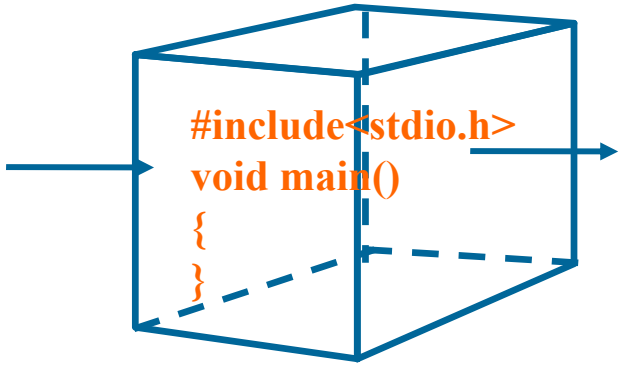
四、白盒测试

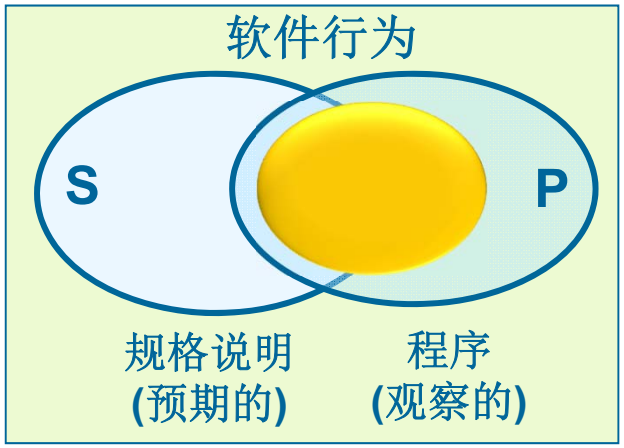
主要内容

1. 白盒测试简介
2. 常见白盒测试方法
3. 覆盖测试
4. 实现覆盖的途径



1. 白盒测试简介

- 也称结构性测试、逻辑驱动测试、基于程序的测试
- 特点
  - 将程序的执行表现与编码意图作比较
  - 关心软件内部设计和程序实现
  - 主要测试依据是代码和设计文档
  - 优点:支持严格定义、数学分析和精确度量
  - 缺点:不验证需求规格，无法功能遗漏等问题



- 主要使用阶段
  - 单元测试阶段：一般由开发人员进行
  - 集成测试阶段：一般由测试人员和开发人员共同完成

白盒黑盒互补，应组合使用，单一使用一种都不全面

主要内容

1. 白盒测试简介
2. **常见白盒测试方法**
3. 覆盖测试
4. 实现覆盖的途径

2. 常见白盒测试方法

- 静态测试
  - 人工代码检查
  - 代码自动审查
  - 风格检查

广义测试

- 动态测试
  - 覆盖测试分析
  - 运行时错误检测
  - 变异测试
  - . . . . .

静态测试

2.1 人工代码检查

- 包括代码审查、代码走查等
- 通常由一组人员来完成
  简化形式之一：结对编程（**Pair programming**）
- 通过正式或非正式会议形式
- 可发现 **30%-70%** 的错误
  – 非常有效!

2.1.1 代码审查(Code Inspection)

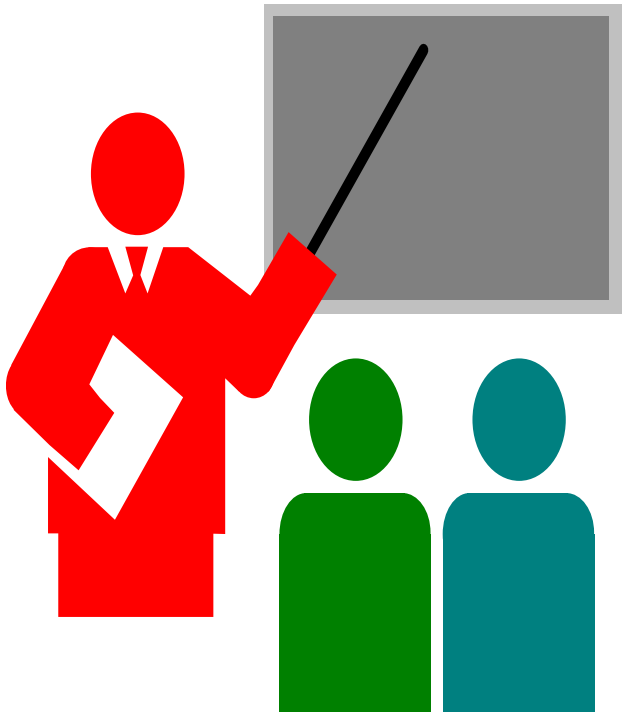
- 背景
  - 1976年**Fagan**
  - 1993年**Gilb and Graham**
  - 1996年最佳软件实践

Michael Fagan

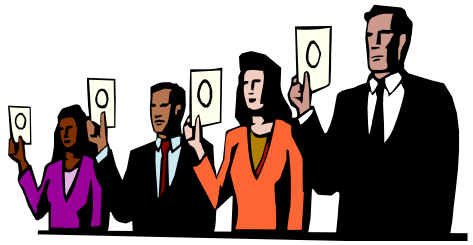
Fagan, M. "Design and Code Inspections to Reduce Errors in Program Development." IBM Systems Journal 15, 3 (1976): 182-211.

代码审查

- 由 **3-5** 人小组进行
  - （1）主持人：多面手
  - （2）作者：解释
  - （3）评论员：找出缺陷
  - （4）记录员：
- 一个会议协调人，负责分发相关材料，记录错误等
- 程序员一行一行解释程序
- 小组成员提问
- 通常审查小组有一个潜在错误的**checklist**以供审查



代码审查清单



- 领域性错误
  - ✓ 各个业务领域相关的功能和非功能错误
- 通用错误
  - ✓ 数据引用错误
  - ✓ 数据声明错误
  - ✓ 计算错误
  - ✓ 比较错误
  - ✓ 控制流程错误
  - ✓ 子程序参数错误
  - ✓ 输出错误
  - ✓ 其他检查

2.1.2 代码走查(Code Walkthrough)

- **3-5**人的团队
  - 协调者
  - 秘书：记录错误
  - 测试员：充当计算机的角色，在一定的测试集下人工推演程序的执行

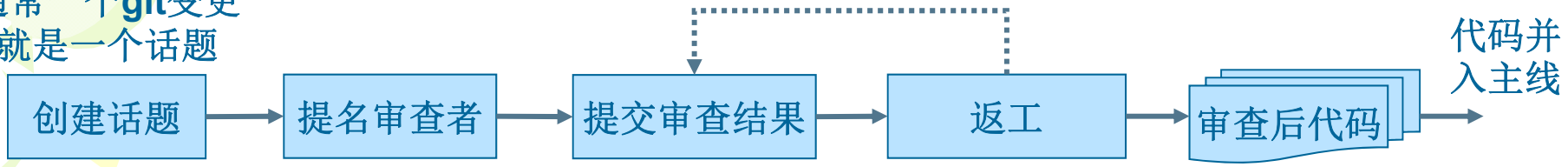
- 有组织的代码走查在编码阶段去除错误的成本是在软件发布阶段去除错误成本的**1/92**

代码审查和代码走查比较

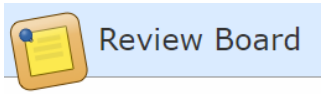
项目	走　查	审　查
准备	通读设计和编码	准备好需求描述文档、程序设计文档、程序的源代码清单、代码编码标准和代码缺陷检查表
形式	非正式会议	正式会议
参加人员	开发人员为主	项目组成员包括测试人员
主要技术方法	无	缺陷检查表
注意事项	限时、不要现场修改代码	
生成文档	会议记录	错误分析报告
目标	代码标准规范，无逻辑错误	

