

# Verificación y Validación de Software

Tecnología de Programación  
Martín Larrea

# Verificación y Validación de Software

Martín L. Larrea, profesor de la materia  
“Verificación y Validación de Software” para la  
Ingeniería en Sistemas de Información

“Testing de Software” materia optativa para la  
Licenciatura en Ciencias de la Computación e  
Ingeniería en Computación

# Verificación y Validación de Software

← → ↻  ☆ 🔔 📧




UNIVERSIDAD NACIONAL DEL SUR  
DEPARTAMENTO DE CIENCIAS E INGENIERÍA DE LA COMPUTACIÓN  
Avenida Alem 1253 - Bahía Blanca - Buenos Aires - Argentina



▼ Institucional
Docentes/Investigadores
▼ Carreras de Grado
▼ Carreras de Posgrado
Investigación
⚙️ Servicios a Terceros
f Noticias DCIC-UNS

**- CÓD. 7993**

Profesor: Martín L. Larrea (mll@cs.uns.edu.ar)

Asistente: (-)

Ayudantes:

**HORARIOS**

Martes de 9-12 hs. Aula 1 DCIC

Viernes de 9-12 hs. Aula 2 DCIC

**ARCHIVOS RECIENTES**

- ▶ VyVS - Clase 05 - Ca(...).zip (05/sep) ⚡
- ▶ VyVS - Clase 03 - In(...).zip (04/sep) ⚡
- ▶ ProyectoTDPParaPrueb(...).rar (04/sep) ⚡
- ▶ VyVS - Clase 04 - In(...).pdf (01/sep) ⚡

Inicio

Programa

Cursado

Fechas Importantes

Downloads

Calendario

---

**LINKS DE INTERÉS**

UN del Sur

DCIC Forums

DCIC Mail

CECom - Estudiantes

📅	04.09.17	Por excepción, la clase del martes 5 de septiembre será a las 8:30
📅	02.08.17	Canal de YouTube de VyVS con las clases <a href="https://www.youtube.com/channel/UCna4jBhoV1z1rwslKNwBfIQ">https://www.youtube.com/channel/UCna4jBhoV1z1rwslKNwBfIQ</a>
		Cuenta de Twitter de VyVS con las novedades <a href="https://twitter.com/unsdcicvyvs">https://twitter.com/unsdcicvyvs</a>
📅	10.09.15	Libro "Voto Electrónico. Los riesgos de una ilusión" <a href="http://www.vialibre.org.ar/wp-content/uploads/2009/03/evoto.pdf">http://www.vialibre.org.ar/wp-content/uploads/2009/03/evoto.pdf</a>
📅	08.09.15	Más sobre voto electrónico <a href="http://gizmodo.com/this-is-why-electronic-voting-is-a-bad-idea-1673093724">http://gizmodo.com/this-is-why-electronic-voting-is-a-bad-idea-1673093724</a>
📅	08.09.15	

**CALENDARIO**

**SEPTIEMBRE, 2017**

Do	Lu	Ma	Mi	Ju	Vi	Sa
					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30

**OCTUBRE, 2017**

Do	Lu	Ma	Mi	Ju	Vi	Sa
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31				



# Verificación y Validación de Software

Twitter, Inc. [US] | https://twitter.com/unsdcicvyvs

Inicio Momentos Notificaciones Mensajes

Buscar en Twitter

Twitter

`...que(array_free_
 ); if (c < 2 * b - 1) { return
 e), this.trigger("click"); } for
 != a[b] && " " != a[b] ||
 _logged").val().`

Tweets 21 Siguiendo 4 Seguidores 9 Me gusta 1

**VyVS**  
@unsdcicvyvs

Bienvenidos a Verificación y Validación de Software (cod. 7993) materia del cuarto año de la Ing. en Sistemas de Información (DCIC/UNS).

Bahía Blanca, Argentina

cs.uns.edu.ar/~mll/vyvs

Tweets Tweets y respuestas Multimedia

Tweet fijado

**VyVS** @unsdcicvyvs · 15 dic, 2016  
Canal de YouTube oficial de la materia

**Verificación y Validación de Software**  
Bienvenidos a Verificación y Validación de Software (cod. 7993) materia del cuarto año de la Ingeniería en Sistemas de Información (DCIC, UNS). En este canal...  
youtube.com

**A quién seguir** · Actualizar · Ver todos

**Florencia** @fmarrocchi

**Tioco Arrillaga** @TiocoA

**Leandro** @leanmarziali



# Verificación y Validación de Software

The screenshot shows a web browser displaying a YouTube channel page. The address bar shows the URL: <https://www.youtube.com/channel/UCna4jBhoV1z1rwsIKNwBfIQ/featured>. The page features a banner image with blurred code snippets, including `this.trigger("click");` and `!= a[b] && " " != a[b]`. Below the banner, the channel name "Verificación y Validación de Software" is displayed with 16 subscribers and a "SUSCRITO 16" button. The navigation menu includes "INICIO", "VÍDEOS", "LISTAS DE REPRODUCCIÓN", "CANALES", and "MÁS INFORMACIÓN". A video thumbnail is visible with the title "Clase 05 - Testing Dinámico" and a duration of 38:40. The thumbnail shows a diagram titled "Grafos Causo-Efecto" with nodes A1, A2, and B, and the text "A1 y A2 implican B". The right sidebar lists popular channels: "dosogas", "Dos Bros", and "Luisito Comunica", each with a "SUSCRIBIRSE" button. The bottom of the page shows a Windows taskbar with the date 06/09/2017 and time 10:03.

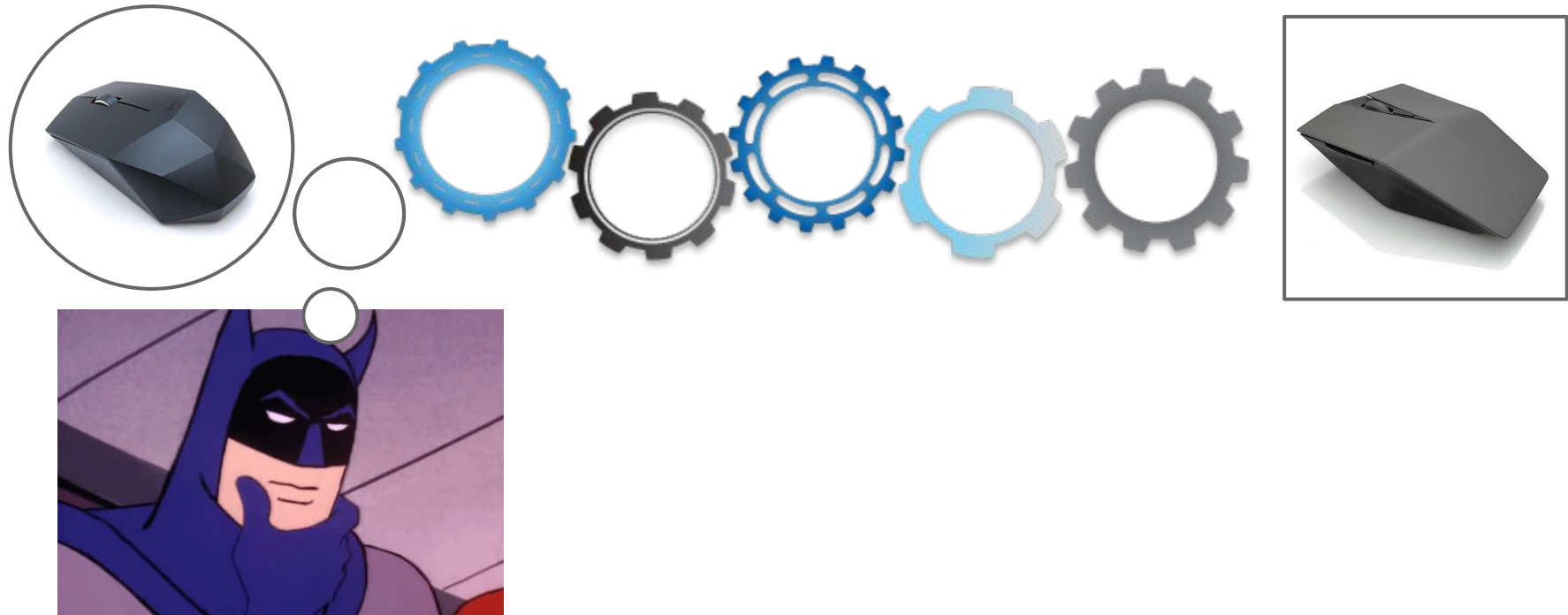
# Verificación y Validación de Software

¿Qué es Verificación y Validación?

# Verificación y Validación de Software



# Verificación y Validación de Software





# Verificación y Validación de Software



Verificación: ¿Construimos el mouse correctamente?

Validación: ¿Hicimos el mouse que el cliente quería?



# Verificación y Validación de Software

Verificación: ¿Construimos  
el mouse correctamente?

Podemos abusar de la terminología y llamar  
a todo esto **testing**

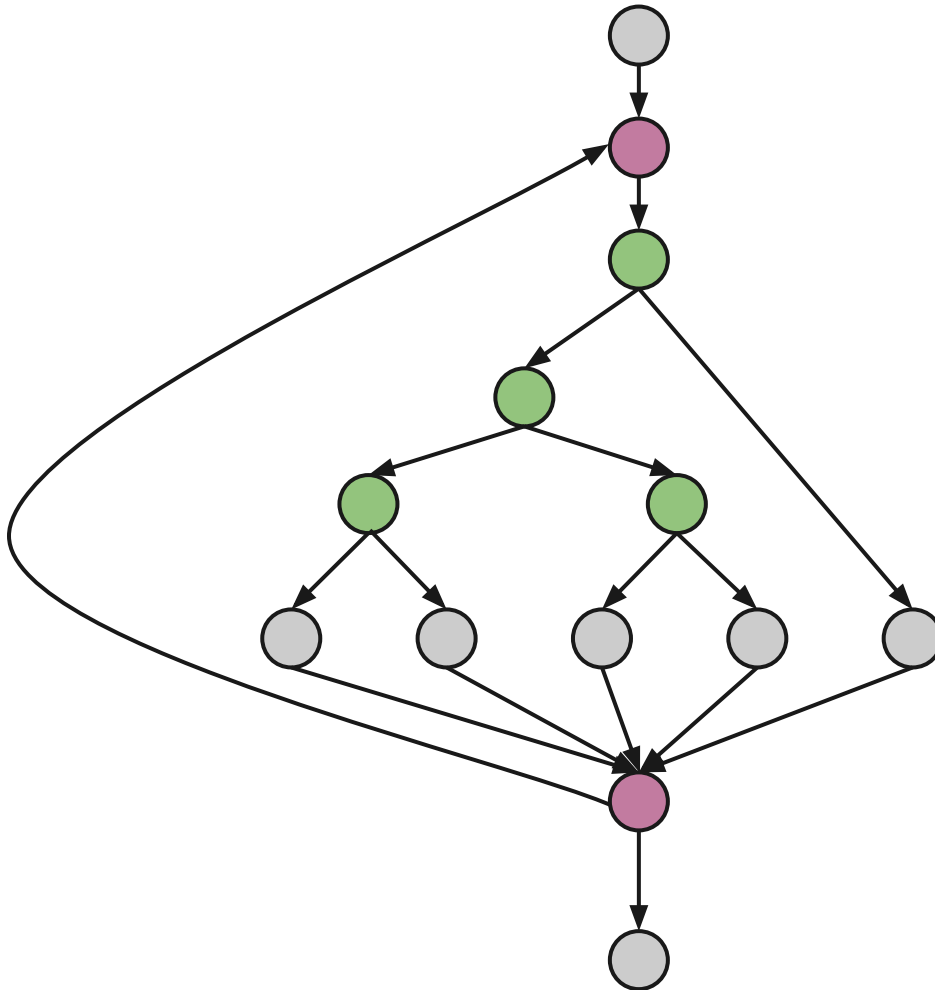
mouse que el cliente quería?

# Testing de Software

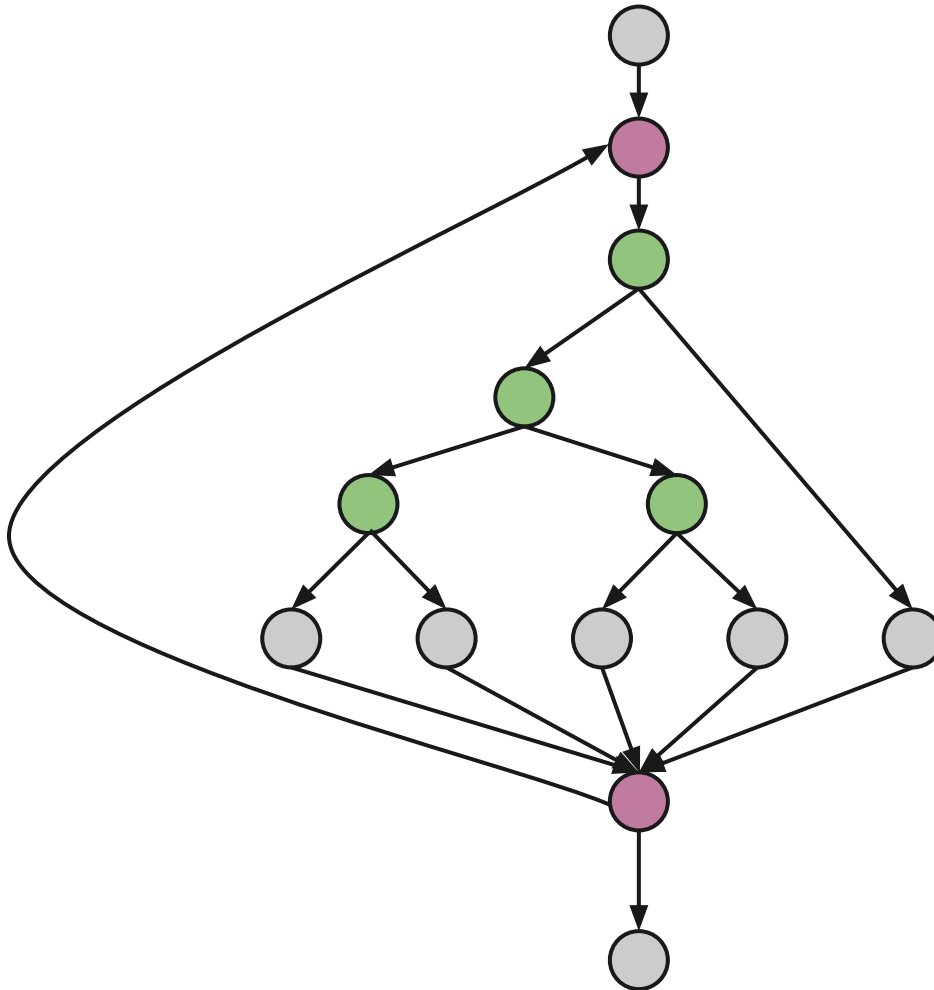


# Testing de Software

Un ejemplo

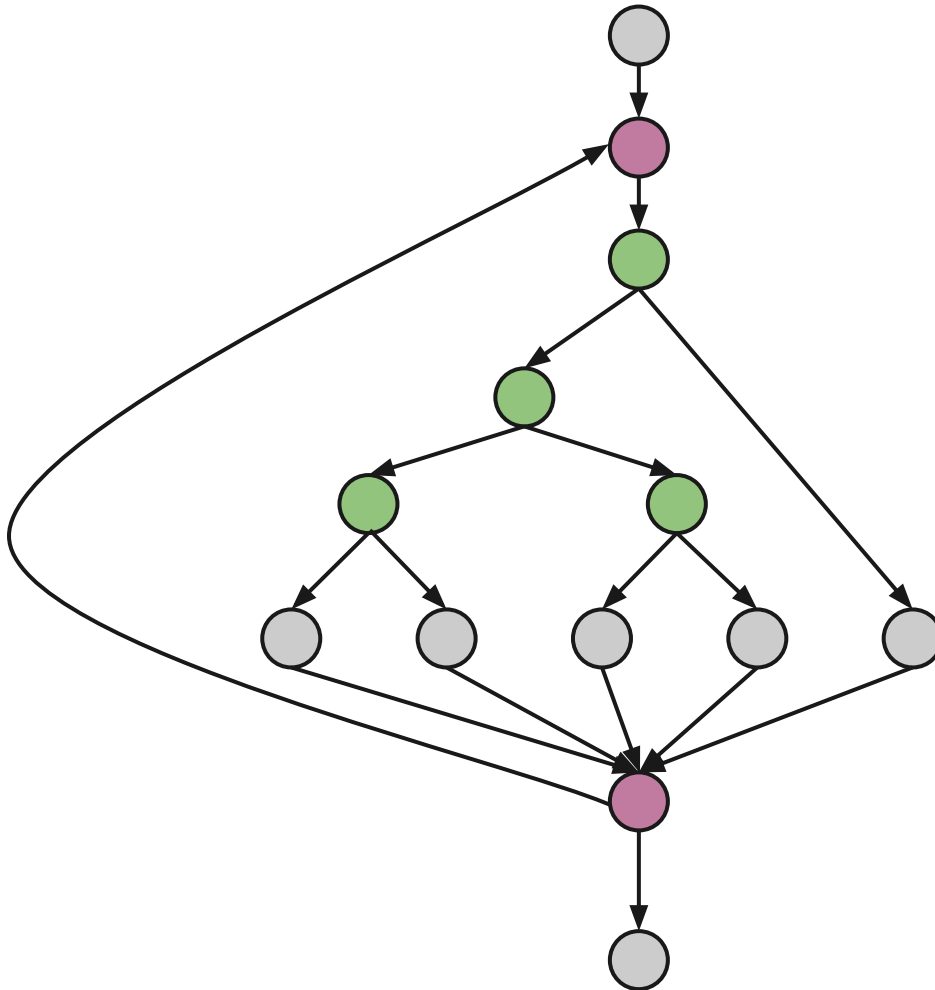


# Testing de Software



Probar este programa en forma exhaustiva significa probar cada posibilidad de camino.

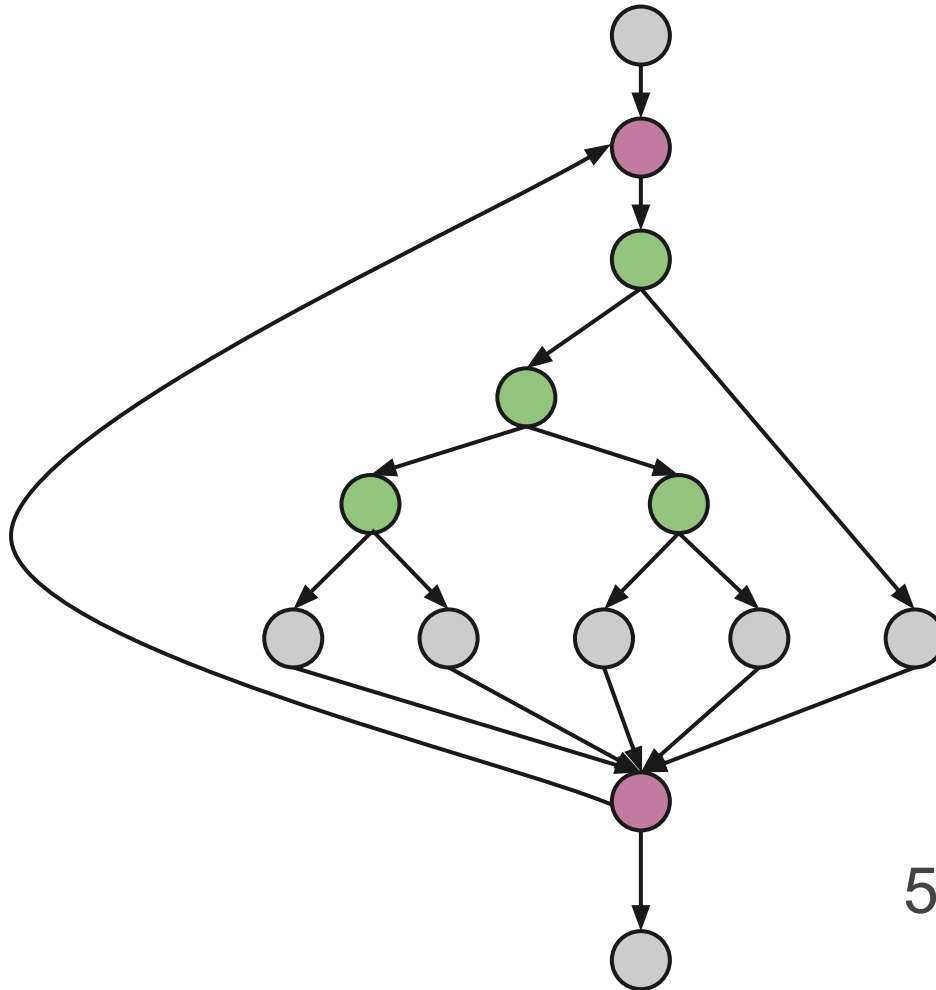
# Testing de Software



Si consideramos 1 iteración del bucle, tenemos 5 posibles caminos.



