；
    - 屏幕刷新区域计算是否正确。

软件攻击测试的分类

- 用户接口数据攻击：3种
  - 制造使内部数据与输入的组合不相容的情况
    - 处理输入的时候是否考虑到内部数据的各种可能的组合。
  - 制造使已有内部数据结构集合溢出的情况
    - 上溢：增加一个元素到集合中是否导致溢出；
    - 下溢：删除集合中最后一个元素，或者从空集合中删除元素是否导致溢出。
  - 制造使已有内部数据结构不符合约束的情况
    - 是否已经同步了初始化代码和修改代码。

软件攻击测试的分类

- 用户接口计算攻击：4种
  - 使用非法的操作数和操作符组合，攻击用户控制的计算要求
    - 是否考虑了操作符和计算要求的合法性。
  - 使函数递归调用自身
    - 是否考虑了循环/递归的中止。
  - 使计算结果溢出
    - 数据结构是否能够正确存储可能的计算结果。
  - 攻击共享数据或互相依赖的功能计算
    - 一个函数在修改共享数据的时候是否考虑了其他函数对这个共享数据的假设/约束。

软件攻击测试的分类

- 文件系统介质攻击：3种
  - 使文件系统超载
    - 是否检查了文件访问函数的返回值；
    - 是否正确处理了文件访问失败的情况。
  - 使介质处于忙或者不可用状态
    - 是否检查了文件/介质访问函数的返回值；
    - 是否正确处理了文件/介质访问失败的情况。
  - 损坏介质 (这时候操作系统可能认为介质可用)
    - 是否检查了文件/介质访问函数的返回值；
    - 是否正确处理了文件/介质访问失败的情况。

- 软件攻击测试的分类
  - 文件系统文件攻击：3种
    - 使用特殊字符/特殊长度/无效的文件名
      - 处理文件名的代码是否考虑了各种情况。
    - 改变文件访问权限
      - 访问文件的函数是否考虑了文件访问权限；
      - 文件访问失败时，是否有正确处理错误的代码。
    - 使文件内容错误，并让系统使用这个文件
      - 是否检查了文件访问函数的返回值；
      - 错误处理代码是否正确。

软件攻击测试的分类

- 操作系统和软件接口攻击：2种
  - 记录-仿真攻击
    - 模拟操作系统和操作环境故障，并记录软件对该故障的反应；
    - 软件是否正确处理了内存故障/网络故障等操作系统和软件接口故障。
  - 观察-失效攻击
    - 观察底层 API 调用，并动态修改 API 调用，制造错误；
    - 软件是否已经正确处理了操作系统和软件 API 调用错误的情况。
    - 观察-失效攻击模式一般应用于对可靠性和稳定性要求非常高的软件；

软件攻击突破口

- 对系统质量影响最大的地方通常也是系统最薄弱、最容易出现错误的地方，这些地方也是软件攻击的突破口。
- 软件攻击突破口测试内容通常包括：
  - 错误处理测试
  - 内存泄漏测试
  - 用户界面测试
  - 性能测试
  - 压力测试
  - 运行时错误检查
  - 回归测试

错误处理测试

- 错误处理
  - 软件采用约定的方法对系统的异常情况进行处理。
  - 错误处理测试检查软件在发生异常时是否进行处理，处理方法是否正确。
- 用户输入非法数据是最为典型的异常情况：
  - 不输入数据
  - 输入无效的数字数据 (如负数和字母数字串)
  - 输入任何被认为是非法的数据类型格式
  - 尝试使用不常用的数据组合
  - 使用零值
  - 输入超过或者短于要求长度的数据等
- 软件的健壮性测试考察软件能否正确处理无效等价类的输入，是软件质量保障的一个重要内容。

- 错误处理测试
  - 其它典型的异常情况：
    - 在系统不支持的平台上运行
    - 网络连接异常
    - 数据文件 (或数据库) 被破坏
    - 数据文件 (或数据库) 中有混乱的数据
    - 计算机断电后启动
    - 在用户界面上执行违反操作步骤的操作

内存泄漏测试

- 内存泄漏是一种典型的程序缺陷。
  - 内存泄漏导致应用程序不断消耗系统内存资源 (或虚拟存储器资源)，使程序运行性能下降，某些功能无法实现，甚至整个系统瘫痪。
- 在某些语言 (如C/C++语言) 程序中，由非熟练程序员编写的程序出现内存泄露是一个极其普遍的问题。
- 内存泄露测试可采用静态测试和动态测试技术。
  - 内存泄露检查工具
    - Compuware – BoundChecker
    - Rational - Purify