基于Web的协同代码审查

通常一个git变更
就是一个话题



工具：



All | Documentation

Open | Merged | Abandoned

Register | Sign In

status:merged Search

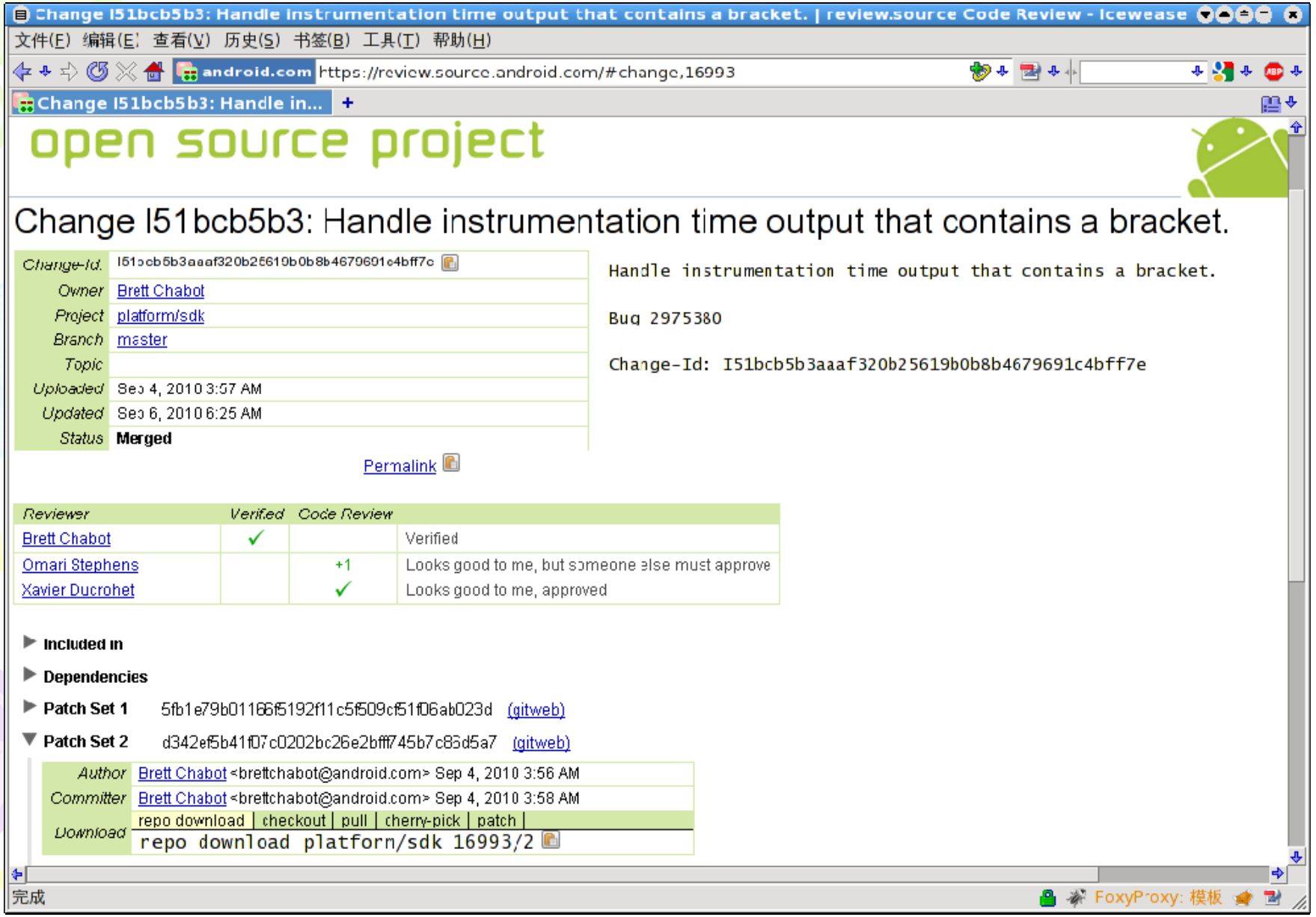
android open source project

Search for status:merged

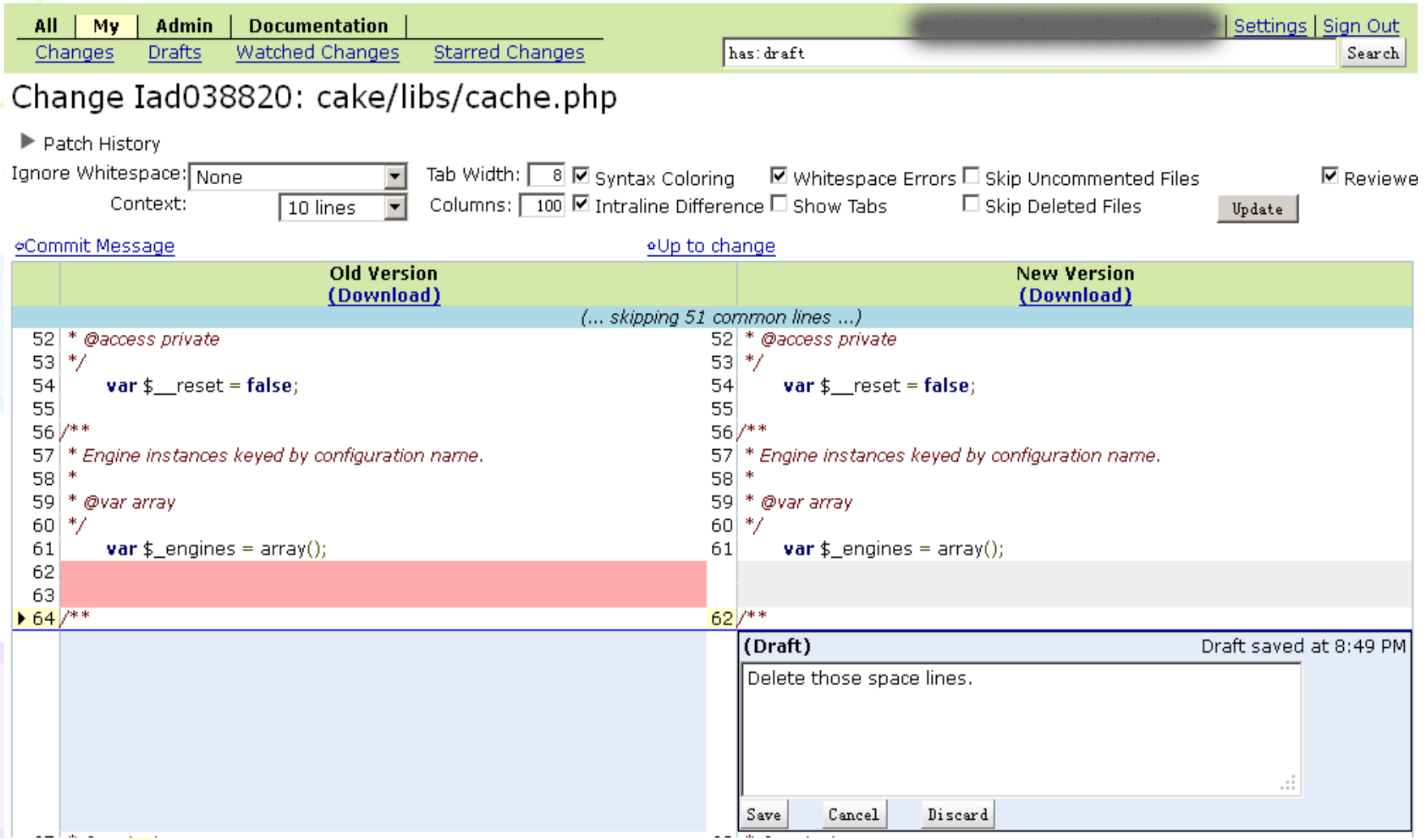
	ID	Subject	Owner	Project	Branch	Updated	V	R
▸	I76d7fcf7	ARM: tegra: usb_phy: Correct utmi power off sequence (MERGED)	Benoit Goby	kernel/tegra	linux-tegra-2.6.36	9:05 AM	✓	✓
	Ibf3ec6a0	Add transient visibility mode for empty containers (MERGED)	Tor Norbye	platform/sdk	master	8:38 AM	✓	✓
	I1ee128e0	Fix ADT to build with the new layoutlib API. (MERGED)	Xavier Ducrohet	platform/sdk	master	8:21 AM	✓	✓
	Ice6324c0	New layoutlib API. (MERGED)	Xavier Ducrohet	platform/sdk	master	8:07 AM	✓	✓
	If976cc25	ARM: tegra: dvfs: Fix dvfs disable config option (MERGED)	Colin Cross	kernel/tegra	linux-tegra-2.6.36	7:54 AM	✓	✓
	I0496cf37	ARM: tegra: dvfs: Add lock to dvfs_reg (MERGED)	Colin Cross	kernel/tegra	linux-tegra-2.6.36	7:43 AM	✓	✓
	I9e3a3cc8	ARM: tegra: dvfs: Fix locking on external dvfs calls (MERGED)	Colin Cross	kernel/tegra	linux-tegra-2.6.36	7:07 AM	✓	✓
	I401ab558	ARM: tegra: dvfs: Add config options to disable dvfs (MERGED)	Colin Cross	kernel/tegra	linux-tegra-2.6.36	7:07 AM	✓	✓
	I643375c3	Fix getDeviceIdentity and getImeiSV for f5521fg modules. (MERGED)	Johan Lindström	platform/vendor/st-ericsson/u300	ericsson-mbm-devices	6:53 AM	✓	✓
	Iff0473dc	[ARM] tegra: cleanup empty functions in mach/fb.h (MERGED)	Erik Gilling	kernel/tegra	linux-tegra-2.6.36	6:05 AM	✓	✓
	Ida529e09	video: add short video mode decode to fbmon	Erik Gilling	kernel/tegra	linux-tegra-2.6.36	6:05 AM	✓	✓

Google gerrit

Google gerrit

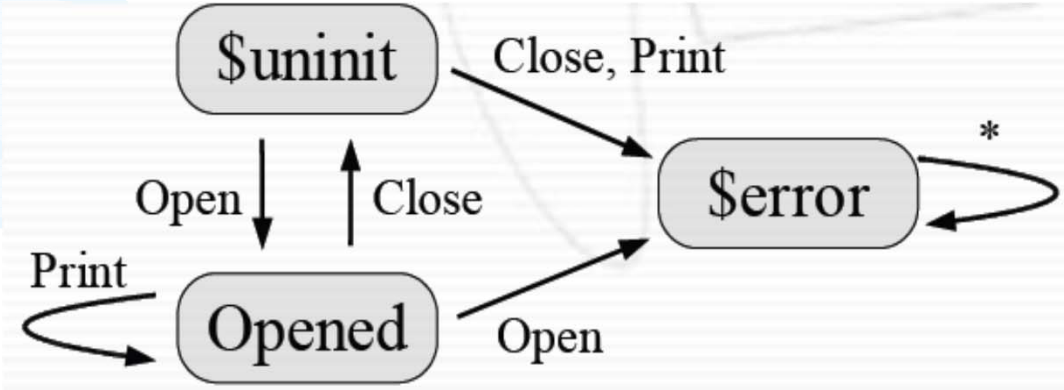


Google gerrit



2.2 代码自动审查

- 通过程序抽象、有限状态机等技术，在不运行程序的情况下，发现其中的错误。



C code pattern	Transition
e=fopen(~)	Open
fclose(e)	Close
fprintf(e,~)	Print

Lint, FindBugs, Coverity, Testbed

FindBugs 2

This page describes the major changes in FindBugs 2. We are well aware that the documentation on the new features in FindBugs 2.0 have not kept up with the implementation. We will be working to improve the documentation, but don't want to hold up the release any longer to improve the documentation.

Anyone currently using FindBugs 1.3.9 should find FindBugs 2.0 to largely be a drop-in replacement that offers better accuracy and performance.

Also check out http://code.google.com/p/findbugs/w/list for more information about some recent features/changes in FindBugs.

The major new features in FindBugs 2 are as follows:

- Bug Rank - bugs are given a rank 1-20, and grouped into the categories scariest (rank 1-4), scary (rank 5-9), troubling (rank 10-14), and of concern (rank 15-20).
  - priority renamed confidence - many people were confused by the priority reported by FindBugs, and considered all HIGH priority issues to be important. To reflect the actually meaning of this attribute of issues, it has been renamed confidence. Issues of different bug patterns should be compared by there rank, not their confidence.
- Cloud storage - having a convent way for developers to share information about when an issue was first seen, and whether it is believed to be a serious problem, is important to successful and cost-effective deployment of static analysis in a large software project.
- update checks - FindBugs will check for releases of new versions of FindBugs. Note: we leverage this capability to count the number of FindBugs users. These update checks can easily be disabled.
- Plugins - FindBugs 2.0 makes it much easier to define plugins that provide various capabilities, and install these plugins either on a per user or per installation basis.
- `fb` command - rather than using the rather haphazard collection of command line scripts developed over the years for running various FindBugs commands, you can now use just one: `fb`.
  - `fb analyze` - invokes the FindBugs analysis
  - `fb gui` - launches the FindBugs GUI
  - `fb list` - lists the issues from a FindBugs analysis file
  - `fb help` - lists the command available.

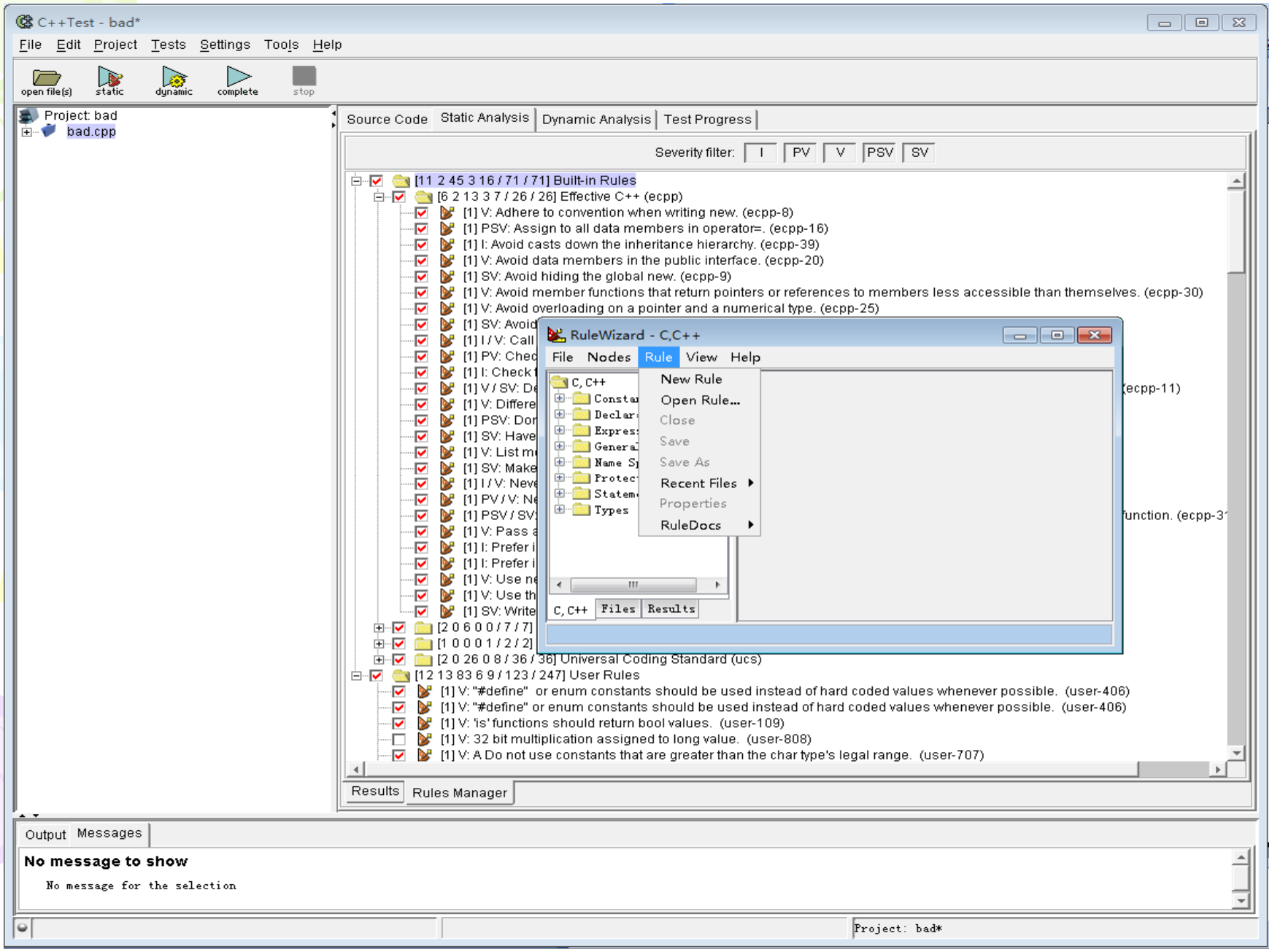
  Plugins can be used to extend the commands that can be invoked via `fb`.