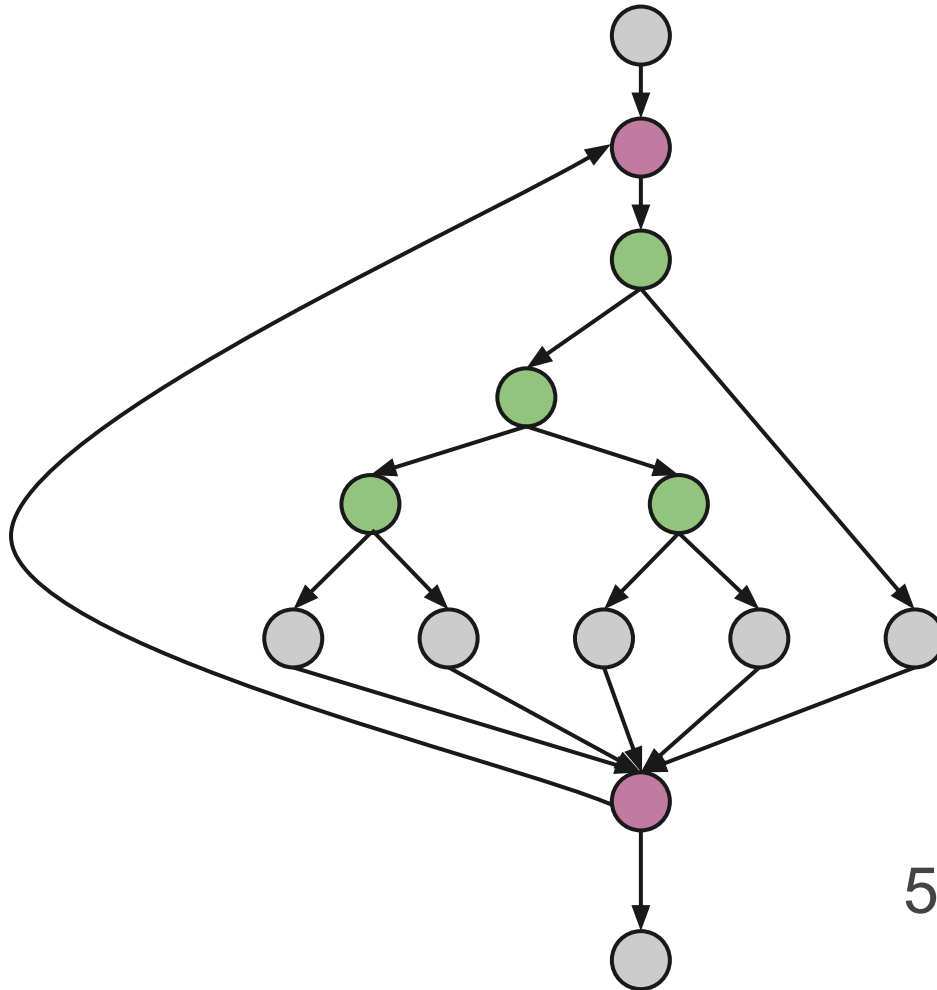
# Testing de Software



Si consideramos 2 iteración del bucle, tenemos 25 posibles caminos.

$$5^1 * 5^1 = 5^2 = 25$$

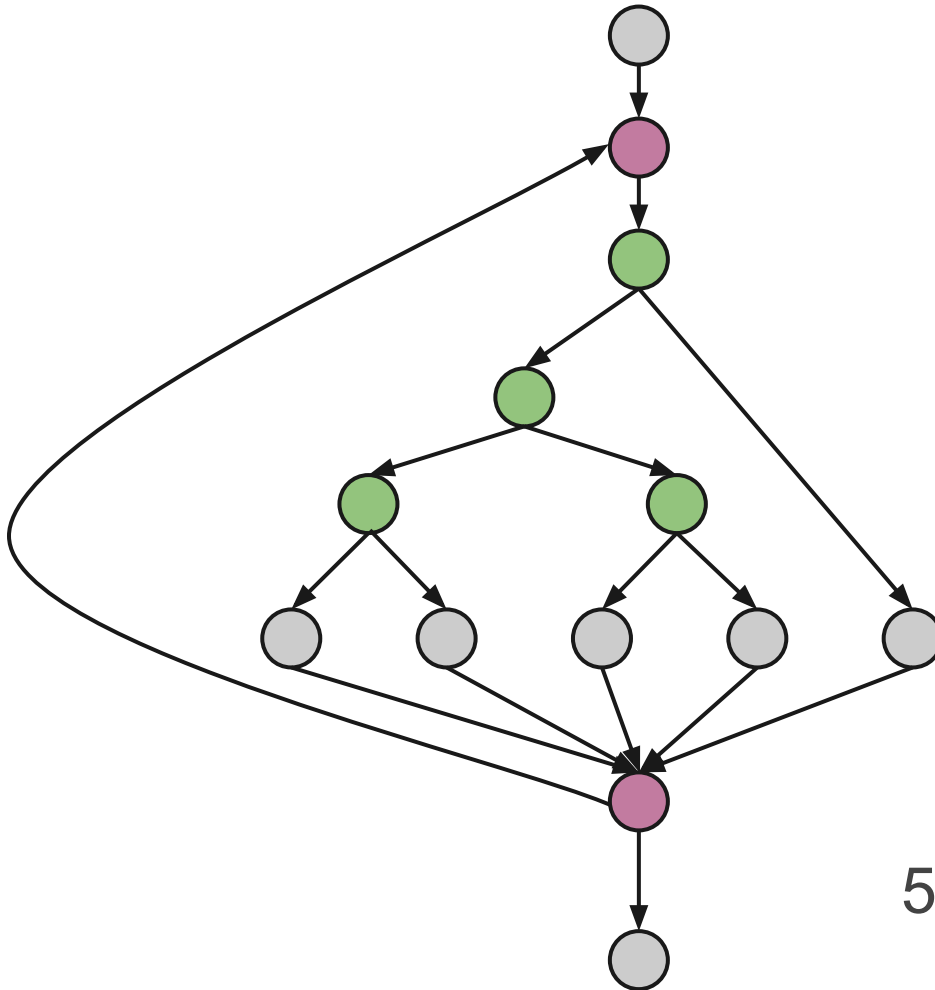
# Testing de Software



Si consideramos 20 iteraciones del bucle, tenemos estos posibles caminos.

$$5^{20} * 5^{19} * \dots * 5^2 * 5^1$$

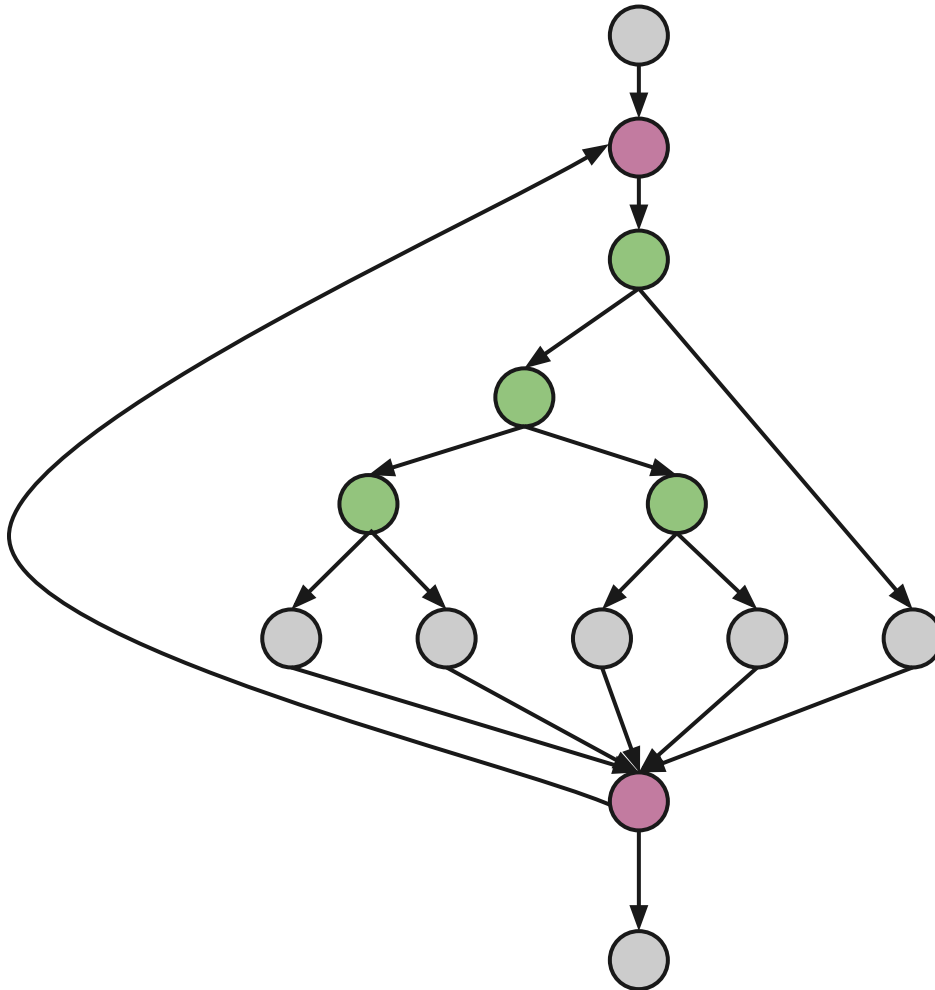
# Testing de Software



Si consideramos 20 iteraciones del bucle, tenemos estos posibles caminos.

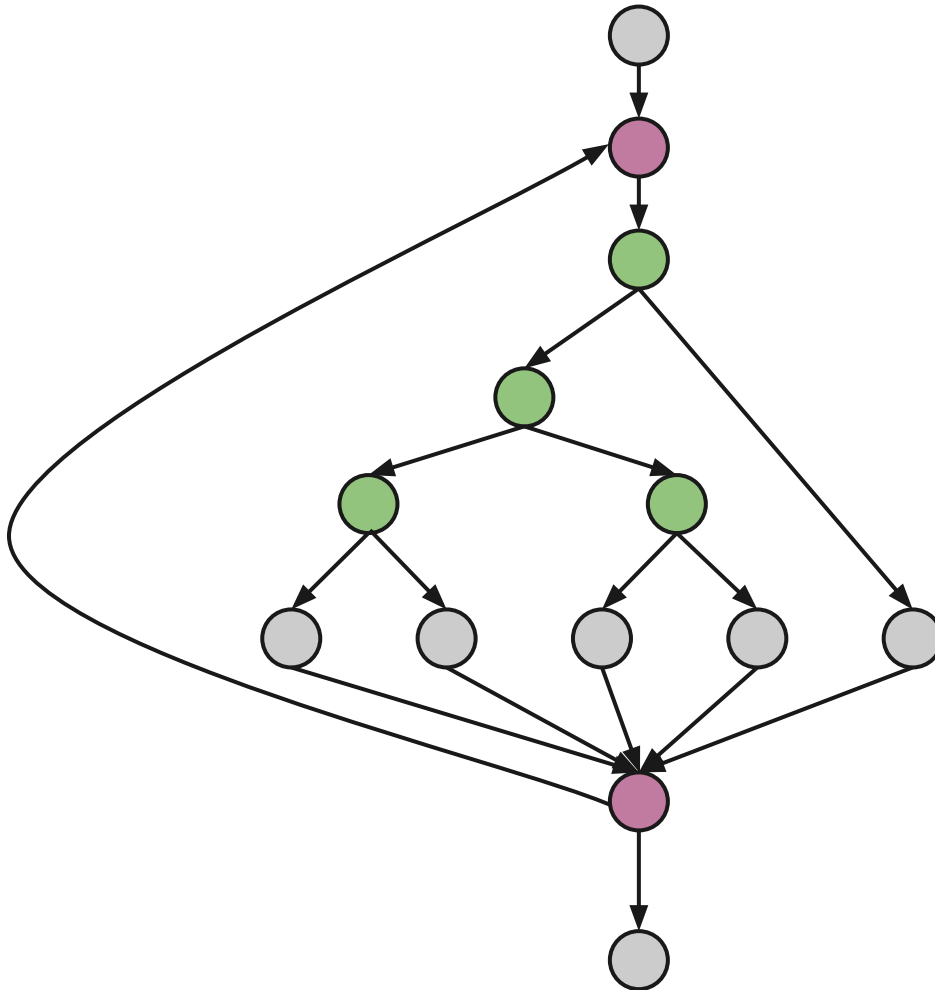
$$5^{20} = 9.5367432e+13$$

# Testing de Software



Si los test se hicieran en forma manual, a 5 minutos por test. Se tardaría mil millones de años.

# Testing de Software



Si los test se hicieran en 5 microsegundos por test. Se tardarían 19 años.

# Testing de Software

Si el testing exhaustivo no es posible...  
¿qué hacemos?



# Testing de Software

Hacemos un testing inteligente

# Testing de Software

“Implemente una función  $f$  que calcule el factorial de un número. El factorial de 1 es 1, y el factorial de  $n$  ( $n > 1$ ) es  $n * f(n-1)$ ”

# Testing de Software

“Implemente una función  $f$  que calcule el factorial de un número. **El factorial de 1 es 1**, y el factorial de  $n$  ( $n > 1$ ) es  $n * f(n-1)$ ”

Hay que probar con: 1

# Testing de Software

“Implemente una función  $f$  que calcule el factorial de un número. El factorial de 1 es 1, y el **factorial de  $n$  ( $n > 1$ ) es  $n * f(n-1)$ ”**

Hay que probar con: 1, 8

# Testing de Software

“Implemente una función  $f$  que calcule

¿Y el cero?

$f(1)$  es 1, y el factorial de  $n$  ( $n > 1$ ) es  $n * f(n-1)$ ”

Hay que probar con: 1, 8, 0

# Testing de Software

“Implemente una función  $f$  que calcule  
**¿Y el entero más grande posible?**  
1 es 1, y el factorial de  $n$  ( $n > 1$ ) es  
 $n * f(n-1)$ ”

Hay que probar con: 1, 8, 0, MAXINT



# Testing de Software

“Implemente una función  $f$  que calcule

¿Y un número negativo?

$f(1)$  es 1, y el factorial de  $n$  ( $n > 1$ ) es  $n * f(n-1)$ ”

Hay que probar con: 1, 8, 0, MAXINT, -5

# Testing de Software

“Implemente una función  $f$  que calcule

Hay que probar con: 1, 8, 0, MAXINT, -5

1 es 1, y el factorial de  $n$  ( $n > 1$ ) es  $n * f(n-1)$ ”

# Testing de Software

Esto no es un testing inteligente

```
1 public class Factorial {
2     public static void main(String[] args) {
3         int numero = factorial(8);
4         System.out.println("El factorial de 8 es: " + numero);
5     }
6     public static int factorial(int n) {
7         if(n == 1) {
8             return 1;
9         } else {
10            return n * factorial(n-1);
11        }
12    }
13 }
```

# Testing de Software

Siempre hacemos lo mismo, comparar lo observado con lo esperado. Esta es la esencia de Verificación y Validación de Software

# Testing de Software

Si yo no sé qué esperar, entonces es poco lo que puedo testear.

Para saber si algo anda mal, necesito saber cuándo anda bien.

# Testing de Software

Si yo no sé qué esperar, entonces es  
necesario lo que puede testear

La especificación del problema es crucial para el testing.

Para saber si algo anda mal, necesito  
saber cuándo anda bien.

# Principios del Testing de Software

**Testing shows the presence of defects, not their absence**

**Exhaustive testing is not possible**

**Testing activities should start as early as possible**

**Defects tend to cluster together**

**The pesticide paradox**

**Test is context dependent**

**The fallacy of assuming that no failures means a useful system**

# Tipos de Testing de Software

Existen diferentes tipos/categorías de testing



# Tipos de Testing de Software

## Estático Vs. Dinámico

# Tipos de Testing de Software

## Estático Vs. Dinámico

En el testing estático el objeto de test no se ejecuta, sólo se observa.

# Tipos de Testing de Software

## Estático Vs. Dinámico



# Tipos de Testing de Software

## Estático Vs. Dinámico

En el testing dinámico, el objeto de test se ejecuta. Se compara la salida del sistema observada con la esperada.

# Testing Estático

## Técnica: Examinación Grupal

Este testing estático se basa en la capacidad humana para comprender, analizar y detectar errores en situaciones complejas.

# Testing Estático

## Técnica: Análisis Estático / Análisis de Complejidad



**CYVIS**  
Software Complexity Visualiser

**CyVis**  
[About CyVis](#)  
[Download](#)  
[Features](#)  
[Release Notes](#)  
[License](#)

**Getting Started**  
[User Manual](#)  
[Graphical Interface](#)  
[Command Interface](#)  
[CyVis Ant Task](#)

**Samples**  
[Screenshots](#)  
[HTML Reports](#)  
[Text Reports](#)  
[XML Dumps](#)

**CyVis Forums**  
[User forums](#)  
[Developer Forums](#)  
[Bug-Tracking](#)

**Download CyVis**

Both the binary and the source for all the releases can be downloaded from CyVis' [SourceForge download page](#).

Refer to the following file naming scheme to see which file release best suits your needs:

Release distribution	Description
cyvis-major.minor-bin.zip	Complete installation, with binaries, documentation, and sample reports (but without source code)
cyvis-major.minor.build-src.zip	Source code distribution, with build instructions and documentation.

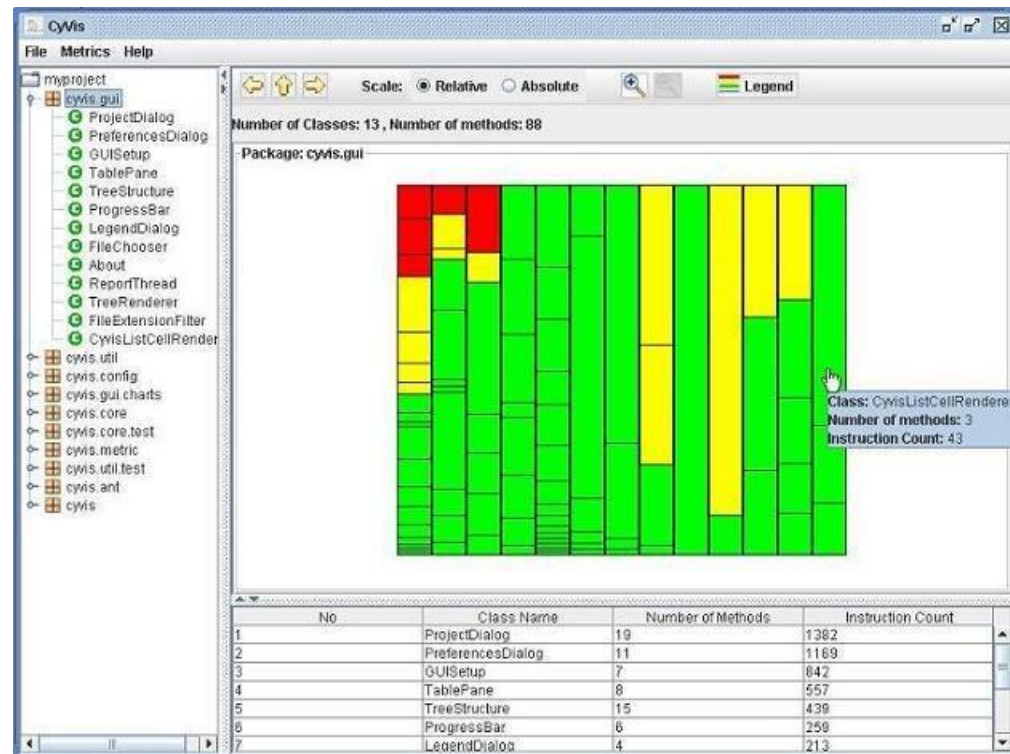
For any two consecutive builds of CyVis, the build with a higher number always indicates a more recent one.

SOURCEFORGE.NET®

© 2005-2006 Pradeep Selvaraj, Vinay Iyer.

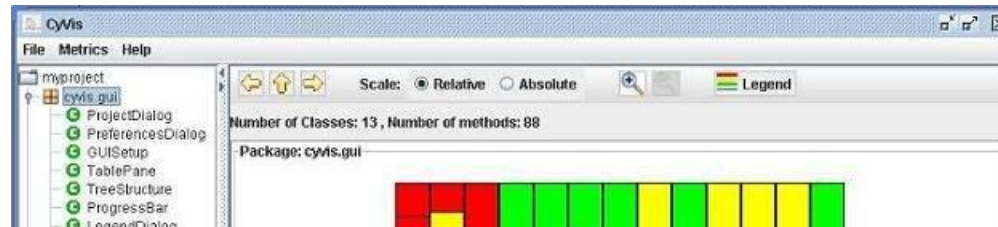
# Testing Estático

## Técnica: Análisis Estático / Análisis de Complejidad



# Testing Estático

## Técnica: Análisis Estático / Análisis de Complejidad



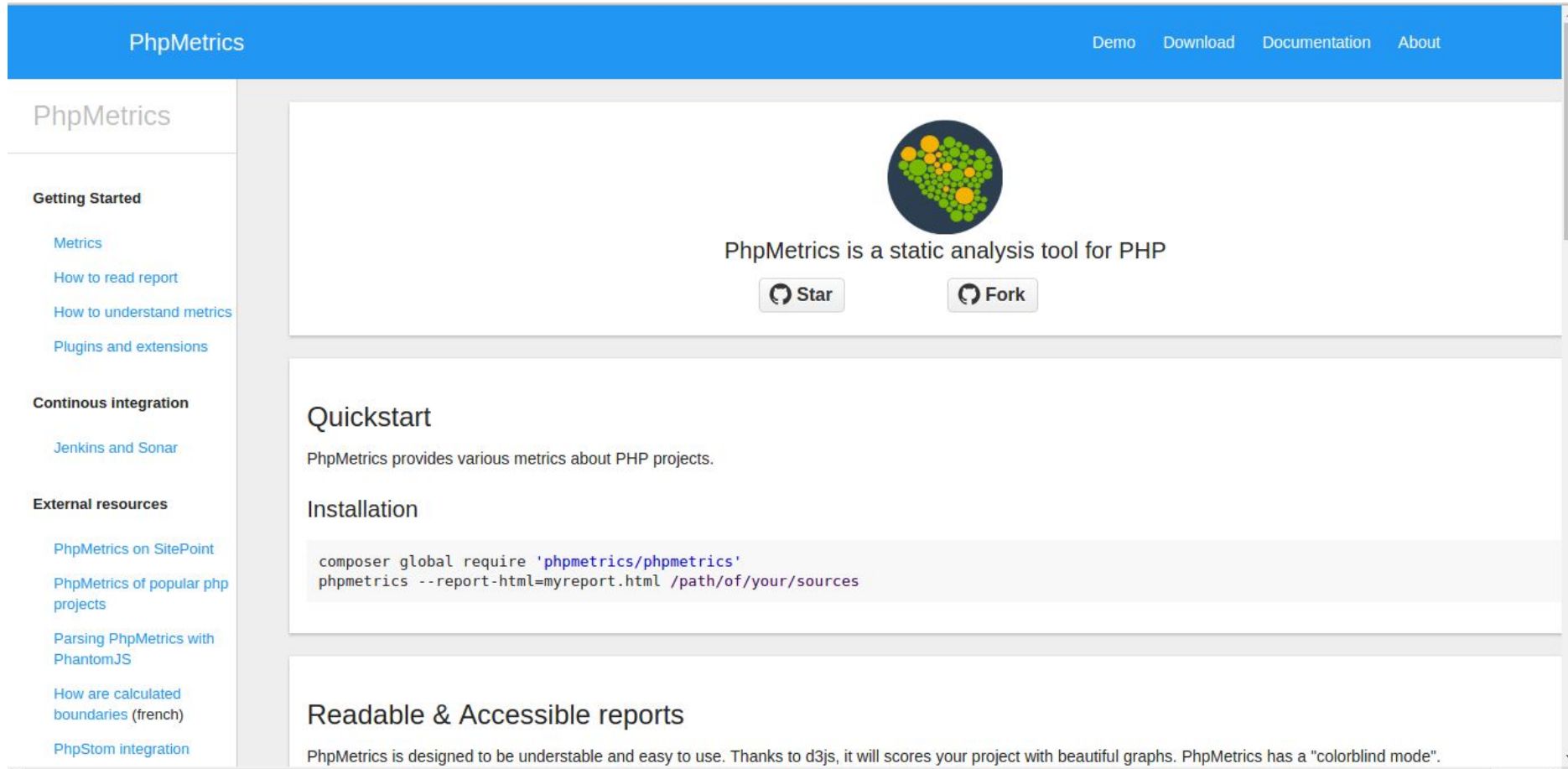
<http://cyvis.sourceforge.net/download.html>

No	Class Name	Number of Methods	Instruction Count
1	ProjectDialog	19	1382
2	PreferencesDialog	11	1169
3	GUISetup	7	842
4	TablePane	8	557
5	TreeStructure	15	439
6	ProgressBar	6	258
7	LegendDialog	4	213



