软件分析与测试
第五次课

严俊

中国科学院大学
University of Chinese Academy of Sciences
ISCAS 中国科學院軟件研究所
Institute of Software, Chinese Academy of Sciences

本次课内容

- 回归测试
- 性能相关的测试
- 新型软件测试技术

本次课内容

- 回归测试、测试用例选择

回归测试 (regression testing)

- 软件系统升级（代码修改）之后，对其重新进行测试，以确认没有引入新的错误。
- 如何做？
  - 重新测试全部用例
  - 选择一部分高价值的测试用例
  - 着重测试修改的部分 （代码修改影响 Change Impact Analysis）
  - ……
- 测试用例库可能很大。 （上千万？）

Research Topics

- Test case prioritization （测试用例优先级排序）
  先执行优先级高的测试用例。
  - Dan Hao et al. Test-case prioritization: achievements and challenges. Frontiers of Computer Science 10(5): 769-777, 2016.
- Test suite minimization （测试用例约减）
- Test Replay （测试重放）
- …

测试集最小化

- 给定一组测试用例 *P*，选择其（最小）子集*Ps*，达到高覆盖度。
- 当测试集合规模不大时
  - 建模为优化问题
  - **(branch coverage)**

    H. S. Wang, S. R. Hsu, and J. C. Lin. A generalized optimal path-selection model for structural program testing. Journal of Systems and Software, 10: 55-63, 1989.

Test case prioritization

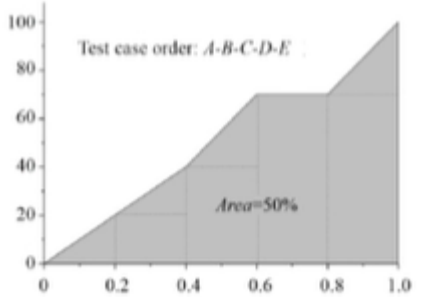
缺陷检测加权百分比（Average of the Percentage of Faults Detected, APFD）

$$APFD = 1 - \frac{TF_1 + TF_2 + \cdots + TF_m}{n * m} + \frac{1}{2n}$$

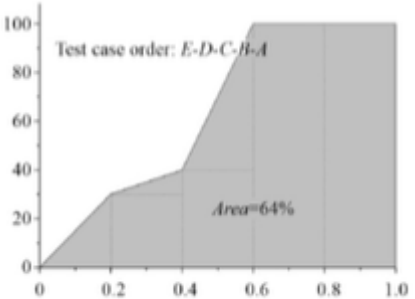
- n 表示测试用例集T 中测试用例数目；
- m 表示该测试用例集可检测软件缺陷的数量；
- TF 表示经过排序后的测试用例集 T' 中首次发现缺陷的测试用例在该序列中的次序。