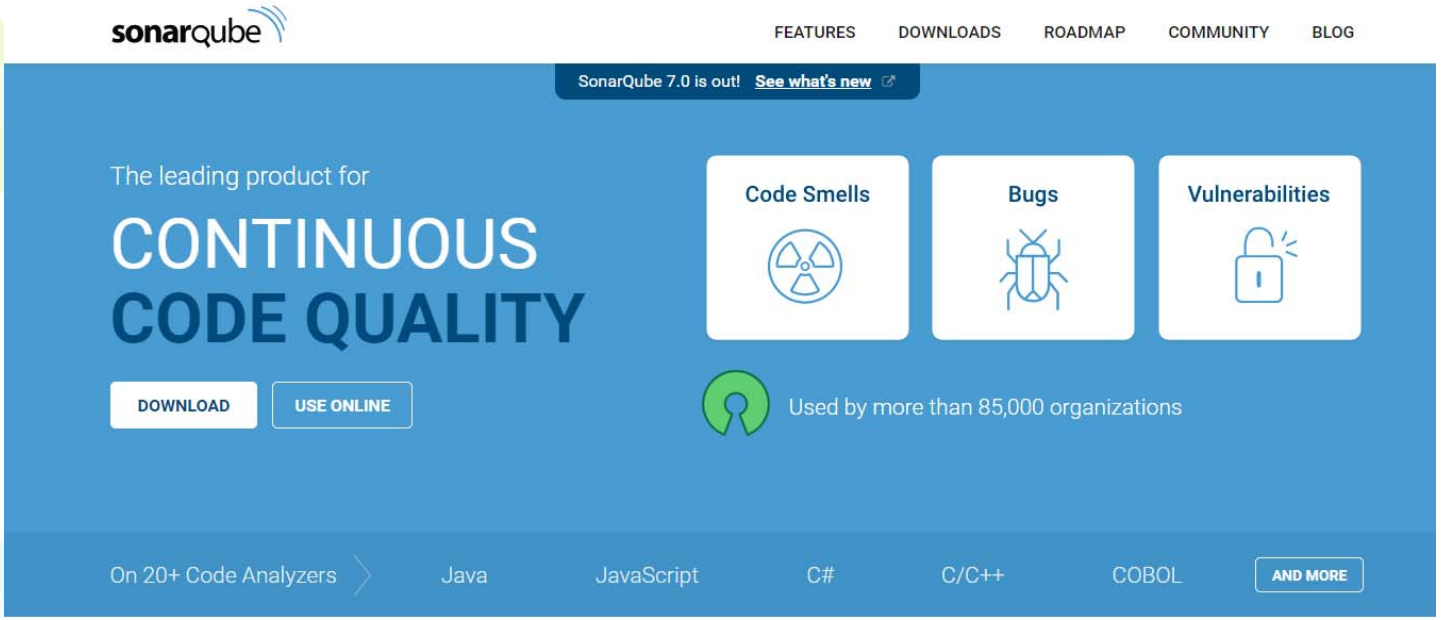
- New bug patterns and detectors, and improved accuracy
- Improved performance: overall, we've seen an average 10% performance improvement over a large range of benchmarks, although a few users have experienced performance regressions we are still trying to understand.
- Guava support - working with Kevin Bourrillion, we have provided additional support for the Guava library, recognizing many common misuse patterns.
- JSR-305 support - improved detection of problems identified by JSR-305 annotations. In particular, we've significantly improved both the accuracy and performance of the analysis of type qualifiers.

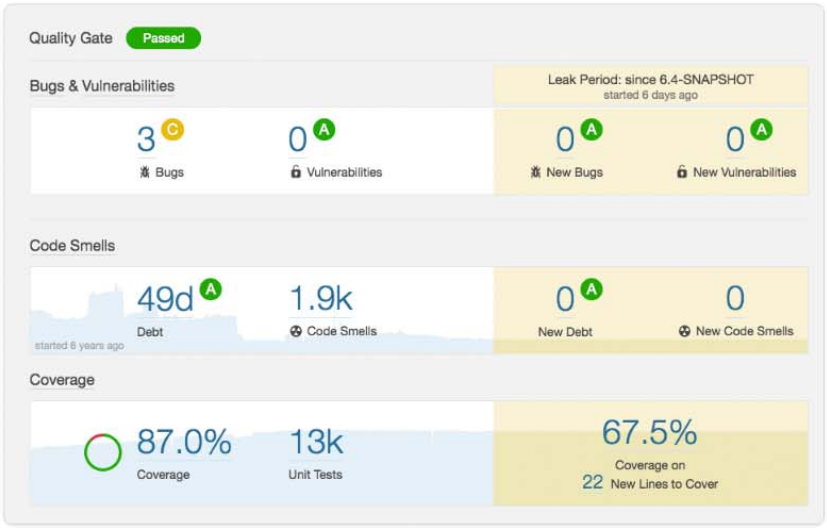
Cloud storage of issue evaluations

For many years, you could store evaluations of FindBugs issues within the XML containing the analysis results. However, this approach did not work well for a team of distributed developers. Instead, we now provide a cloud based mechanism for storing this information. We are providing a free communal cloud (hostied by Google appengine) for storing evaluations of FindBugs issues. You can set up your own private cloud for storing issues, but at the moment this checking out a copy of FindBugs, making some modifications and building the cloud storage plugin from source.

Parasoft C++ Test







Continuous Inspection

SonarQube provides the capability to not only show health of an application but also to highlight issues newly introduced. With a Quality Gate in place, you can fix the leak and therefore improve code quality systematically.

Read more →

在线缺陷扫描工具SonarQube

2.3 风格检查(Style Checking)

- 好的风格对于程序设计具有关键性作用。进行编程风格的检查，可增加：可靠性、可读性/维护性、移植性、……
- 优秀的风格：清楚简单、直截了当、自然的表达式、通行的语言使用方式
- 风格检查
  - 编码风格：针对程序指令、运算符、代码结构、声明等方面制定规则并检查
    - 如为了保证程序模块的结构化，规定不得使用GOTO语句
  - 命名风格：对程序中局部变量、全局变量、类等的命名制定规则并检查，以利于程序的理解、维护
    - 命名应尽量面向待解决的问题，而不是具体实现手段

      坏名字：**list, tree**；好名字：**studentList, familyTree**

https://github.com/SalGnt/cscs

README.md

Coding Style Conventions and Standards

A curated list of Coding Style Conventions and Standards.

Table of Contents

Programming Languages

Arduino

- **Arduino Style Guide**.
- The ArduPirates Coder's Bible.

C

- C Coding Standard.
- Recommended C Style and Coding Standards.
- SEI CERT C Coding Standard.

C#

- C# Coding Standards and Naming Conventions.
- **Microsoft C# Coding Conventions (C# Programming Guide)**.
- Mono Coding Guidelines.
- The Official raywenderlich.com C# Style Guide - C# Style Guide for Unity Tutorials.

C++

- Apache OpenOffice Cpp Coding Standards
- C++ Coding Standard.
- C++ Coding Standard.
- Google C++ Style Guide.
- High Integrity C++ Coding Standard.
- SEI CERT C++ Coding Standard.

Clojure

- The Clojure Style Guide - A community coding style guide for the Clojure programming language.

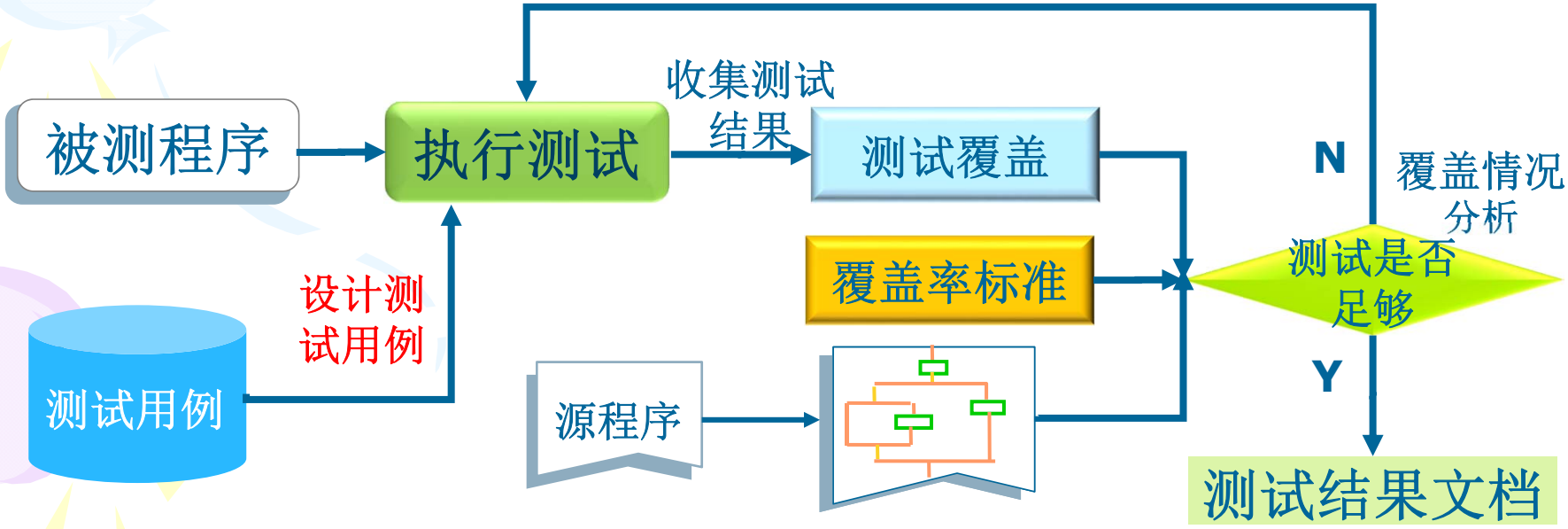
Common Lisp

- Ariel Networks Common Lisp Style Guide.
- Google Common Lisp Style Guide.