# Testing Estático

## Técnica: Análisis Estático / Análisis de Complejidad



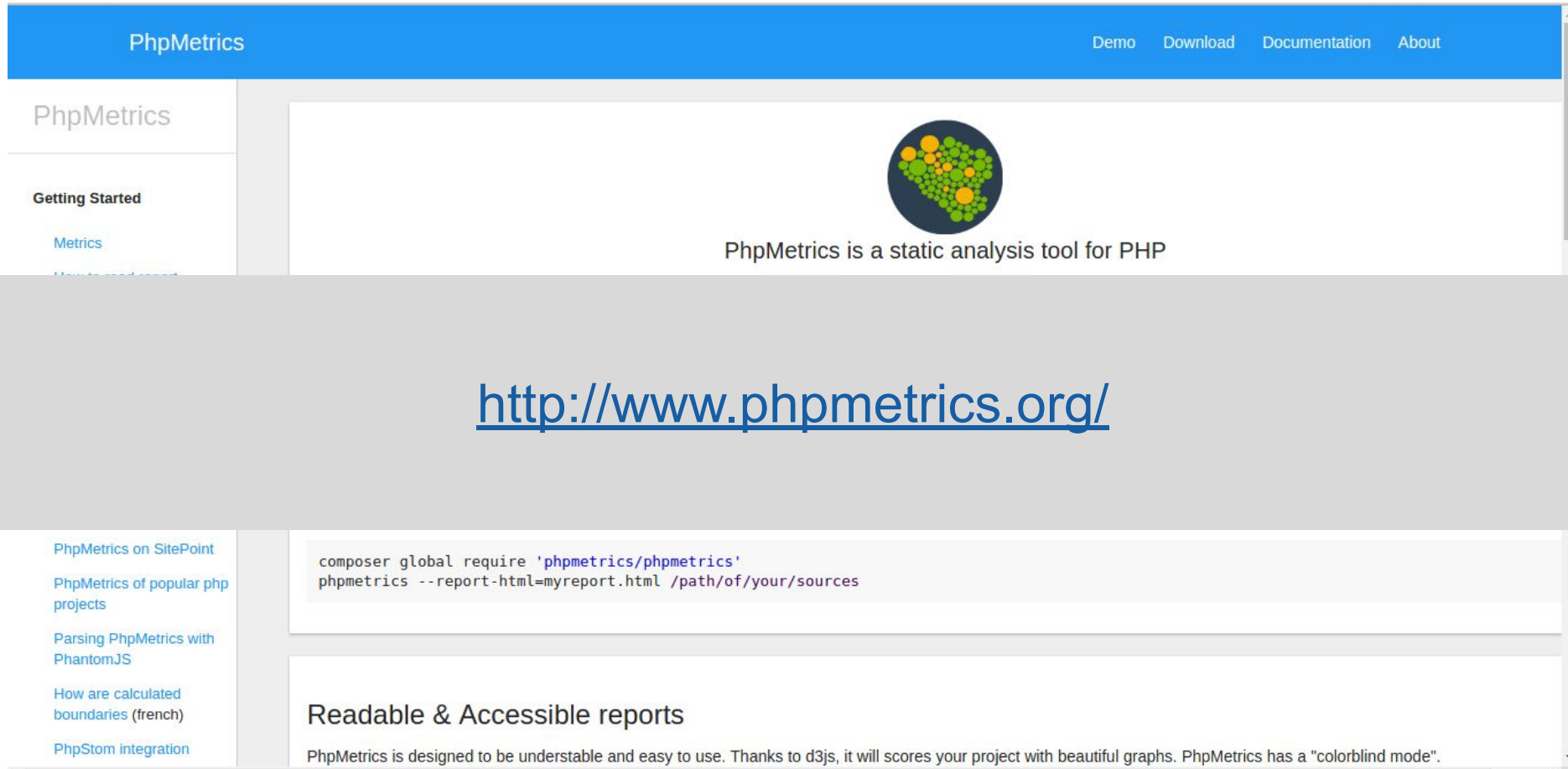
The screenshot shows the homepage of the PhpMetrics website. The page has a blue header with the 'PhpMetrics' logo on the left and navigation links for 'Demo', 'Download', 'Documentation', and 'About' on the right. A left sidebar contains a 'Getting Started' section with links to 'Metrics', 'How to read report', 'How to understand metrics', and 'Plugins and extensions'. Below this is a 'Continuous Integration' section with a link to 'Jenkins and Sonar', and an 'External resources' section with links to 'PhpMetrics on SitePoint', 'PhpMetrics of popular php projects', 'Parsing PhpMetrics with PhantomJS', 'How are calculated boundaries (french)', and 'PhpStom integration'. The main content area features a large circular logo of green and yellow dots, followed by the text 'PhpMetrics is a static analysis tool for PHP' and buttons for 'Star' and 'Fork'. Below this is a 'Quickstart' section stating 'PhpMetrics provides various metrics about PHP projects.' and an 'Installation' section with a code block: 

```
composer global require 'phpmetrics/phpmetrics'  
phpmetrics --report-html=myreport.html /path/of/your/sources
```

 At the bottom, there is a 'Readable & Accessible reports' section stating 'PhpMetrics is designed to be understandable and easy to use. Thanks to d3js, it will scores your project with beautiful graphs. PhpMetrics has a "colorblind mode".'

# Testing Estático

## Técnica: Análisis Estático / Análisis de Complejidad



The screenshot shows the homepage of the PhpMetrics website. The header is blue with the text 'PhpMetrics' on the left and navigation links 'Demo', 'Download', 'Documentation', and 'About' on the right. Below the header, there is a large white area with a circular logo containing green and yellow dots. The text below the logo reads 'PhpMetrics is a static analysis tool for PHP'. Below this, there is a large blue link: <http://www.phpmetrics.org/>. At the bottom, there is a section titled 'Readable & Accessible reports' with a paragraph of text. On the left side, there is a sidebar with several links: 'Getting Started', 'Metrics', 'How to install', 'PhpMetrics on SitePoint', 'PhpMetrics of popular php projects', 'Parsing PhpMetrics with PhantomJS', 'How are calculated boundaries (french)', and 'PhpStom integration'. The main content area also contains a code block with the following text: 

```
composer global require 'phpmetrics/phpmetrics'  
phpmetrics --report-html=myreport.html /path/of/your/sources
```

# Testing Estático

## Técnica: Análisis Estático

Secure | <https://www.codacy.com>

C O D A C Y

Product Resources Pricing Enterprise Login [Sign up](#)

# Review less, merge faster

Check code style, security, duplication, complexity and coverage on every change while tracking code quality throughout your sprints.

[Sign up with GitHub](#)

[Or sign up with a different account](#)

Scala Java JS python php And more →

# Testing Estático

## Técnica: Análisis Estático

opensource / jquery Product Pricing Enterprise [Sign Up](#)

**Automate your code reviews!** Codacy is 100% free for open source projects. [Get started for free!](#)

Dashboard master ▾

### A Project Certification

Code Complexity <b>No Patterns</b>	Code Style <b>84%</b>
Compatibility <b>100%</b>	Documentation <b>No Patterns</b>
Error Prone <b>88%</b>	Performance <b>96%</b>
Security <b>100%</b>	Unused Code <b>98%</b>

### Issues Breakdown

**196**  
Total Issues

Code Style	95
Error Prone	70
Performance	23
Others	8

### Coverage

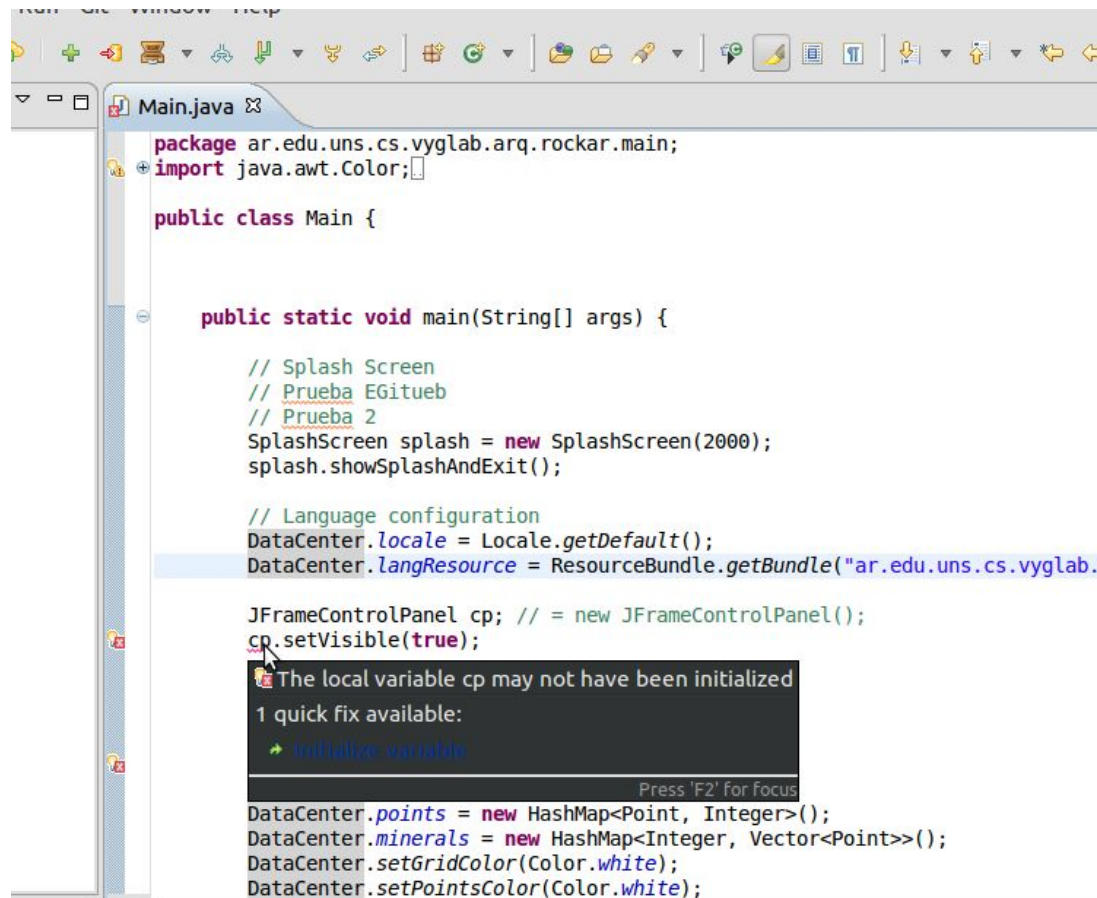
**Setup Coverage**

Start getting coverage results and notifications.

[Setup Coverage](#)

# Testing Estático

## Técnica: Análisis de Flujo de Datos



```
package ar.edu.uns.cs.vyglab.arq.rockar.main;
import java.awt.Color;

public class Main {

    public static void main(String[] args) {

        // Splash Screen
        // Prueba EGitueb
        // Prueba 2
        SplashScreen splash = new SplashScreen(2000);
        splash.showSplashAndExit();

        // Language configuration
        DataCenter.locale = Locale.getDefault();
        DataCenter.langResource = ResourceBundle.getBundle("ar.edu.uns.cs.vyglab.

        JFrameControlPanel cp; // = new JFrameControlPanel();
        cp.setVisible(true);

        DataCenter.points = new HashMap<Point, Integer>();
        DataCenter.minerals = new HashMap<Integer, Vector<Point>>();
        DataCenter.setGridColor(Color.white);
        DataCenter.setPointsColor(Color.white);
    }
}
```

The local variable cp may not have been initialized  
1 quick fix available:  
Initialize variable

# Testing Estático

## Técnica: Análisis de Flujo de Datos

El Set-Use Pairs es un sistema de notación para el análisis de flujo de datos. El ciclo de vida de las variables se divide en tres etapas:

**d:** Esta etapa incluye la creación, definición o inicialización de una variable.

**u:** Esta etapa incluye el uso de la variable. El uso puede ser de cálculo o en una decisión.

**k:** Esta etapa incluye la destrucción, deallocate o salida del área de alcance.

# Testing Estático

## Técnica: Análisis de Flujo de Datos

~d	~u	~k	dd	d~
du	ud	ku	uu	u~
dk	uk	kd	kk	k~

# Testing Estático

## Técnica: Análisis de Flujo de Datos

Eclipse documentation - Current Release Eclipse Luna		
overridden in a subclass, so if you make a "potentially static" method static, this may break existing clients.		
<b>Potential programming problems</b>		
Comparing identical values ('x == x')	When enabled, the compiler will issue an error or a warning if a comparison is involving identical operands (e.g. 'x == x').	Warning
Assignment has no effect (e.g. 'x = x')	When enabled, the compiler will issue an error or a warning whenever an assignment has no effect (e.g. 'x = x').	Warning
Possible accidental boolean assignment (e.g. 'if (a = b)')	When enabled, the compiler will issue an error or a warning whenever it encounters a possible accidental boolean assignment (e.g. 'if (a = b)').	Ignore
Boxing and unboxing conversions	When enabled, the compiler will issue an error or a warning whenever it encounters a boxing or unboxing conversion. Autoboxing may affect performance negatively.	Ignore
Using a char array in string concatenation	When enabled, the compiler will issue an error or a warning whenever a char[] expression is used in String concatenations. "hello" + new char[]{'w', 'o', 'r', 'l', 'd'}	Warning
Inexact type match for vararg arguments	When enabled, the compiler will issue an error or a warning whenever it encounters an inexact type match for vararg arguments.	Warning
Empty statement	When enabled, the compiler will issue an error or a warning whenever it encounters an empty statement (e.g. a superfluous semicolon).	Ignore
Unused object allocation	When enabled, the compiler will issue an error or a warning when it encounters an allocated object which is not used, e.g. <pre>if (name == null)     new IllegalArgumentException();</pre>	Ignore
Incomplete 'switch' cases on enum	When enabled, the compiler will issue an error or a warning whenever it encounters a 'switch' statement which does not contain a 'default' case nor case statements for every enum constant of the referenced enum. This warning is recommended by the Java Language Specification <a href="#">14.11</a> . It helps to ensure that 'switch' statements cover all possible enum values.	Warning
Signal even if 'default' case exists	When enabled, the compiler additionally will issue an error or a warning if an enum constant is not covered by a case, even if a 'default' case exists. This option helps to catch missing case statements when a new enum constant is added.	Off
'switch' is missing 'default' case	When enabled, the compiler will issue an error or a warning if a 'switch' statement lacks a 'default' case. Consequently, a missing 'default' will be flagged even if all possible values are otherwise covered by 'case' statements. This option helps to ensure that new 'switch' expression values are handled explicitly, rather than being skipped. It can also help to explain compile errors for un-initialized variables after a 'switch' statement: The set of legal values can grow in the future, so the variable also needs to be initialized in the 'default' case.	Ignore



# Testing Estático

La principal ventaja del Testing Estático es su bajo costo y la facilidad de uso de las herramientas

# Testing Dinámico

En el testing dinámico, el objeto de test se ejecuta. Se compara la salida del sistema observada con la esperada

# Testing Dinámico

## Caja Negra Vs. Caja Blanca

# Testing Dinámico

## Caja Negra Vs. Caja Blanca

También llamada “Basada en Especificación”.  
El testing se hace en base al comportamiento del sistema, sin tomar en cuenta los detalles de implementación para generar los casos de test

# Testing Dinámico

## Caja Negra Vs. Caja Blanca

Los casos de test se diseñan en base a cómo el sistema fue construido. El código es una documentación fundamental para este tipo de test.

# Testing de Caja Negra

## Partición de Equivalencias

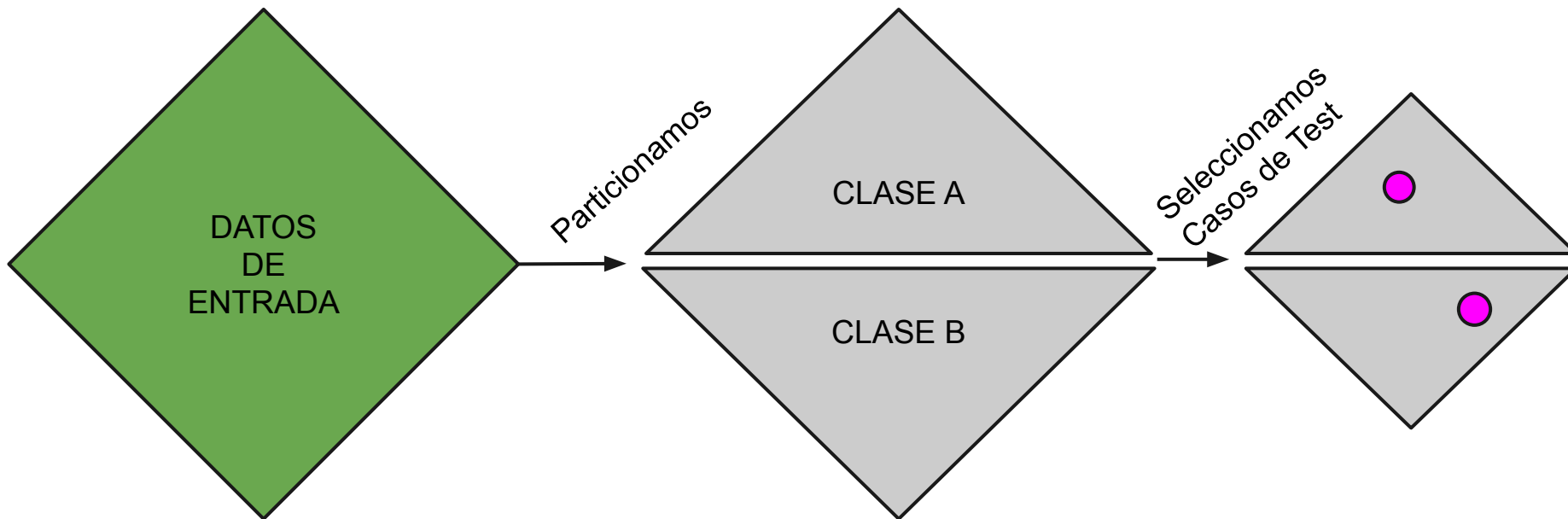
Se testean varios grupos los cuales se esperan que sean manejados de igual forma por el sistema.

El Modelo que se utiliza es el de clases de equivalencias. Clases de datos de entrada, salida, cálculos... cualquier cosa. Las clases son equivalentes porque el sistema las maneja igual.

Podemos tener clases de equivalencias válidas e inválidas. Las válidas son las que son manejadas por el sistema en forma normal. Las inválidas son rechazadas por el sistema.

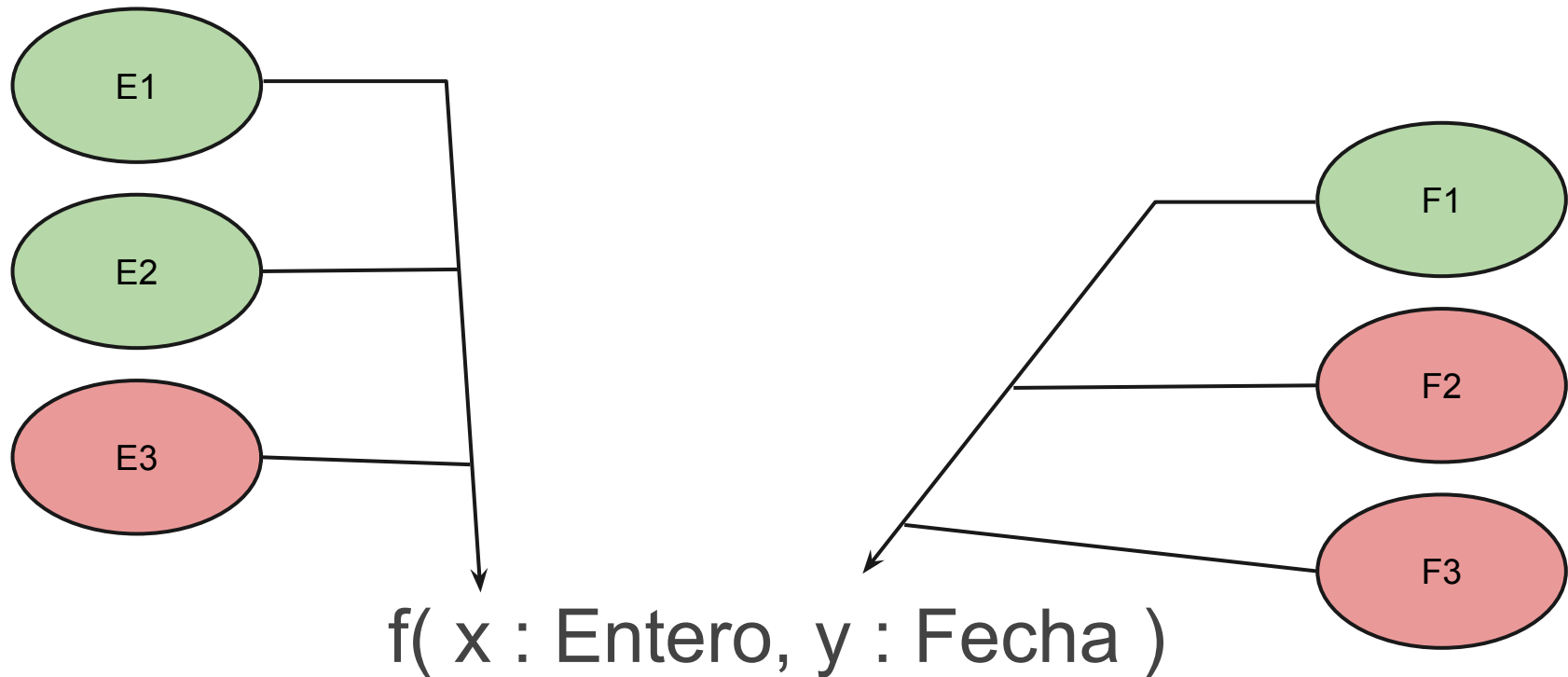
# Testing de Caja Negra

## Partición de Equivalencias



# Testing de Caja Negra

## Partición de Equivalencias

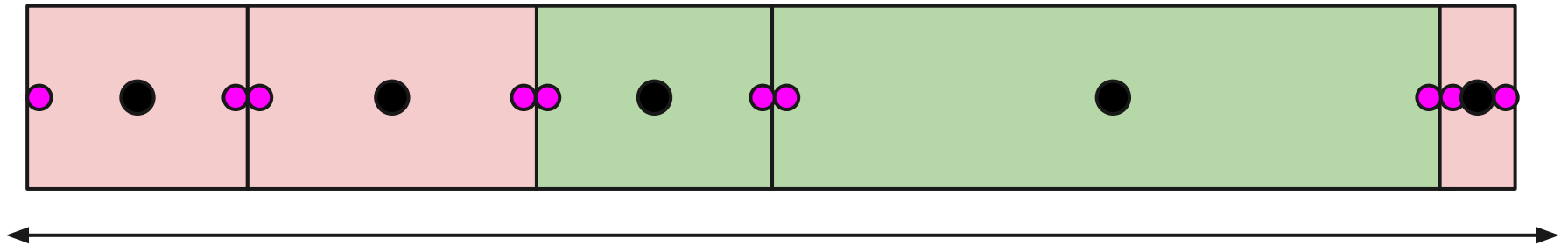




# Testing de Caja Negra

Partición de Equivalencias

Análisis de Valores Límites



# Testing de Caja Negra

Estructuras de Datos

Trabajo Práctico N°1

2017

- c) Implemente un programa en Java que lea dos números por teclado e imprima el cociente entre los mismos. En el caso en que el divisor sea cero, el programa debe capturar una `ArithmeticException` e imprimir el mensaje de error correspondiente.

