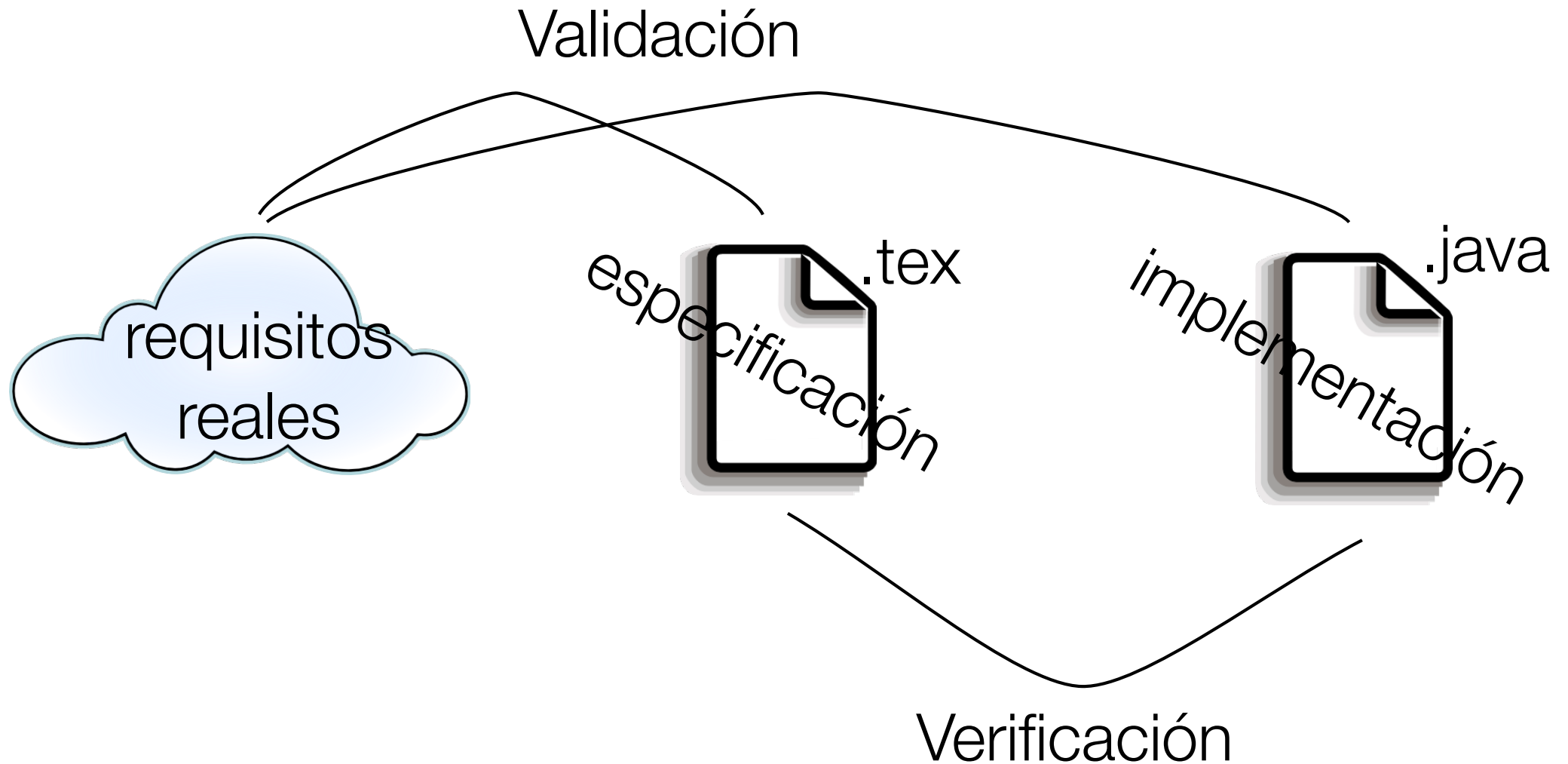


Generación Automática de Tests

Generación Aleatoria | Randoop

Validación vs. Verificación



Verificación

Spec

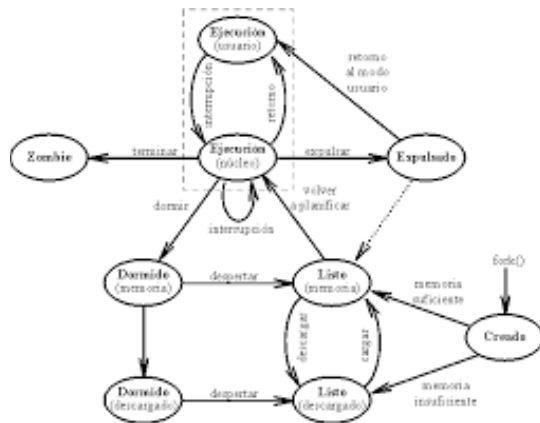
Verificación

Spec

¿Se comporta **Sys** de acuerdo a su especificación?

Verificación

Spec



Sys

¿Se comporta **Sys** de acuerdo a su especificación?

