EVALUACIÓN DE HERRAMIENTAS DE GENERACIÓN DE CONJUNTOS DE TESTS PARA COBERTURA DE CAMINOS PRIMARIOS

EVALUATION OF TEST SET GENERATION TOOLS FOR COVERAGE OF PRIMARY PATHS

TRABAJO FIN DE MÁSTER
CURSO 2021-2022

AUTOR
YIFEI LIU

DIRECTOR
MARÍA DE LAS MERCEDES GARCÍA MERAYO

MÁSTER EN INGENIERÍA INFORMÁTICA
FACULTAD DE INFORMÁTICA
UNIVERSIDAD COMPLUTENSE DE MADRID

EVALUACIÓN DE HERRAMIENTAS DE GENERACIÓN DE CONJUNTOS DE TESTS PARA COBERTURA DE CAMINOS PRIMARIOS

EVALUATION OF TEST SET GENERATION TOOLS FOR COVERAGE OF PRIMARY PATHS

TRABAJO DE FIN DE MÁSTER EN INGENIERÍA INFORMÁTICA
DEPARTAMENTO DE SISTEMA INFORMÁTICOS Y COMPUTACIÓN

AUTOR
YIFEI LIU

DIRECTOR
MARÍA DE LAS MERCEDES GARCÍA MERAYO

CONVOCATORIA: SEPTIEMBRE 2022
CALIFICACIÓN: 8. 5

MÁSTER EN INGENIERÍA INFORMÁTICA
FACULTAD DE INFORMÁTICA
UNIVERSIDAD COMPLUTENSE DE MADRID

DÍA DE SEPTIEMBRE DE 2022

AGRADECIMIENTOS

En primer lugar, a mi tutora, por la orientación y la ayuda que me brindó, por su apoyo y paciencia para la realización de este proyecto.

A mis padres que viven lejos en China, por su comprensión, motivación y apoyo durante el transcurso de toda mi vida académica.

A mi compañera de piso por estar escuchando mis quejas, por sus palabras de ánimo que me sirvieron para calmar y seguir con el proyecto.

Por último, me gustaría agradecer a Yi Teng y Javier por ayudarme con el problema del idioma y corregir mis errores gramaticales en la redacción de esta memoria.

RESUMEN

Evaluación de herramientas de generación de conjuntos de tests para cobertura de caminos primarios

Testing es una parte integral del aseguramiento de la calidad del software. Hoy en día en el mercado ya existen muchas herramientas de generación de pruebas automatizadas que pueden generar automáticamente casos de prueba para el código que se está probando. Esas pruebas no siempre son perfectas, pero ha reducido bastante el trabajo para los probadores. La "cobertura de prueba" es una de las medidas de calidad más importante para las pruebas, las herramientas de prueba existentes se calculan principalmente como cobertura del código.

En este proyecto, se implementó una herramienta web utilizando el framework Spring Boot, y se utilizó la tecnología Antlr4 para generar grafo de control de flujo mediante análisis de sintaxis del método en el código (JAVA) introducido por el usuario desde la página web, se registra los nodos y luego se calcula los requisitos de test del código de entrada.

Para este proyecto, se eligieron EvoSuite y Diffblue, dos herramientas de pruebas automatizadas, para generar casos de prueba automatizados cada uno. Se analizó el conjunto de pruebas de estos archivos de test automatizados para obtener la cobertura del camino primario. Los resultados de estos cálculos se muestran uno al lado del otro en una página web, lo que permite al usuario ver hasta qué punto el código de los casos de prueba generados por las dos herramientas cumple los requisitos de la prueba, y seleccionar los casos de prueba que satisfacen sus necesidades.

Palabras clave

Testing, Criterio de cobertura, Requisitos de test, Caminos primarios.

ABSTRACT

Evaluation of test set generation tools for coverage of primary paths

Testing is an essential component of software quality assurance. There are already many automated test generation tools on the market that can generate test cases for the code being tested automatically. Those tests are not always perfect, but it has reduced the workload for the testers quite a bit. "Test coverage" is one of the most important quality measures for testing, existing testing tools are mainly calculated as code coverage.

