软件分析与测试
第四次课

严俊

中国科学院大学
University of Chinese Academy of Sciences
ISCAS 中国科学院软件研究所
Institute of Software, Chinese Academy of Sciences

本次课内容

- 测试用例生成技术
- 回归测试
- 智能软件测试技术

测试准则 Test Criterion

- 功能
  - 作为测试可以结束的终止条件;
  - 作为软件质量的一个度量，一个测试集合代表了一个质量等级
  - 用于生成测试用例。如果两个测试集合符合相同的测试准则，我们称他们为等效的。
- 一般定义为某种覆盖（Coverage）

Hierarchy (Aditya P. Mathur)

- 需求 (requirement)
- 函数 (function/method)
- 语句 (statement)
- 判定 (decision)
- 数据流 (data-flow)
- …

错误检测率与覆盖率增长的关系（Gittens）

- 经验研究
  - 商用软件，221,629行，221个模块，
  - 代码块覆盖
- 饱和效应
  - 在系统测试中，覆盖率达到51%-60%之前，错误检测率与覆盖率相关
  - 回归测试中，覆盖率达到61%-70%之前，错误检测率与覆盖率相关

覆盖率与找错

- 测试即使达到100%的语句、分支、MC/DC覆盖率，也可能会漏掉很多错误。
- 例.

```
int a[10], i, j;
INPUT(i);
if (i > 4)
  j = i-1;
else j = i+1;
a[j] = j;        // 可能越界
```

两个测试用例
1. i = 4
2. i = 5

巧合正确性

- Coincidental Correctness
- 执行了语句/路径，但未发现其中的错误。

- 例：
  - y = x * 2;
  - y = x ^ 2;

    - 如果输入数据 x==2, …

白盒测试用例生成与选择

Research problems

- Test data/case generation: How can we generate a test suite to achieve 100% statement/branch/… coverage? 测试生成

- Does high coverage indicate better fault detection capability? 发现错误的能力

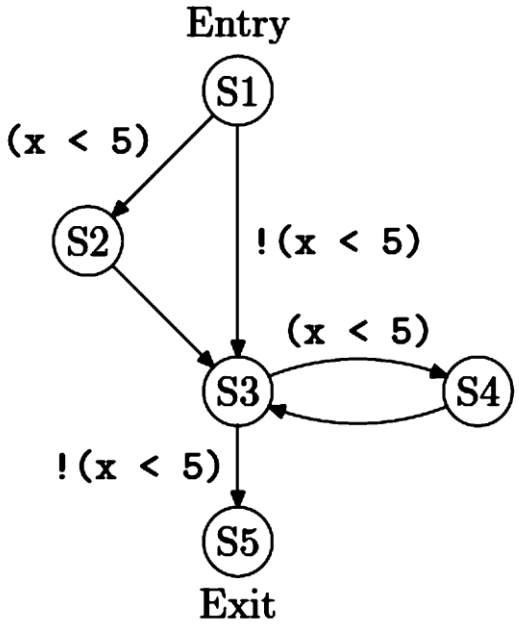
简单的测试生成

```
TS = EmptySet;
   do {
      产生测试数据 t;
      采用 t 执行程序，检查覆盖率;
      if (覆盖率增加)
            将 t 加入测试集TS;
   } while (覆盖率不达标);
   return TS;
```

测试路径

- CFG上的路径
  - 完整路径：*A path that starts at an initial node and ends at a final node*
  - 例：S1-S2-S3-S4-S3-S5
- 不可行路径
  - S1-S3-S4-S3-S5

```
void f(int x）{
/*S1*/ if (x < 5）
/*S2*/ y = 2;
/*S3*/ while (x < 5）
/*S4*/ x++;
/*S5*/ return;
}
```



基于路径的测试生成过程

```
TS = EmptySet;
    do {
      生成一条对覆盖率有贡献的测试路径p;
      if (可从p产生测试数据t)
            将t加入测试集TS;
      根据p统计覆盖率;
   } while (覆盖率不达标);
   return TS;
```