# Testing de Caja Negra

Estructuras de Datos

Trabajo Práctico N°1

2017

- c) Implemente un programa en Java que lea dos números por teclado e imprima el cociente entre los mismos. En el caso en que el divisor sea cero, el programa debe capturar una `ArithmeticException` e imprimir el mensaje de error correspondiente.

Dato de Entrada A

Enteros

Dato de Entrada B

Enteros

# Testing de Caja Negra

Estructuras de Datos

Trabajo Práctico N°1

2017

- c) Implemente un programa en Java que lea dos números por teclado e imprima el cociente entre los mismos. En el caso en que el divisor sea cero, el programa debe capturar una `ArithmeticException` e imprimir el mensaje de error correspondiente.

Dato de Entrada A



Enteros

Dato de Entrada B

Enteros



Casos de Prueba: (-9,300)

# Testing de Caja Negra

Estructuras de Datos

Trabajo Práctico N°1

2017

- c) Implemente un programa en Java que lea dos números por teclado e imprima el cociente entre los mismos. En el caso en que el divisor sea cero, el programa debe capturar una `ArithmeticException` e imprimir el mensaje de error correspondiente.

Dato de Entrada A

Enteros

Dato de Entrada B

Enteros &lt; 0



0



Enteros &gt; 0

Casos de Prueba: (-9,300), (-10,0), (-5000, 50)

# Testing de Caja Negra

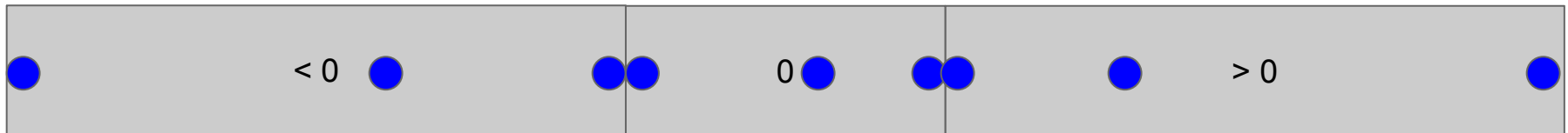
Estructuras de Datos

Trabajo Práctico N°1

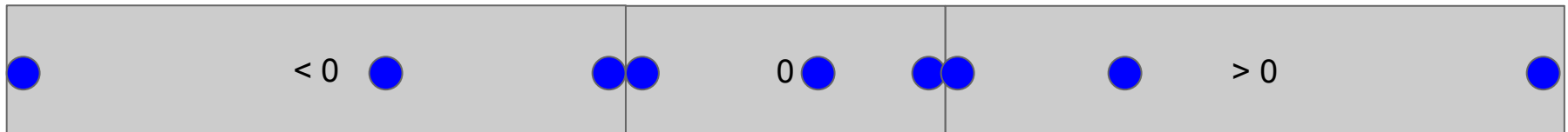
2017

- c) Implemente un programa en Java que lea dos números por teclado e imprima el cociente entre los mismos. En el caso en que el divisor sea cero, el programa debe capturar una `ArithmeticException` e imprimir el mensaje de error correspondiente.

Dato de Entrada A



Dato de Entrada B



# Testing Dinámico

## Caja Blanca

Los casos de test se diseñan en base a cómo el sistema fue construido. El código es una documentación fundamental para este tipo de test.

# Testing de Caja Blanca

## Testing del Flujo de Control

El flujo de control de un programa, es la estructura que determina los “caminos” que se pueden tomar a lo largo del código. Los if, while, repeat, case, forman parte de la estructura de control.



# Testing de Caja Blanca

## Grafo de Flujo de Control

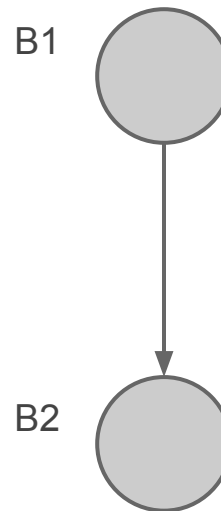
Control-Flow Graph

El Grafo de Flujo de Control es el modelo/representación que nos va a facilitar realizar/evaluar los diferentes cubrimiento.

# Testing de Caja Blanca

## Grafo de Flujo de Control

Control-Flow Graph



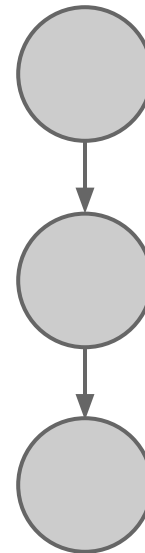
Dos sentencias de código en secuencia

# Testing de Caja Blanca

## Grafo de Flujo de Control

Control-Flow Graph

```
int x = 23;  
x++;  
f( x );
```

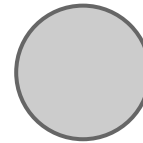


# Testing de Caja Blanca

## Grafo de Flujo de Control

Control-Flow Graph

```
int x = 23;  
x++;  
f( x );
```

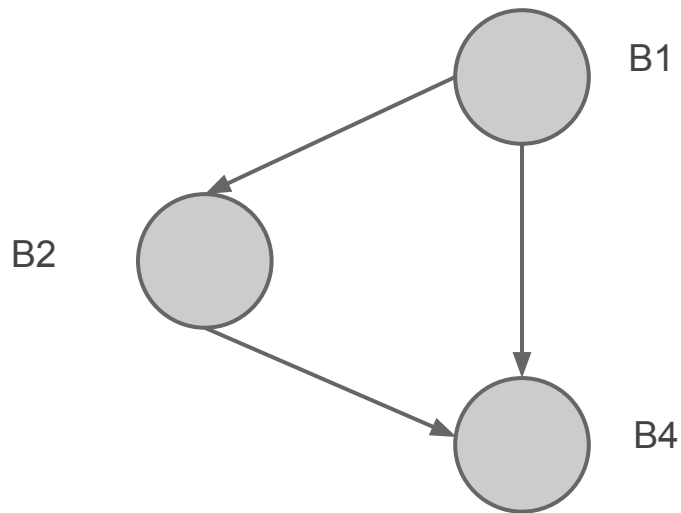


Como, en este ejemplo, no se modifica el flujo de control vamos a considerarlo un único nodo.

# Testing de Caja Blanca

## Grafo de Flujo de Control

Control-Flow Graph



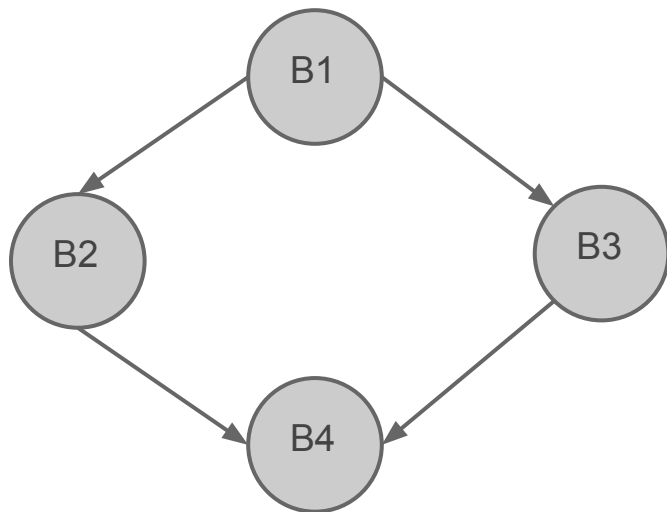
```
IF B1 THEN {  
    B2;  
}  
B4
```

IF-THEN

# Testing de Caja Blanca

## Grafo de Flujo de Control

Control-Flow Graph



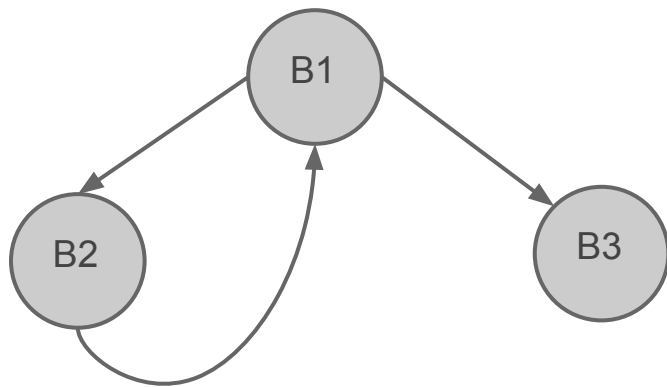
```
IF B1 THEN {  
    B2; }  
ELSE {  
    B3; }  
B4
```

IF-THEN-ELSE

# Testing de Caja Blanca

## Grafo de Flujo de Control

Control-Flow Graph



```
WHILE B1 {  
    B2;  
}  
B3;
```

CICLO

# Testing de Caja Blanca

## Grafo de Flujo de Control

Control-Flow Graph

```
PROGRAM
VAR x,y:REAL
BEGIN
    Read(x);
    Read(y);
    IF (x != 0)
        THEN x = x + 10;
    y = y / 10;
    Write(x);
    Write(y);
END
```

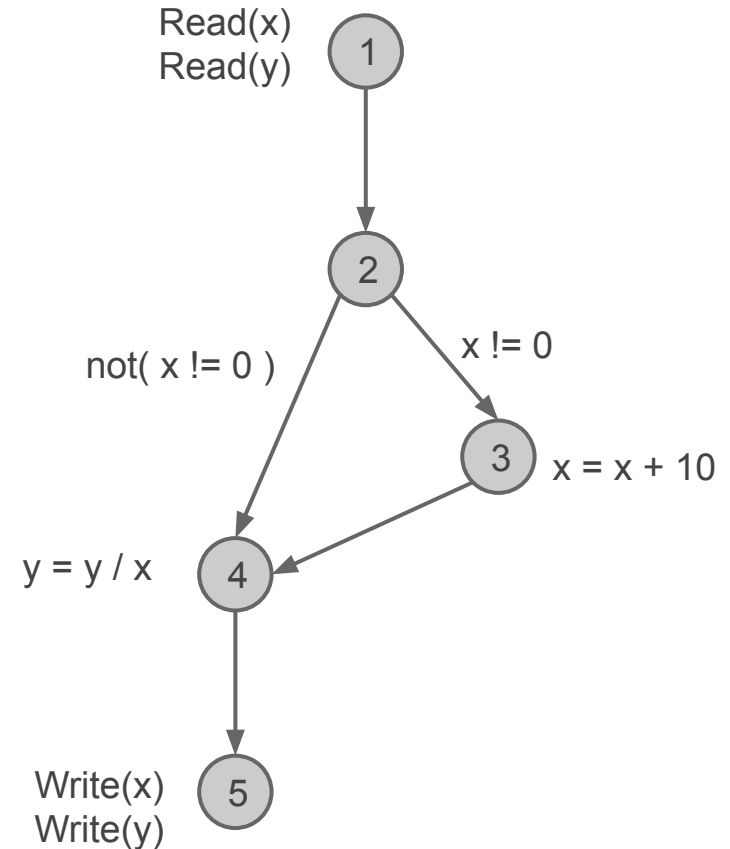


# Testing de Caja Blanca

## Grafo de Flujo de Control

Control-Flow Graph

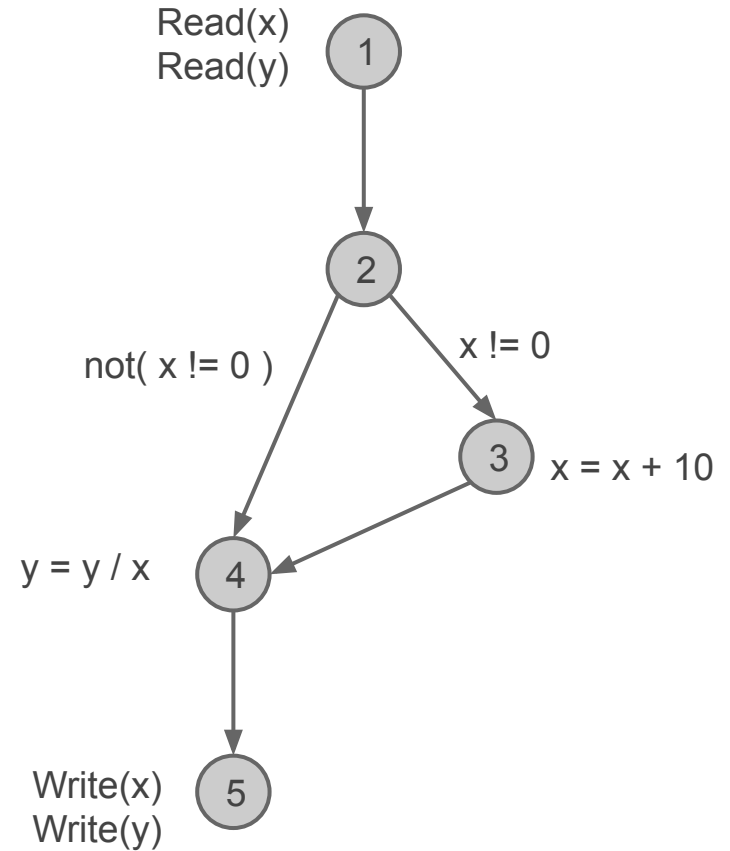
```
PROGRAM  
VAR x,y:REAL  
BEGIN  
  Read(x);  
  Read(y);  
  IF (x != 0)  
    THEN x = x + 10;  
  y = y / 10;  
  Write(x);  
  Write(y);  
END
```



# Testing de Caja Blanca

## Criterios de Cubrimiento

Criterio de Comandos



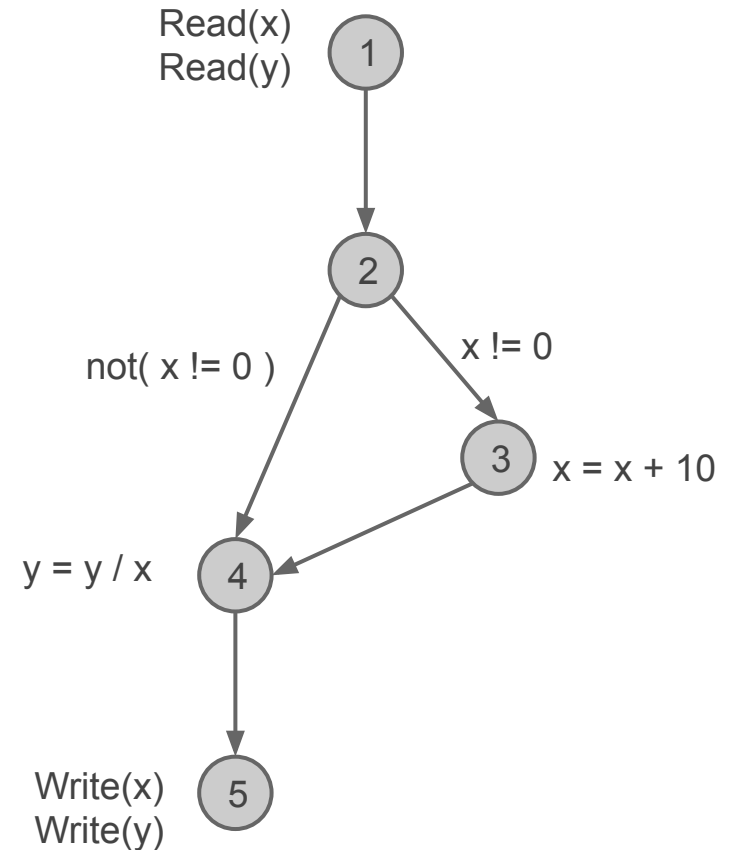
# Testing de Caja Blanca

## Criterios de Cubrimiento

### Criterio de Comandos

Un test T satisface este criterio ssi cada comando (sentencia) ejecutable del programa es ejecutado al menos una vez.

Visito todos los nodos.



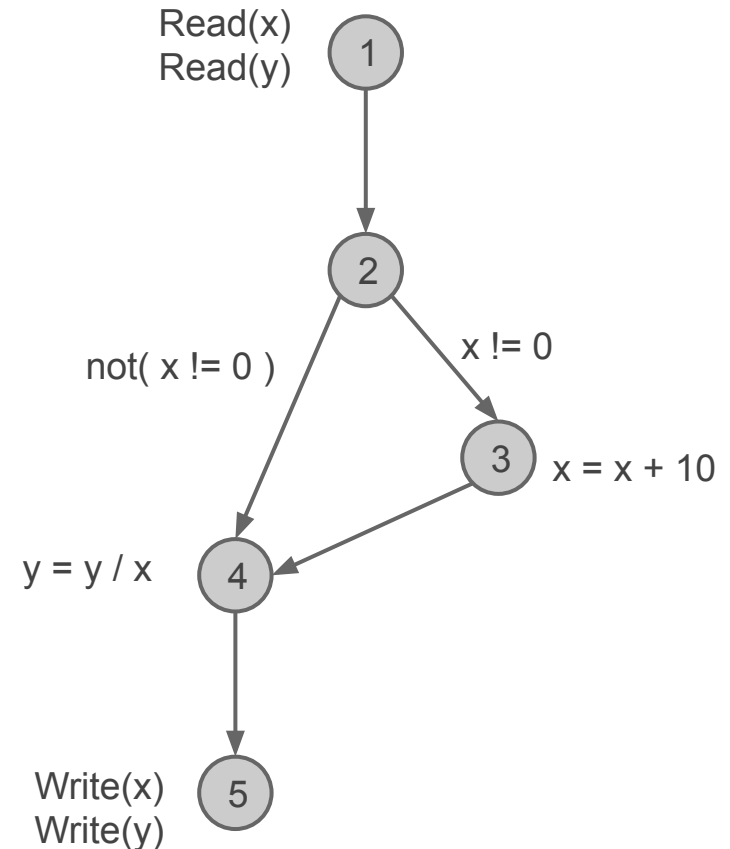
# Testing de Caja Blanca

## Criterios de Cubrimiento

### Criterio de Comandos

Un test  $T$  satisface este criterio ssi cada comando (sentencia) ejecutable del programa es ejecutado al menos una vez.

El camino  $[1, 2, 3, 4, 5]$  visita todos los nodos del grafo. Si pudiese encontrar un par  $(x, y)$  (o más pares) tal que al usarlos como inputs se ejecutase dicho camino, entonces podríamos decir que el test  $T = \{(x,y)\}$  satisface el Criterio de Comandos.



# Testing de Caja Blanca

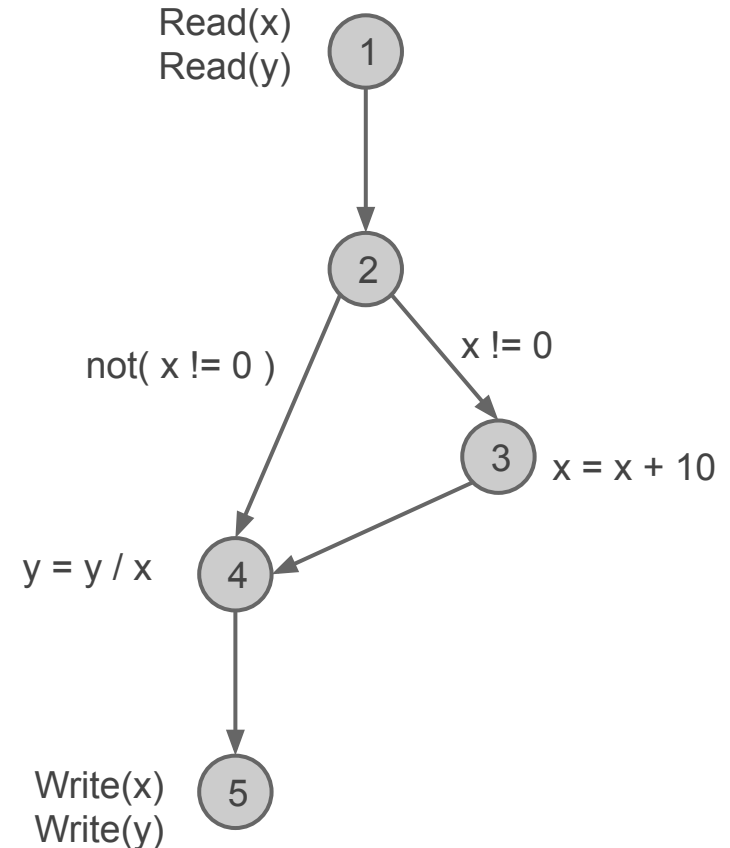
## Criterios de Cubrimiento

### Criterio de Comandos

Un test  $T$  satisface este criterio ssi cada comando (sentencia) ejecutable del programa es ejecutado al menos una vez.

El camino  $[1, 2, 3, 4, 5]$  visita todos los nodos del grafo. Si pudiese encontrar un par  $(x, y)$  (o más pares) tal que al usarlos como inputs se ejecutase dicho camino, entonces podríamos decir que el test  $T = \{(x,y)\}$  satisface el Criterio de Comandos.

$$T_{CC} = \{(x=20, y=30)\}$$



# Testing de Caja Blanca

## Criterios de Cubrimiento

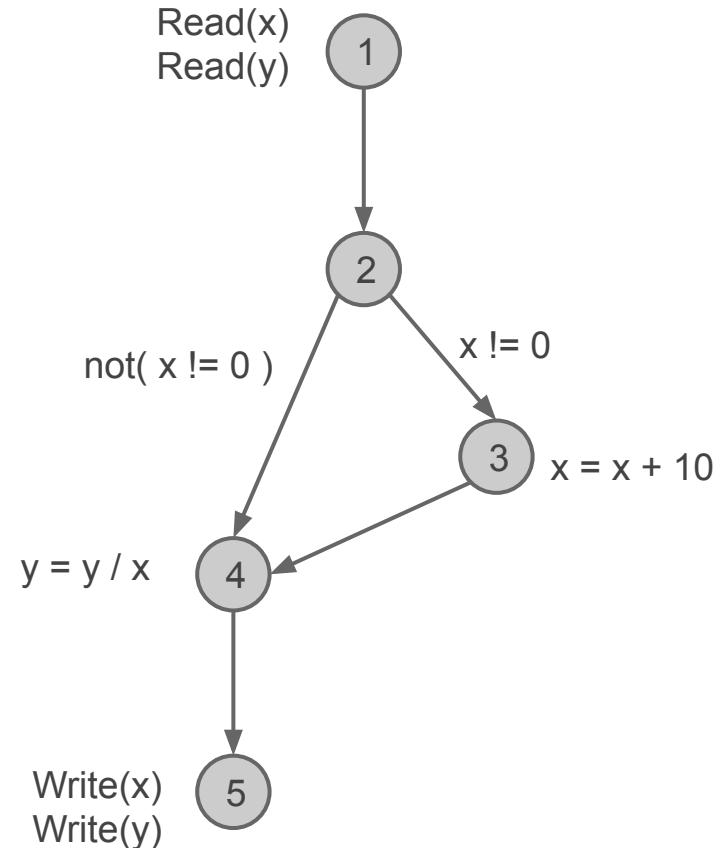
### Criterio de Comandos

Un test  $T$  satisface este criterio ssi cada comando (sentencia) ejecutable del programa es ejecutado al menos una vez.