动态测试

2.6 覆盖测试分析

- 衡量软件被测试执行的程度
- 在尽可能多地执行程序的路径，进行逻辑覆盖的同时，考察程序执行表现是否异常，尤其是某些复杂的和"正常"情况下不易执行的路径。



2.7 运行时错误检测

- 在程序中注入监控代码，监控程序运行，检测是否有错误发生

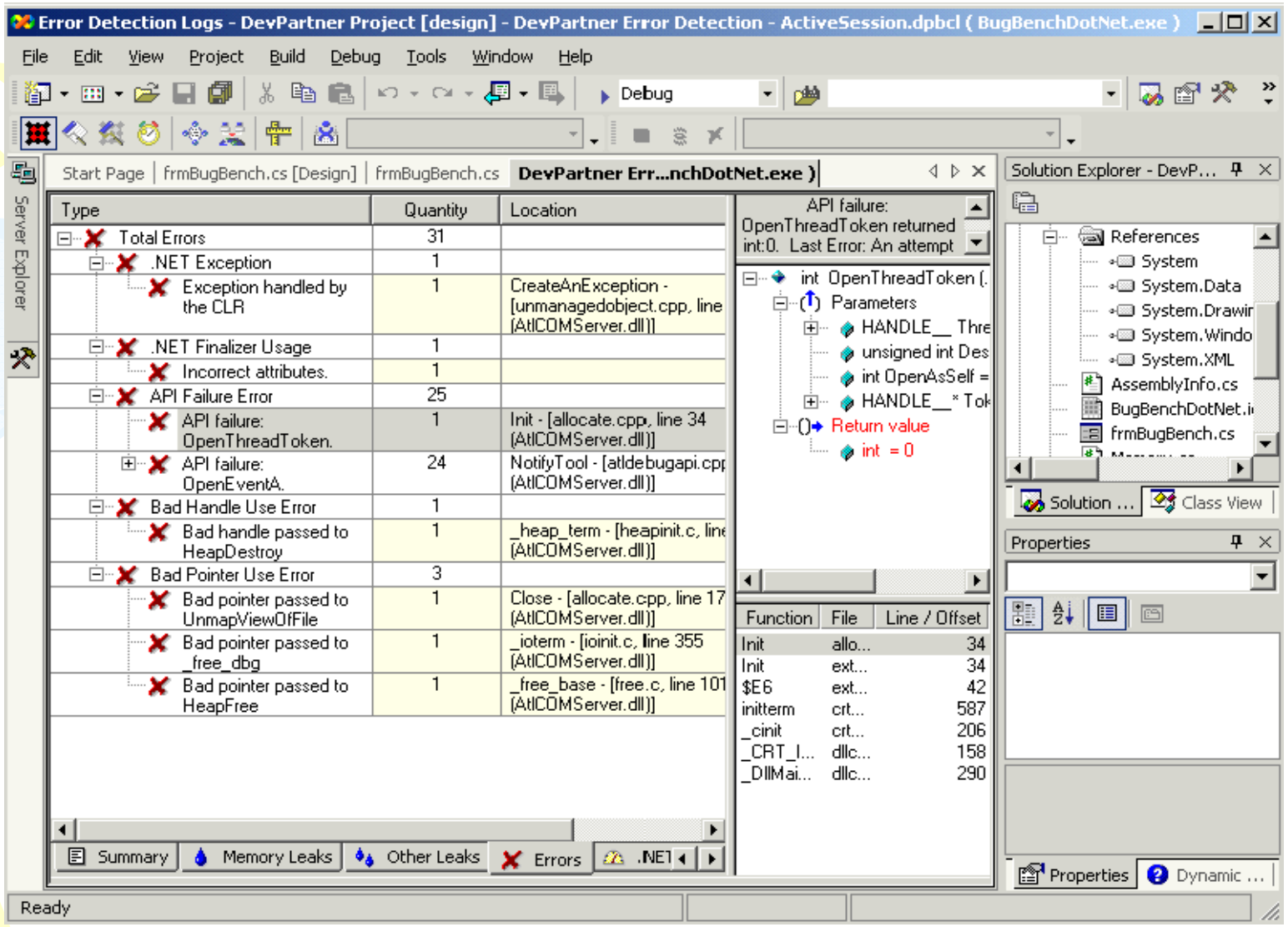
监控代码：check(p)
正常代码：p->next = …

- 工具
  - **IBM Rational Purify**
  - **Compuware DevPartner**
  - **……**

DevPartner Bounds Checker

- C/C++指针、内存错误
- 泄露错误

API与OLE错误
验证ActiveX控件使用正确性



变异测试(Mutation Testing)

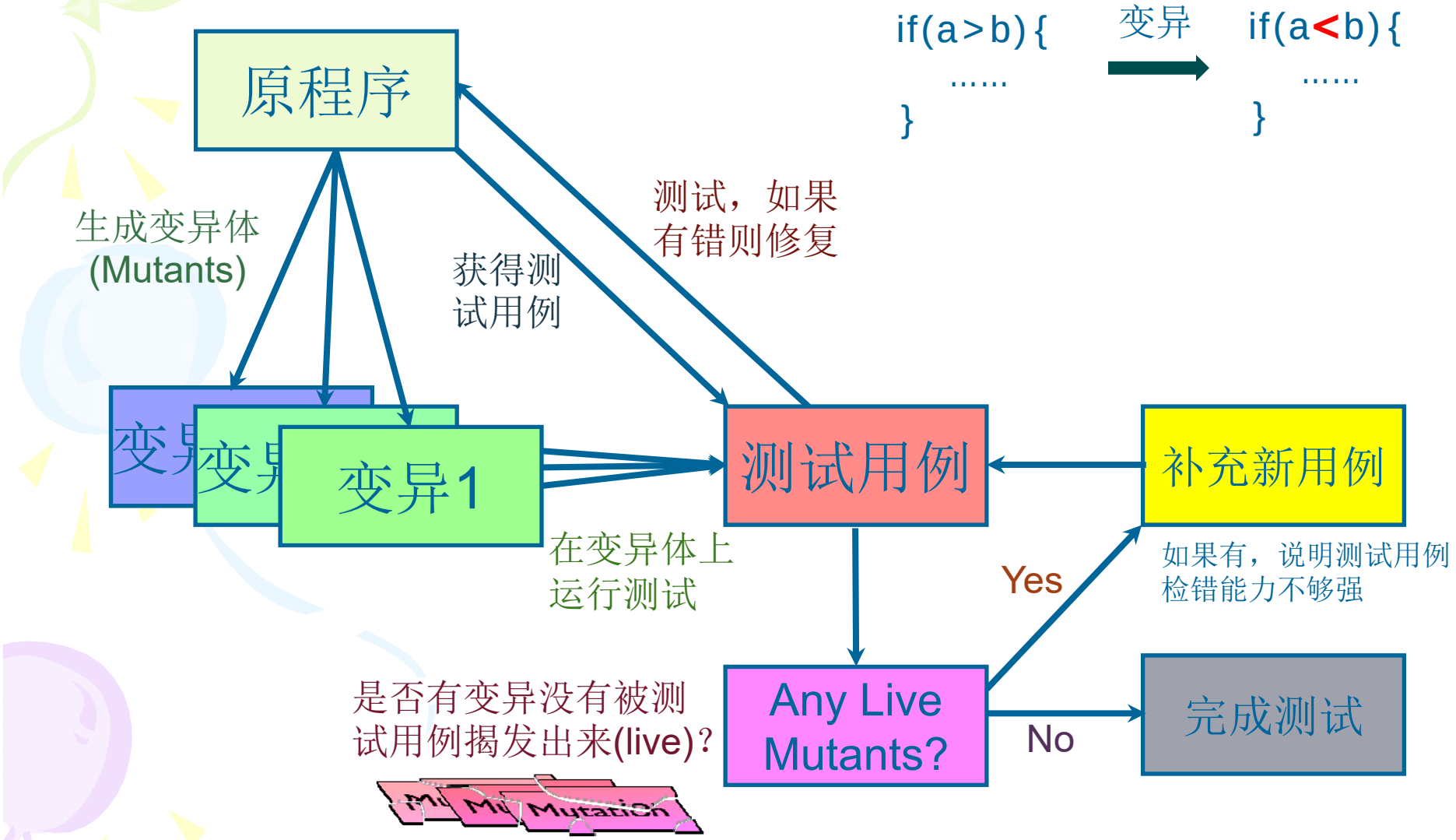
- 一种检查测试集检错是否充分的方法

  在程序中人为植入错误，检查在给定测试集下，人为植入的错误是否能够被发现。
  - 能够发现：测试集发现问题能够好
  - 不能发现：测试集检错能力弱，应补充测试用例。

- 植入错误的版本称为mutants

- 变异算子(Mutation Operators)：植入错误的具体手段
  - 删除语句
  - 修改比较运算符，例如：== 变 >=、< 变<=、a || b变*true*
  - 修改算术运算符：例如：+ 变 -
  - 替换变量

变异测试过程


if(a>b){
.....
}
变异
if(a<b){
.....
}
原程序
生成变异体
(Mutants)
获得测
试用例
测试，如果
有错则修复
变异1
测试用例
补充新用例
在变异体上
运行测试
如果有，说明测试用例
检错能力不够强
Yes
是否有变异没有被测
试用例揭发出来(live)?
Any Live
Mutants?
No
完成测试
Mutation

主要内容

1. 白盒测试简介
2. 常见白盒测试方法
3. **覆盖测试**
4. 实现覆盖的途径

3. 覆盖测试

- 3.1 逻辑覆盖方法
- 3.2 数据流测试
- 3.3 其它覆盖方法

3.1 逻辑覆盖方法

- 以覆盖一定的程序逻辑结构为测试充分性标准，生成测试用例对程序进行测试。

- 逻辑覆盖包括
  1、语句覆盖
  2、判定覆盖
  3、条件覆盖
  4、判定/条件覆盖
  5、条件组合覆盖
  6、修正条件/判定覆盖
  7、路径覆盖

语句覆盖

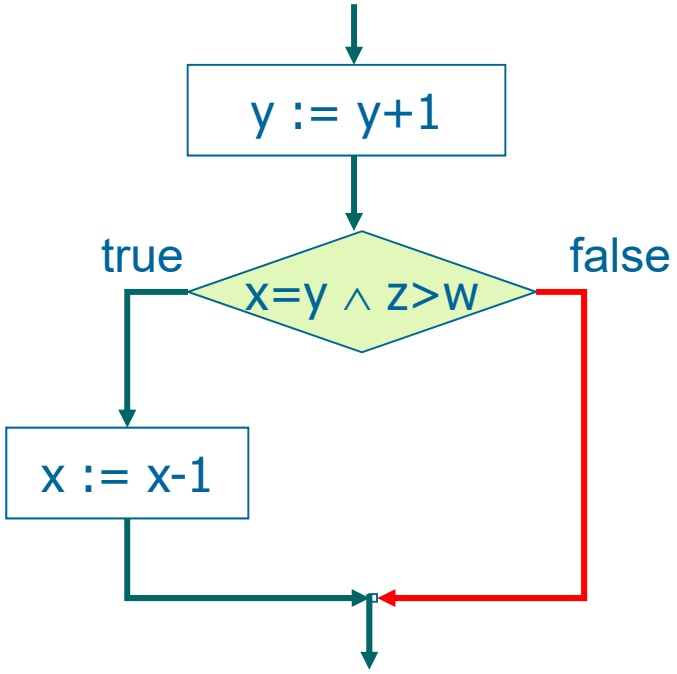
- 每条语句至少被执行一次（一个测试用例）
- 优点
  - 简单直观，无须知道每个判定表达式的取值
- 问题：

  假分支没有得到检查!

输入：{x=2, y=1, z=4, w=3}

判定执行前各个变量的值：

{x=2, y=2, z=4, w=3}



判定（分支/边）覆盖

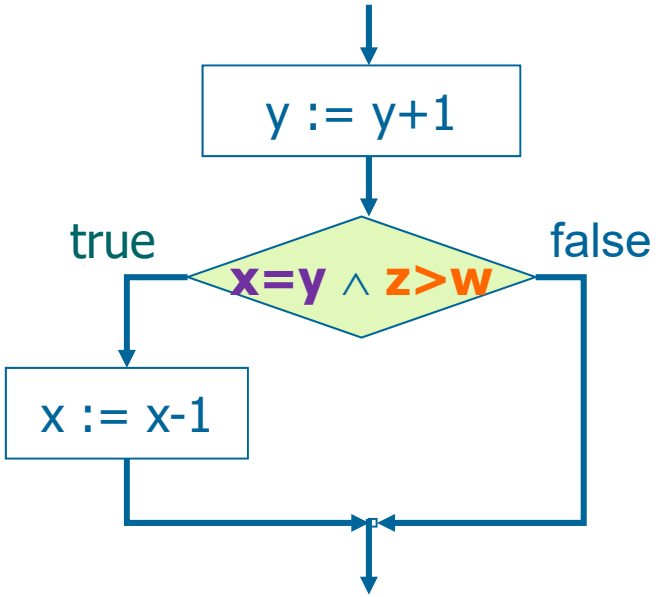
- 使程序中每个判定真假的分支至少执行一次
- 优点
  - 相比语句覆盖，多几乎一倍的测试路径，具有更强的测试能力
  - 简单性，无须知道判定中每个条件的取值
- 问题