	1	2	3	4	5	6	7	8	9	10
A	×				×					
B	×				×	×	×			
C	×	×	×	×	×	×	×			
D					×					
E								×	×	×

缺陷检测信息示例：该程序有5个测试用例和10个缺陷，如测试用例B可以检测出编号为1,5,6和7的缺陷。



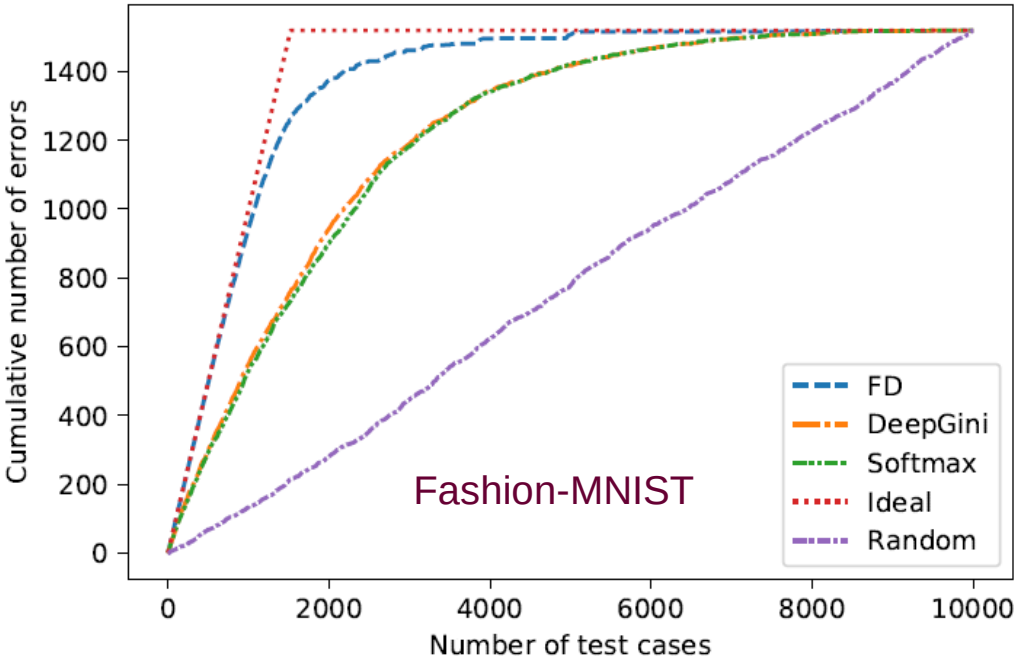
执行序列A-B-C-D-E 的APFD值



执行序列E-D-C-B-A的APFD值

例：基于神经元激活的测试用例排序方法

- 按熟悉度进行升序排列，熟悉度低的样本优先级高



$$APFD = 1 - \frac{\sum_{i=1}^{m} TF_i}{n * m} + \frac{1}{2n}$$

$$= 0.901$$

Kai Zhang, *et al.*, Neuron Activation Frequency Based Test Case Prioritization. TASE 2020

实例研究：automated regression testing

- **A practical solution for functional regression testing that captures the changes in database state (due to data manipulations) during the execution of the SUT.**
- **Test case prioritization: The classification tree models are used to prioritize test cases.**

E. Rogstad et al. (Simula Research Lab). Industrial experiences with automated regression testing of a legacy database application, ICSM 2011.

- **SUT**
  - **SOFIE: tax accounting system in Norway**
- **Challenges for testing Sofie:**
  - 这个系统必须处理大量的数据。
  - 构建自动化测试oracle是很困难的
  - 批处理
  - …

Approach and Tool

- **DART, a tool for regression testing of database applications (for batch jobs)**
- 在完全相同的输入数据和初始数据库状态上执行系统测试两次，一次使用系统的原始版本(baseline)，另一次使用更改后的系统版本(delta)。
- 计算两次运行时数据库状态的差异。差异要么是由于有效的更改，要么是由于回归错误造成的。

差分测试 (Differential Testing)

- 同样的测试输入，对于不同的系统，应该有一致的结果
  - 同一系统的不同版本（纵向）
  - 不同的系统（横向）
- 测试应用
  - 编译器测试
  - 浏览器测试
  - ……
- 与蜕变测试的关系？