Path Feasibility – Ex.

```
int  i, j;
bool good;
if ((i > 2) && (j > 3))
{
     j = j-1;
     if (i+2j < 5)
        good = FALSE;
     else  good = TRUE;
}
```

- **Path 1:**

@((i > 2) && (j > 3))

j = j-1;

@(i+2j < 5)

good = FALSE;

Two paths

- **Path 1:**

```
@((i > 2) && (j > 3))
  j = j-1;
@(i+2j < 5)
  good = FALSE;
```

- **Path 2:**

```
@((i > 2) && (j > 3))
  j = j-1;
@! (i+2j < 5)
  good = TRUE;
```

如何分析路径产生测试数据

- 符号执行
- 自动推理/约束求解

```
@((i > 2) && (j > 3))
  j = j-1;
@!(i+2j < 5)
  good = FALSE;
```

```
@((i0 > 2) && (j0 > 3))
  j1 = j0-1;
@(i0+j1 < 5)
```

i0 > 2, j0 > 3, i0+ j0-1 >= 5

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。
- **(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.

路径的覆盖度矩阵

Branch

Path	Br_1	Br_2	…	Br_j	…
P_1	b_{11}	b_{12}	…	b_{1j}	…
P_2	b_{21}	b_{22}	…	b_{2j}	…
…					
P_i	b_{i1}	b_{i2}	…	b_{ij}	…
…					

b_{ij}: coverage frequency for path P_i over branch Br_j.

建模为优化问题

- Bool X_i = ITE(P_i is selected, 1, 0)
- 优化问题

$$min. \sum_i X_i$$

$$s.t. \ \sum_i X_i b_{ij} \geq 1 \text{ for all } j$$

（对每条边 Br_j）

- 整数规划/伪布尔优化（PBO, Pseudo-Boolean Optimization）

Branch Path Matrix

Path No.	branch No. / branch	1 a	2 b	3 c	4 d	5 e	6 f	7 g	8 h	9 i	10 j	11 k	12 l	13 m	14 n	15 o	16 p	17 q	18 r	19 s	20 t	P_BN
1>	ac	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2
2>	bdedfghjkmprti	0	1	0	2	1	1	1	1	1	1	1	0	1	0	0	1	0	1	0	1	14
3>	bdedfghjkmqsti	0	1	0	2	1	1	1	1	1	1	1	0	1	0	0	0	1	0	1	1	14
4>	bdedfghjlnkmprti	0	1	0	2	1	1	1	1	1	1	1	1	1	1	0	1	0	1	0	1	16
5>	bdedfghjlnkmqsti	0	1	0	2	1	1	1	1	1	1	1	1	1	1	0	0	1	0	1	1	16
6>	bdedfghjlnloprti	0	1	0	2	1	1	1	1	1	1	0	2	0	1	1	1	0	1	0	1	16
7>	bdedfghjlnloqsti	0	1	0	2	1	1	1	1	1	1	0	2	0	1	1	0	1	0	1	1	16
8>	bdedfghjloprti	0	1	0	2	1	1	1	1	1	1	0	1	0	0	1	1	0	1	0	1	14
9>	bdedfghjloqsti	0	1	0	2	1	1	1	1	1	1	0	1	0	0	1	0	1	0	1	1	14
10>	bdedfgi	0	1	0	2	1	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	7
11>	bdfghjkmprti	0	1	0	1	0	1	1	1	1	1	1	0	1	0	0	1	0	1	0	1	12
12>	bdfghjkmqsti	0	1	0	1	0	1	1	1	1	1	1	0	1	0	0	0	1	0	1	1	12
13>	bdfghjlnkmprti	0	1	0	1	0	1	1	1	1	1	1	1	1	1	0	1	0	1	0	1	14
14>	bdfghjlnkmqsti	0	1	0	1	0	1	1	1	1	1	1	1	1	1	0	0	1	0	1	1	14
15>	bdfghjlnloprti	0	1	0	1	0	1	1	1	1	1	0	2	0	1	1	1	0	1	0	1	14
16>	bdfghjlnloqsti	0	1	0	1	0	1	1	1	1	1	0	2	0	1	1	0	1	0	1	1	14
17>	bdfghjloprti	0	1	0	1	0	1	1	1	1	1	0	1	0	0	1	1	0	1	0	1	12
18>	bdfghjloqsti	0	1	0	1	0	1	1	1	1	1	0	1	0	0	1	0	1	0	1	1	12
19>	bdfgi	0	1	0	1	0	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	5