没有检查每个独立条件，比如 x≠y 的情况没有得到有效检查

判定执行前各个变量的值：

{x=2, y=2, z=4, w=3} TRUE

{x=3, y=3, z=5, w=7} FALSE

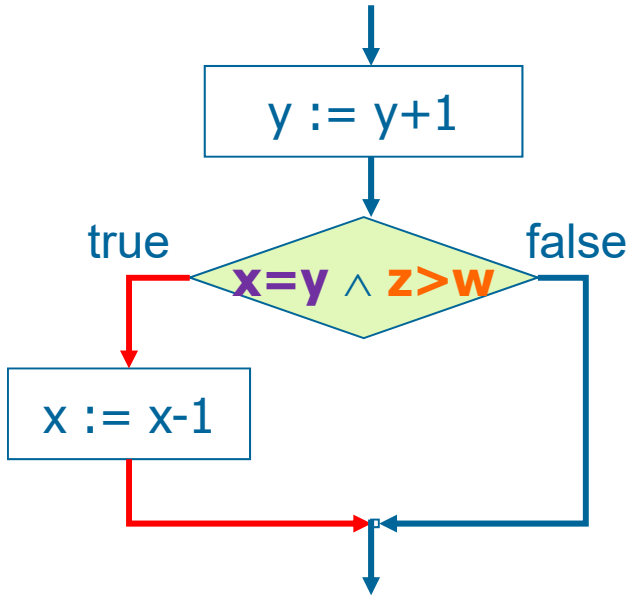


条件覆盖

- 每个条件的真假可能取值均至少被取到一次
- 通常比判定覆盖强，因为它使判定表达式中每个条件都取到了两个不同的结果
- 但也可能有相反的情况：虽然每个条件都取到了不同值，但判定表达式却始终只取一个值

{x=2, y=3, z=4, w=3}
{x=3, y=3, z=5, w=7}
{x=3, y=3, z=7, w=5}

x =y	z >w	
F	T	F
T	F	F
T	T	F



条件覆盖不一定包含判定覆盖
判定覆盖也不一定包含条件覆盖

判定/条件覆盖

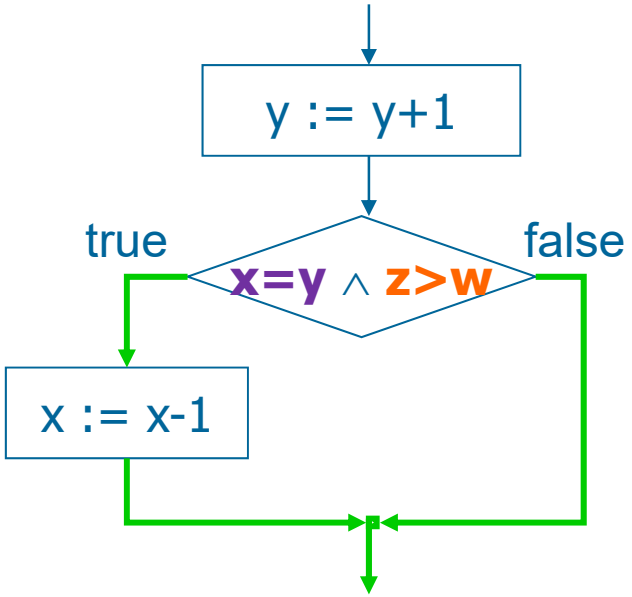
- 在覆盖条件的同时，要求也覆盖判定（分支/边）
- 问题
  - 未考虑条件的组合情况
- 表面上，它测试了所有条件的取值，但事实并非如此，往往某些条件掩盖了另一些条件，会遗漏某些条件取值错误的情况。

T {x=2, y=2, z=4, w=3}

F {x=3, y=2, z=5, w=7}

如果考虑条件短路，还要补充一条

{x=3, y=3, z=5, w=7}



多条件覆盖(条件组合覆盖)

- 设计足够多的测试用例，使得每个判定中条件的各种可能组合都至少出现一次

A ∨ (B ∧ C)

8 = 2^3

指数级爆炸

x=y ∧ z>w

$\{x \mapsto 2, y \mapsto 2, z \mapsto 4, w \mapsto 3\}$

$\{x \mapsto 2, y \mapsto 2, z \mapsto 5, w \mapsto 7\}$

$\{x \mapsto 2, y \mapsto 3, z \mapsto 4, w \mapsto 3\}$

$\{x \mapsto 2, y \mapsto 3, z \mapsto 5, w \mapsto 7\}$

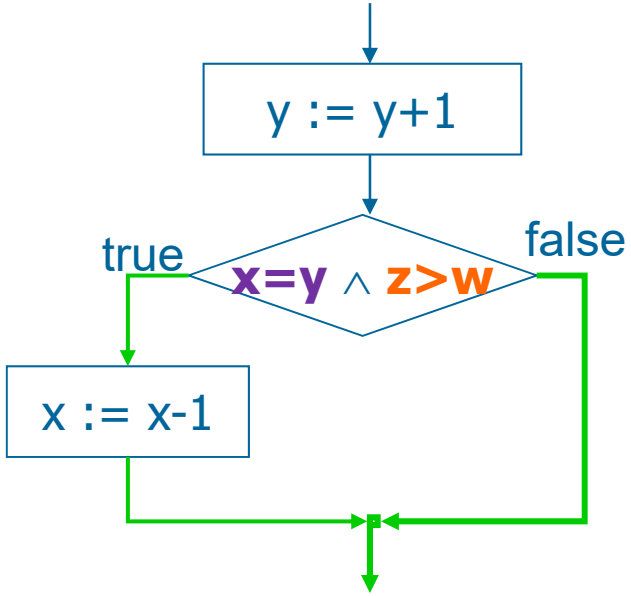
修正条件/判定覆盖(MC/DC)

- 程序的每个入口和出口都必须执行一次
- 程序中每个条件必须至少取到其所有可能值各一次
- 判定中，每个条件应独立影响判定结果至少一次

F $\{x \mapsto 2, y \mapsto 2, z \mapsto 5, w \mapsto 7\}$

T $\{x \mapsto 2, y \mapsto 2, z \mapsto 4, w \mapsto 3\}$

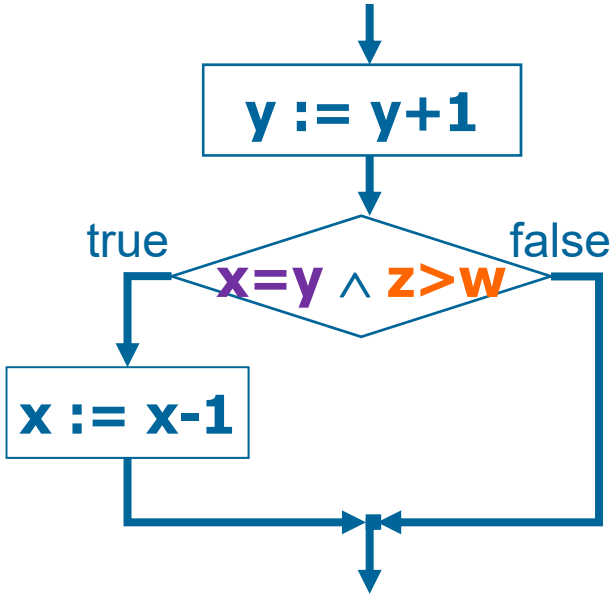
F $\{x \mapsto 3, y \mapsto 2, z \mapsto 4, w \mapsto 3\}$



路径覆盖

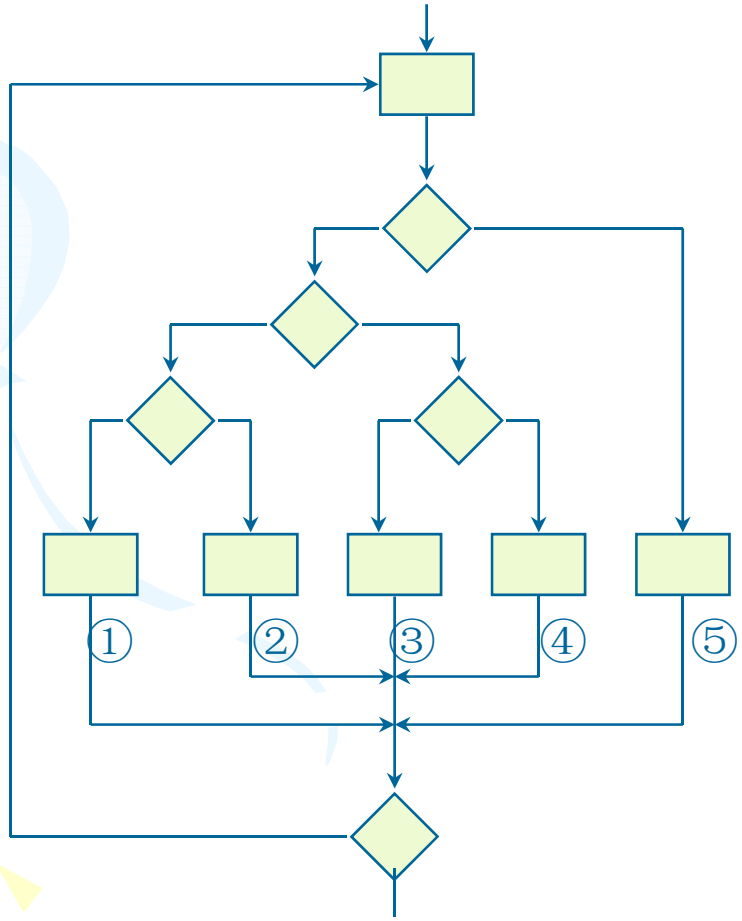
- 设计足够多的测试用例，覆盖程序中的每条路径.
- 实际程序中往往有无限多路径——不可能全部覆盖
- 可以考虑路径片段的覆盖
- 即使一条路径也可能对应不同行为（考虑有数组）

$$\mathbf{F}\ \{x \mapsto 2, y \mapsto 2, z \mapsto 5, w \mapsto 7\}$$
$$\mathbf{T}\ \{x \mapsto 2, y \mapsto 2, z \mapsto 4, w \mapsto 3\}$$



路径覆盖

- 如果循环执行**20**次，将对应5^{20} 条不同路径**!**



假使对每一条路径进行测试需要**1**毫秒，一天工作**24**小时，一年工作**365** 天， 那么要想把如图所示的小程序的所有路径测试完，则需要**3170**年。

覆盖准则总结

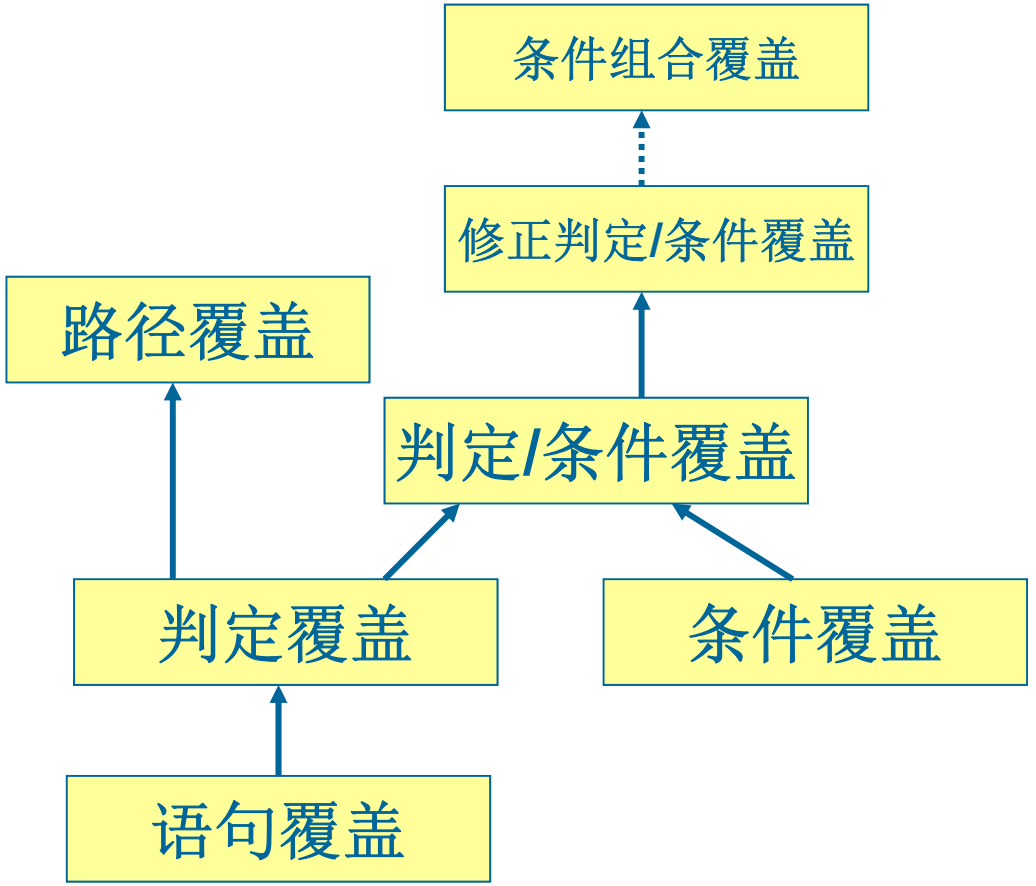
覆盖准则总结		
1	语句覆盖	保证程序中的每条语句都执行一遍
2	判定覆盖	保证每个判断取True和False至少一次
3	条件覆盖	保证每个判断中的每个条件的取值至少满足一次
4	判定条件覆盖	保证每个条件和由条件组成的判断的取值…
5	条件组合覆盖	保证每个条件的取值组合至少出现一次…
6	修正条件/判定覆盖	保证每个条件取到其所有可能值各一次 保证每个条件独立影响判定结果至少一次
7	路径覆盖	覆盖程序中所有可能路径