El camino  $[1, 2, 3, 4, 5]$  visita todos los nodos del grafo. Si pudiese encontrar un par  $(x, y)$  (o más pares) tal que al usarlos como inputs se ejecutase dicho camino, entonces podríamos decir que el test  $T = \{(x,y)\}$  satisface el Criterio de Comandos.

$$T_{CC} = \{(x=20, y=30)\}$$

Puedo llegar a necesitar más pares...



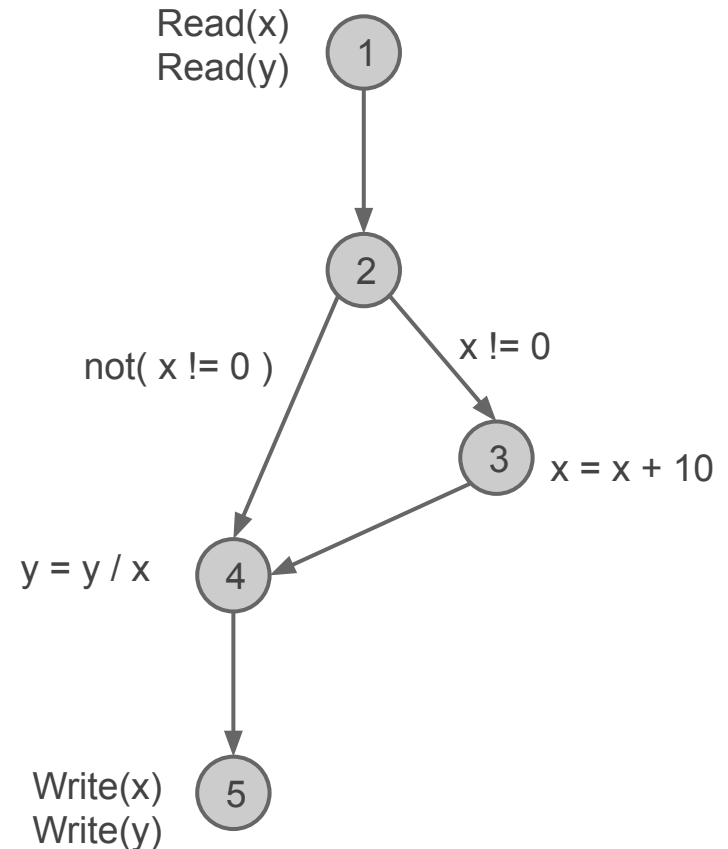
# Testing de Caja Blanca

## Criterios de Cubrimiento

### Criterio de Comandos

Un test  $T$  satisface este criterio ssi cada comando (sentencia) ejecutable del programa es ejecutado al menos una vez.

Con  $T_{CC} = \{(x=20, y=30)\}$  cubro todas las líneas ejecutables del programa. ¿Es un buen test? Si el test da bien, ¿me puedo quedar tranquilo de que no hay errores?



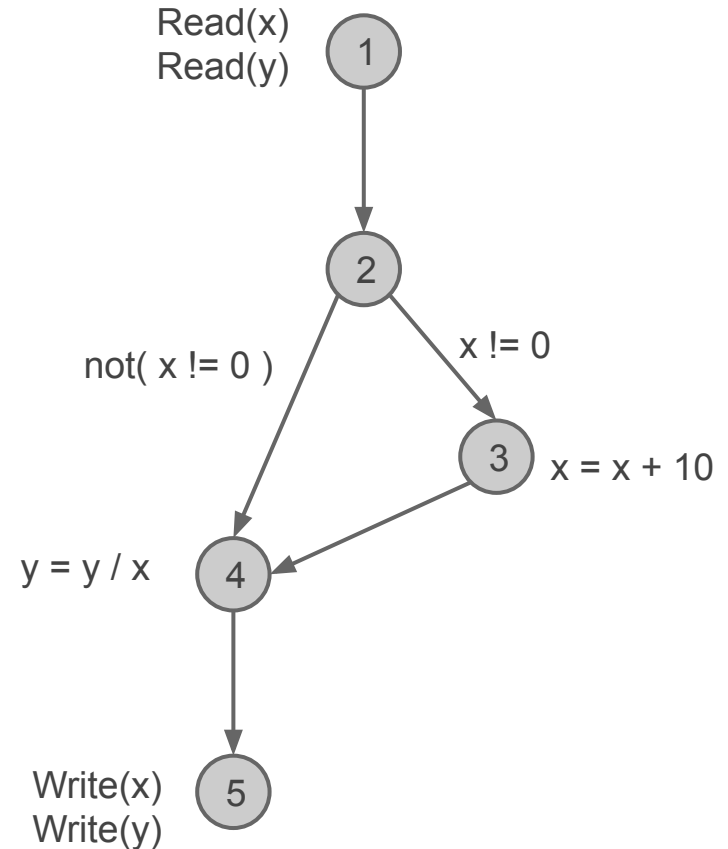
# Testing de Caja Blanca

## Criterios de Cubrimiento

### Criterio de Decisiones

Un test T satisface este criterio ssi cada arco del grafo de flujo de control es visitado al menos una vez.

Visito todos los arcos.





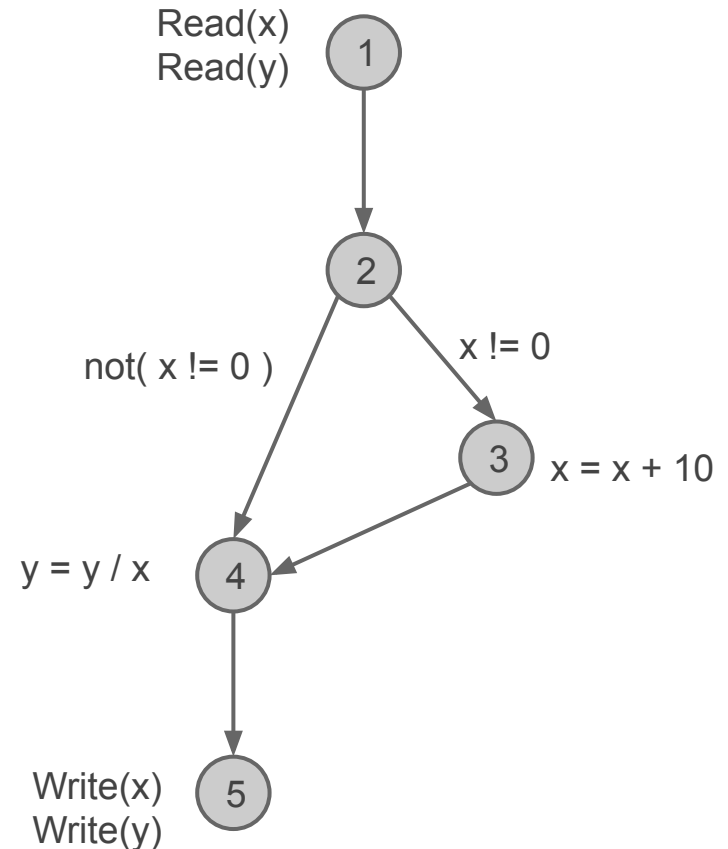
# Testing de Caja Blanca

## Criterios de Cubrimiento

### Criterio de Decisiones

Un test  $T$  satisface este criterio ssi cada arco del grafo de flujo de control es visitado al menos una vez.

El camino  $[1, 2, 3, 4, 5]$  **no** visita todos los arcos del grafo. Por lo que  $T_{CC} = \{(x=20, y=30)\}$  no satisface el Criterio de Decisiones.



# Testing de Caja Blanca

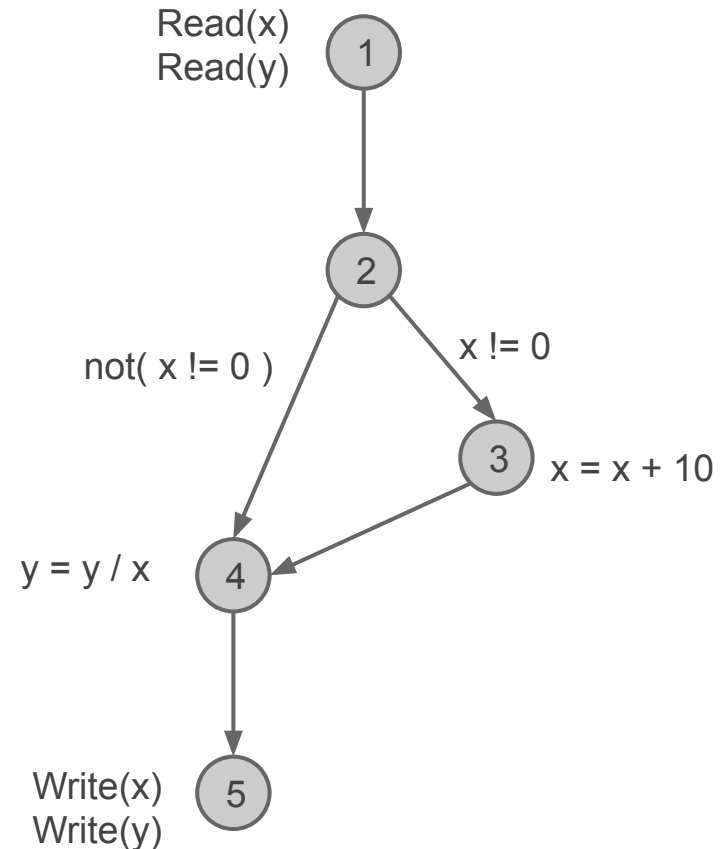
## Criterios de Cubrimiento

### Criterio de Decisiones

Un test  $T$  satisface este criterio ssi cada arco del grafo de flujo de control es visitado al menos una vez.

El camino  $[1, 2, 3, 4, 5]$  **no** visita todos los arcos del grafo. Por lo que  $T_{CC} = \{(x=20, y=30)\}$  no satisface el Criterio de Decisiones.

El conjunto de caminos  $([1, 2, 3, 4, 5], [1, 2, 4, 5])$  **sí** visita todos los arcos. Si puedo encontrar  $(x_1, y_1)$  y  $(x_2, y_2)$  que recorren dichos caminos entonces podría decir que  $T = \{(x_1, y_1), (x_2, y_2)\}$  satisface el Criterio de Decisiones.



# Testing de Caja Blanca

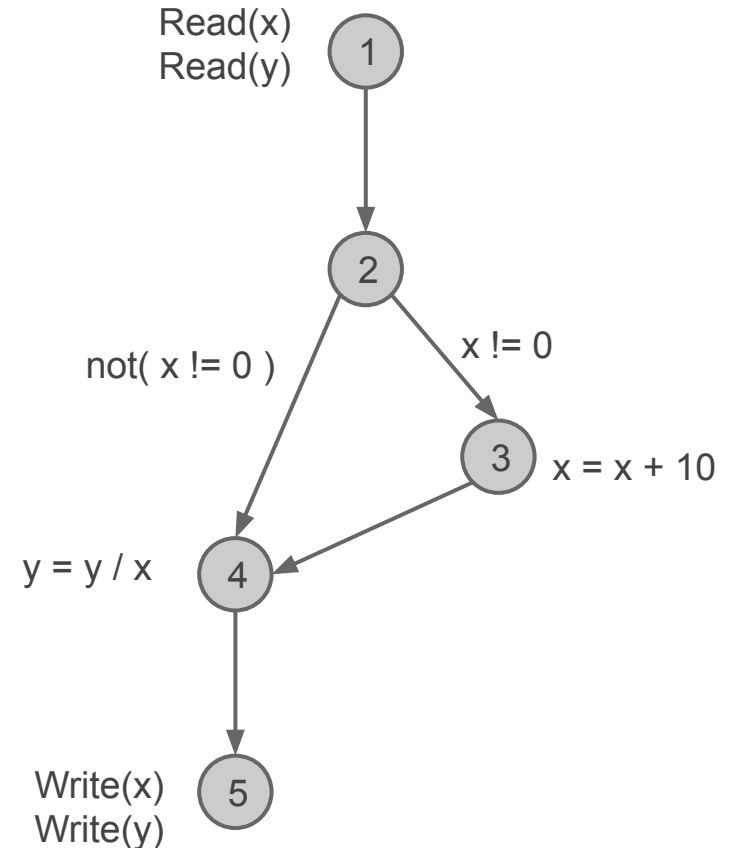
## Criterios de Cubrimiento

### Criterio de Decisiones

Un test  $T$  satisface este criterio ssi cada arco del grafo de flujo de control es visitado al menos una vez.

El conjunto de caminos  $([1, 2, 3, 4, 5], [1, 2, 4, 5])$  **sí** visita todos los arcos. Si puedo encontrar  $(x_1, y_1)$  y  $(x_2, y_2)$  que recorren dichos caminos entonces podría decir que  $T = \{(x_1, y_1), (x_2, y_2)\}$  satisface el Criterio de Decisiones.

$$T_{CD} = \{(x=20, y=30), (x=0, y=30)\}$$



# Testing de Caja Blanca

## Criterios de Cubrimiento

### Criterio de Condiciones y Decisiones

Un test T satisface este criterio ssi cada condición simple (atómica) que aparece en cada decisión del programa es evaluada al menos una vez a TRUE y al menos una vez a FALSE para diferentes datos de test.

Una condición es simple (atómica) cuando no presenta operadores lógicos como AND, OR y NOT. Puede tener sí operadores relacionales como  $>$ ,  $>=$ ,  $<$ ,  $<=$  e  $=$ .

# Testing de Caja Blanca

## Criterios de Cubrimiento

### Criterio de Condiciones y Decisiones

Un test T satisface este criterio ssi cada condición simple (atómica) que aparece en cada decisión del programa es evaluada al menos una vez a TRUE y al menos una vez a FALSE para diferentes datos de test.

```
boolean                                cond1;  
boolean                                cond2;  
...  
if( cond1 AND cond2 ) THEN {  
... }
```

# Testing de Caja Blanca

## Criterios de Cubrimiento

### Criterio de Condiciones y Decisiones

Un test T satisface este criterio ssi cada condición simple (atómica) que aparece en cada decisión del programa es evaluada al menos una vez a TRUE y al menos una vez a FALSE para diferentes datos de test.

```
boolean  
boolean  
...  
if( cond1 AND cond2 ) THEN {  
... }
```

Condiciones Simples		Condición
cond1	cond2	cond1 AND cond2

# Testing de Caja Blanca

## Criterios de Cubrimiento

### Criterio de Condiciones y Decisiones

Un test T satisface este criterio ssi cada condición simple (atómica) que aparece en cada decisión del programa es evaluada al menos una vez a TRUE y al menos una vez a FALSE para diferentes datos de test.

```
boolean  
boolean  
...  
if( cond1 AND cond2 ) THEN {  
... }
```

Condiciones Simples		Condición
cond1	cond2	cond1 AND cond2
TRUE	TRUE	TRUE
FALSE	FALSE	FALSE

# Testing de Caja Blanca

## Criterios de Cubrimiento

### Criterio de Condiciones Múltiples

Un test T satisface este criterio ssi cada condición simple (atómica) que aparece en cada decisión del programa es evaluada a TRUE y a FALSE en todas las combinaciones de la tabla de verdad, para diferentes datos de test.



# Testing de Caja Blanca

## Crterios de Cubrimiento

### Criterio de Condiciones Múltiples

Un test T satisface este criterio ssi cada condición simple (atómica) que aparece en cada decisión del programa es evaluada a TRUE y a FALSE en todas las combinaciones de la tabla de verdad, para diferentes datos de test.

```
boolean  
boolean  
...  
if( cond1 AND cond2 ) THEN {  
... }
```

Condiciones Simples		Condición
cond1	cond2	cond1; cond1 AND cond2; cond2;
TRUE	TRUE	TRUE
FALSE	FALSE	FALSE
TRUE	FALSE	FALSE
FALSE	TRUE	FALSE

# Testing de Caja Blanca

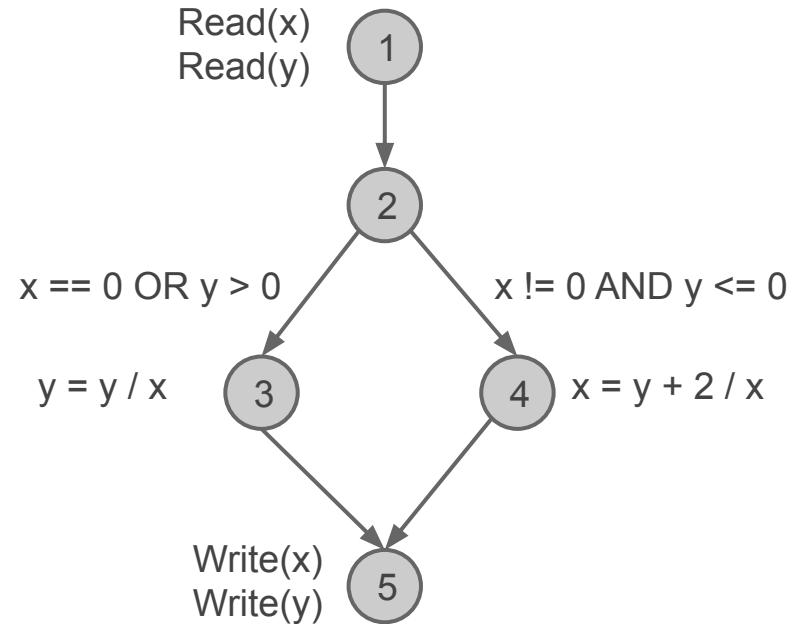
## Criterios de Cubrimiento

```
PROGRAM
VAR x,y:REAL
BEGIN
Read(x);
Read(y);
IF (x == 0 OR y > 0 )
THEN
    y = y / x;
ELSE
    x = y + 2 / x;
Write(x);
Write(y);
END
```

# Testing de Caja Blanca

## Criterios de Cubrimiento

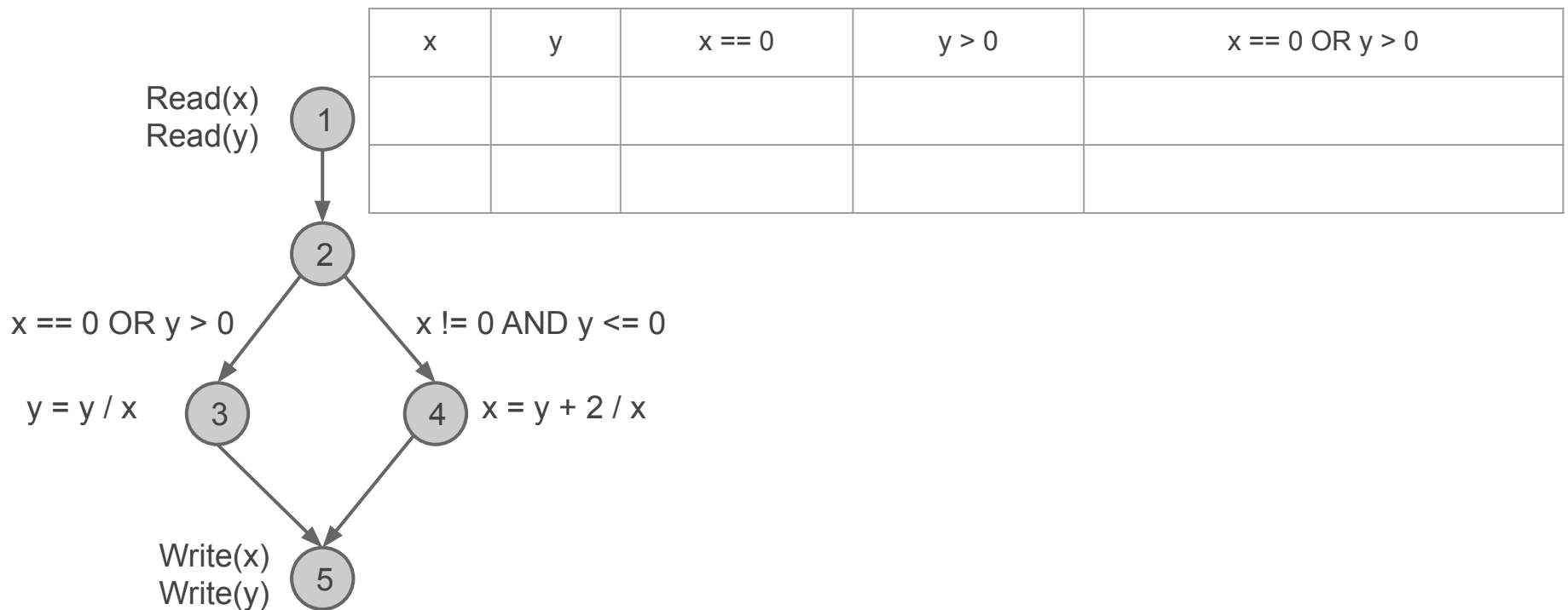
```
PROGRAM
VAR x,y:REAL
BEGIN
Read(x);
Read(y);
IF (x == 0 OR y > 0 )
THEN
    y = y / x;
ELSE
    x = y + 2 / x;
Write(x);
Write(y);
END
```