BRANCH PATH MATRIX

```
   *** STATEMENT-TESTING test path recommendation ***

 testpath set  1 , the min. path number=3
       path no.  1    12F
       path no.  4    134345678A89BCE6F
       path no.  9    134345678ABDE6F


   *** BRANCH-TESTING test path recommendation ***

 testpath set  1 , the min. path number=3
       path no.  1    ac
       path no.  4    bdedfghjlnkmprti
       path no.  7    bdedfghjlnloqsti
```

复杂约束下的优化问题

- Linear Programming
- SMT 优化

min. f(X)

s.t. **Ax <= b**

min. f(X)

s.t. **SMT约束**

约束可以是逻辑公式，而不只是线性不等式。

可以混合整数、实数、布尔变量，可以有蕴含、合取、析取、取反，甚至还能嵌套。

一个小例子

min. x - y
subject to:
$(((y + 3x < 3) \rightarrow (30 < y)) \vee (x <= 60))$
$\wedge ((30 < y) \rightarrow \neg (x > 3) \wedge (x <= 60))$

Stress Testing – test data gen.

- **Extract paths from some model (e.g., activity diagram).**

从某种模型中抽取路径

- **From the path, obtain resource consumption information.**

从路径中得到资源消耗信息

- **Generate constraint solving/optimization problems.**

- **Solve them !** 生成约束求解 / 优化问题

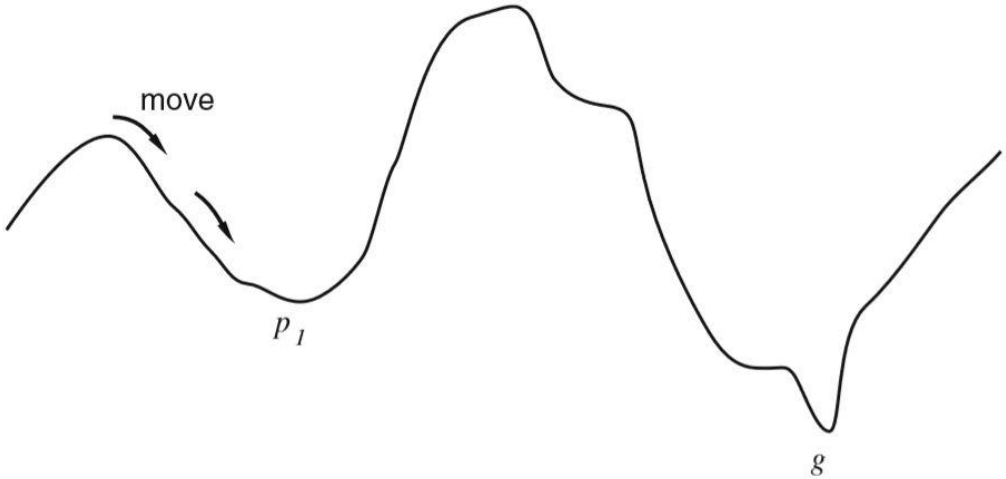
```
min:-100 (x00-x01) -1200
   (x10-x11) -1800 (x20-x21) -
   56 (x30-x31) -1500 (x40-x41)
   -150 (x50-x51) -1440 (x60-
   x61) -25 (x70-x71) -200
   (x80-x81) -230 (x90-x91)

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; ...
```