无论哪种覆盖方法，都无法绝对保证程序的正确性

各种覆盖方法之间的关系

- 路径覆盖未必条件组合覆盖
- 条件组合覆盖未必路径覆盖
  e.g., 考虑循环
- 高覆盖未必找到更多错误，只是找到的可能性更大

在实际的测试用例设计过程中，可以根据需要将不同的覆盖方法组合起来使用，以实现最佳的测试用例设计 。



覆盖率使用的基本原则

1. 覆盖率不是目的，只是一种手段

测试目标是发现错误，应先根据需求挖掘测试用例，再使用覆盖准则来丰富和完善测试用例；
仅以覆盖率来指导测试，会遗漏很多有用的用例，陷入追求覆盖率的数字游戏

2. 不可能针对所有的覆盖率去测试

会造成测试成本十分高昂，难以实现；现代软件需要对市场快速响应，要求尽量使用精简的，能够让产品质量迅速稳定下来的测试；

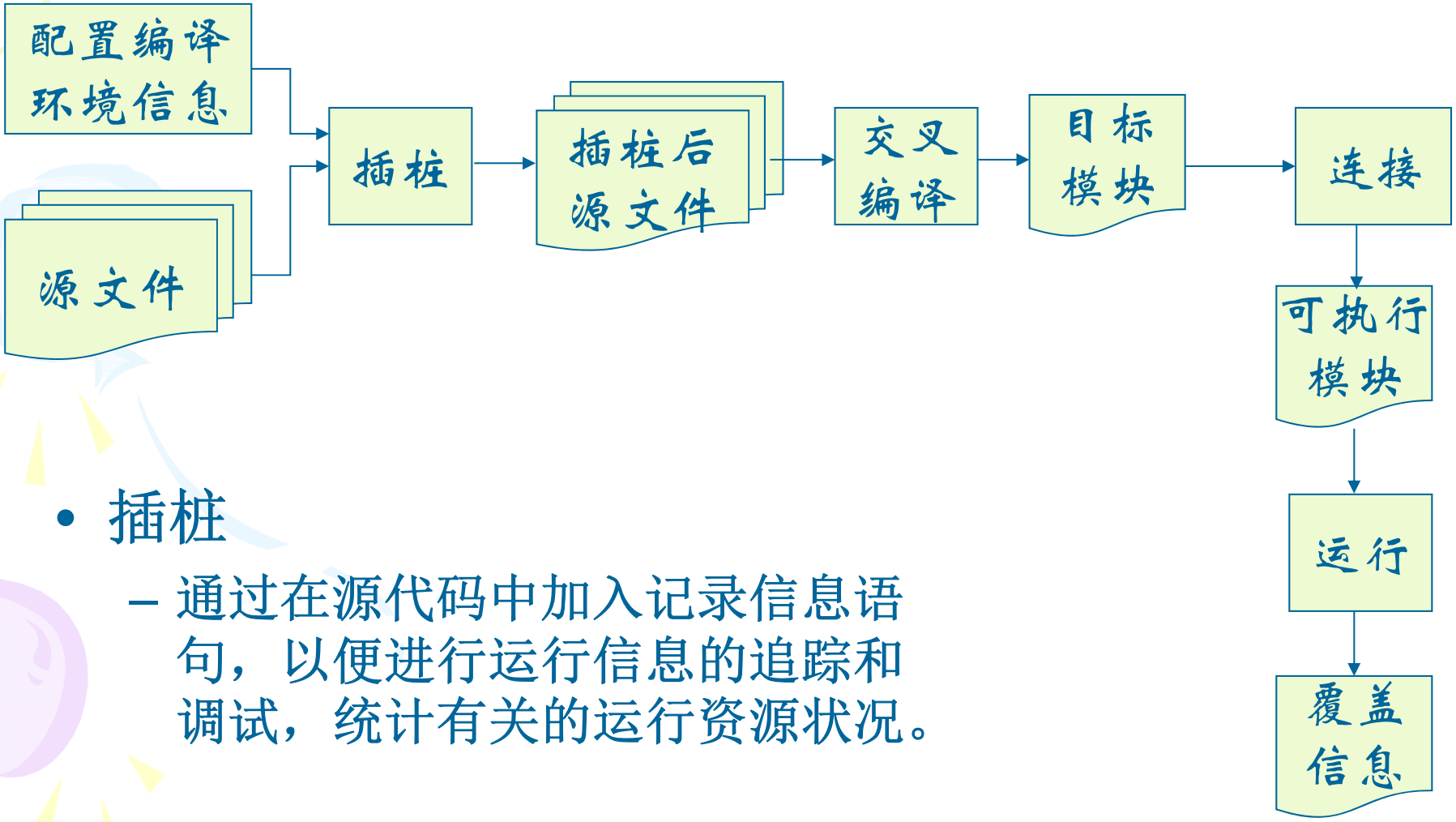
3. 只考虑一种覆盖标准不恰当

每种覆盖标准都有其针对的重点，考虑一种会遗漏重要的测试内容；
综合使用能发现更多错误

4. 不要追求绝对100%的覆盖率

达到100%覆盖率成本巨大，实践中即使语句覆盖也很难达到100%
对一般软件应设置合理的覆盖标准；覆盖要均匀，也要考虑重点

覆盖信息的收集



- 插桩
  - 通过在源代码中加入记录信息语句，以便进行运行信息的追踪和调试，统计有关的运行资源状况。

程序插桩

- 插桩的实现

  程序→ **AST→ New AST→**程序

- 插桩的主要问题
  - **1**：探测什么信息？

    运行时间

    函数调用次数

    语句执行次数
  - **2**：插桩位置？
  - **3**：注入什么代码？

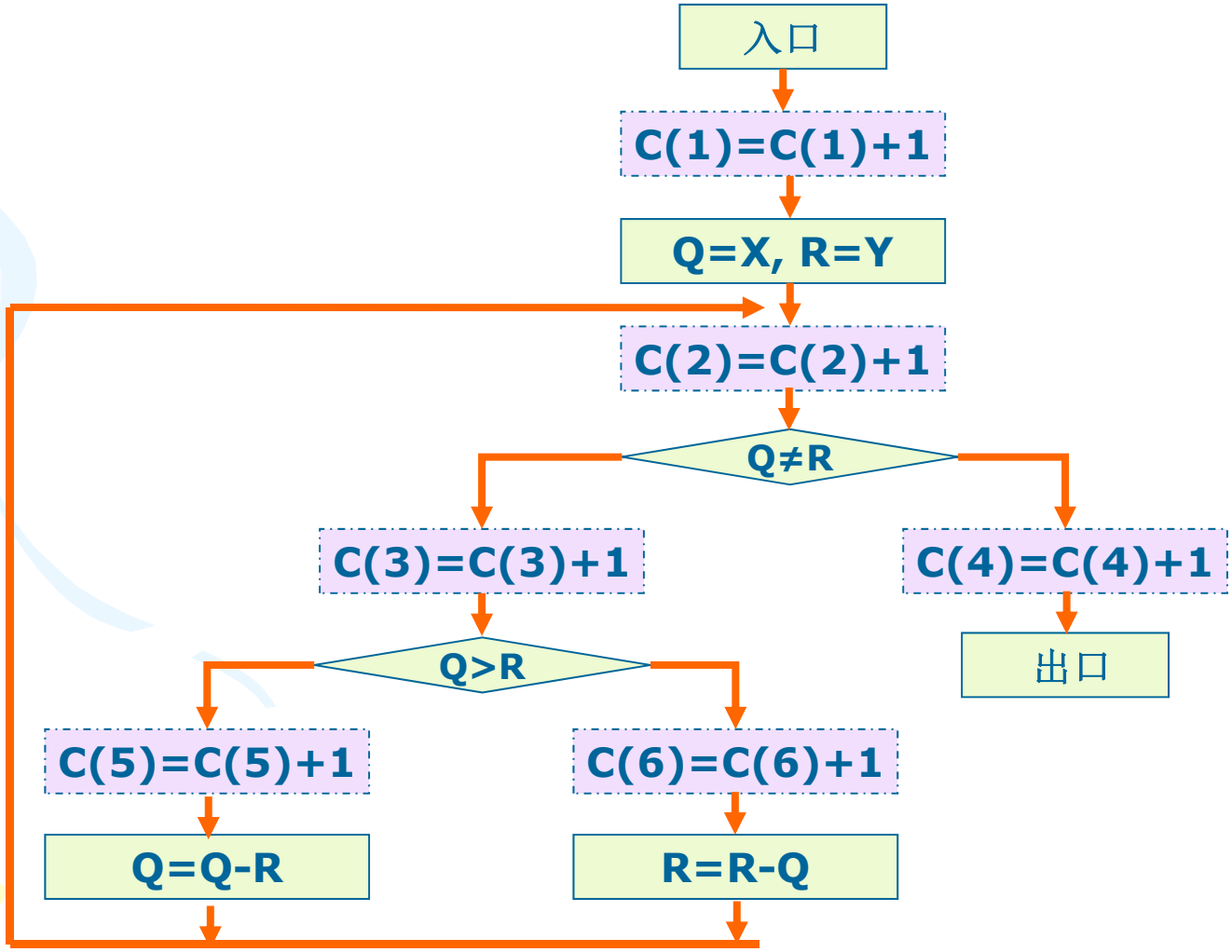
插桩**return**语句示例：

```
for ( j=0;j<10000;j++)
{
    if ( j == k)
        return ;
}
```

```
for ( j=0;j<10000;j++)
{
    if ( j == k){
        log("return");
        return ;
    }
}
```

例：分析覆盖率和测试用例有效性

求整数**X**和**Y**的最大公约数程序的插桩



CodeCover

- 图形化展示测试用例的代码覆盖情况
- 统计测试覆盖率，帮助确定未覆盖的代码

CodeCoverTest.java

```
import java.util.Scanner;

public class CodeCoverTest {
    public static void main(String[] args) {
        int x, y, z;

        Scanner s = new Scanner(System.in);
        System.out.println("输入x, y, z: ");
        x = s.nextInt();
        // System.out.println("x=" + x);
        y = s.nextInt();
        z = s.nextInt();
        DoWork(x, y, z);

    }

    static void DoWork(int x, int y, int z) {
        int k = 0, j = 0;
        if (x > 3 && z < 10) {
            k = x * y - 1;
            j = k + 1;
        }
        if (x == 4 || y > 5)
            j = x * y + 10;
        j = j % 3;
        System.out.println("k=" + k);
        System.out.println("j=" + j);

    }
}
```

Test Sessions | Coverage | Boolean Analyzer | Pick Test Cases | Correlation

Test Session Container: CodeCover 2016-12-4 23:33:47

Name

- 2016-12-5 0:21:57
  - UNNAMED TESTCASE
- 2016-12-5 0:23:02
  - UNNAMED TESTCASE

Test Sessions | Coverage | Boolean Analyzer | Pick Test Cases | Correlation | Problems | Console