思考

- **组合测试如何做回归测试？**
- 用户可能会要求某些取值组合必须被测试，这种取值组合被称为种子(seed)。

  例：AETG 这一段测试规范

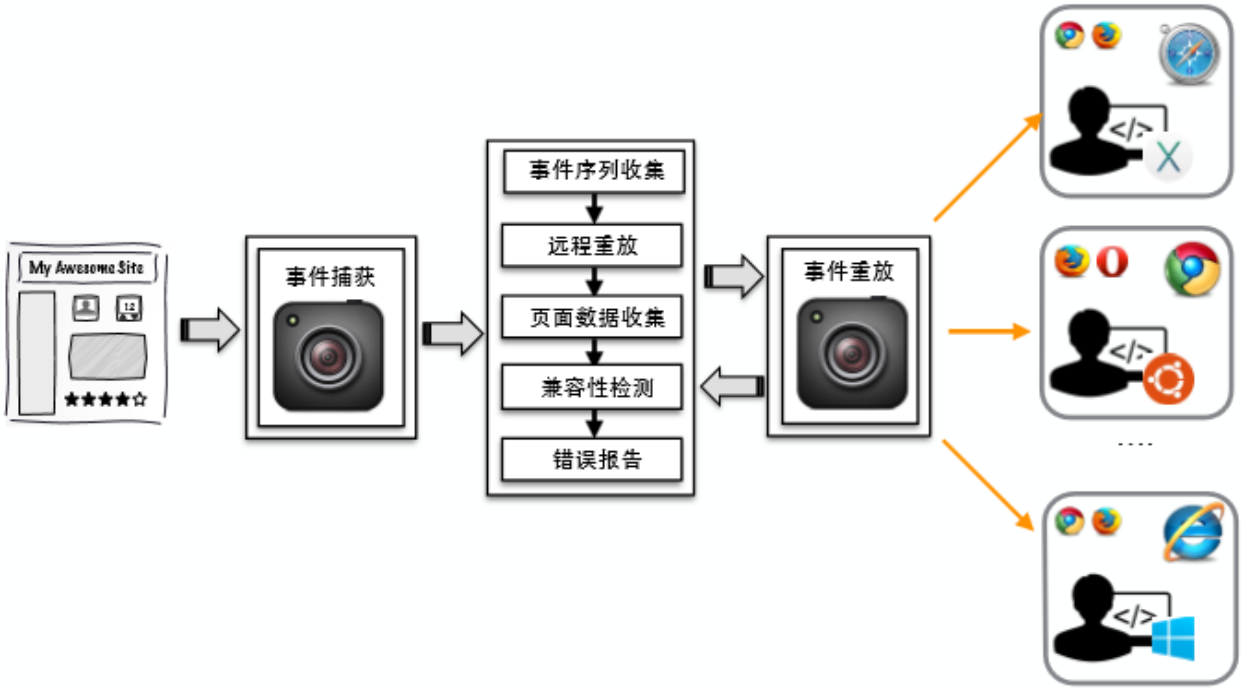
  ```
  seed {
  a b c d
  2 4 8 3
  }
  ```

  表示(a, b, c, d) = (2, 4, 8, 3) 这种组合形式在测试用例集中至少出现一次。

捕获/重放技术（吴国全）

- 目的：支持对Web应用的错误分析与定位。
- 跨浏览器的捕获/重放技术
  - Mugshot[NSDI'10]是一套针对客户端JavaScript应用的调试工具，核心是用标准的JavaScript实现对非确定性事件的录制和重放。
  - Jalangi[ESEC/FSE'13]、JSBench
- 针对特定浏览器的捕获/重放技术
  - Timelapse[UIST'13]受虚拟机捕获/重放技术启发，通过对Webkit插装支持捕获/重放各种非确定性事件。
  - WaRR[DSN'11]

基于捕获/重放的跨浏览器检测方法



X-CHECK的总体流程图

本次课内容

- 性能测试、负载测试、压力测试

性能/负载测试

- 软件不仅功能正确，还具有一定的性能指标。
  - non-functional requirements （非功能性需求）
- 这不仅取决于软件本身的设计和实现，还依赖于其他因素：
  - 硬件（如服务器的配置）；网络带宽；基础软件（数据库管理系统、web应用服务器）；用户个数、使用习惯；…
- 模拟实际软件系统所承受的不同负载，观察被测软件系统的响应时间和数据吞吐量等指标，找出其问题（性能瓶颈）。

基本概念

- 性能测试 (performance testing)
  - 性能指标（响应时间、资源使用、能耗等）
- 负载测试 (load testing)
  - 评估系统在负载下的行为，以检测与负载相关的问题
- 压力测试 (stress testing)
  - 高强度、极端条件下的测试