常用优化算法

- 部分完备算法
  - LP
  - Max-SAT
  - SMT-OPT
- 贪心
- 元启发式搜索

局部最优 vs 全局优化



更为复杂的优化问题

- 单目标优化

$min\ f(x)$

$s.t.\ constraints$

- 多目标优化（既要，又要）

$min\ f_1(x), f_2(x), \ldots, f_k(x)$

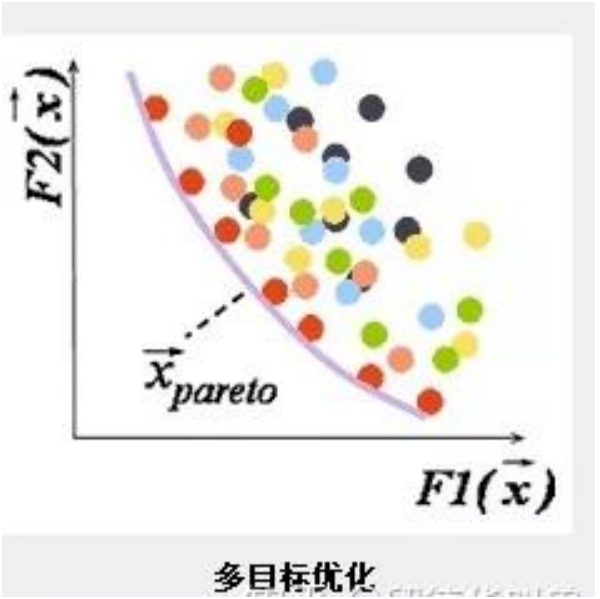
$s.t.\ constraints$

- 例：测试时间、测试成本、测试充分性

多目标优化

- x是搜索空间中一点，说 x为Pareto最优解，当且仅当不存在 x' (在搜索空间可行性域中)使得 $f_k(x') \leq f_k(x)$成立

- 它的解并非唯一，而是存在一组由众多 Pareto最优解组成的最优解集合，集合中的各个元素称为Pareto最优解或非劣最优解。



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

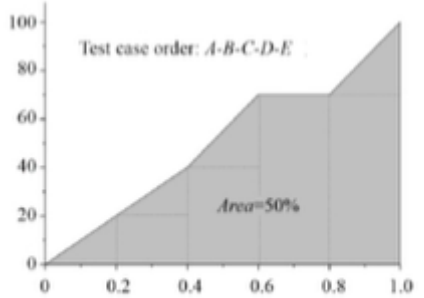
缺陷检测加权百分比（Average of the Percentage of Faults Detected, APFD）

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

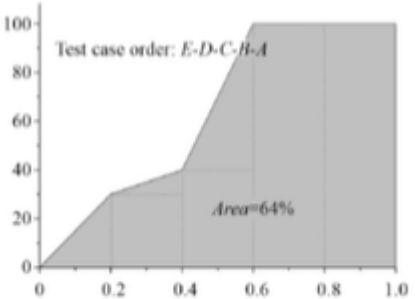
- n 表示测试用例集T 中测试用例数目；
- m 表示该测试用例集可检测软件缺陷的数量；
- TF 表示经过排序后的测试用例集 T' 中首次发现缺陷的测试用例在该序列中的次序。