Show methods with Statement Coverage <= 90.5 %

Name	Statement	Branch	Loop	Term	?-Operator	Synchronized
CodeCover	84.6 %	75.0 %	–	62.5 %	–	–
CodeCoverTest	84.6 %	75.0 %	–	62.5 %	–	–
DoWork	71.4 %	75.0 %	–	62.5 %	–	–
main	100.0 %	–	–	–	–	–

CodeCoverTest.java

```
static void DoWork(int x, int y, int z) {
    int k = 0, j = 0;
    if (x > 3 && z < 10) {
        k = x * y - 1;
        j = k + 1;
    }
```

Test Sessions | Coverage | Boolean Analyzer | Pick Test Cases | Cor

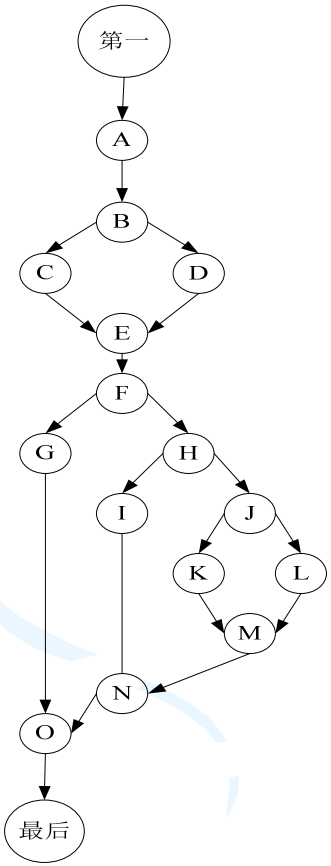
Class: CodeCoverTest Condition: (x > 3 AND z < 10)

(	x > 3	AND	z < 10	)	Result	Test Cases (Number of Executions)
F	F		x		0	UNNAMED TESTCASE (1) Coverage: 25.0
T	F		F		0	UNNAMED TESTCASE (1) Coverage: 25.0
T	T		T		1	UNNAMED TESTCASE (1) Coverage: 50.0

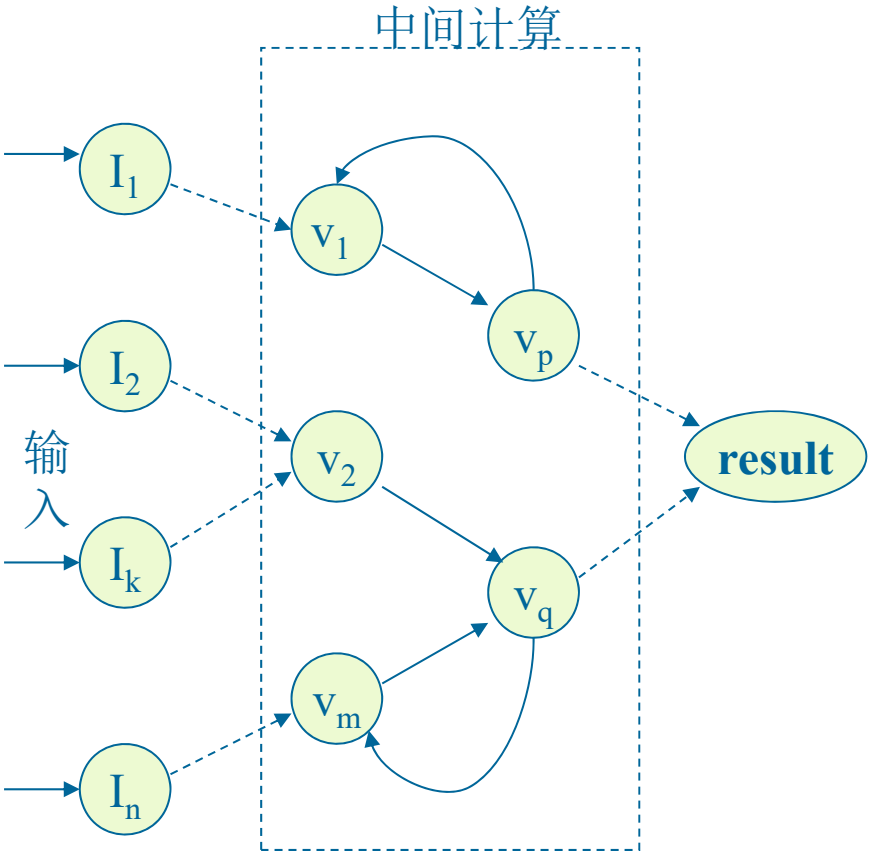
可以帮助分析条件的覆盖情况

3.2 数据流测试

- 关于程序的不同角度理解



程序的核心是语句间的控制转移



程序的核心是变量间的值传递

3.3 数据流测试

- 前面的测试针对程序的控制流路径，检查程序在各种条件值、各种决策下是否有错。
- 数据流测试主要测试程序中的数值流（覆盖值传递路径），检测变量定义与使用的情况。
- 它比较容易发现下列类型的错误
  - 变量被定义，但是从来没有使用。
  - 所使用的变量没有被定义。
  - 变量在使用之前被定义两次。
  - 其它定义不当或使用不当的情况