用户界面测试

- 用户图形界面测试的重点是正确性、易用性和视觉效果。
  - 用户界面测试的重要环节是界面中的文字检查和拼写检查；
  - 用户界面测试有时依赖于测试人员的主观判断。
- 用户界面测试的基本原则：
  - 易用性
  - 规范性
  - 合理性
  - 美观与协调性
  - 独特性
  - 快捷方式的组合
  - 排错性

用户界面测试

指 标	检 查 项 目
合适性和正确性	用户界面是否与软件的功能相融洽
	是否所有界面元素的文字和状态都正确无误
容易理解性	用户能否不必阅读手册就能使用常用功能
	是否所有界面元素 (比如图标) 都不会导致误解
	是否所有界面元素都提供了充分而且必要的提示
	界面结构是否清晰反映了工作流程
	用户是否容易清楚了解自己在界面中的位置而不至迷失
	是否提供了联机帮助文档
信息反馈	是否提供进度条、动画等反映正在进行的比较耗费时间的过程
	是否为重要的操作返回必要的结果信息
出错处理	是否对重要的输入数据进行校验
	执行有风险的操作时，是否给出“确认”、“放弃”选项
	是否根据用户的权限自动屏蔽某些功能
	是否提供了一些操作的“撤销”功能

用户界面测试

指 标	检 查 项 目
风格一致性	同类的界面元素是否有相同的视觉效果和操作方式
	界面字体是否一致
	是否符合广大用户使用同类软件的一般习惯
用户适合性	是否所有界面元素都具备充分且必要的键盘操作和鼠标操作
	是否初学者和专家都有操作这个界面的适当方式
	色盲或者色弱者是否能够正常操作这个界面
国际化	是否采用了国际通行的图标和语言
	度量单位、日期格式、人名等的定义使用是否符合国际惯例
布局合理性	界面的布局是否符合软件的功能逻辑
	界面元素是否在水平或垂直方向对齐
	界面元素的尺寸是否合理、行列间隔是否保持一致
	是否恰当地利用窗体和空间的空白以及分割线条
	窗口切换、移动、改变大小时，界面是否正常

用户界面测试

指 标	检 查 项 目
色彩设计	界面的色调是否和谐、令人满意
	是否用醒目的色彩强调了重要对象
	色彩的使用是否符合行业习惯
个性化	是否具有与众不同的、让人记忆深刻的界面设计
	是否在具备必要的一致性的前提下突出了个性化设计

- 性能测试
  - 性能测试包括并发性能测试、强度测试、破坏性测试等内容。
    - 并发性能测试
      - 评估系统交易或业务在渐增式并发情况下处理瓶颈问题的能力。
    - 强度测试
      - 在资源受限的情况下，发现因资源不足或资源竞争导致的错误。
    - 破坏性测试
      - 重点关注超出系统正常负荷若干倍的情况下，错误出现的状态和出现比率以及系统的错误恢复能力。
  - 性能测试可以通过黑盒测试或者白盒测试方法进行。

性能测试

- 性能测试的应用场合
  - 软件中某个模块涉及到复杂的计算，特别是一些基于人工智能的分析；
  - 涉及到大量数据的读写和通讯；
  - 涉及到数据检索，而被检索的对象具有大数据量；
  - 具有多个并发用户；
  - 软件在运行时，可用资源 (特别是 CPU 和内存) 可能在某些情况下出现紧缺 (例如一些嵌入式系统软件中)。

压力测试

- 压力测试的目的是获取系统能够正常运行的极限状态，是一种广义上的负载测试。
- 压力测试用于检查软件在面对大数据量时是否可以正常运行。
  - “大数据量”是相对与系统的可用资源而言的。
  - “大数据量”往往被视为小概率事件。
- 压力测试主要涉及的数据量包括：
  - 数据库规模
  - 磁盘空间
  - 内存空间
  - 数据通信量

运行时错误检查

- 软件运行时错误是指应用程序在运行期间执行了非法操作或某些操作失败时出现的错误。运行时错误在所有的软件错误中最具风险。
- 运行时错误检查在软件实际运行时检测程序，通过选择适当的测试用例，并对程序的运行状态进行监控以期发现程序的运行时错误。检测过程依靠系统编译程序和动态检查工具实现。
- 优点：
  - 检测的结果比较接近程序的真实运行状态，对于内存使用不当、空指针引用等错误的检测比较准确。
- 缺点：
  - 检测的全面性严重依赖于测试用例对代码的覆盖情况，检测过程运行时间较长。
  - 对于程序中递归调用等过程，动态检测很难覆盖全部的情况。