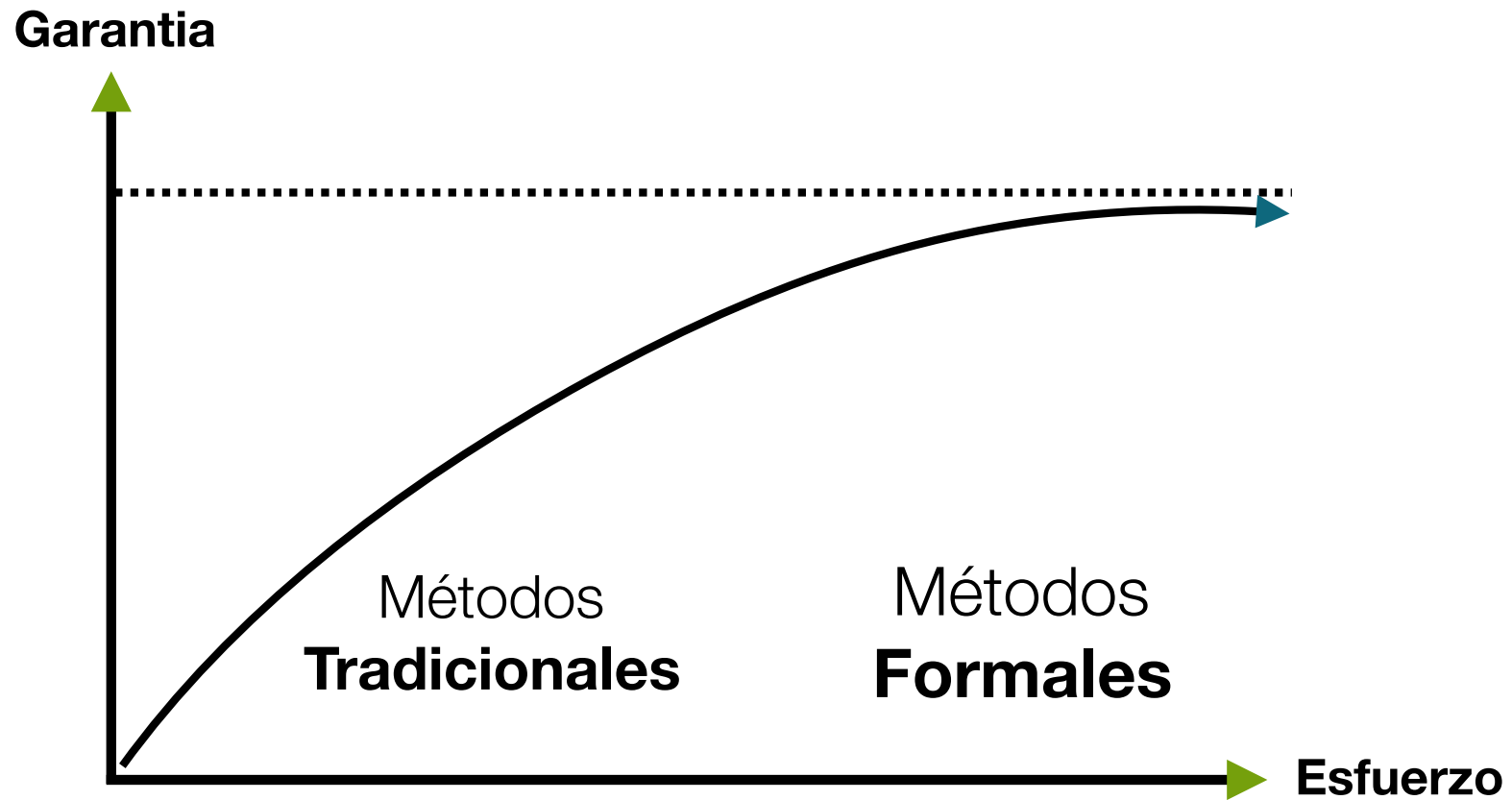
Enfoques

Garantia

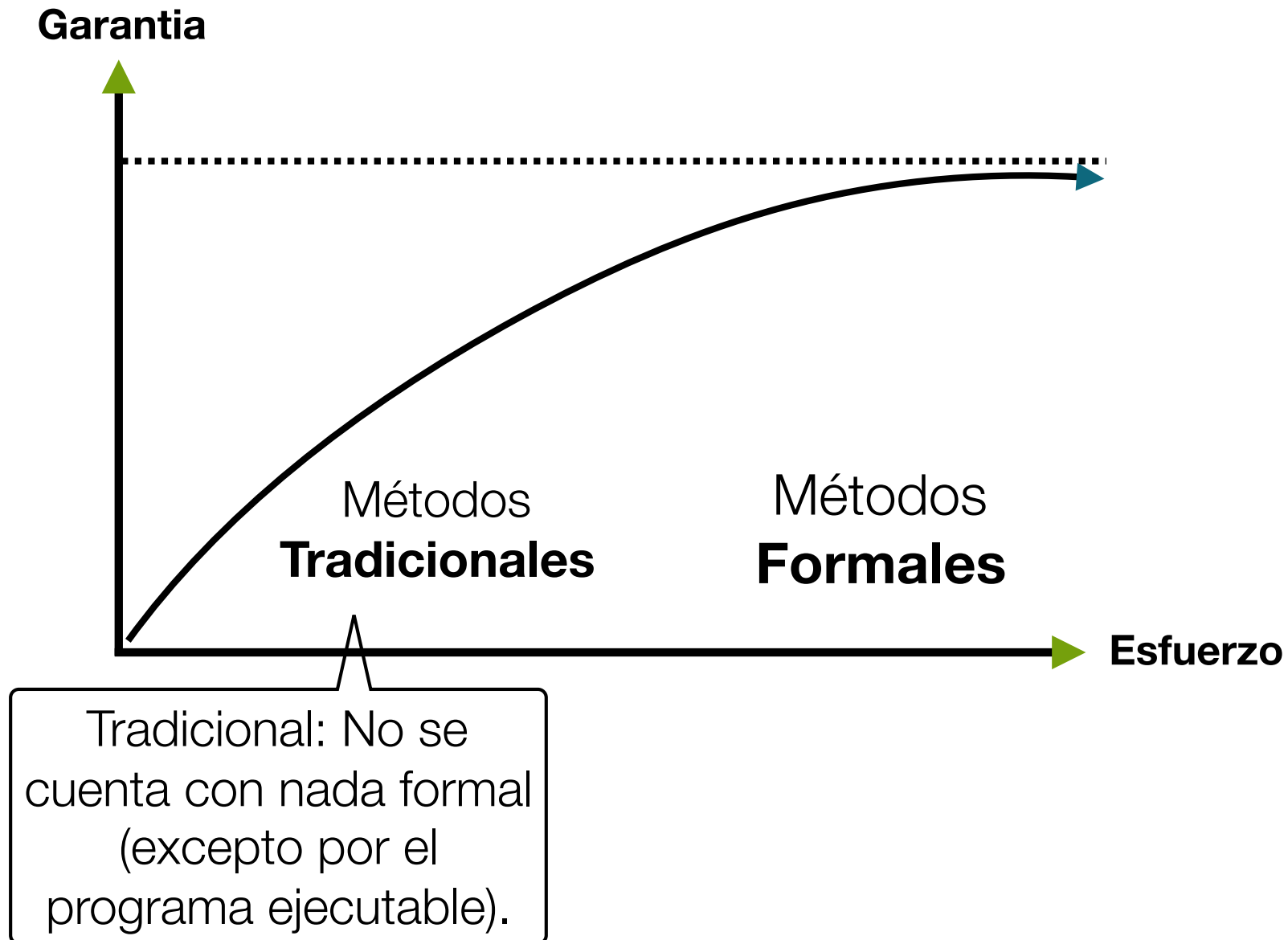


Esfuerzo

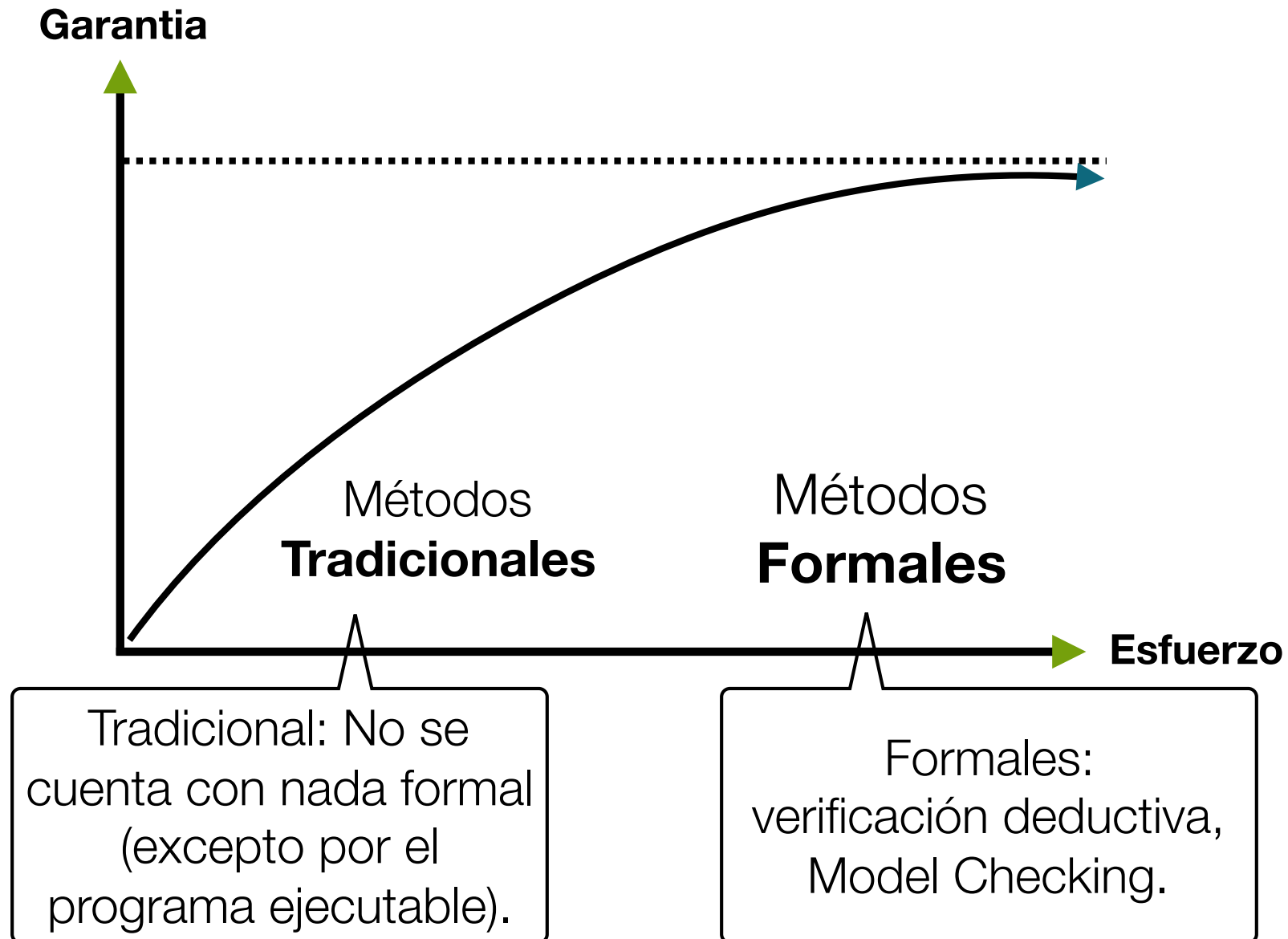
Enfoques



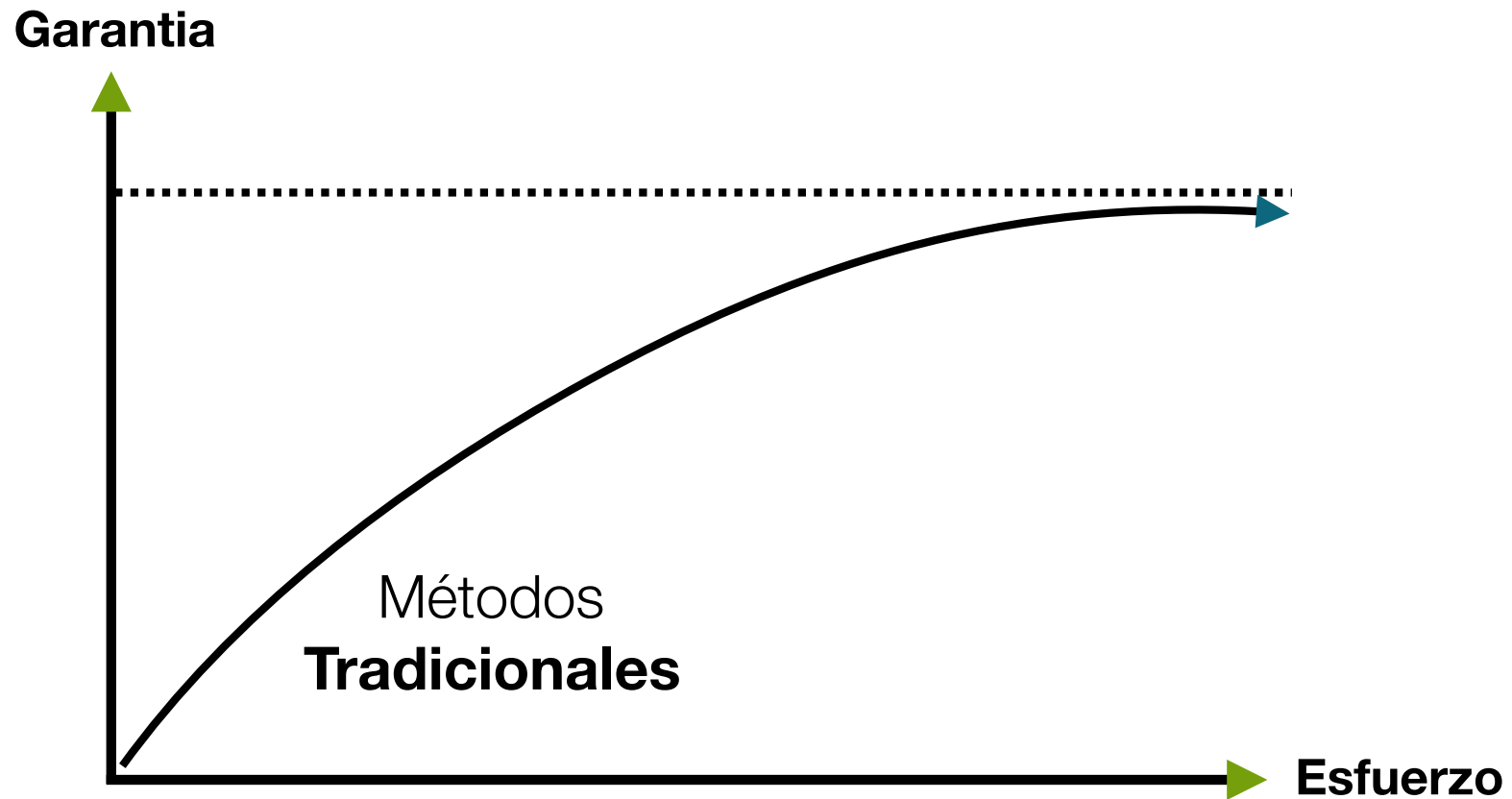
Enfoques



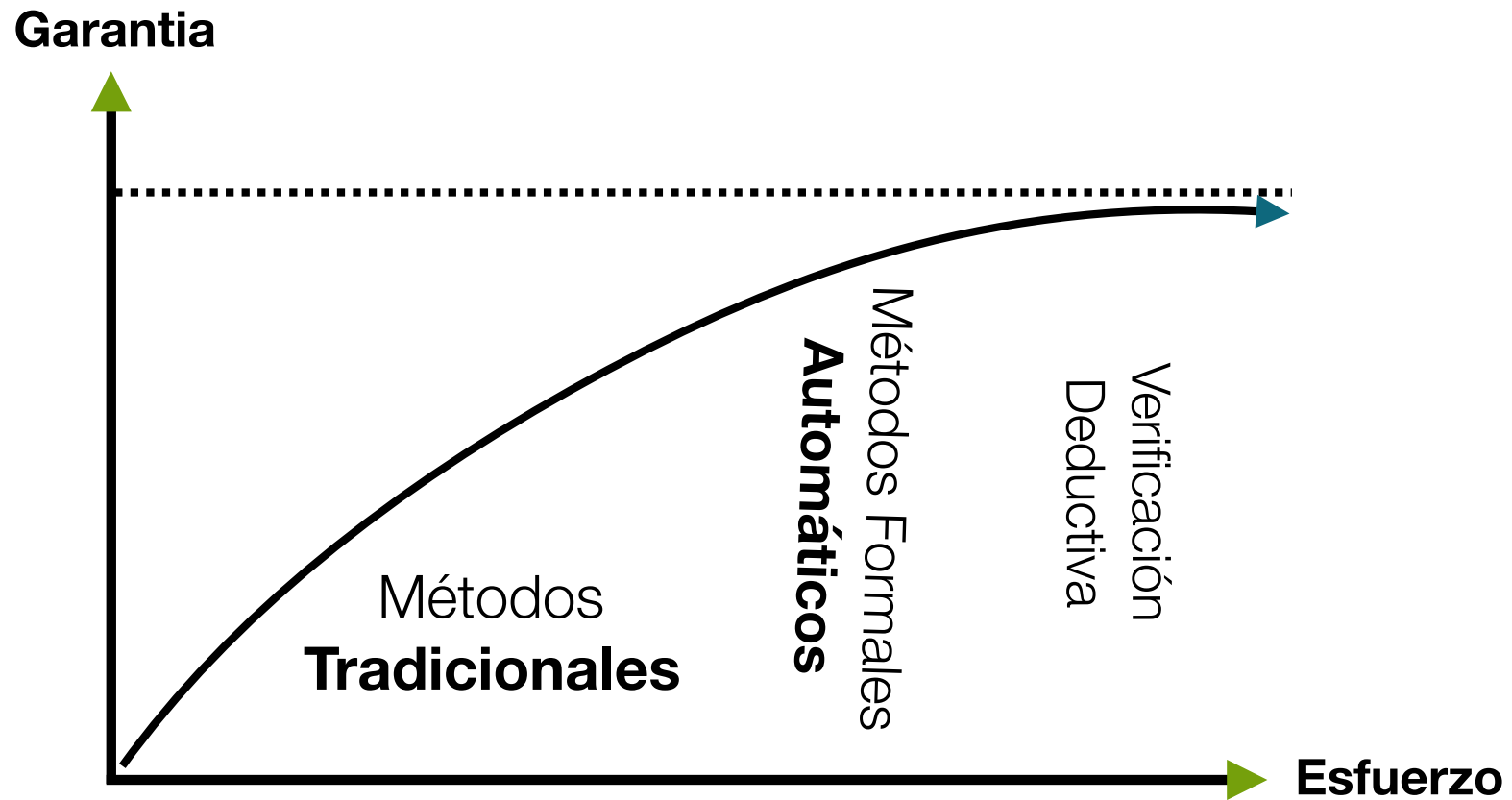
Enfoques



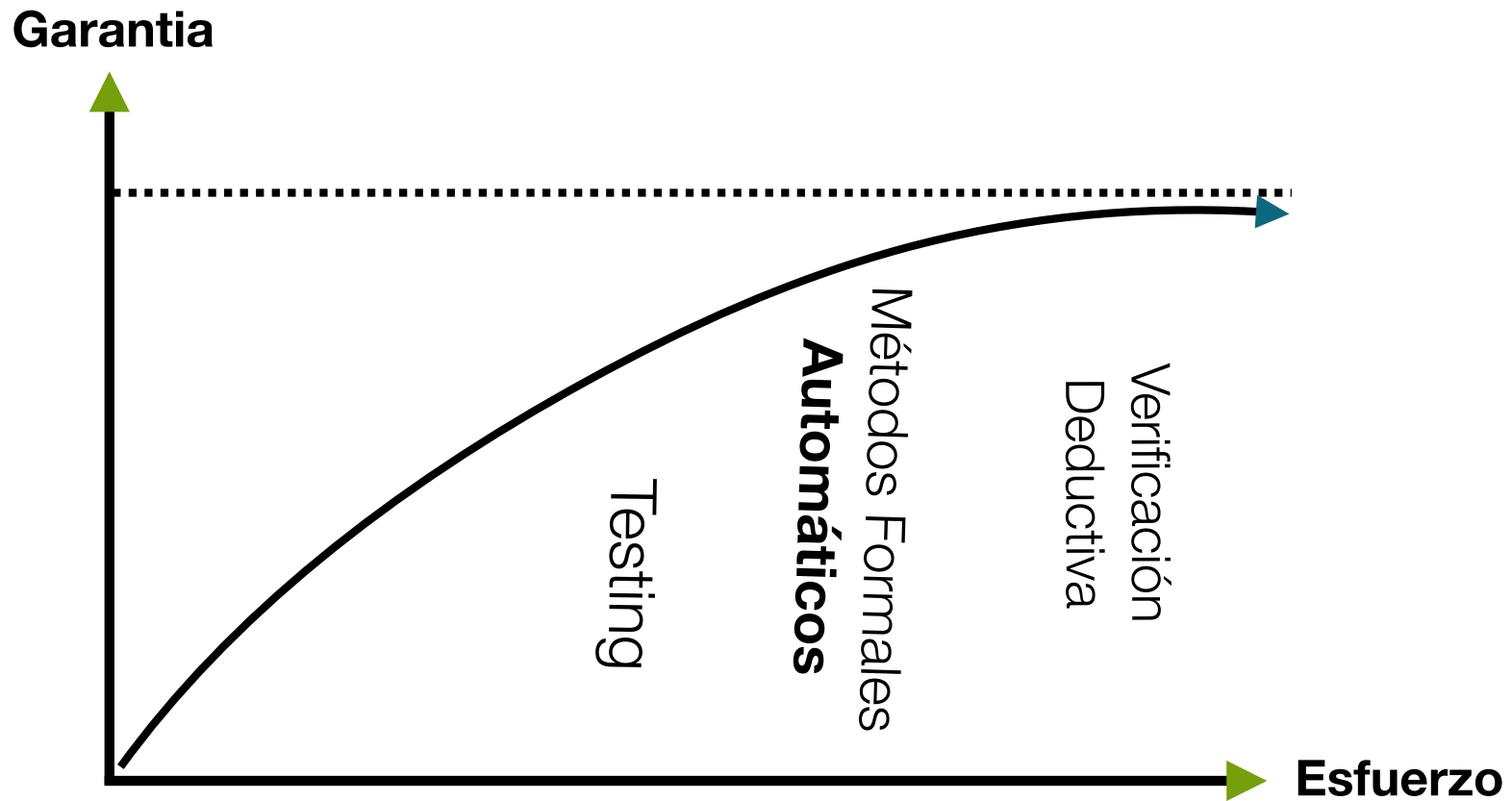
Enfoques



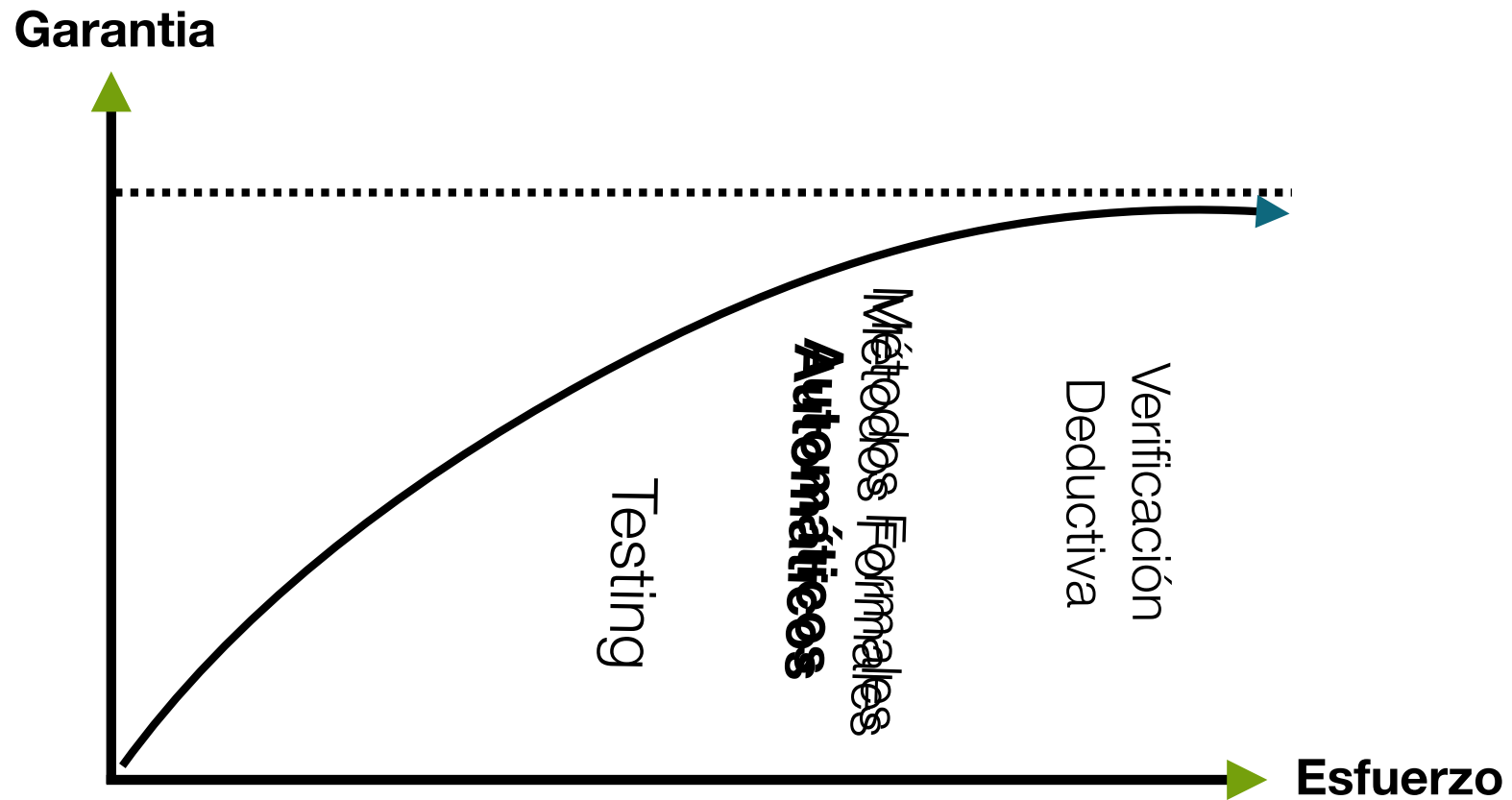
Enfoques



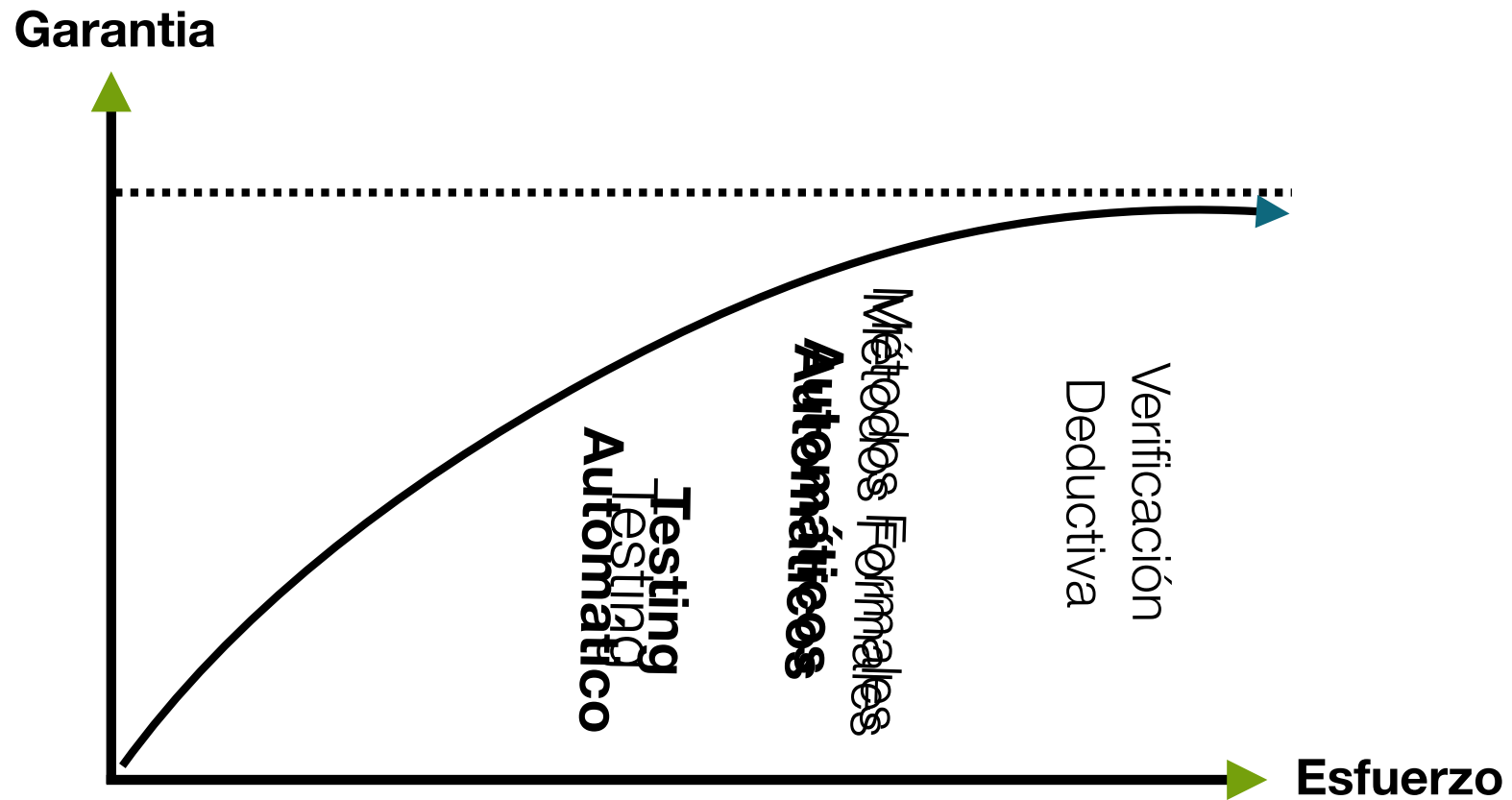
Enfoques



Enfoques



Enfoques



Generación Automática de Tests

La construcción de tests involucra como mínimo lo siguiente:

- la construcción de inputs para el SUT, que lo ejerciten en una variedad de situaciones (en general, teniendo en cuenta un criterio de cobertura)

- la evaluación de cuál debería ser el resultado en cada caso, y la producción de los “asserts” correspondientes