# Testing de Caja Blanca

## Criterios de Cubrimiento

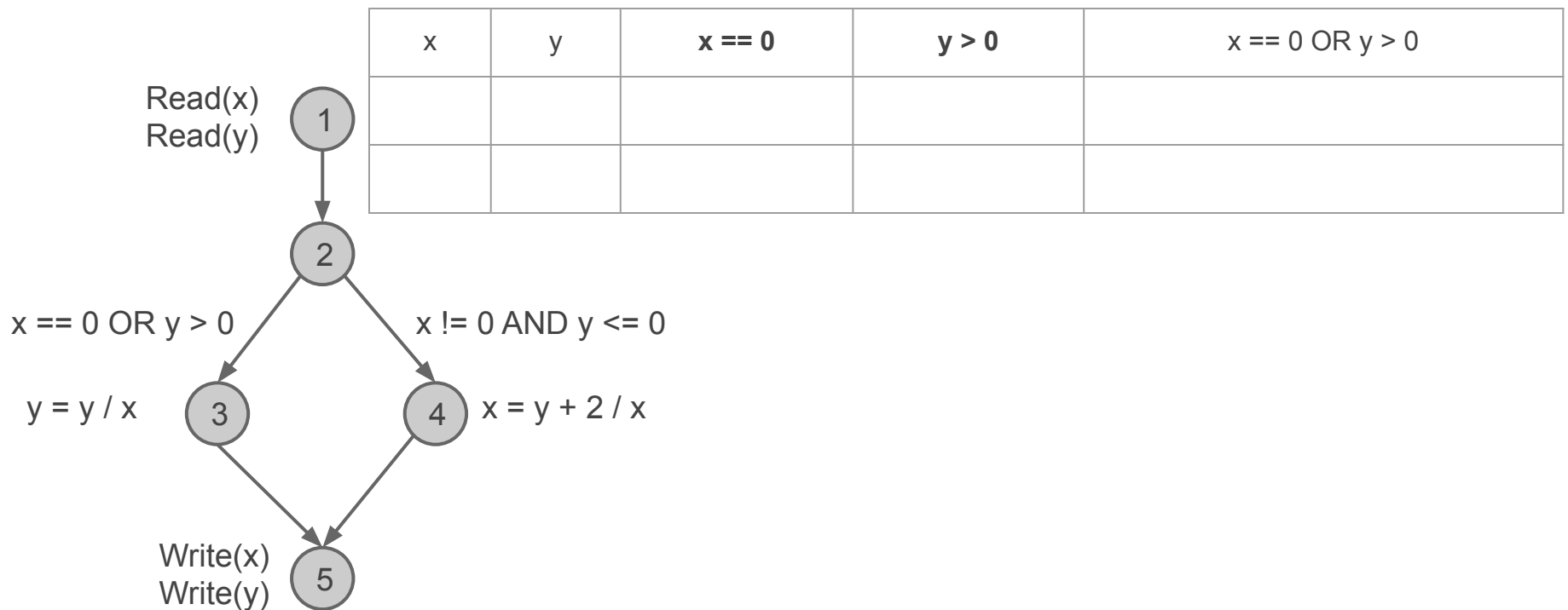
Criterio de Condiciones y Decisiones



# Testing de Caja Blanca

## Criterios de Cubrimiento

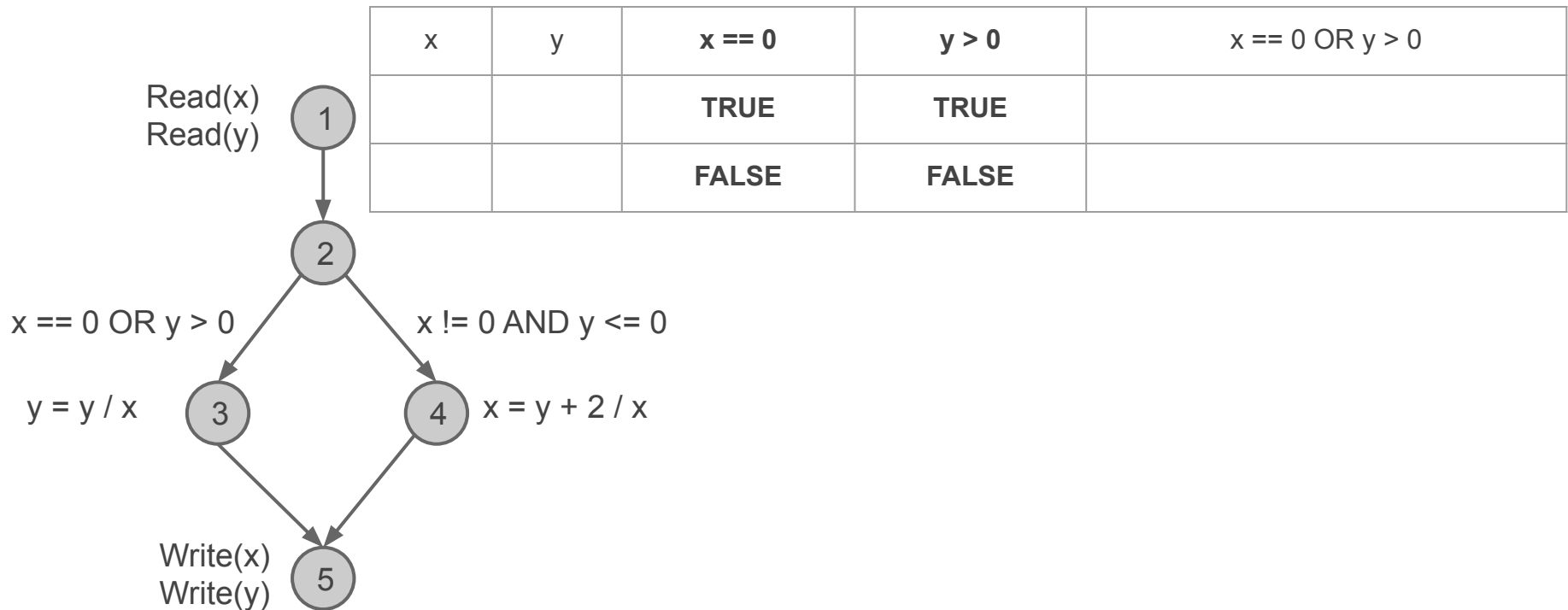
Criterio de Condiciones y Decisiones



# Testing de Caja Blanca

## Criterios de Cubrimiento

Criterio de Condiciones y Decisiones

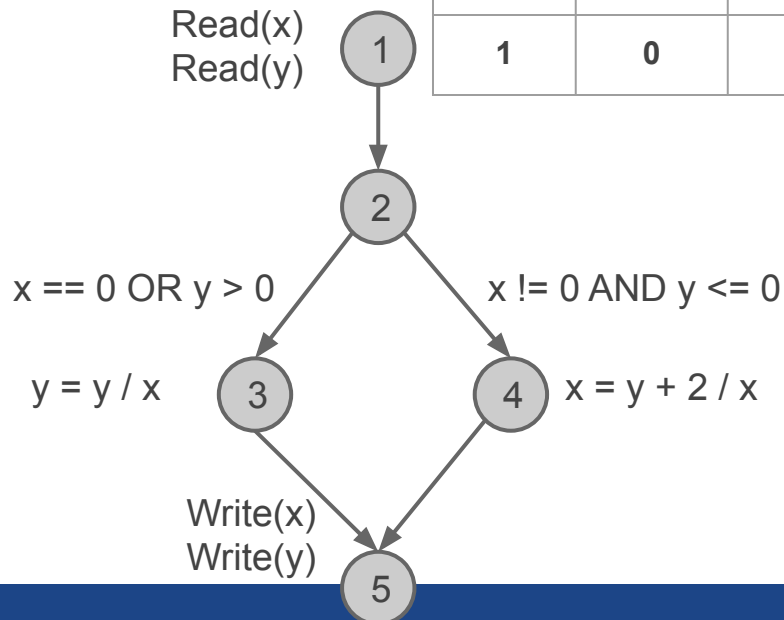


# Testing de Caja Blanca

## Criterios de Cubrimiento

Criterio de Condiciones y Decisiones

x	y	$x == 0$	$y > 0$	$x == 0 \text{ OR } y > 0$
0	1	TRUE	TRUE	
1	0	FALSE	FALSE	

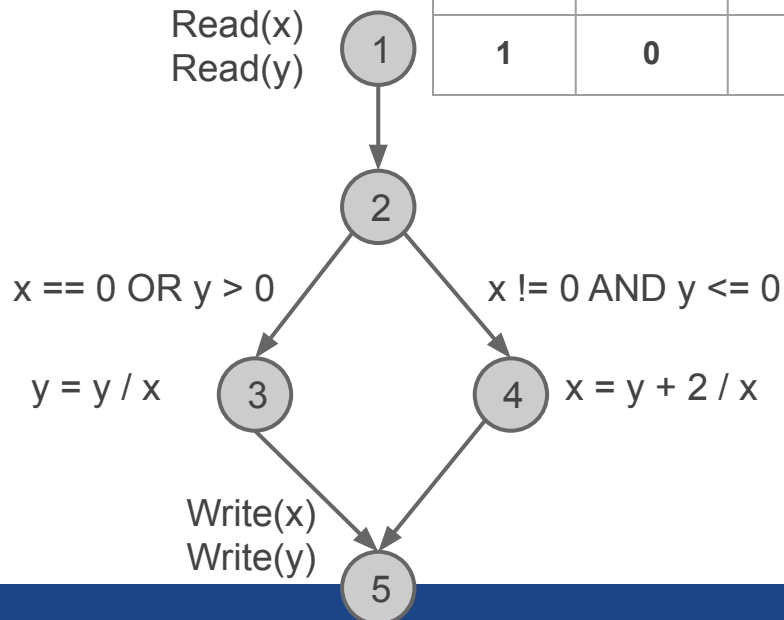


# Testing de Caja Blanca

## Criterios de Cubrimiento

Criterio de Condiciones y Decisiones

x	y	$x == 0$	$y > 0$	$x == 0 \text{ OR } y > 0$
0	1	TRUE	TRUE	TRUE
1	0	FALSE	FALSE	FALSE



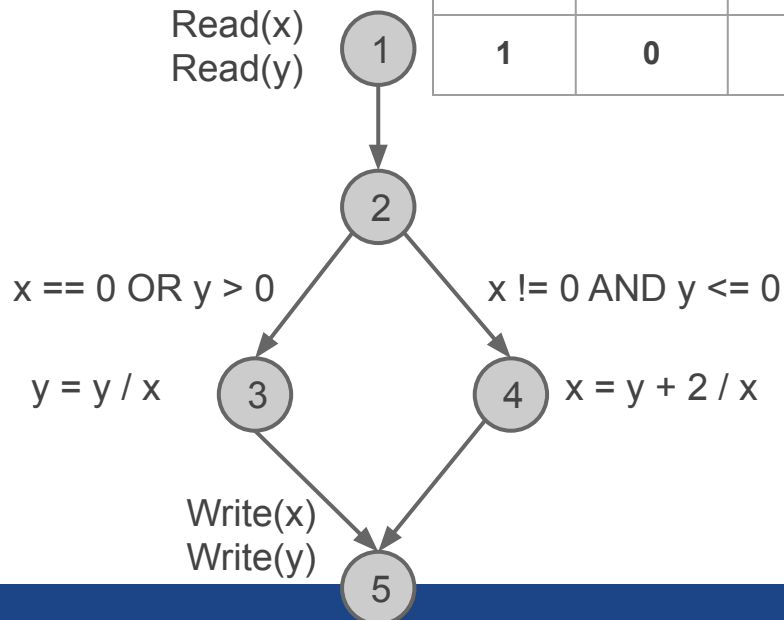


# Testing de Caja Blanca

## Criterios de Cubrimiento

Criterio de Condiciones y Decisiones

x	y	$x == 0$	$y > 0$	$x == 0 \text{ OR } y > 0$
0	1	TRUE	TRUE	TRUE
1	0	FALSE	FALSE	FALSE

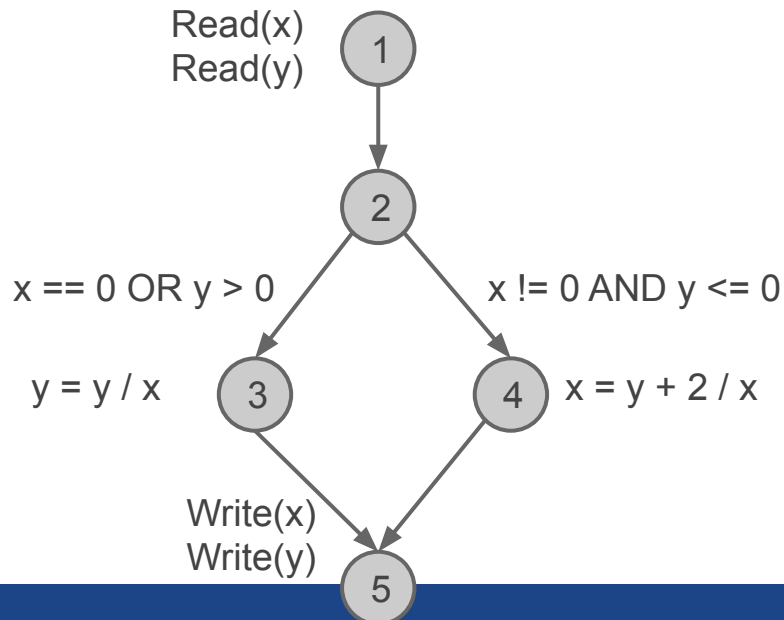


$$T_{\text{CCD}} = \{(x=1, y=0), (x=0, y=1)\}$$

# Testing de Caja Blanca

## Criterios de Cubrimiento

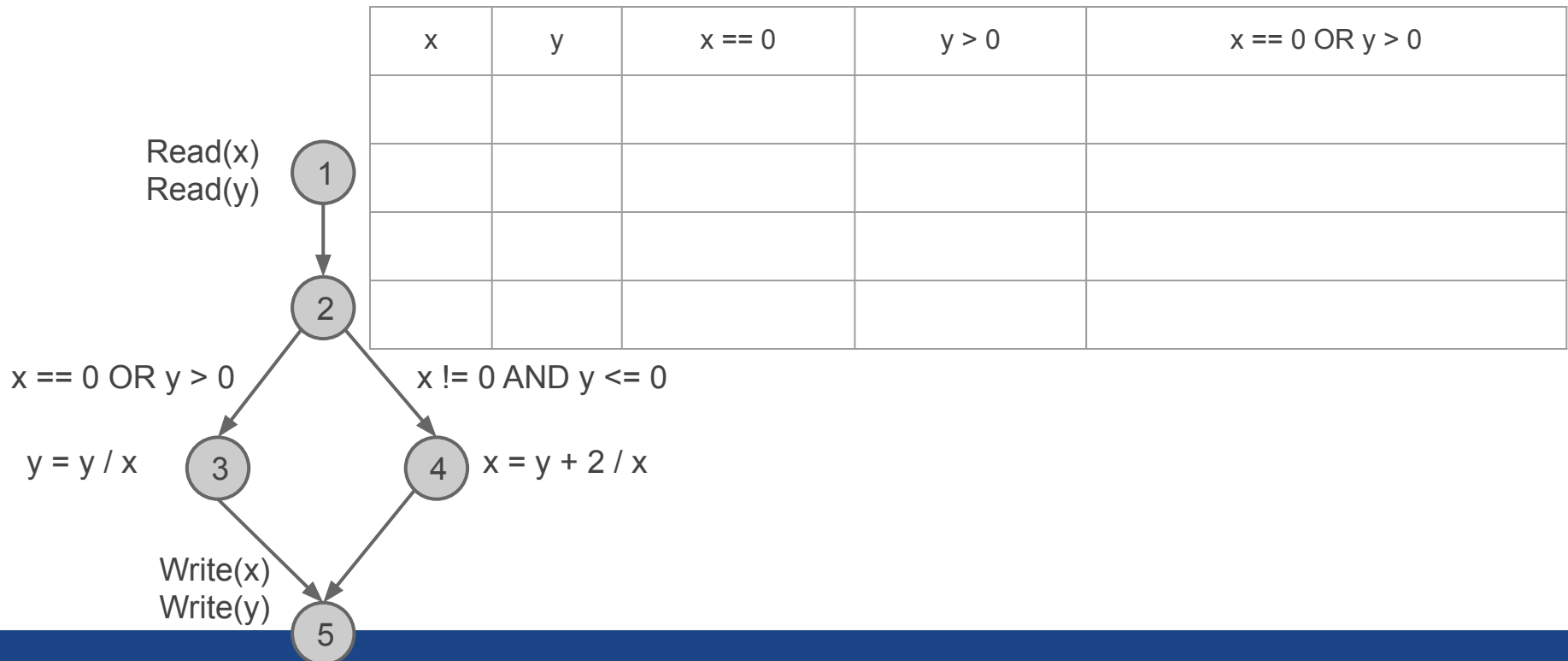
Criterio de Condiciones Múltiples



# Testing de Caja Blanca

## Criterios de Cubrimiento

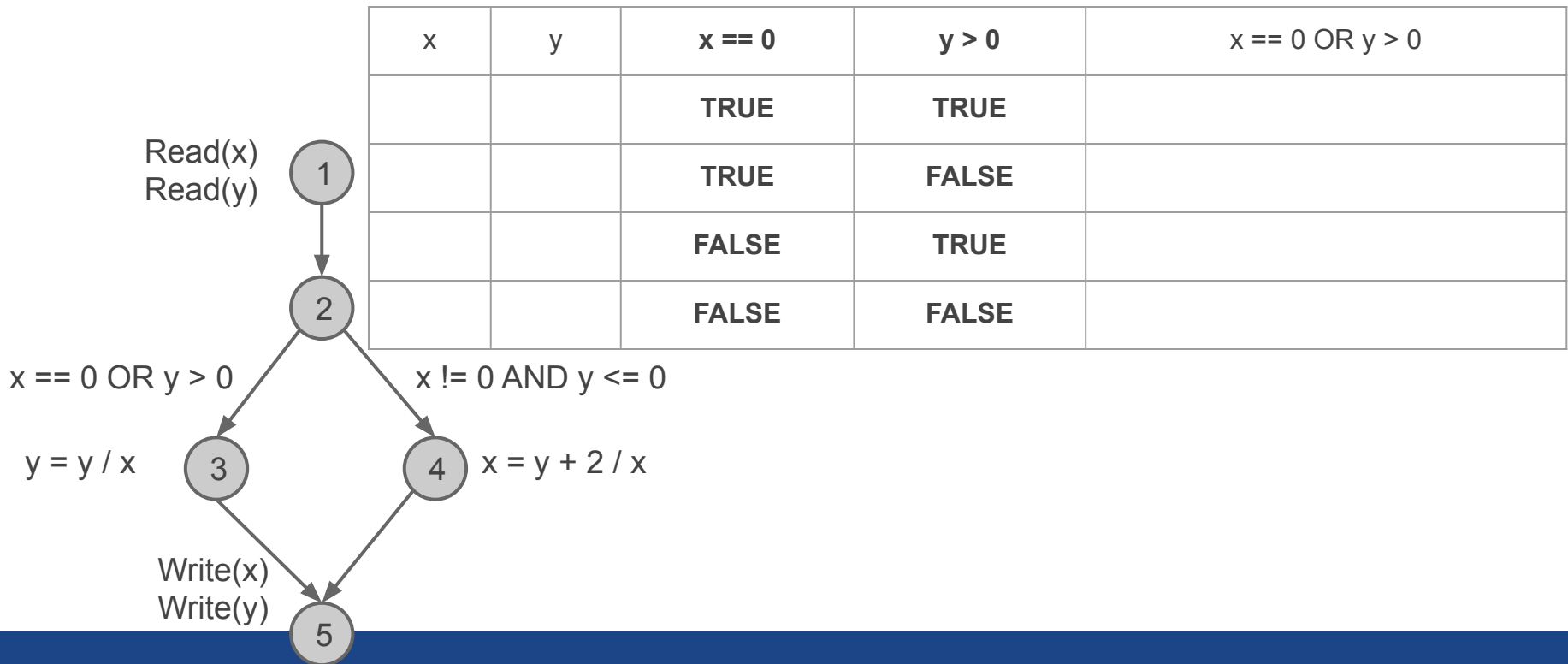
Criterio de Condiciones Múltiples



# Testing de Caja Blanca

## Criterios de Cubrimiento

### Criterio de Condiciones Múltiples

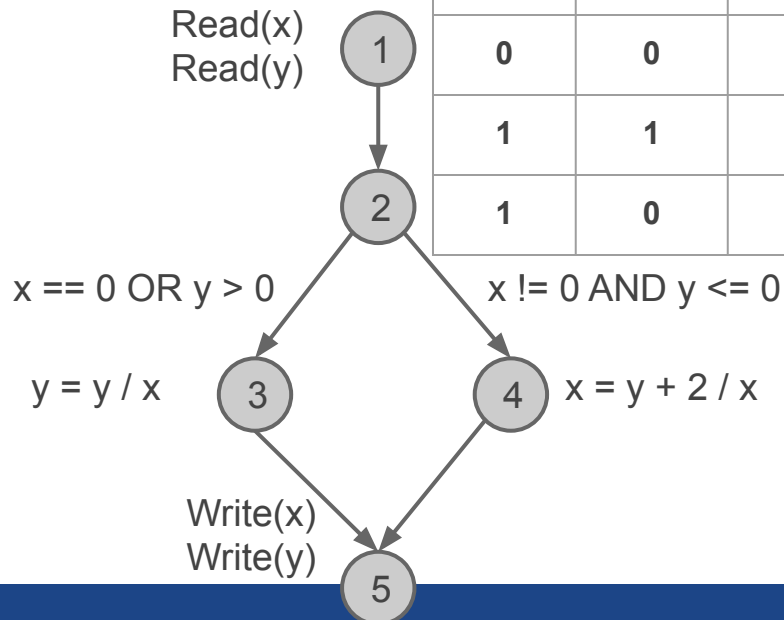


# Testing de Caja Blanca

## Criterios de Cubrimiento

### Criterio de Condiciones Múltiples

x	y	x == 0	y > 0	x == 0 OR y > 0
0	1	TRUE	TRUE	
0	0	TRUE	FALSE	
1	1	FALSE	TRUE	
1	0	FALSE	FALSE	

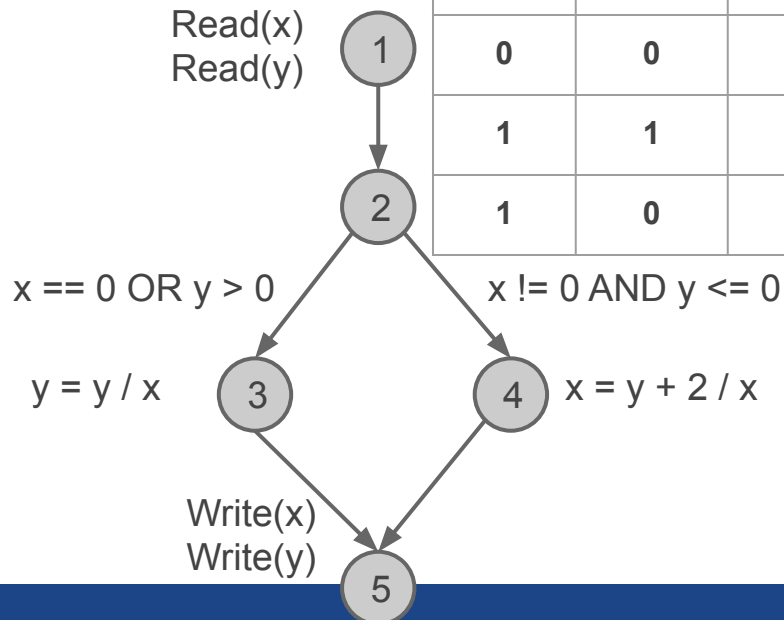


# Testing de Caja Blanca

## Criterios de Cubrimiento

### Criterio de Condiciones Múltiples

x	y	$x == 0$	$y > 0$	$x == 0 \text{ OR } y > 0$
0	1	TRUE	TRUE	TRUE
0	0	TRUE	FALSE	TRUE
1	1	FALSE	TRUE	TRUE
1	0	FALSE	FALSE	FALSE

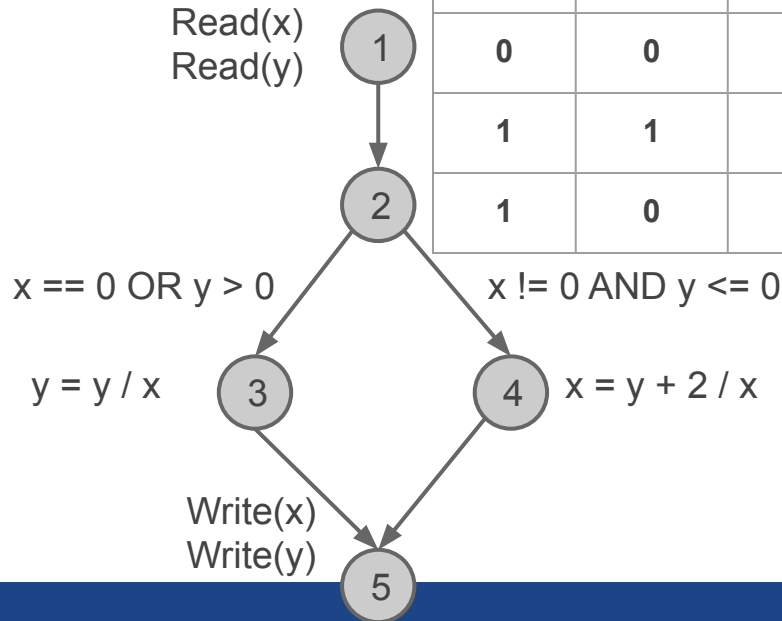


# Testing de Caja Blanca

## Criterios de Cubrimiento

### Criterio de Condiciones Múltiples

x	y	x == 0	y > 0	x == 0 OR y > 0
0	1	TRUE	TRUE	TRUE
0	0	TRUE	FALSE	TRUE
1	1	FALSE	TRUE	TRUE
1	0	FALSE	FALSE	FALSE



$$T_{CCM} = \{(x=1, y=0), (x=0, y=1), (x=1, y=1), (x=0, y=0)\}$$

# Testing de Caja Blanca

## Criterios de Cubrimiento

### Criterio de Caminos

Un test  $T$  satisface este criterio ssi todos los caminos desde el nodo inicial del grafo, al nodo final son atravesados.