负载测试

- **负载设计**
  - 采用真实的负载模拟设计中可能出现的负载
  - 在负载中植入错误，用于暴露于负载相关的缺陷
- **负载测试执行**
  - 采用真实的用户手工执行
  - 采用负载生成器（ load drivers ）自动生成
  - 特定的平台（ e.g., a platform which enables deterministic test executions）
- **负载测试结果分析**
  - verifying against known thresholds (e.g., detecting violations in reliability requirements)
  - 检查已知问题(例如，内存泄漏检测)
  - 推断异常系统行为

An example (by Grig Gheorghiu)

SUT: web app. with database back-end

1. 从连接到数据库服务器的一个Web/应用程序服务器开始。
2. 在Web服务器上运行一个负载测试，从10个并发用户开始，每个用户总共向服务器发送1000个请求。增加用户数量，以10为增量，直到达到100个用户。
3. 分析（ profile ）应用程序和数据库，看看代码/SQL查询/存储过程中是否有需要优化的热点（hot spots）。

Example

4. 假设Web服务器的回复率在50个用户左右开始趋向平缓。→ SQL查询中存在热点，需要优化
   - 调优代码和数据库查询
   - 在服务器上投入更多的硬件
5. 从第2步重新运行负载测试，在性能开始下降之前，现在有75个并发用户。
6. 在实际启动应用程序之前，在“实际”生产环境中重复上述步骤。

性能/负载测试工具 open source tools

httperf (https://github.com/httperf/httperf)

-- a testing tool to measure the performance of web servers

Apache JMeter (http://jmeter.apache.org)

- **100% pure Java application designed to load test functional behavior and measure performance.**

性能/负载测试工具 LoadRunner

HP (Mercury) LoadRunner

(https://software.microfocus.com/zh-cn/software/loadrunner)

- **automated performance and load testing tool**
- **can simulate many users concurrently using the SUT, recording and analyzing its performance**
- **available in the Cloud**

Stress testing -- Examples

- 北京2008奥运售票系统
- 铁路售票系统 12306

- 金融：

2017年9月26日，“网联”平台与各大银行以及支付宝等全国交易量最大的商业银行和支付机构，针对高发支付交易场景开展了联合生产压力测试。

压力测试数据自动生成

- 不仅依赖于用户数（进程数），还取决于程序逻辑、执行路径
- 如何寻找压力很大的执行路径（场景）？ Model-based testing
  - Jian Zhang and S. C. Cheung. Automated test case generation for the stress testing of multimedia systems. Software Pract. Exper. 32(15): 1411-1435 (2002)
  - Vahid Garousi, Lionel C. Briand and Yvan Labiche. Traffic-aware stress testing of distributed systems based on UML models. ICSE 2006.
  - Vahid Garousi, Lionel C. Briand and Yvan Labiche. Traffic-aware stress testing of distributed real-time systems based on UML models using genetic algorithms. Journal of Systems and Software 81(2): 161-185 (2008)

Stress Testing – test data generation

Model-based testing

- 从一些模型(例如，活动图)中提取路径。
- 从每个路径中获取资源消耗信息。
- 建模成约束求解/优化问题。
- **求解**

Stress Testing – test data gen.

- Extract paths from some model (e.g., activity diagram).
- From the path, obtain resource consumption information.
- Generate constraint solving/optimization problems.
- Solve them !

```
min:-100 (x00-x01) -1200
   (x10-x11) -1800 (x20-x21) -
   56 (x30-x31) -1500 (x40-x41)
   -150 (x50-x51) -1440 (x60-
   x61) -25 (x70-x71) -200
   (x80-x81) -230 (x90-x91)

u1 = 1; r1 = 1; v0 <= 25; s0
   <= 25; u0 = v1; r0 = s1; u1
   <= u0; x01 = u1 or x11 = u1
   or x21 = u1; x00 <= u0; x10
   <= u0; x20 <= u0; x01 = x21;
   x10 = x20; x00 - x01 <= 5;
   x10 - x11 <= 6; x20 - x21
   <= 15; v1 <= v0; x31 = v1
   or x41 = v1; …
```