Veremos una variedad de técnicas para automatizar estas tareas, particularmente la primera.

Técnicas Automáticas

Generación Aleatoria

Generación Exhaustiva Acotada

Generación Basada en Constraint Solving

Generación Basada en Algoritmos Genéticos

Generación Aleatoria de Tests

Una forma de construir tests automáticamente es produciendo **aleatoriamente** entradas para los programas a testear

- simple si el SUT recibe sólo entradas de tipos simples

 - utilización de mecanismos de generación aleatoria de valores numéricos

- no tan trivial si el SUT recibe entradas de tipos complejos (e.g., objetos)

Generación Aleatoria Simple

Consideremos la siguiente técnica:

Sea **prog**($x_1: T_1, \dots, x_k: T_k$) una rutina a testear.

Para cada x_i hacer

Si T_i es un tipo básico, generar un valor v_i para asignar a x_i utilizando un generador de números aleatorios

Si T_i es una clase, elegir aleatoriamente entre:

asignar **null** a x_i

elegir uno de los constructores sin parámetros de T_i para construir un objeto a asignar a x_i

Generación Aleatoria Simple: Ejemplo 1

Consideremos la siguiente rutina:

```
public static int max(int a, int b) {  
    if (a>b) {  
        return a;  
    }  
    else {  
        return b;  
    }  
}
```

Para ésta, generamos aleatoriamente valores para a y b:

{(a=124,b=3), (a=-15,b=0), ... }

Generación Aleatoria Simple: Ejemplo 2

Consideremos ahora la siguiente rutina:

```
public static Integer largest(Integer[] list) {  
    int index = 0;  
    int max = Integer.MIN_VALUE;  