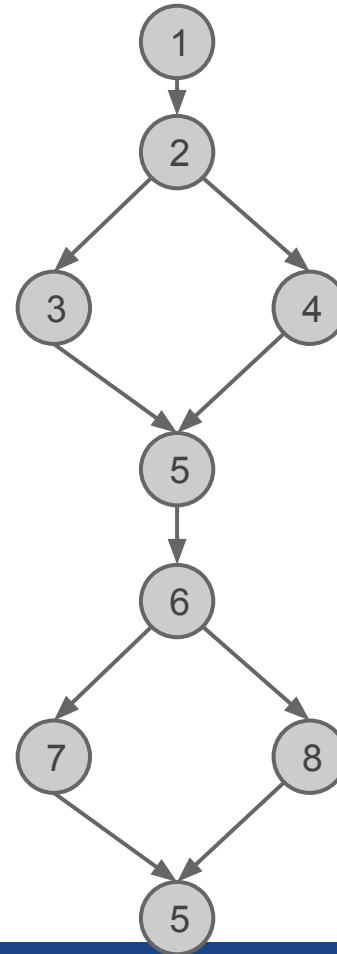


# Testing de Caja Blanca

## Criterios de Cubrimiento

### Criterio de Caminos

Un test T satisface este criterio ssi todos los caminos desde el nodo inicial del grafo, al nodo final son atravesados.



# Testing de Caja Blanca

## Criterios de Cubrimiento

### Criterio de Caminos

Un test T satisface este criterio ssi todos los caminos desde el nodo inicial del grafo, al nodo final son atravesados.

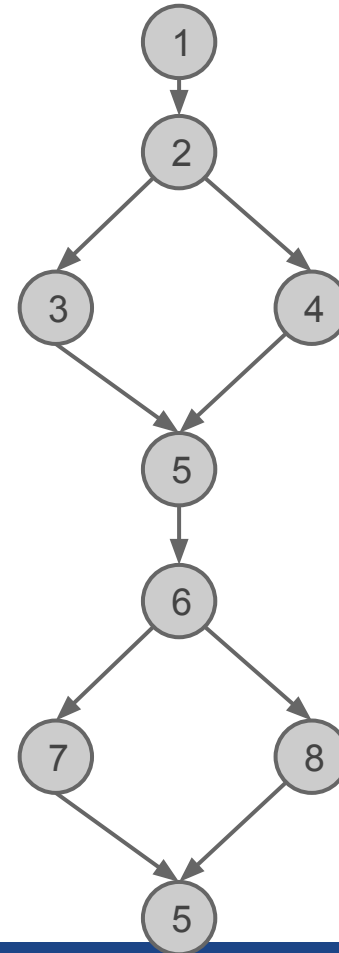
Todos los caminos:

[1, 2, 3, 5, 6, 7, 5]

[1, 2, 3, 5, 6, 8, 5]

[1, 2, 4, 5, 6, 7, 5]

[1, 2, 4, 5, 6, 8, 5]



# Testing de Caja Blanca

## Criterios de Cubrimiento

### Criterio de Bucles

Los bucles son casos especiales. Se deberían cubrir los siguientes casos, cuando sea posible.

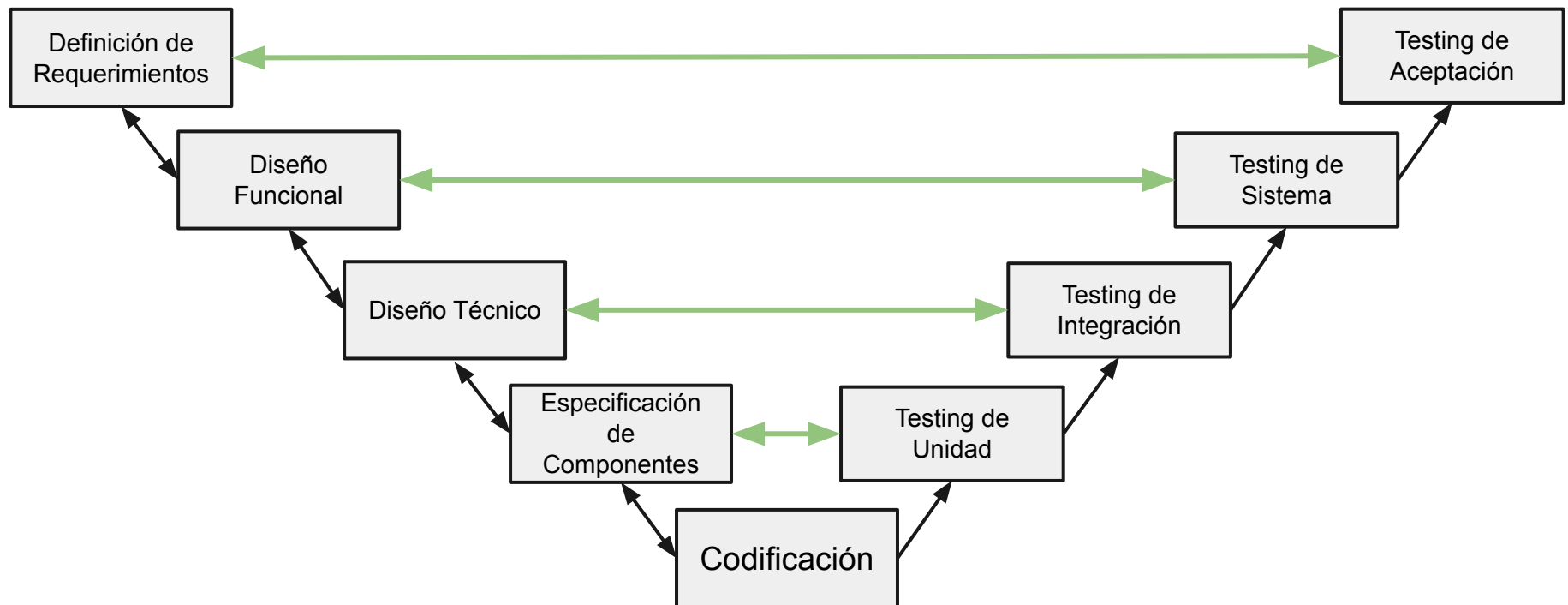
Ejecutar 0 veces el bucle, o el mínimo número de veces.

Ejecutar el caso máximo posible de veces, si es determinable.

Ejecutar un número promedio de veces, si es un valor calculable.

# Testing en el Ciclo de Vida

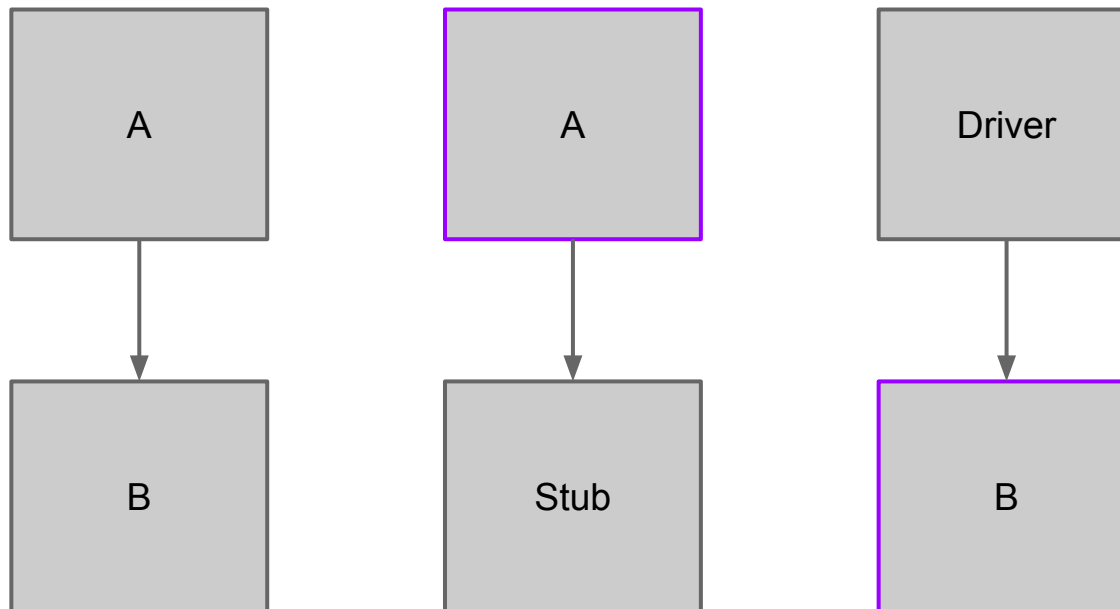
En este modelo, el Testing toma el mismo nivel de importancia que el Diseño y el Desarrollo.



# Testing en el Ciclo de Vida

## Testing de Unidad

Component Testing



# Testing en el Ciclo de Vida

## Testing de Integración

Integration Testing

Segunda etapa del testing. En esta etapa se asume que las componentes (unidades) fueron testeadas y sus defectos fueron resueltos.

# Testing en el Ciclo de Vida

## Testing de Integración

Integration Testing

Se define como “integración” a la conexión entre componentes para formar un sistema o unidad mayor.

# Testing en el Ciclo de Vida

## Testing de Integración

Integration Testing

Se define como “integración” a la conexión entre componentes para formar un sistema o unidad mayor.

El objetivo en esta etapa es detectar fallas en las interfaces e interacciones entre componentes. También se pueden incluir test de integración con sistemas externos.



# Testing en el Ciclo de Vida

## Testing de Integración

Integration Testing

Algunos de los errores que se pueden encontrar son:

Formatos de interfaces erróneos

Errores en el intercambio de información

# Testing en el Ciclo de Vida

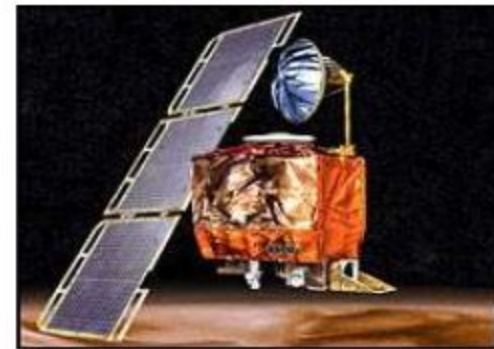
## Testing de Integración

Integration Testing

### **Metric Mishap Caused Loss of NASA Orbiter**

\* By Kathy Sawyer & Robin Lloyd (Washington Post Staff Writer & CNN Interactive Senior Writer)

NASA's Mars Climate Orbiter was lost in space last week because engineers failed to make a simple conversion from English units to metric, an embarrassing lapse that sent the \$125 million craft fatally close to the Martian surface.



Scientists do not yet know what caused the Mars Orbiter to lose contact with Earth. (AP)

[http://mytoe.org/docs/Mars\\_closereading\\_art.pdf](http://mytoe.org/docs/Mars_closereading_art.pdf)

# Testing en el Ciclo de Vida

## Testing de Integración

Integration Testing

Se define como “integración” a la conexión entre componentes para formar un sistema o unidad mayor.

El objetivo en esta etapa es detectar fallas en las interfaces e interacciones entre componentes.

# Testing en el Ciclo de Vida

## Testing de Integración

Integration Testing

No todas las unidades se terminan al mismo tiempo, esto obliga a manejar diferentes tiempos de testing. El scheduling del desarrollo de componentes es un input importante para la planificación del testing de integración.

# Testing en el Ciclo de Vida

## Testing Isolating Code Under Test with Microsoft Fakes

Integration Testing

Visual Studio 2013 | [Other Versions](#) ▾

### Stubs .NET

Stubs is a lightweight classes, type-safe Stubs are part of

Stubs is a lightweight may be used on minimal overhead rewriter and enc

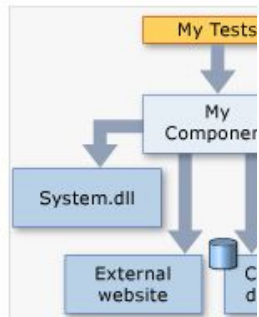


[Download](#)  
[Read more](#)

Microsoft Fakes help you isolate the code you are testing by replacing other parts of the application with *stubs* or *shims*. These are s there and not somewhere else. Stubs and shims also let you test your code even if other parts of your application are not working ye

Fakes come in two flavors

- A *stub* replaces a c classes that are de
- A *shim* modifies th



Deployed component dep are incomplete or exhibit

### Moles - Isolation framework for .NET

The Fakes Framework in Visual Studio 2012 is the next generation of Moles & Stubs. Fakes is different from Moles, however, so moving from Moles to Fakes will require some modifications to your code. The Moles framework will not be supported in Visual Studio 2012.

variable behavior.

work under the control of the tests.

# Testing en el Ciclo de Vida

## Testing de Integración

Integration Testing

Existen diferentes estrategias de integración:

Top-Down, Bottom-Up, Ad-hoc, Backbone, Big-Bang.

# Testing en el Ciclo de Vida

## Testing de Integración

Integration Testing

Existen diferentes estrategias de integración:

Top-Down, Bottom-Up, Ad-hoc, Backbone, Big-Bang.

No requiere Test Drivers / Requiere Stubs

# Testing en el Ciclo de Vida

## Testing de Integración

Integration Testing

Existen diferentes estrategias de integración:

Top-Down, Bottom-Up, Ad-hoc, Backbone, Big-Bang.

No requiere Stubs / Requiere Test Drivers



# Testing en el Ciclo de Vida

## Testing de Integración

Integration Testing

Existen diferentes estrategias de integración:

Top-Down, Bottom-Up, Ad-hoc, Backbone, Big-Bang.

Se ahorra tiempo / Requiere Stubs y Drivers

# Testing en el Ciclo de Vida

## Testing de Integración

Integration Testing

Existen diferentes estrategias de integración:

Top-Down, Bottom-Up, Ad-hoc, **Backbone**, Big-Bang.

El orden no es importante / **Buscar el Backbone**

# Testing en el Ciclo de Vida

## Testing de Integración

Integration Testing

Existen diferentes estrategias de integración:

Top-Down, Bottom-Up, Ad-hoc, Backbone, Big-Bang.



# Verificación y Validación de Software

El objetivo del testing es encontrar errores, diferencias entre lo observado y lo esperado.

# Verificación y Validación de Software

El objetivo del testing es encontrar errores, diferencias entre lo observado y lo esperado.

Si consideramos que un conjunto de test  $T$  es adecuado y un programa  $P$  pasa  $T$ , entonces podemos decir que el programa está libre de errores\*.

# Verificación y Validación de Software

Si un programa tiene errores, entonces el test debería señalarlos.

# Verificación y Validación de Software

Entonces...

# Verificación y Validación de Software

Entonces...

Sea  $P$  un programa



# Verificación y Validación de Software

Entonces...

Sea  $P$  un programa, para este ejemplo  $P$  es un programa que determina si un número es par o no.

```
1
2 public class Main {
3
4     public boolean esPar(int x) {
5         return (x % 2 == 0);
6     }
7
8 }
9
```

# Verificación y Validación de Software

Entonces...

Sea  $P$  un programa, para este ejemplo  $P$  es un programa que determina si un número es par o no.

Sea  $T$  un conjunto de test para  $P$ . En particular supongamos que  $T$  adecuado y que  $P$  pasa  $T$ .

$P$  no tiene errores.

# Verificación y Validación de Software

Entonces...

$$T = T^{SI} \cup T^{NO}$$

$$T^{SI} = \{ 6, 12, 24 \}$$

$$T^{NO} = \{ 1, 7, 11 \}$$

```
1
2 public class Main {
3
4     public boolean esPar(int x) {
5         return (x % 2 == 0);
6     }
7
8 }
9
```

# Verificación y Validación de Software

Entonces...

$$T = T^{SI} \cup T^{NO}$$

$$T^{SI} = \{ 6, 12, 24 \}$$

$$T^{NO} = \{ 1, 7, 11 \}$$

*T* nos asegura que *P*  
no tiene errores

```
1  
2 public class Main {  
3  
4     public boolean esPar(int x) {  
5         return (x % 2 == 0);  
6     }  
7  
8 }  
9
```

# Verificación y Validación de Software

## ENTONCES...

Si  $T$  es tan bueno, si yo ingresara un error en  $P$  a propósito,  $T$  debería ser capaz de detectarlo.

$$T = T^{SI} \cup T^{NO}$$

$$T^{SI} = \{ 6, 12, 24 \}$$

$$T^{NO} = \{ 1, 7, 11 \}$$

```
1  
2 public class Main {  
3  
4     public boolean esPar(int x) {  
5         return (x % 2 == 0);  
6     }  
7  
8 }  
9
```

# Verificación y Validación de Software

## ENTONCES...

Si  $T$  es tan bueno, si yo ingresara un error en  $P$  a propósito,  $T$  debería ser capaz de detectarlo.

$$T = T^{SI} \cup T^{NO}$$

$$T^{SI} = \{ 6, 12, 24 \}$$

$$T^{NO} = \{ 1, 7, 11 \}$$

```
1  
2 public class Main {  
3  
4     public boolean error(int x) {  
5         return (x % 2 == 0);  
6     }  
7  
8 }  
9
```

# Verificación y Validación de Software

## ENTONCES...

Sea  $P^1$  el resultado de introducir un cambio sintáctico en  $P$ .  
Introduce un error, por lo tanto T debería detectarlo.

$$T = T^{SI} \cup T^{NO}$$

$$T^{SI} = \{ 6, 12, 24 \}$$

$$T^{NO} = \{ 1, 7, 11 \}$$

```
1
2 public class Main {
3
4     public boolean esPar(int x) {
5         return (x % 3 == 0);
6     }
7
8 }
9
```

# Verificación y Validación de Software

## ENTONCES...

Sea  $P^1$  el resultado de introducir un cambio sintáctico en  $P$ .  
Introduce un error, por lo tanto T debería detectarlo.

T no reconoce el error

$$T = T^{SI} \cup T^{NO}$$

$$T^{SI} = \{ 6, 12, 24 \}$$

$$T^{NO} = \{ 1, 7, 11 \}$$

```
1  
2 public class Main {  
3  
4     public boolean esPar(int x) {  
5         return (x % 3 == 0);  
6     }  
7  
8 }  
9
```



# Verificación y Validación de Software

## ENTONCES...

Sea  $P^1$  el resultado de introducir un cambio sintáctico en  $P$ .

Introduce un error.

Introduce el error

$T = T^{SI} \cup T^{NO}$

$T^{SI} = \{6, 1\}$

$T^{NO} = \{1, 7\}$



```
5  
6 }  
7  
8 }  
9  
return (x % 5 == 0);  
(int x) {
```

# Verificación y Validación de Software

## ENTONCES...

Esta experiencia me sirve para ver que  $T$  no es un buen conjunto de test y además me sirve para mejorar  $T$ . Si analizo el error que introduje, puedo mejorar los casos de test.

$$T = T^{SI} \cup T^{NO}$$

$$T^{SI} = \{ 6, 12, 24 \}$$

$$T^{NO} = \{ 1, 7, 11 \}$$

```
1  
2 public class Main {  
3  
4     public boolean esPar(int x) {  
5         return (x % 3 == 0);  
6     }  
7  
8 }  
9
```

# Verificación y Validación de Software

## ENTONCES...

Esta experiencia me sirve para ver que  $T$  no es un buen conjunto de test y además me sirve para mejorar  $T$ . Si analizo el error que introduje, puedo mejorar los casos de test.

$$D = D^{SI} \cup D^{NO}$$

$$D^{SI} = \{ 6, 12, 24, 2 \}$$

$$D^{NO} = \{ 1, 7, 11, 9 \}$$

```
1
2 public class Main {
3
4     public boolean esPar(int x) {
5         return (x % 3 == 0);
6     }
7
8 }
9
```

# Verificación y Validación de Software

## ENTONCES...

El nuevo conjunto de test  $D$  reconoce que  $P^1$  tiene errores, por lo que es un mejor testeo que  $T$ .

$$D = D^{SI} \cup D^{NO}$$

$$D^{SI} = \{ 6, 12, 24, 2 \}$$

$$D^{NO} = \{ 1, 7, 11, 9 \}$$

```
1  
2 public class Main {  
3  
4     public boolean esPar(int x) {  
5         return (x % 3 == 0);  
6     }  
7  
8 }  
9
```

# Verificación y Validación de Software

## Testing por Mutación

Mutation Testing

# Verificación y Validación de Software

## Testing por Mutación

Mutation Testing

Una mutación es un pequeño cambio en un programa. Estos cambios buscan reflejar los pequeños errores que se producen al programar.

# Verificación y Validación de Software

## Testing por Mutación

Mutation Testing

El Testing por Mutación es una técnica de caja blanca que permite evaluar/mejorar la calidad de nuestra suite de test.

# Verificación y Validación de Software

## Testing por Mutación

Mutation Testing

Dado un programa  $P$  y una suite de test  $T$  el proceso es:

Aplicamos mutaciones sobre  $P$  para obtener una secuencia  $P^1 \dots P^n$  de mutantes de  $P$ .



# Verificación y Validación de Software

## Testing por Mutación

Mutation Testing

Dado un programa  $P$  y una suite de test  $T$  el proceso es:

Aplicamos mutaciones sobre  $P$  para obtener una secuencia  $P^1 \dots P^n$  de mutantes de  $P$ .