	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。



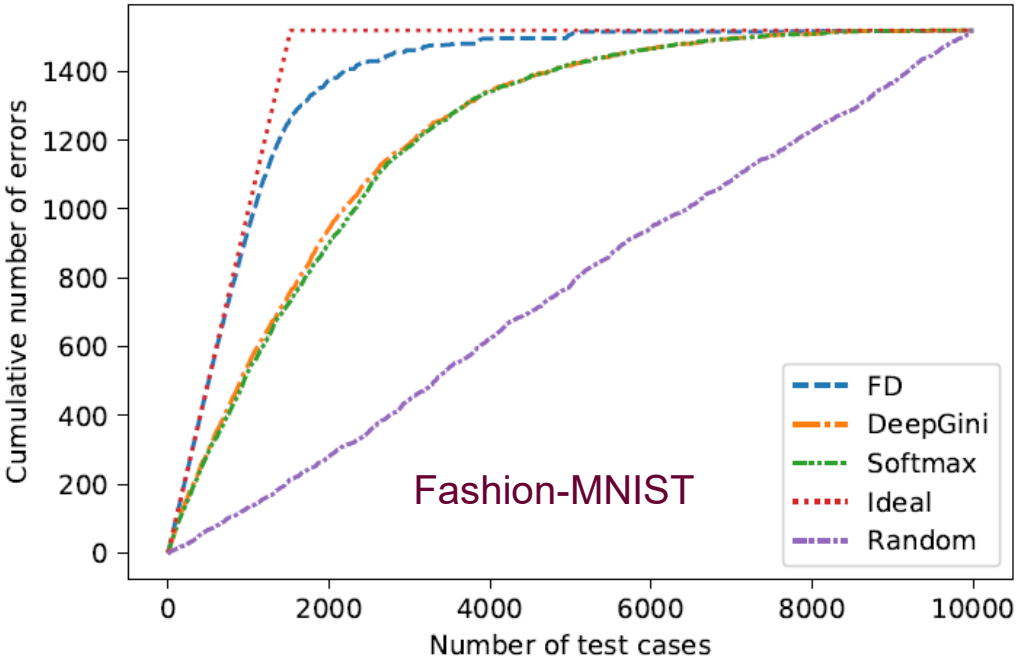
执行序列A-B-C-D-E 的APFD值



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.** 回归测试的对象是“变化”。
- **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** System Under Test
  - **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
  - …

我们在一个真实的、数据库密集型的税务系统（SUT）上做回归测试。
这个系统数据量大、规则复杂、oracle 难写。
所以我们用"数据库状态变化"作为判错依据，并用分类树模型对测试用例进行排序，从而实现**可扩展的自动化回归测试。**

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
  - ……

思考

- **如何采用组合测试方法开展回归测试？**
- 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

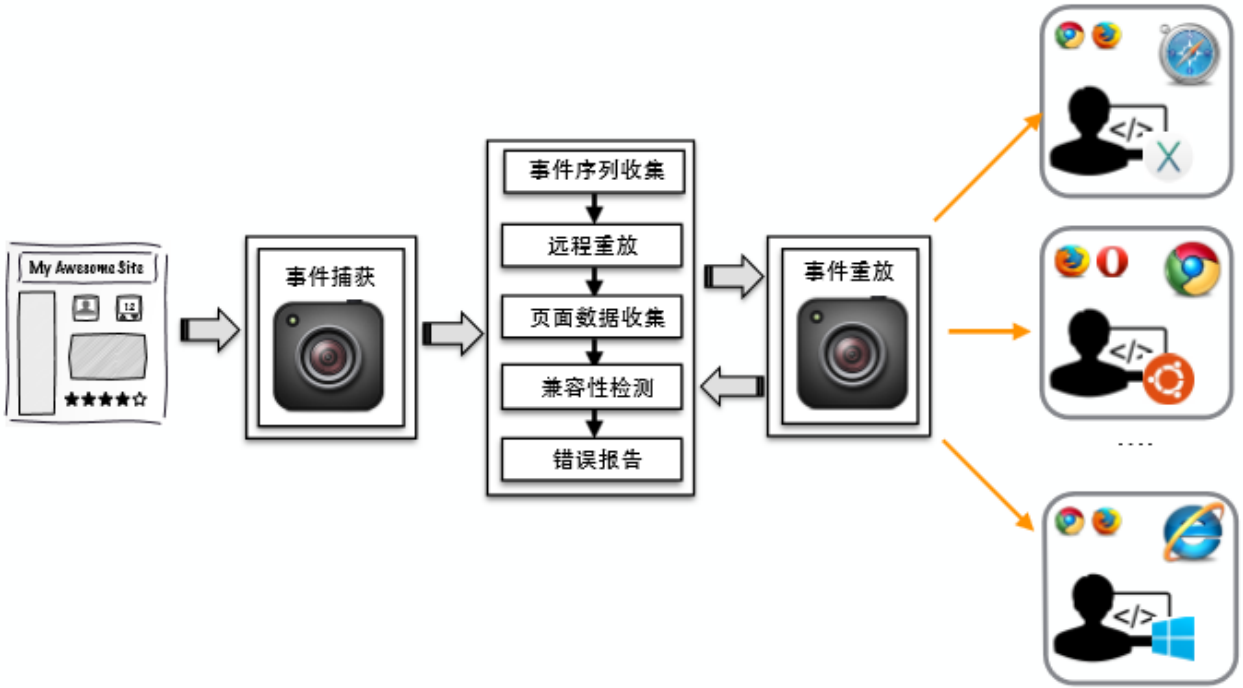
  ```
  seed {
  a b c d
  2 4 8 3
  }
  ```