    while (index <= list.length-1) {  
        if (list[index] > max) {  
            max = list[index];  
        }  
        index++;  
    }  
    return max;  
}
```

Para ésta, podemos usar null o los constructores básicos de arreglos. Obtendríamos:

{ list = null, list = [null,null], list = [null, null, null, null, null], ... }

Limitaciones de la Generación Aleatoria Simple

La generación aleatoria simple tiene limitaciones

Para objetos, puede generar en general objetos muy elementales, los construibles directamente a partir de constructores simples.

Cada generación es independiente de las anteriores, lo cual puede producir muchos tests “redundantes” e “ilegales”, que ejerciten el código de la misma manera, o violen la precondition del SUT, respectivamente.

Generación de Objetos “Complejos”

Para generar objetos más complejos que los “alcanzables” por los constructores simples, se puede seguir el siguiente proceso:

Sea **prog**(**x**₁: **T**₁, ..., **x**_k: **T**_k) una rutina a testear.

Para cada **x**_i hacer

Elegir aleatoriamente entre lo siguiente:

usar un generador de números aleatorios para generar un valor **v**_i a asignar a **x**_i, si **T**_i es un tipo básico

asignar **null** a **x**_i (si **T**_i es una clase)

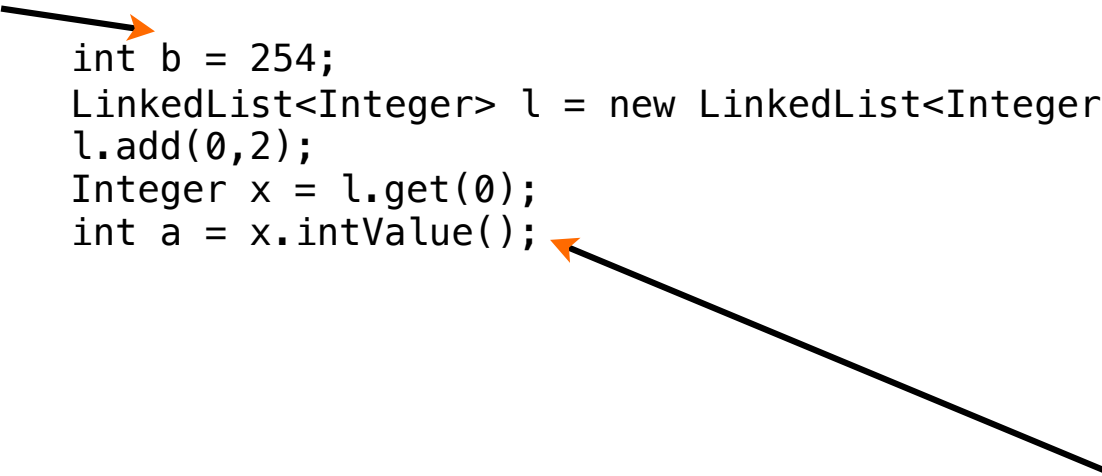
elegir un método **m**(**y**₁:**T**'₁, ... **y**_{k'}:**T**'_{k'}) cuyo tipo de retorno sea **T**_i para construir un objeto a asignar a **x**_i

Repetir el proceso para construir valores para cada **y**_i

Generación Aleatoria: Ejemplo 1 rev.