回归测试

- 回归测试指对某些已经被测试过的内容进行重新测试。
- 回归测试的必要性
  - 软件的增量影响软件的结构，导致新的缺陷；
  - 软件内容修改后引入新的缺陷。
- 回归测试策略
  - 通过用例的相关性分析，找到与软件修改部分相关的测试用例进行测试；
  - 在不进行任何分析的前提下对测试用例的全集进行测试；
  - 必要时可能会生成新的测试用例专门用来进行回归测试；
  - 每两周进行一次完整的回归测试；
  - 修复的缺陷数量累计到50个时进行一次完整的回归测试；
  - 在产品递交用户前5个工作日，进行一次完整的回归测试；
  - 回归测试通常可以使用自动化测试工具。

软件故障注入技术

- 软件故障注入 (Software fault injection, 或称软件故障植入) 按照选定的软件故障模型，人为地在目标系统中植入错误或强制程序进入某种特定状态，加速系统可能的失效的发生，进而对系统的异常处理机制 (异常代码) 进行评估。
- 软件故障注入通过分析系统对植入故障的反应，验证系统的容错性、健壮性、安全性等能力，是一类实现系统接口攻击的测试技术。
- 软件故障注入的目标是强制程序执行异常代码。
  - 功能代码
    - 程序中执行实现用户需求而完成软件任务的部分代码。
  - 异常代码
    - 程序中执行异常处理机制来处理程序错误的部分代码。

- 软件故障注入技术
  - 采用软件故障注入的原因：
    - 真正的攻击测试成本太高
      - 比如物理毁坏介质。
    - 真正的攻击测试工作量太大
      - 比如使用大文件填充硬盘、使用多任务抢占 CPU/网络/内存等资源。
    - 存在一些不容易实现的测试
      - 比如动态监控和修改 API 调用。

软件故障注入方法

- 软件故障注入方法包括静态注入方法和动态注入方法。
- 静态注入方法
  - 静态注入也称编译期注入 (Compile-time injection)。
  - 静态注入通过程序变异的方法，在源代码中人为地插入引发软件失效的代码，从而在被测软件的二进制代码文件内静态引入软件故障，使其在运行时出现错误。
  - 优点：
    - 静态注入方法占用的系统资源较少，不需要额外的系统状态监控进程，基本保持了被测系统原有的时序结构。
  - 缺点：
    - 静态注入方法需要针对具体的源代码设计并插入失效代码，灵活性不足，一般不能重用。
    - 源代码的修改和恢复会带来一定的风险。

- 软件故障注入方法
  - 动态注入方法
    - 动态注入也称运行期注入 (Runtime injection)。
    - 动态注入在被测系统正常运行过程中，在特定条件或状态下，修改被测系统的二进制代码映像，实现故障植入。
    - 动态注入方法的有效实现高度依赖设计者的业务水平，其系统复杂性带来了附加的风险。
    - 动态注入需要一个高优先级的监控进程进行故障注入管理，占用比较大的系统开销。

- 软件故障注入方法
  - 动态注入方法 (续)
    - 优点：
      - 不需要修改源代码
        - 系统测试阶段通常没有源代码的支持。
      - 达到模拟环境故障的目的
        - 环境故障如：网络中断、网络繁忙、内存匮乏。
      - 采用 API 调用失败的方式模拟故障
        - 在程序看来，任何环境故障实质上都是一系列的 API 调用失败。
      - 只影响被注入的程序
        - 使用模拟 API 调用失败方式，因此只会影响被注入的程序。

- 软件故障注入方法
  - 动态注入方法 (续)
    - 实现机制：截获 API
      - 基于调用源截获
        - 找出程序中调用的 API 的名字或 DLL，将其替换成测试函数。
      - 路径内截获
        - 如果程序中使用虚函数表 VTable 等技术间接调用 API 地址，则可以直接修改 VTable 中的 API 地址，将其替换成测试函数。
      - 目的地截获
        - 修改被调用 API 的地址内容，修改其头部代码，强制转向测试函数 (病毒方式)。

- 软件故障注入方法
  - 动态注入方法 (续)
    - 实现策略
      - 基于模式的故障动态注入
        - 模拟环境故障；记录故障特征和影响到的 API；修改影响到的 API。
        - 例如：记录-仿真攻击方法。
      - 基于调用的故障动态注入
        - 观察使用到的 API；独立地、细粒度地修改使用到的 API。
        - 例如：观察-失效攻击方法。