Ejecutamos  $T$  sobre cada mutante. Se dice que  $T$  mata al mutante  $P^j$  si  $T$  detecta un error en  $P^j$ .

# Verificación y Validación de Software

## Testing por Mutación

Mutation Testing

Dado un programa  $P$  y una suite de test  $T$  el proceso es:

Aplicamos mutaciones sobre  $P$  para obtener una secuencia  $P^1 \dots P^n$  de mutantes de  $P$ .

Ejecutamos  $T$  sobre cada mutante. Se dice que  $T$  mata al mutante  $P^j$  si  $T$  detecta un error en  $P^j$ .

Si  $T$  mata a  $k$  mutantes entonces se dice que la eficacia (adequacy) de  $T$  es  $k/n$ .  $T$  es **mutation adequacy** si  $k = n$ .

# Verificación y Validación de Software

## Testing por Mutación

Mutation Testing

Para poder aprovechar esta técnica debemos poder sistematizar el proceso de mutación. Esto implica definir reglas para la aplicación de mutaciones.

# Verificación y Validación de Software

## Testing por Mutación

Mutation Testing

Para poder aprovechar esta técnica debemos poder sistematizar el proceso de mutación. Esto implica definir reglas para la aplicación de mutaciones.

Obviamente que, para una simple línea de código podemos pensar en muchas mutaciones. Esto llevado a un programa complejo implica que la cantidad de mutantes, si queremos mutar todo lo posible, es astronómica.

# Verificación y Validación de Software

## Testing por Mutación

Mutation Testing

Mutación de Valor

Mutación de Decisión

Mutación de Sentencia

# Verificación y Validación de Software

## Testing por Mutación

Mutation Testing

Mutación de Valor

Modificamos el valor de una constante

Mutación de Decisión

Mutación de Sentencia

# Verificación y Validación de Software

## Testing por Mutación

Mutation Testing

Mutación de Valor

Mutación de Decisión

Modificamos condiciones en decisiones

Mutación de Sentencia

# Verificación y Validación de Software

## Testing por Mutación

Mutation Testing

Mutación de Valor

Mutación de Decisión

Mutación de Sentencia

Eliminamos sentencias, las duplicamos o intercambiamos



# Verificación y Validación de Software

## Testing por Mutación

Mutation Testing

### Mutación de Valor

El objetivo es reflejar errores en el razonamiento de la solución por parte del programador.

*Recorrer un arreglo a partir de 1 en lugar de 0.*

*El caso base de factorial en 1 retorna 0.*

Cubrimiento: Modificar cada constante del programa por lo menos una vez (o dos).

# Verificación y Validación de Software

## Testing por Mutación

Mutation Testing

### Mutación de Decisión

El objetivo es reflejar errores en el razonamiento de las decisiones en la solución, por parte del programador.

*Controlar un if por < cuando debería ser <=*

*Ciclar sobre un && cuando debería ser ||*

Cubrimiento: Modificar cada condición del programa una vez. Modificar una vez cada operador lógico y de relación una vez.

# Verificación y Validación de Software

## Testing por Mutación

Mutation Testing

### Mutación de Sentencia

El objetivo es reflejar errores en la edición del código,  
muchas veces por copiar y pegar

*Borrar, duplicar e intercambiar una sentencia*

Cubrimiento: Aplicar cada una de estas acciones en cada una de las sentencias del programa.

# Verificación y Validación de Software

## Testing por Mutación

Mutation Testing

Esta técnica es práctica sólo si podemos automatizar el proceso de mutación y testeo.

# Verificación y Validación de Software

## Testing por Mutación

Mutation Testing

IEEE TRANSACTIONS ON SOFTWARE ENGINEERING

1

## An Analysis and Survey of the Development of Mutation Testing

Yue Jia *Student Member, IEEE*, and Mark Harman *Member, IEEE*

**Abstract**—Mutation Testing is a fault-based software testing technique that has been widely studied for over three decades. The literature on Mutation Testing has contributed a set of approaches, tools, developments and empirical results. This paper provides a comprehensive analysis and survey of Mutation Testing. The paper also presents the results of several development trend analyses. These analyses provide evidence that Mutation Testing techniques and tools are reaching a state of maturity and applicability, while the topic of Mutation Testing itself is the subject of increasing interest.

**Index Terms**—mutation testing, survey

Besides using Mutation Testing at the software implementation level, it has also been applied at the design level to test the specifications or models of a program. For example, at the design level Mutation Testing has been applied to Finite State Machines [20], [28], [88], [111], State charts [95], [231], [260], Estelle Specifications [222], [223], Petri Nets [86], Network protocols [124], [202], [216], [238], Security Policies [139], [154], [165], [166], [201] and Web Services [140], [142], [143], [193], [245], [259].

Mutation Testing has been increasingly and widely studied since it was first proposed in the 1970s. There has been much research work on the various kinds of techniques seeking to

# Verificación y Validación de Software

## Testing por Mutación

Mutation Testing

### MuJava : An Automated Class Mutation System

Yu-Seung Ma<sup>1\*</sup>, Jeff Offutt<sup>2†</sup>, and Yong Rae Kwon<sup>1</sup>

<sup>1</sup>Division of Computer Science  
Department of Electrical Engineering and Computer Science  
Korea Advanced Institute of Science and Technology, Korea  
{*ysma, kwon*}@salmosa.kaist.ac.kr

<sup>2</sup>Department of Information and Software Engineering  
George Mason University  
*ofut@ise.gmu.edu*

#### Abstract

Several module and class testing techniques have been applied to object-oriented programs, but researchers have only recently begun developing test criteria that evaluate the use of key OO features such as inheritance, polymorphism, and encapsulation. Mutation testing is a powerful testing technique for generating software tests and evaluating the quality of software. However, the cost of mutation testing has traditionally been so high it cannot be applied without full automated tool support.

This paper presents a method to reduce the execution cost of mutation testing for OO programs by



# Verificación y Validación de Software

## Testing por Mutación

Mutation Testing

Language Feature	Operator	Description
Access Control	AMC	Access modifier change
Inheritance	IHD	Hiding variable deletion
	IHI	Hiding variable insertion
	IOD	Overriding method deletion
	IOP	overriding method calling position change
	IOR	Overriding method rename
	ISK	<i>super</i> keyword deletion
	IPC	Explicit call of a parent's constructor deletion
Polymorphism	PNC	<i>new</i> method call with child class type
	PMD	Instance variable declaration with parent class type
	PPD	Parameter variable declaration with child class type
	PRV	Reference assignment with other comparable type
Overloading	OMR	Overloading method contents change
	OMD	Overloading method deletion
	OAQ	Argument order change
	OAN	Argument number change
Java-Specific Features	JTD	<i>this</i> keyword deletion
	JSC	<i>static</i> modifier change
	JID	Member variable initialization deletion
	JDC	Java-supported default constructor creation
Common Programming Mistakes	EOA	Reference assignment and content assignment replacement
	EOC	Reference comparison and content comparison replacement
	EAM	Accessor method change
	EMM	Modifier method change

# Verificación y Validación de Software

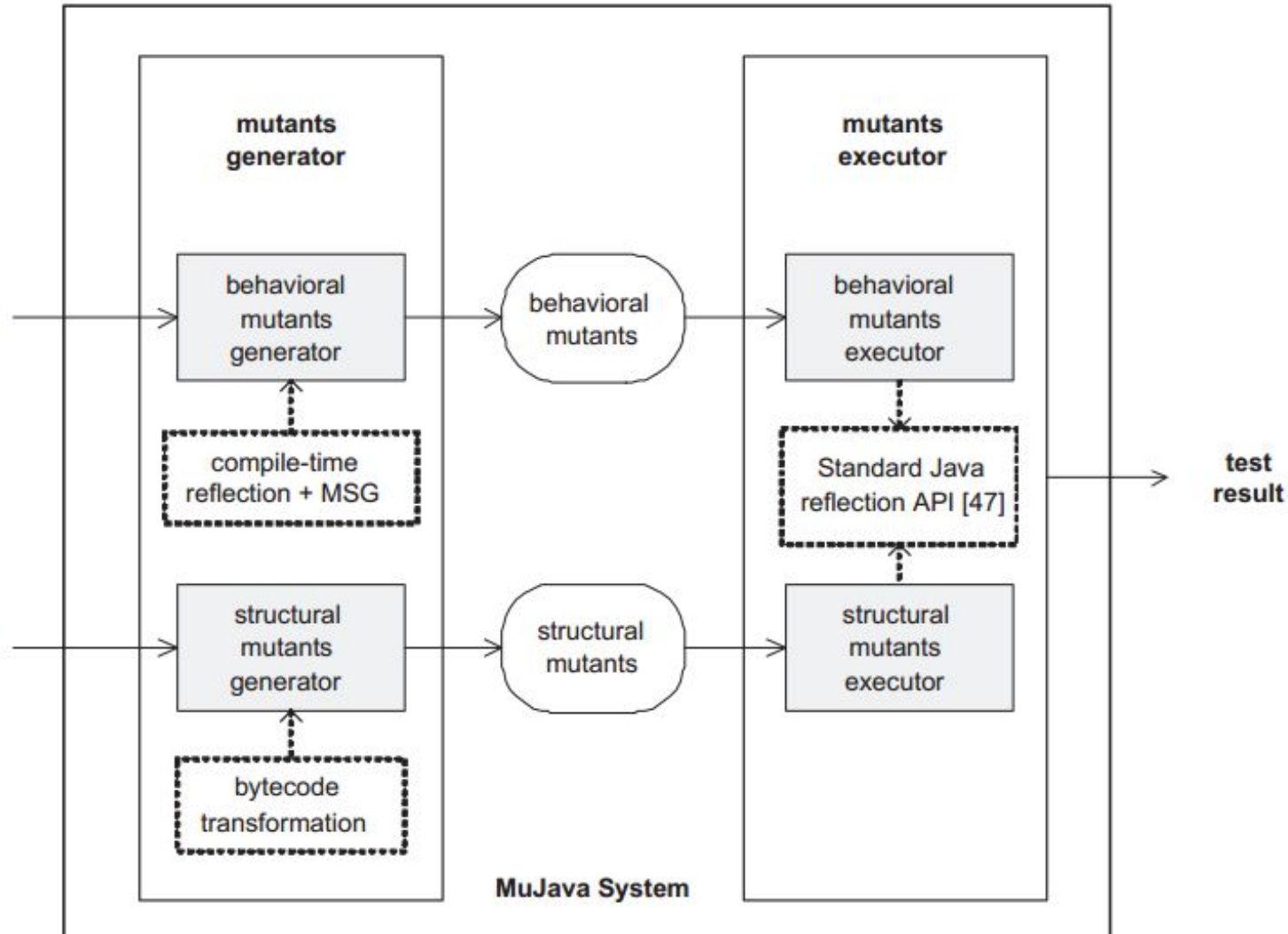
## Testing por Mutación

Mutation Testing

original program  
( source code )

compile

original program  
( byte code )





# Verificación y Validación de Software

## Testing por Mutación

Mutation Testing



The screenshot shows a web browser window with the address bar containing the URL <https://cs.gmu.edu/~offutt/mujava/#Links>. The page title is **μJava Home Page**, with a subtitle **Version 4, June 2013** and a note: **JUnit tests, Java 1.6 features (including generics), and fault fixes. (minor bug fixes, May 2014)**. The main content includes a paragraph about the April 2015 release on GitHub under the Apache license, a command line interface, and the Version 3 page. A section describes μJava as a mutation system for Java programs, mentioning its collaboration with KAIST and the University of Florida. A list of links is provided: 

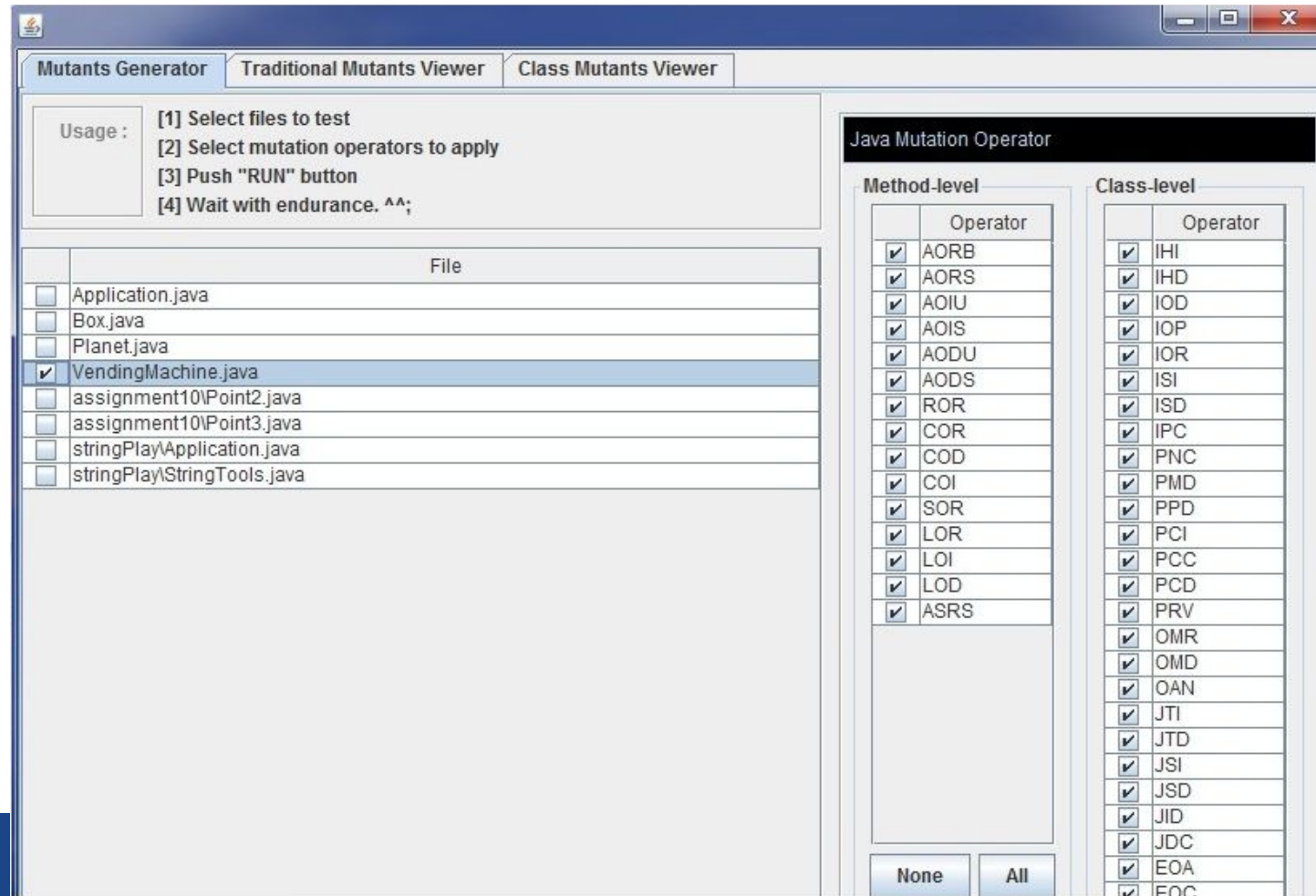
- [An overview](#) of object-oriented Java and μJava.
- [Links](#) to downloadable files.
- [Detailed description](#) of how to install, set up, and run μJava.
- [References](#) to published papers.

A final note mentions a March 2007 modification by Laurie Williams and Ben Smith of NC State to be an Eclipse plugin.

# Verificación y Validación de Software

## Testing por Mutación

Mutation Testing



# Verificación y Validación de Software

Usage : [1] Select files to test  
[2] Select mutation operators to apply  
[3] Push "RUN" button  
[4] Wait with endurance. ^^;

File
<input checked="" type="checkbox"/> Main.java

Log level 1

None All

Generate

### Java Mutation Operator

Method-level		Class-level	
	Operator		Operator
<input checked="" type="checkbox"/>	AORB	<input type="checkbox"/>	IHI
<input checked="" type="checkbox"/>	AORS	<input type="checkbox"/>	IHD
<input checked="" type="checkbox"/>	AOIU	<input type="checkbox"/>	IOD
<input checked="" type="checkbox"/>	AOIS	<input type="checkbox"/>	IOP
<input checked="" type="checkbox"/>	AODU	<input type="checkbox"/>	IOR
<input checked="" type="checkbox"/>	AODS	<input type="checkbox"/>	ISI
<input checked="" type="checkbox"/>	ROR	<input type="checkbox"/>	ISD
<input checked="" type="checkbox"/>	COR	<input type="checkbox"/>	IPC
<input checked="" type="checkbox"/>	COD	<input type="checkbox"/>	PNC
<input checked="" type="checkbox"/>	COI	<input type="checkbox"/>	PMD
<input checked="" type="checkbox"/>	SOR	<input type="checkbox"/>	PPD
<input checked="" type="checkbox"/>	LOR	<input type="checkbox"/>	PCI
<input checked="" type="checkbox"/>	LOI	<input type="checkbox"/>	PCC
<input checked="" type="checkbox"/>	LOD	<input type="checkbox"/>	PCD
<input checked="" type="checkbox"/>	ASRS	<input type="checkbox"/>	PRV
<input checked="" type="checkbox"/>	SDL	<input type="checkbox"/>	OMR
<input checked="" type="checkbox"/>	VDL	<input type="checkbox"/>	OMD
<input checked="" type="checkbox"/>	CDL	<input type="checkbox"/>	OAN
<input checked="" type="checkbox"/>	ODL	<input type="checkbox"/>	JTI
		<input type="checkbox"/>	JTD
		<input type="checkbox"/>	JSI
		<input type="checkbox"/>	JSD
		<input type="checkbox"/>	JID
		<input type="checkbox"/>	JDC
		<input type="checkbox"/>	EOA
		<input type="checkbox"/>	EOC
		<input type="checkbox"/>	EAM
		<input type="checkbox"/>	EMM

None All

None All

# Verificación y Validación de Software

The screenshot shows a software mutation testing tool interface. At the top, there are three tabs: "Mutants Generator", "Traditional Mutants Viewer", and "Class Mutants Viewer". Below the tabs, there are two dropdown menus: "Select a class:" set to "Main" and "Select a method:" set to "All method".

On the left side, there is a table titled "\* Summary \*" with two columns: "Op" and "#".

Op	#
AO...	4
AO...	0
AOIU	0
AOIS	4
AO...	0
AO...	0
ROR	7
COR	0
COD	0
COI	1
SOR	0
LOR	0
LOI	1
LOD	0
ASRS	0
SDL	2
VDL	1
CDL	1
ODL	2

Below the table, it says "Total : 23".

In the center, there is a list of mutants: AOIS\_1, AOIS\_2, AOIS\_3, AOIS\_4, AORB\_1, AORB\_2, AORB\_3, AORB\_4, CDL\_1, COL\_1, LOI\_1, ODL\_1, ODL\_2, ROR\_1, ROR\_2, ROR\_3, ROR\_4, ROR\_5, ROR\_6, ROR\_7, SDL\_1, SDL\_2, and VDL\_1. AOIS\_1 is selected.

On the right, there are two code blocks. The top one is labeled "Original" and shows the following code:

```
(line 9) boolean_esPar(int):x => ++x  
  
Original  
1 // This is a mutant program.  
2 // Author : ysma  
3  
4 public class Main  
5 {  
6  
7     public static boolean esPar( int x )  
8     {  
9         return x % 2 == 0;  
10    }  
11  
12 }
```

The bottom one is labeled "Mutant" and shows the following code:

```
Mutant  
1 // This is a mutant program.  
2 // Author : ysma  
3  
4 public class Main  
5 {  
6  
7     public static boolean esPar( int x )  
8     {  
9         return ++x % 2 == 0;  
10    }  
11  
12 }
```



# Verificación y Validación de Software

The screenshot shows the Eclipse IDE with the following components:

- Package Explorer:** Shows a project named 'Ejemplo1'.
- Code Editor:** Displays the content of 'testMain.java':

```
1 import static org.junit.Assert.*;
4
5 public class testMain {
6
7     @Test
8     public void testEsPar() {
9         assertTrue(Main.esPar(6));
10        assertTrue(Main.esPar(12));
11        assertTrue(Main.esPar(24));
12
13        assertTrue(!Main.esPar(1));
14        assertTrue(!Main.esPar(7));
15        assertTrue(!Main.esPar(11));
16
17    }
18 }
19
20 }
21
```
- Task List:** Contains a 'Connect Mylyn' notification.
- Outline:** Shows the class structure: 'testMain' with a method 'testEsPar() : void'.
- Problems:** Shows '0 items'.
- Status Bar:** Displays 'Writable', 'Smart Insert', and '21 : 1'.

The Windows taskbar at the bottom shows the time as 23:53 on 21/10/2015.

# Verificación y Validación de Software

The screenshot displays a software testing tool interface with three tabs: "TestCase Runner", "Traditional Mutants Viewer", and "Class Mutants Viewer". The "Class Mutants Viewer" tab is active, showing configuration for the "VendingMachine" class, "All method", and "testVendingMachineEdgeCoverage" test case. The "Execute all mutants" option is selected. The "Time-Out" is set to 3 seconds. A yellow "RUN" button is visible.

**Execute all mutants**

Class : VendingMachine  
Method : All method  
TestCase: testVendingMachineEdgeCoverage  
Time-Out : 3 seconds

**Traditional Mutants Result**

Live Mutants #	27
Killed Mutants #	94
Total Mutants #	121
Mutant Score	77.0%

**Class Mutants Result**

Live Mutants #	3
Killed Mutants #	1
Total Mutants #	4
Mutant Score	25.0%

**Op #**

AORB	8
AORS	0
AOIU	5
AOIS	38
AO...	0
AODS	0
ROR	42
COR	6
COD	0
COI	10
SOR	0
LOR	0
LOI	12
LOD	0
ASRS	0

**Op #**

IHI	0
IHD	0
IOD	0
IOP	0
IOR	0
ISI	0
ISD	0
IPC	0
PNC	0
PMD	0
PPD	0
PCI	0
PCC	0
PCD	0
PRV	0
OMR	0
OMD	0
OAN	0
JTI	0
JTD	0
JSI	2
JSD	1
JID	0
JDC	1
EOA	0
EOC	0

**Traditional Mutants Result**

Live	Killed
AOIS_11	AOIS_1
AOIS_12	AOIS_10
AOIS_19	AOIS_13
AOIS_20	AOIS_14
AOIS_23	AOIS_15
AOIS_24	AOIS_16
AOIS_9	AOIS_17
LOI_1	AOIS_18
LOI_3	AOIS_2
ROR_1	AOIS_21
ROR_2	AOIS_22
ROR_20	AOIS_3
ROR_22	AOIS_4
ROR_6	AOIS_5
AOIS_29	AOIS_6
AOIS_30	AOIS_7
AOIS_33	AOIS_8
AOIS_34	AOIU_1

**Class Mutants Result**

Live	Killed
JSD_1	JDC_1
JSI_1	
JSI_2	

# Verificación y Validación de Software

Estructuras de Datos  
Martín Larrea