Consideremos nuevamente la rutina **max**. Para ésta, podríamos generar aleatoriamente valores para **a** y **b** de la siguiente manera:

generado con un
gen. de números
aleatorios



```
int b = 254;  
LinkedList<Integer> l = new LinkedList<Integer>();  
l.add(0,2);  
Integer x = l.get(0);  
int a = x.intValue();
```

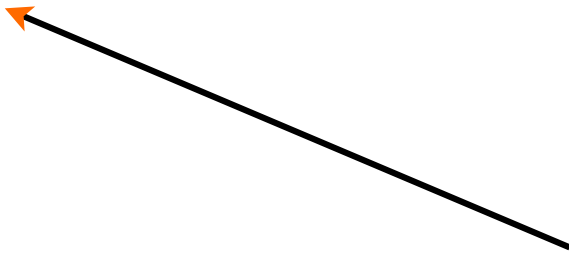
The diagram consists of two arrows. The first arrow originates from the text 'generado con un gen. de números aleatorios' and points to the line 'int b = 254;'. The second arrow originates from the text 'a se genera a partir de un encadenamiento aleatorio de métodos, que resultan en la generación de un valor entero' and points to the line 'int a = x.intValue();'.

a se genera a partir de un encadenamiento
aleatorio de métodos, que resultan en la
generación de un valor entero

Generación Aleatoria: Ejemplo 2 rev.

Consideremos nuevamente la rutina **largest**. Para ésta, podríamos generar aleatoriamente valores para **list** de la siguiente manera:

```
LinkedList<Integer> l = new LinkedList<Integer>();  
l.add(0,2);  
l.add(0,l.getFirst());  
Integer[] list = l.toArray();
```



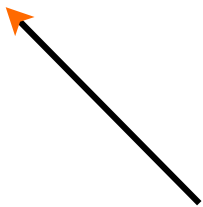
list se genera a partir de un
encadenamiento aleatorio de métodos, que
resultan en la generación de un arreglo de
Integer

Eliminación de Redundancia: Aprovechamiento de Feedback

En la generación aleatoria de tests, es posible caer en las siguientes situaciones:

```
Set s = new HashSet();  
s.add("hi");  
assertTrue( ... );
```

```
Set s = new HashSet();  
s.add("hi");  
s.isEmpty();  
assertTrue( ... );
```



si se identificara la redundancia del test,
no se incluiría en la suite

Eliminación de Redundancia: Aprovechamiento de Feedback

En la generación aleatoria de tests, es posible caer en las siguientes situaciones:

```
Set s = new HashSet();  
s.add("hi");  
assertTrue( ... );
```

```
Set s = new HashSet();  
s.add("hi");  
s.isEmpty();  
assertTrue( ... );
```



si se identificara la redundancia del test,
no se incluiría en la suite

Eliminación de Redundancia: Aprovechamiento de Feedback

En la generación aleatoria de tests, es posible caer en las siguientes situaciones:

```
Set s = new HashSet();  
s.add("hi");  
assertTrue( ... );
```

```
Set s = new HashSet();  
s.add("hi");  
s.isEmpty();  
assertTrue( ... );
```



si se identificara la redundancia del test,
no se incluiría en la suite

```
Date d = new Date(2006,2,14);  
d.setMonth(-1); // pre:argument>0  
assertTrue( ... );
```

```
Date d = new Date(2006,2,14);  
d.setMonth(-1); // lanza excepción  
d.setDay(5);  
assertTrue( ... );
```



si se sabe que el test anterior falla en
producir una entrada válida, este no
debería producirse

Eliminación de Redundancia: Aprovechamiento de Feedback

En la generación aleatoria de tests, es posible caer en las siguientes situaciones:

```
Set s = new HashSet();  
s.add("hi");  
assertTrue( ... );
```

```
Set s = new HashSet();  
s.add("hi");  
s.isEmpty();  
assertTrue( ... );
```



si se identificara la redundancia del test,
no se incluiría en la suite

```
Date d = new Date(2006,2,14);  
d.setMonth(-1); // pre:argument>0  
assertTrue( ... );
```

```
Date d = new Date(2006,2,14);  
d.setMonth(-1); // lanza excepción  
d.setDay(5);  
assertTrue( ... );
```



si se sabe que el test anterior falla en
producir una entrada válida, este no
debería producirse

Generación Aleatoria Guiada por Feedback

Only normally-behaving sequences are used to generate new sequences, as it makes little sense to extend a sequence that contains an already-corrupted state.