软件故障注入方法

- Holodeck 故障植入软件测试工具
  - Holodeck is a unique test tool that uses fault injection to simulate real-world application and system errors for Windows applications and services - allowing testers to work in a controlled, repeatable environment to analyze and debug error-handling code. Holodeck is the first commercial fault-simulation tool and was developed by leading researchers in the application quality field. It is used by organizations like Microsoft, Adobe, EMC and McAfee.
  - Developers: *Joe Basirico*, Security Innovation

软件系统的可用性

- 软件系统的可用性指在特定环境下，软件为特定用户用于特定目的时所具有的有效性、效率和主观满意度 (ISO 9241-11)。
  - 有效性是用户完成特定任务和达成特定目标时所具有的正确和完整程度。
  - 效率是用户完成任务的正确和完成程度与所用资源 (如时间) 之间的比率。
  - 主观满意度是用户在使用软件过程中所感受到的主观满意和接受程度。
- 软件可用性测试通过观察、访谈或二者相结合的方法，发现软件或软件原型存在的可用性问题，为设计改进提供依据。
- 软件可用性测试不是用来评估软件整体的用户体验，而主要用于发现系统中潜在的误解，或系统功能在使用时存在的错误。

系统可用性分级方法

- 五级划分法
  - 5级：无关紧要的错误；
  - 4级：问题虽小但却让用户焦虑；
  - 3级：中等程度，耗费时间但不会丢失数据；
  - 2级：导致数据丢失的严重问题；
  - 1级：灾难性错误，导致数据的丢失或者软硬件的损坏。
- 三级划分法
  - 低：会让参加者心烦或沮丧，但不会导致任务失败；
  - 中：与任务的失败有一定关系但不直接导致任务的失败；
  - 高：直接导致任务失败的问题。

- 系统可用性分级方法
  - 二维划分法
    - 根据可用性问题相关错误的出现频率和影响严重性进行划分。
  - 决策树法
    - 决策树法由可用性问题相关错误的频率 (偶然的还是经常的)、影响 (容易克服的还是很难克服的) 和持续性 (一次性的还是持续性的) 三个因素综合确定可用性级别。
  - 多维划分法
    - 先判断可用性问题所属的范围；
      - 结构设计错误、交互设计错误、视觉设计错误、语辞定义错误、软件代码错误、质量控制错误。
    - 再根据遇到可用性问题的人数判断严重程度。

系统可用性分级策略

- 根据产品特性和一些外围限制条件 (如时间、精力、资金、干系人特点等) 来选择响应的方法。
  - 对一些重要的软件系统，需要选择维度较全面的分类方法，例如多维划分法。
  - 对一些时间要求紧迫的软件项目，可以选择分类较少、容易操作的方法，例如五级、三级分类方法。
  - 对非专业人士或者新手，可以选择决策树方法。
- 可用性问题分级方法没有最好，只有最适合。
  - 分级词语准确，便于判别。
  - 分级的名称和具体的定义清晰准确。
  - 分级评估全面，能够包含发现的全部可用性问题。

系统完整性测试

- 软件系统的完整性
  - 软件系统的完整性指软件所包含文件内容正确完整，目录结构正确，同时基本功能表现正常。
  - 软件系统的完整性测试是整个软件产品发布过程中极其重要的综合性环节。它覆盖面广、涉及测试技术多样，测试内容包括兼容性测试、安装测试、基本功能测试等，同时需要大量的物理资源。
    - 软件系统完整性测试的基础是软件的完整性级别划分。
    - 软件完整性级别是指描述已经达成一致的软件完整性级别的分类标准，以及按照这个标准制定的相应类别和模块间的对应关系。

系统完整性测试

- 软件完整性级别
  - 根据软件失效所造成后果的危害程度，计算机软件的完整性级别被分为 A、B、C、D 四个等级。
    - 《GB/T 18492-2001 信息技术 系统及软件完整性级别》
  - 不同完整性级别软件的安全性要求不同，对软件的测试内容、测试要求和测试所采用的方法也有所不同。
  - 针对各种不同的软件测试类别，应有安全性方面的考虑。

动态测试工具

- 动态测试工具
  - 单元测试
    - JUnit
    - CppUnit
  - 代码覆盖
    - EclEmma
    - Gcov
  - … …

Thank you!