In this project, a web tool was implemented by using Spring Boot, and Antlr4 technology was used to generate a control flow graph by syntax analysis of the method in the code (JAVA) entered by the user from the web page. The nodes are registered and then the test requirements of the input code are calculated.

For this project, EvoSuite and Diffblue, two automated testing tools, were chosen to generate automated test cases each. The test suite of these automated test files was analyzed to obtain the primary path coverage. The results of these calculations are displayed side-by-side on a web page, allowing the user to see to what extent the test case code generated by the two tools meets the test requirements and to select test cases that meet their needs.

Keywords

Testing, Coverage criteria, Test requirements, Primary paths.

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

Capítulo 1 - Introducción

El primer capítulo introducirá la motivación de este tema como proyecto, y presentará los objetivos a alcanzar en este proyecto, así como el plan de trabajo desarrollado para completarlos.

1.1 Motivación

Software testing, como un medio importante para garantizar la calidad y confiabilidad del software, es una de las partes más importantes e indispensables del proceso de desarrollo de software. Con el rápido desarrollo de la informática y la tecnología, el testing de hoy en día ya no significa sólo el proceso de prueba tradicional después de la finalización de la codificación, sino un proceso continuo a lo largo de todo el ciclo de vida de cada producto, desde su desarrollo hasta la explotación. Este cambio se trata de un avance significativo para la industria de la información, la participación temprana en el proceso de desarrollo y el descubrimiento de errores y vulnerabilidades ocultos lo antes posible sin duda ahorrará el costo de los recursos humanos y el tiempo requerido para solucionarlos, y también reducirá aún más el riesgo de que el desarrollo de software se atasque en las fases finales debido a la acumulación de errores anteriores.

A partir de los resultados de la investigación publicados en "Software Testing Trends 2022"[1], podemos encontrar que las pruebas de API y la automatización de pruebas están recibiendo cada vez más atención por parte de la profesión en la actualidad. Como la tendencia principal del desarrollo de software hoy en día, la arquitectura de microservicios, el desarrollo y la aplicación orientados a API, las pruebas de API están destinadas a convertirse en una de las herramientas de verificación más importantes de su tipo. Además, las pruebas de API también son objetivamente más fáciles de automatizar.

Tuve la suerte de matricularme en un curso opcional sobre testing en el grado y aprendí algunos conceptos básicos relacionados con las pruebas y, poco a poco, me interesé por ellas. Durante el proceso de aprendizaje, descubrí que al realizar análisis

estáticos del programa dado, como dibujar CFG[1], requisitos de test, cálculos de caminos primarios, etc., todo se hacía con lápiz y papel. Un código un poco más largo suele necesitar verificarse varias veces, es bastante propenso a errores y produce una gran cantidad de borradores. Desafortunadamente, algunas de las herramientas que he encontrado no siempre satisfacen las necesidades y tienen sus defectos, por lo que decidí completar una herramienta que pueda mostrar el CFG, los requisitos de test y la cobertura de casos de prueba del programa en este proyecto.

1.2 Objetivos

El objetivo principal del proyecto se puede dividir en tres partes: transformación del código a CFG -> cálculo de requisitos de test -> generación de la cobertura de casos de prueba de los requisitos de test.

Antes de comenzar con la primera parte, es necesario crear una interfaz de aplicación simple que permita al usuario ingresar el código de prueba, mostrar la información textual devuelta por el proyecto y mostrar el CFG generado. Una vez listo, vamos con el primer objetivo: la elección inicial para iterar a través del programa bajo prueba es Antlr4[2], que necesita registrar los nodos y las conexiones de unión a medida que el programa bajo prueba encuentra la sintaxis de la estructura especificada, y seleccionar la herramienta de dibujo apropiada para dibujar los nodos y las aristas en un gráfico CFG de acuerdo con las reglas. El CFG necesita mostrar alguna información básica en lugar de mostrar simplemente 1,2,3,4, para evitar que el usuario tenga que comparar repetidamente los métodos de programación introducidos.