  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

- **自动化测试辅助工具**
  - 测试执行工具
  - 覆盖度统计工具

Unit Testing Frameworks

- https://en.wikipedia.org/wiki/List_of_unit_testing_frameworks
- **A test automation framework**
  - integrates the function libraries, test data sources, object details and various reusable modules
  - has the advantage of low maintenance cost.
- This page is a list of tables of code-driven unit testing frameworks for various programming languages. Some but not all of these are based on xUnit.

Testing frameworks for various PLs

Contents [hide]

JUnit

- JUnit: a unit testing framework for Java
- It provides
  - annotations to identify test methods.
  - assertions for checking expected results.
  - test runners for running tests.
- JUnit tests can be run automatically and they check their own results.

Example: MessageUtil.java

```
public class MessageUtil {
   private String message;
   //Constructor
   //@param message to be printed
   public MessageUtil(String message){
      this.message = message;
   }
   // prints the message
   public String printMessage(){
      System.out.println(message);
      return message;
   }
}
```

Example： TestJunit.java

```
import org.junit.Test;
import static org.junit.Assert.assertEquals;

public class TestJunit {

   String message = "Hello World";
   MessageUtil messageUtil = new MessageUtil(message);

   @Test
   public void testPrintMessage() {
      assertEquals(message, messageUtil.printMessage());
   }
}
```

Example： TestRunner.java

```
import org.junit.runner.JUnitCore;
import org.junit.runner.Result;
import org.junit.runner.notification.Failure;

public class TestRunner {
   public static void main(String[] args) {
      Result result = JUnitCore.runClasses(TestJunit.class);

      for (Failure failure : result.getFailures()) {
         System.out.println(failure.toString());
      }
      System.out.println(result.wasSuccessful());
   }
}
```

案例-JUSTIN工具

- 自动生成Java单元测试数据的工具
  - 以Java字节码为输入
  - 以public方法为测试单元
  - 生成单元测试源码
  - 生成结果可在JUnit框架下编译执行
- 所需环境
  - JDK 1.8
  - Junit
  - 命令行
- 测试数据生成方法
  - 随机测试，模糊测试，……

- 方法参数的自动生成
  - 基本类型
    - 包含 int | short | byte | long | float | double | char | Boolean
    - String 类型的常量生成
  - 自定义类
    - 支持复杂对象参数的生成
  - 数组
    - 支持多维数组
    - 支持元素为基本类型和自定义类
    - 数组中自动添加元素
- 方法调用

Justin产生的测试用例

- 源代码

```java
package cn.ios.test;
public class GasStation {
    public int canCompleteCircuit(int[] gas, int[] cost) {
        int [] rest = new int [gas.length];

        int res = 0;

        for(int i = 0; i < gas.length; ++i){
            rest[i] = gas[i] - cost[i];
            res += rest[i];
        }

        if(res < 0)
            return -1;
        int len = rest.length;
        for(int i = 0; i < rest.length; ++i){
            int start = (i+1)%len;
            int end = i;
            int total = rest[i];
            if(total < 0)
                continue;
            while(start != end){
                total += rest[start];
                if(total < 0)
                    break;
                start = (start + 1)%len;
            }
            if(total >= 0)
                return i;
            else
                continue;
        }
        return -1;

    }
}
```