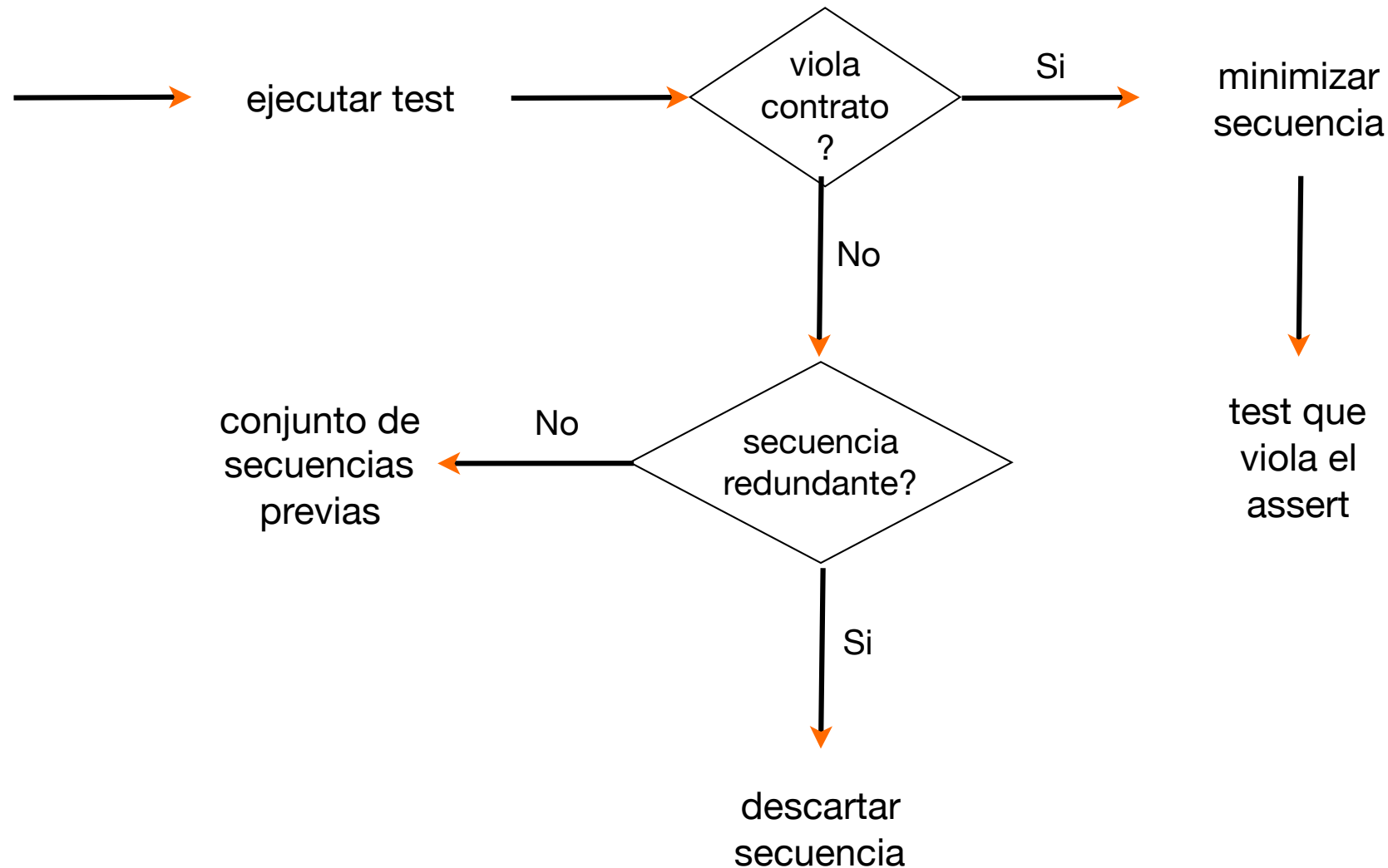
La generación aleatoria de tests guiada por feedback complementa el proceso de generación descripto anteriormente, mediante la construcción incremental de entradas

Tan pronto como se produzca una secuencia de generación, se ejecuta la misma

si la secuencia es inválida, se descarta

si la secuencia es válida, se producen nuevas extendiendo a ésta y otras anteriores

Clasificación de Tests generados Aleatoriamente



Cómo Determinar si una Secuencia es Redundante

Para determinar si una secuencia es redundante se realiza lo siguiente:

- Durante la generación se mantiene el conjunto de todos los objetos creados.

- Cada vez que se produce una nueva secuencia válida, se compara con los objetos creados previamente

 - si durante su ejecución no genera nuevos objetos (usando el **equals** para comparar), entonces la secuencia generada se descarta (redundante)

 - si el input no fue creado previamente, se almacena la secuencia, y se guarda el nuevo input generado en el conjunto de objetos

Randoop: Una Herramienta para la Generación Aleatoria Guiada por Feedback

Randoop es una herramienta para la generación de tests unitarios basada en generación aleatoria

incorpora las técnicas de generación guiadas por feedback descriptas para:

- intentar eliminar casos de test inválidos

- intentar no producir casos de test redundantes

Existen versiones de Randoop tanto para Java como para .NET

Randoop: Una Herramienta para la Generación Aleatoria Guiada por Feedback

Randoop recibe como **entrada**:

- un conjunto de clases (SUT)

- un tiempo límite de generación/cantidad máxima de test

Randoop produce como **salida**:

- una suite de tests, que

 - violan “contratos” básicos generados automáticamente, o

 - son reconocidos como tests válidos

Randoop y Asserts Generados Automáticamente

Regression tests can discover inconsistencies between two different versions of the software

Para determinar si un test corresponde a comportamiento correcto o no, Randoop genera asserts básicos que, independientemente del contexto, siempre deberían ser válidos.

Contratos sobre equals(), hashCode(), toString(), ...

Ej: assertTrue(o.equals(o)).

Además, Randoop genera asserts capturando comportamiento de la implementación actual a ser utilizados para **testing de regresión**

```
Object o = new Object();
LinkedList l = new LinkedList();
l.addFirst(o);
l.add(o);
assertEquals(2, l.size());           // expected to pass
assertEquals(false, l.isEmpty());    // expected to pass
```

Randoop e Invariantes de Representación

Además del chequeo de asserts básicos, Randoop también puede utilizar una especificación provista por el usuario por ejemplo, en forma de invariante de clase.

Los invariantes de clase forman parte de la especificación de una clase. Establecen propiedades que deben mantenerse a lo largo de la ejecución de los objetos de la clase.

Es decir,

- todos los constructores deben establecer, al finalizar, la propiedad

- todas las rutinas de la clase deben preservar la propiedad

Invariantes de Representación Imperativos

Muchas herramientas, Randoop entre ellas, recibe la especificación de invariantes de representación en forma de código, mediante un método de la clase que comprueba que el invariante se satisface.

Por ejemplo:

```
public class SinglyLinkedList {  
  
    public Entry header;  
  
    private int size = 0;  
  
    ...  
}  
  
public class Entry {  
  
    Object element;  
  
    Entry next;  
}
```

```
public boolean repOK() {  
    if (header == null)  
        return false;  
    if (header.element != null)  
        return false;  
    Set<Entry> visited = new java.util.HashSet<Entry>();  
    visited.add(header);  
    Entry current = header;  
    while (true) {  
        Entry next = current.next;  
        if (next == null)  
            break;  
        if (next.element == null)  
            return false;  
        if (!visited.add(next))  
            return false;  
        current = next;  
    }  
    if (visited.size() - 1 != size)  
        return false;  
    return true;  
}
```

Efectividad de Generación Aleatoria Guiada por Feedback (cont.)

También ha resultado ser muy efectiva para descubrir bugs

	test cases output	error-revealing tests cases	distinct errors
JDK	32	29	8
Apache commons	187	29	6
.Net framework	192	192	192
Total	411	250	206

Otras Herramientas basadas en Generación Aleatoria

Existen otras herramientas de generación automática de tests basadas en generación aleatoria.

AutoTest: es la herramienta de generación automática de tests para Eiffel, integrada a EiffelStudio

QuickCheck: es una herramienta de generación aleatoria de tests para programas funcionales (Haskell específicamente)

Randooop

<https://randooop.github.io/randooop/>

[http://people.csail.mit.edu/cpacheco/publications/
randooopjava.pdf](http://people.csail.mit.edu/cpacheco/publications/randooopjava.pdf)