La segunda parte consiste básicamente en una traducción de lo que era un cálculo en papel, una derivación paso a paso, a un programa que requiere la información almacenada en el proceso de generación del CFG para obtener las partes necesarias para calcular la ruta de requisitos del test.

Para el tercer objetivo, primero es necesario encontrar un método adecuado para no sólo ejecutar el caso de prueba de entrada, sino también registrar la ruta de

[1] Grafo de control de flujo
[2] Another Tool for Language Recognition V4

nodo que recorre el caso de prueba al ejecutar el programa introducido por el usuario. A continuación, se calcula a partir de la ruta de ejecución para obtener su cobertura de los requisitos de test obtenidos en el objetivo anterior. Una vez conseguido esto, buscar una herramienta de prueba automatización adecuada para reemplazar el módulo en el que se introdujeron manualmente los casos de prueba y modificar el programa para que se ejecute correctamente.

1.3 Trabajo relativo

Antes de empezar mi trabajo, busqué algunas herramientas similares como referencia. Para el trazado de CFG, encontré Eclipse Flow Chart Generator[2], es un plug-in de Eclipse desarrollado por estudiante, que genera un CFG haciendo clic con el botón derecho y seleccionando el método en el programa a ejecutar, calculando los nodos, el número de conexiones y la métrica McCabe para calcular la complejidad del bucle.

El usuario puede arrastrar y organizar los nodos CFG generados por este plug-in, lo cual es muy flexible y también acepta clics en los nodos para ampliar el código detallado. Sin embargo, la desventaja es que cada línea de código que recibe se convierte en un nodo, lo que hace que el diagrama sea muy largo e incompresible cuando hay varias líneas de código como declaraciones de función, impresión y otras declaraciones simples. Podemos intentar mejorarlos en este desarrollo.

Para el cálculo de la cobertura, Cobertura[3] es una herramienta de código abierto para Java, basada en el desarrollo de Jcoverage. Mide la cobertura de la prueba inspeccionando el código básico y observando qué código se ejecuta y cuál no cuando se ejecuta el paquete de prueba. Además de encontrar código no probado y descubrir errores, Cobertura también puede optimizar el código marcando el código inútil y no ejecutado. Estos me han servido para tener las ideas del desarrollo.

1.4 Plan de trabajo

Para asegurar la implementación del proyecto, opté por desarrollarlo en un modelo en cascada. Como se muestra en el diagrama de Gantt, el desarrollo se dividió en 4 fases: Investigación, Desarrollo, Test y modificación, y Redacción de la memoria.