- 生成的测试代码

```java
package cn.ios.test;

import org.junit.Test;

public class GasStation_Test {

  @Test
  public void test_GasStation_canCompleteCircuit_0(){

      int[] intArray1 = {-1852899294, -27147850, -1423120474};
      int[] intArray2 = {1513584829};
      cn.ios.test.GasStation gasStation0 = new cn.ios.test.GasStation();
      gasStation0.canCompleteCircuit(intArray1, intArray2);

  }


}
```

发现的bug（Justin-str，ISSRE2022）

JDK 开发者确认或修复的Bug 信息

Bug 类型：	ArrayIndexOutOfBoundsException (2)
Issue ID：	I4MWI1 (Commit ID), 8279422
Bug 类型：	StringIndexOutOfBoundsException (14)
Issue ID：	8278186, 8279128, 8279129, 8279198, 8279218, 8279336, 8279341, 8279342, 8279362, 8279423, **21212bd18(Commit ID), 8279424, **411a404a9(Commit ID), **8baba7d11(Commit ID)
Bug 类型：	Infinite Loop (1)
Issue ID：	8278993

GUnit

https://github.com/cpp-testing/GUnit

- Based on GoogleTest/GoogleMock

Google Test (Google's C++ testing framework), by Zhanyong Wan

Gunit

http://googletesting.blogspot.com/2012/10/why-are-there-so-many-c-testing.html

- 系统的可扩展性
- C++应用太广。很难写一个在各种环境下都能工作的 C++ 测试框架。
- Google had a huge number of C++ projects that got compiled on various operating systems (Linux, Windows, Mac OS X, and later Android, among others) with different compilers and all kinds of compiler flags, and we needed a framework that worked well in all these environments and could handle many different types and sizes of projects.

覆盖率统计工具：llvm-cov

- 编译
  - clang -fprofile-instr-generate -fcoverage-mapping test.cc -o test
- 运行
  - LLVM_PROFILE_FILE=“test.profraw" ./test
- 对profraw文件索引
  - llvm-profdata merge -sparse test.profraw -o test.profdata
- 生成覆盖率报告
  - llvm-cov show ./test -instr-profile=test.profdata
  - llvm-cov report -show-region-summary=false ./test -instr-profile=test.profdata

覆盖率报告

- 第一列为程序的行号
- 第二列为该行代码被覆盖的次数
- Branch (line : column) 标记了分支的位置
- [True:10 , False:1]表示满足条件和不满足条件的覆盖次数
- 标红的行表示未被测试用例覆盖

```
  1|       |#include <stdio.h>
  2|       |
  3|       |int main(void)
  4|      1|{
  5|      1|    int i,total;
  6|      1|    total = 0;
  7|     11|    for(i=0;i<10;i++)
------------------
|  Branch (7:13): [True: 10, False: 1]
------------------
  8|     10|        total += i;
  9|      1|    if(total != 45)
------------------
|  Branch (9:8): [True: 0, False: 1]
------------------
 10|      0|        printf("Failure\n");
 11|      1|    else
 12|      1|        printf("Success\n");
 13|       |    /*else
 14|       |        printf("just test\n");*/
 15|      1|    return 0;
 16|      1|}
 17|       |
```

覆盖率总结

- Functions为程序中所含函数数量
- Missed Functions 为未被覆盖的函数
- Executed 为函数覆盖率
- Lines 代码总行数
- Missed Lines 未被覆盖的行的数量
- Cover 行覆盖率
- Branches 条件总数
- \- Missed Branches 未被覆盖了条件的数量

Filename	Functions	Missed Functions	Executed	Lines	Missed Lines	Cover	Branches	Missed Branches	Cover
/home/hurx/test/test.c	1	0	100.00%	11	1	90.91%	4	1	75.00%
TOTAL	1	0	100.00%	11	1	90.91%	4	1	75.00%

覆盖率总结报告