约束相关的问题

- **找一组解 - 约束满足问题**
  - **SAT，LP, SMT…**
- **找最优解 - 最优化问题**
  - **MAX-SAT, LP, SMT-OPT**
- **找所有解的个数 - 计数问题**
  - **Counting**

	找一个解	找最优解
逻辑与算术	SMT	SMT-OPT
线性算术约束（不等式）	LP, …	Linear programming

常用优化算法

- 部分完备算法
  - 线性规划 LP
  - Max-SAT
  - SMT-OPT
- 贪心算法、梯度下降算法
- 元启发式搜索
  - 模拟退火
  - 遗传算法
  - ……

本次课内容

- 回归测试、测试用例选择
- 性能测试、负载测试、压力测试
- 应用研讨（实例研究）

案例：网站测试

据不完全统计，近几年，淘宝崩过、拼多多崩过、微信崩过、百度地图崩过、美团崩过、钉钉崩过、知乎崩过、哈啰崩过，豆瓣等更是多次崩了。可谓是**“没有崩过的不足以称之为互联网企业”**。甚至不少崩过的互联网企业也做云服务，帮助别人解决服务器问题。

案例：2021.07.13 B站

2021年7月13日22:52，SRE收到大量服务和域名的接入层不可用报警，客服侧开始收到大量用户反馈B站无法使用，同时内部同学也反馈B站无法打开，甚至APP首页也无法打开。

- **23:25 - 23:55** 未恢复的业务暂无其他立即有效的止损预案，此时尝试恢复主机房的SLB。我们通过Perf发现SLB CPU热点集中在Lua函数上，怀疑跟最近上线的Lua代码有关，开始尝试回滚最近上线的Lua代码。
- **01:10 - 01:27** 使用Lua 程序分析工具跑出一份详细的火焰图数据并加以分析，发现 CPU 热点明显集中在对 lua-resty-balancer 模块的调用中，从 SLB 流量入口逻辑一直分析到底层模块调用，发现该模块内有多个函数可能存在热点。
- **01:39 - 01:58** 在分析 debug 日志后，发现 lua-resty-balancer模块中的 _gcd 函数在某次执行后返回了一个预期外的值：nan，同时发现了触发诱因的条件：某个容器IP的weight=0。

哔哩哔哩技术，2021.07.13 我们是这样崩的，2022-07-12 12:00。
https://mp.weixin.qq.com/s/nGtC5lBX_Iaj57HIdXq3Qg

罪魁祸首

- Lua 是动态类型语言。Lua在对一个数字字符串进行算术操作时，会尝试将这个数字字符串转成一个数字。
- 执行数学运算 n % 0，结果为 NaN（Not A Number）。
- _gcd函数对入参没有做类型校验，允许参数b传入："0"。同时因为"0" != 0，所以此函数第一次执行后返回是 _gcd("0",nan)。如果传入的是int 0，则会触发[if b == 0]分支逻辑判断，不会死循环。
- _gcd("0",nan)函数再次执行时返回值是_gcd(nan,nan)，然后Nginx worker开始陷入死循环，进程 CPU 100%。

```
17  local _gcd
18  _gcd = function (a, b)
19      if b == 0 then
20          return a
21      end
22
23      return _gcd(b, a % b)
24  end
25
```

部分经验

- 运维团队做项目有个弊端，开发完成自测没问题后就开始灰度上线，没有专业的测试团队介入。此组件太过核心，需要引入基础组件测试团队，对SLB输入参数做完整的异常测试。
- 跟社区一起，Review使用到的OpenResty核心开源库源代码，消除其他风险。基于Lua已有特性和缺陷，提升我们Lua代码的鲁棒性，比如变量类型判断、强制转换等。
- 招专业做LB的人。我们选择基于Lua开发是因为Lua简单易上手，社区有类似成功案例。团队并没有资深做Nginx组件开发的同学，也没有做C/C++开发的同学。