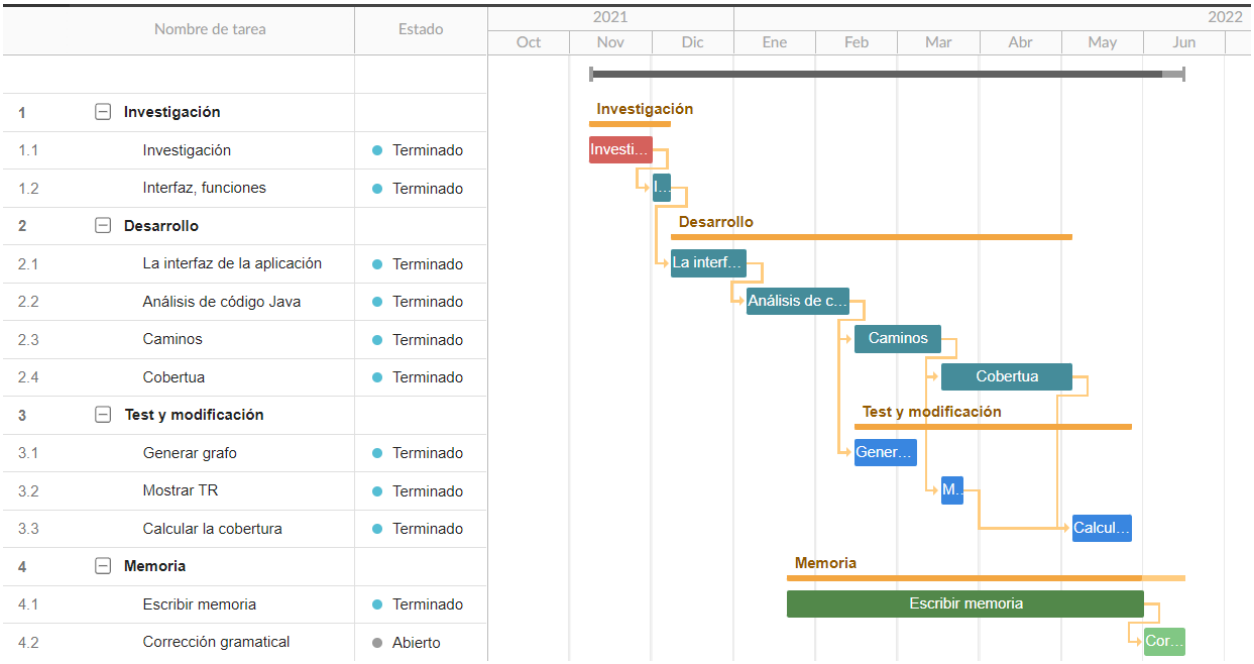


Figura 1-1. Diagrama de gantt

Durante la fase de investigación, primero hice un análisis detallado de lo que quería hacer para el proyecto. Determiné qué técnicas se necesitaban para el proyecto y qué técnicas estaban disponibles para un periodo de estudio inicial de aproximadamente un mes. Por supuesto, parte de este tiempo también se reservó para repasar los conocimientos del curso de testing que se había estudiado. A partir de la recopilación y el aprendizaje en la fase de preparación, determinar las principales técnicas a utilizar y también tenía pensado preparar 1-2 opciones alternativas para los objetivos mencionados en el capítulo 1.2. Una vez decidido el plan, hice un diseño simple de la interfaz y las funciones del programa.

Tras el análisis de los objetivos y requisitos de desarrollo se comprobó que cada módulo tenía una clara división funcional y que el primer objetivo era la base y el núcleo para garantizar la correcta presentación de todo el proyecto, el resto de los objetivos

deben desarrollarse y perfeccionarse sobre esta base, por lo que finalmente se adoptó el modelo incremental para este proyecto. Cada módulo se probó a tiempo tras su finalización, y el programa se corrigió hasta que estuvo libre de errores antes de pasar a la siguiente fase de desarrollo y pruebas, con el fin de reducir los riesgos de desarrollo. El código de este proyecto se subirá finalmente a mi repositorio de GitHub, al que se puede acceder desde el siguiente enlace: https://github.com/luojuee/toolTFM.

Por último, el documento de la memoria se redactará a lo largo del desarrollo del software hasta el final, manteniendo un registro del estado de desarrollo y las ideas de desarrollo del programa. De este modo se evita que se olviden los detalles del trabajo anterior cuando se produzcan cambios repentinos durante el desarrollo. Así mismo, puede facilitar la cumplimentación posterior del texto. Una vez terminada la redacción de la memoria, se reservará un tiempo para las correcciones lingüísticas y sintácticas.

Capítulo 2 - Tecnologías empleadas

En este capítulo se dará una descripción simplificada de las herramientas y técnicas utilizadas en la preparación y realización del proyecto, se explicarán las razones de su uso y se presentarán sus ventajas e inconvenientes.

2.1 IDEA

El nombre completo de IDEA es IntelliJ IDEA, es un entorno integrado para el desarrollo en lenguaje Java, pero por supuesto que también puede comprender otros lenguajes. IntelliJ como producto desarrollado por JetBrains, es reconocido como una de las mejores herramientas de desarrollo Java del sector, especialmente por sus funciones del asistente de código inteligente, la solicitud automática de código, la refactorización, la compatibilidad con J2EE, la compatibilidad con múltiples lenguajes y framework, JUnit, la revisión de código, el diseño innovador de GUI y otras características que pueden considerarse excelentes[4].