数据流测试—基本概念

- 节点集

def(x): 定义变量 x 的节点的集合

def(x) = {1}

def(y) = {2, 3, 7, 8}

use(x): 使用变量 x 的节点的集合

use(x) = {5, 6, 7, 8}

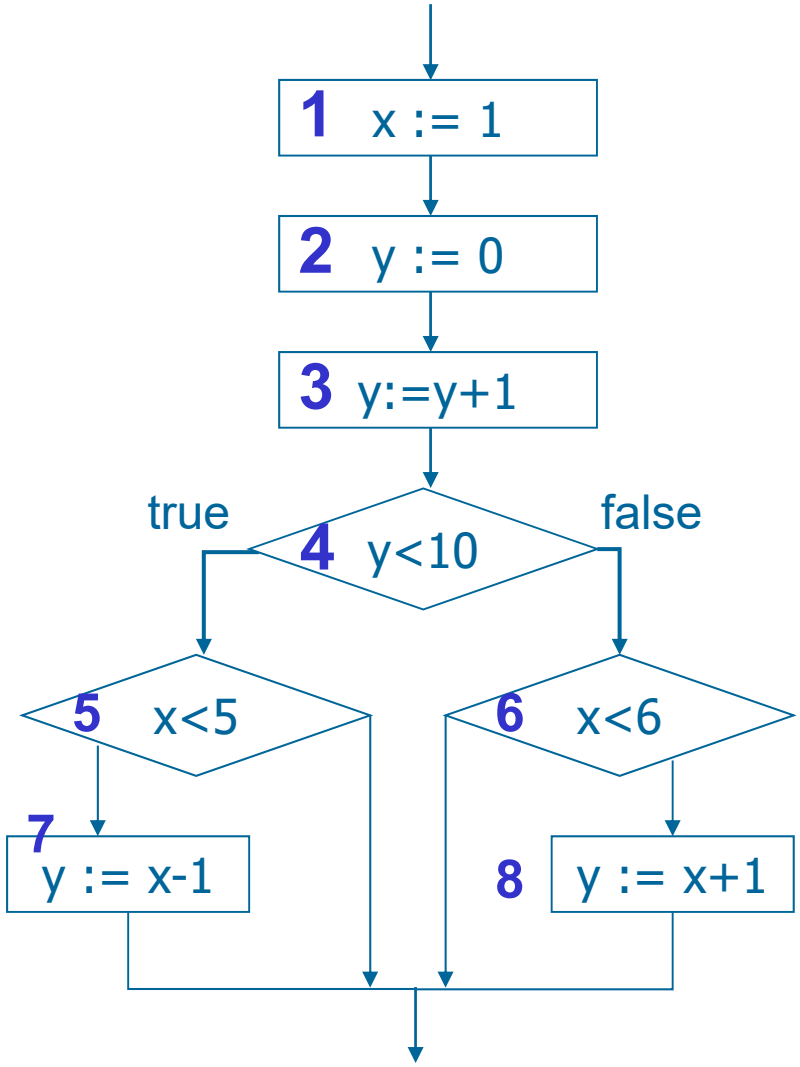
use(y) = {3, 4}

du(s, x): 节点集，其中的每个节点 s’满足s’∈use(x)，且从 s 到 s’ 有一条路径，其上变量x没有被重新定值 。

du(1, x) = {5, 6, 7, 8}

du(2, y) = {3}

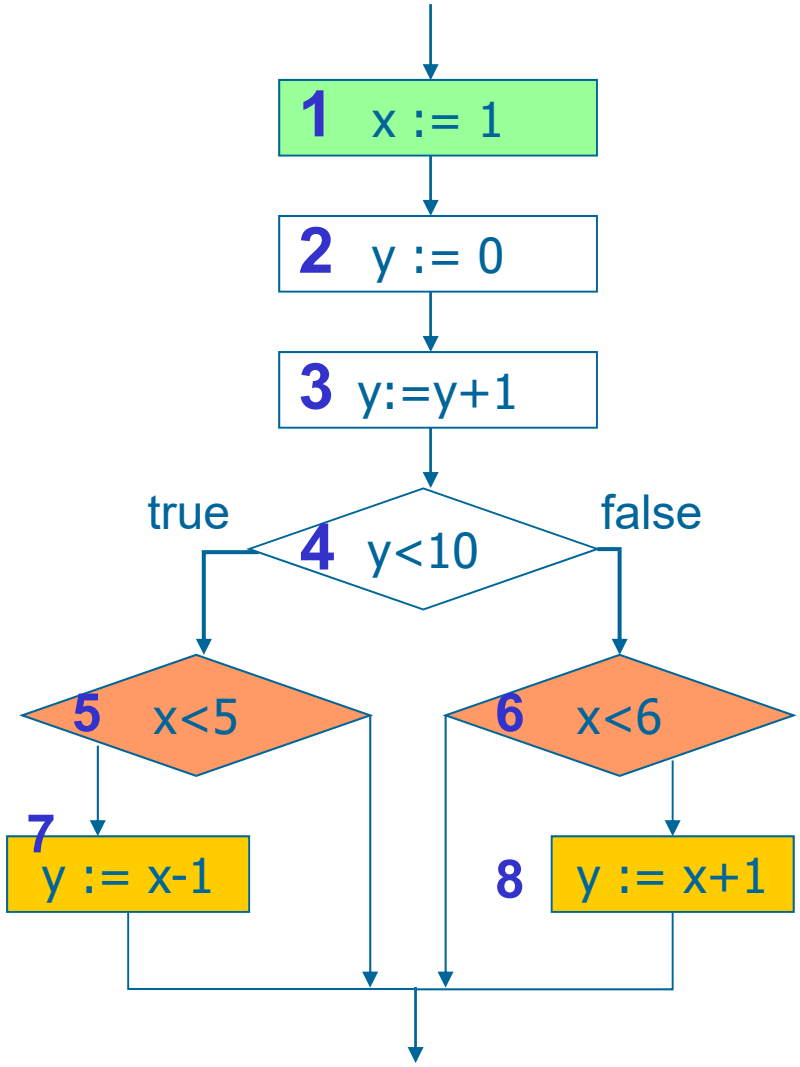
du(3, y) = {4}



数据流测试—覆盖准则

- 全定义(all-defs)
  - 对任一变量 x，和它的任一定义点 s∈def(x)，测试执行至少包含到 du(s, x) 中某个节点的一条路径

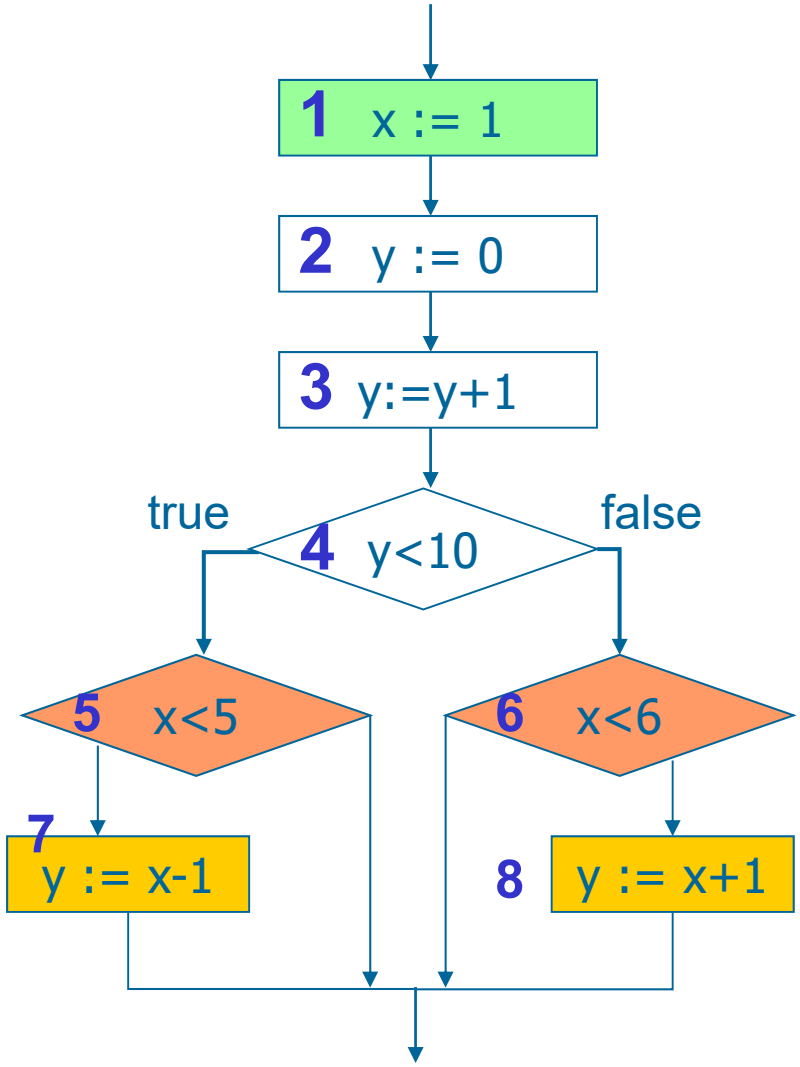
所有定义至少使用一次



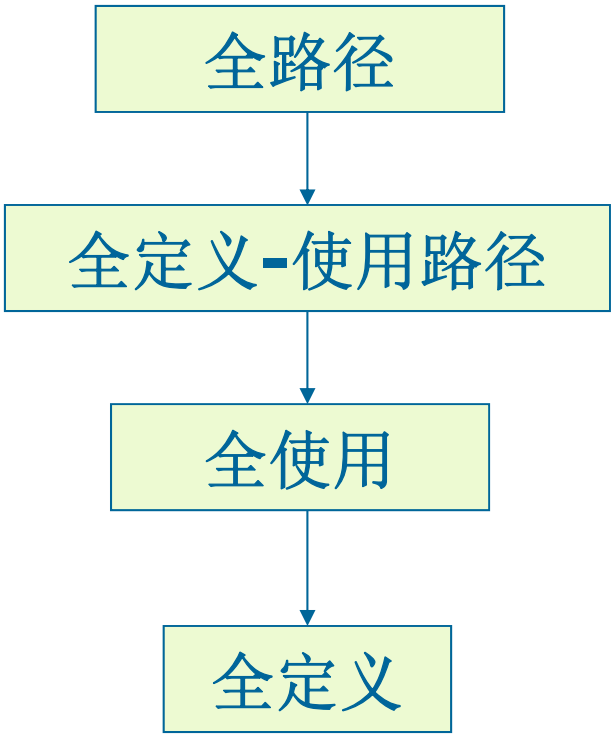
数据流测试—覆盖准则

- 全使用 **(all-uses)**
  - 对任一变量x，和它的任一定义点 s ∈ def(x)，测试执行至少包含到 du(s,x) 中每个节点的一条路径。

- 全定义-使用路径 **(all-du-paths)**
  - 对任一变量x，和它的任一定义点s ∈ def(x)，测试执行包含到 du(s,x)中每个节点的所有路径。



数据流测试—包含关系



数据流分析—优势与缺点

- 因为程序内的语句因变量的定义和使用而彼此相关，所以用数据流测试方法更能有效地发现软件缺陷。

- 但是，在度量测试覆盖率和选择测试路径的时候，数据流测试很困难。
  - 很难知道全覆盖到底是什么（需要数据流分析）
  - 不容易设计测试用例

3.3 其它覆盖准则

逻辑覆盖、数据流覆盖、基路径覆盖等仍可能不充分，为此人们提出了其它一些覆盖准则

- 数据域覆盖(Data Domain Coverage)：
  覆盖数据取值的每个数据域

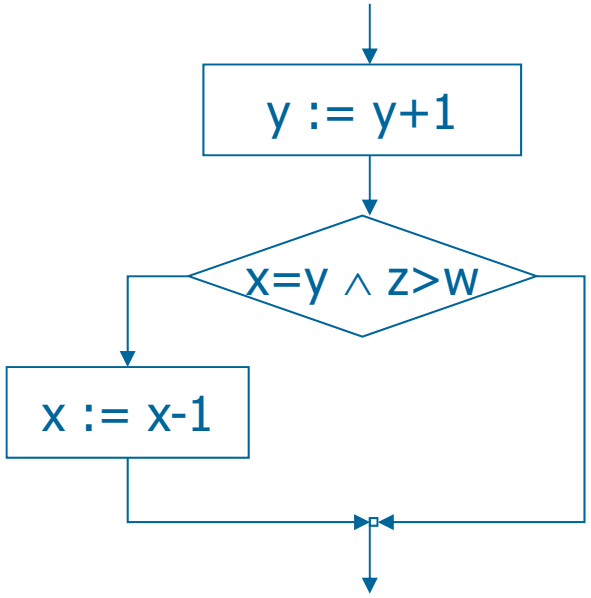
- 风险覆盖(Risk Coverage)：
  覆盖可能导致软件失效的风险点

- 状态图覆盖
  覆盖状态节点
  覆盖状态迁移
  ……

主要内容

1. 白盒测试简介
2. 常见白盒测试方法
3. 覆盖测试
4. 实现覆盖的途径

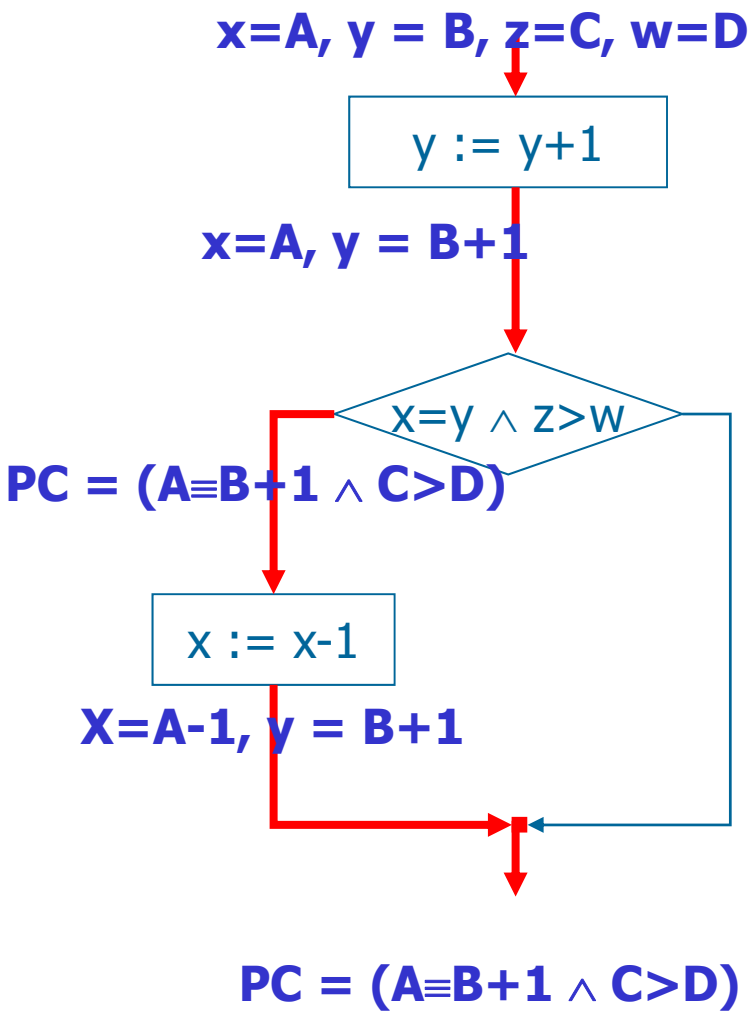
4. 实现覆盖的途径



- 希望能设定一个测试，使之按照给定的路径来执行，从而实现期望的覆盖。
- 但是，如何为这样一个测试给定输入？
  - 随机生成测试输入，大量尝试
  - 基于路径条件分析的方法

推导路径条件(path condition)

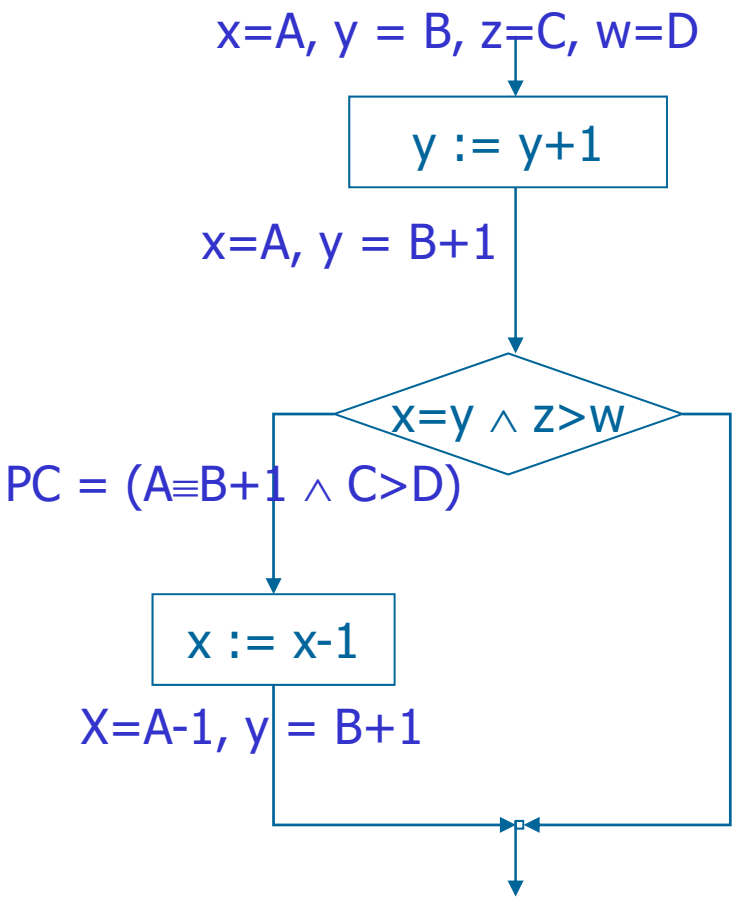
- 用符号表示输入变量的取值
  - A表示x，B表示y
- 顺着路径自上而下进行分析，对于赋值，用输入上的符号表达式表示当前变量的计算结果
- 对于判定分支，带入各变量的符号表达式，用符号关系表示所走分支应该满足的约束，将该约束加入到该路径的路径条件中



求解路径约束，获得测试用例

- 所走路径应满足
  (A≡B+1 ∧ C>D)
- 用约束求解工具可获得满足条件的一组值

  x=2, y=1,
  z=4, w=3



PC = (A≡B+1 ∧ C>D)

课堂练习

```
    void DoWork(int x, int y, int z) {
1           int k = 0, j = 0;
2           if ((x > 2) && (z < 8))
            {
3                   k = x * y - 1;
4                   j = sqrt(k);
            }
5           if ((x == 5) || (y > 10))
6                   j = x * y + 10;
7           j = j % 3;
        }
```

分别以语句覆盖、判定覆盖、条件覆盖、判定/条件覆盖、MC/DC覆盖方法设计测试用例，给出分析过程