El proyecto se desarrolló con Eclipse al principio, pero fue sustituido por IDEA a mitad del proyecto. Aparte de la incomodidad inicial, IDEA realmente mejoró mi eficiencia de desarrollo en términos de cambio entre diferentes versiones del entorno Java, la provisión de plug-ins y algunas de las herramientas utilizadas ha eliminado la necesidad de rellenar los argumentos de la VM. En la parte de revisión del código, IDEA me indica directamente qué código está duplicado ofrece una comparación, lo que facilita la reducción de la tasa de duplicación de código y mejora en gran medida la mantenibilidad y legibilidad del código.

2.2 JavaFX & Spring Boot

JavaFX es una aplicación de interfaz gráfica de usuario desarrollada con el lenguaje de programación Java[5]. Los programadores suelen utilizar librerías como AWT, Swings, etc., pero JavaFX incluye todas estas funcionalidades en una única librería y permite a los usuarios diseñar la interfaz mediante la función de arrastrar y soltar a través de la aplicación Scene Builder. El código generado se ha conservado, tal y como se

describe en el capítulo 3, y se puede habilitar si la versión del cliente se reactiva en una futura versión.

Spring Boot es un proyecto construido sobre Spring Framework. Proporciona una manera fácil y rápida de instalar, configurar y ejecutar aplicaciones sencillas basadas en la web[6]. Integra un gran número de configuraciones comunes de bibliotecas de terceros, tanto desde el punto de vista del desarrollador del programa como del usuario, reduciendo el tiempo de instalación y despliegue de varios softwares que necesitaba ejecutar el cliente antes de iniciar. Y la existencia de una gran cantidad de plantillas excelentes para el desarrollo web, que evita la duplicación de trabajo y ahorra tiempo de desarrollo, ha sido una razón por las que el proyecto se trasladó al desarrollo web en una fase posterior.

2.3 Antlr4

Antlr4, conocido como Another Tool for Language Recognition, es una herramienta de generación de gramática de código abierto basada en el desarrollo de Java, capaz de generar gramática correspondiente basada en archivos de reglas gramaticales[7]. En el repositorio oficial de GitHub de Antlr4[3], ya existe una gran cantidad de grammers para diferentes idiomas, y los ASTs correspondientes pueden ser generados por Antlr4 grammer para satisfacer las necesidades de desarrollo de diferentes idiomas.

Este proyecto se ha utilizado el método Listener de Antlr4 para escuchar los diferentes comportamientos de los nodos AST que se necesitan: entrar en el nodo, terminar el nodo. Durante el uso, Antlr4 recorrerá el AST generado, "ParseTreeWalker". Dado que no hay valor de retorno en este modo, se agregan estructuras de datos adicionales para almacenar la información requerida cuando se activa el nodo. Este proyecto debería encajar bien con el simple uso de la función de lenguaje de análisis Antlr4.

[3] https://github.com/antlr/grammars-v4

2.4 Graphviz

GraphViz es un software de visualización de imágenes de código abierto que utiliza DOT como lenguaje de scripting[8], y luego utiliza un motor de diseño para analizar este script y completar el diseño automático. Graphviz también ofrece un formato de exportación enriquecido, como formato de imagen común, SVG, formato PDF, etc.

Como se muestra en la Figura 2-1, DOT, como lenguaje de descripción de imágenes en texto plano, es fácil de aprender y muy fácil de leer. Requiere menos información para construir imágenes y es muy conveniente usarlo para construir CFG como uso auxiliar en este proyecto.

```
digraph G {
    rankdir = LR;
    node [shape = circle, color
= red]
    a -> b[label = "next"];
    b -> c;
}
```

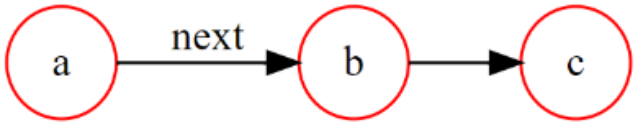


Figura 2-1. Ejemplo de conversión de DOT lenguaje a imagen

2.5 Herramientas generadoras de test automáticos

EvoSuite es una herramienta de código abierto desarrollada conjuntamente por Sheffield y otras universidades para generar automáticamente conjuntos de casos de prueba. Los casos de prueba generados por esta herramienta son compatibles con Junit y pueden ejecutarse directamente en Junit[9]. Una de las características de EvoSuite es la minimización de casos de prueba individuales, sólo se conservan los casos de prueba individuales que contribuyen a la cobertura, lo que mejora en gran medida la eficiencia del desarrollo de los probadores.

Diffblue Cover es una herramienta automatizada de creación de pruebas unitarias de Diffblue, una empresa fundada por la Universidad de Oxford. Analiza las aplicaciones Java existentes y utiliza algoritmos de aprendizaje por refuerzo, código automatizado de inteligencia artificial para generar pruebas unitarias. Diffblue soporta

proyectos estándar de Java 8 y 11, Spring y Spring Boot, donde obtiene la estructura del proyecto y las dependencias a través de Maven y el usuario puede elegir ejecutar una clase o un método[10].

Ambas herramientas tienen plug-ins en IDEA para facilitar su uso. Se compararán en este proyecto y se enumerarán en el capítulo 4.

Capítulo 3 - Descripción de trabajo

Este capítulo se divide en dos secciones principales según el orden de desarrollo del proyecto: dibujo CFG y cobertura de la ruta. El cálculo de los requisitos de test mencionados en el plan de trabajo se fusiona en la sección sobre el cálculo de la cobertura. Ampliaré en detalle la programación realizada durante el proceso de desarrollo, los algoritmos involucrados, el conocimiento y el proceso de implementación de los mismos.

3.1 Generación de CFG

3.1.1 Diseño

Antes de comenzar a escribir el código para el proyecto, he diseñado primero la interfaz de la herramienta basándome en los objetivos del proyecto, con un diseño como el que se muestra en la Figura 3-1, que se divide aproximadamente en tres partes según los requisitos del proyecto.

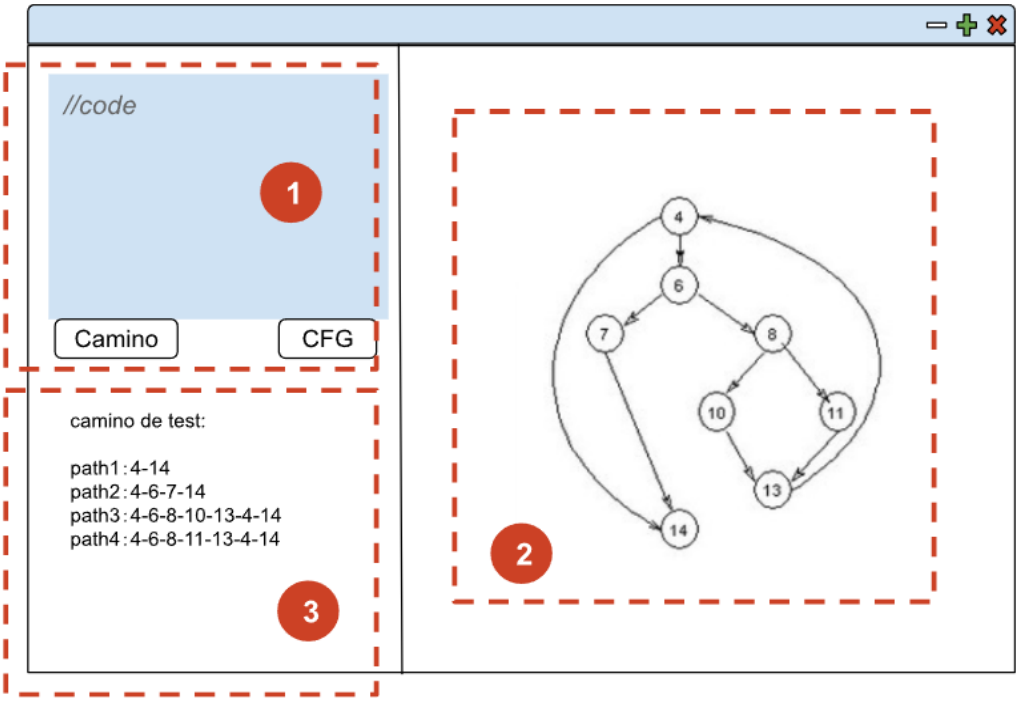


Figura 3-1. Boceto de diagrama de interfaz del proyecto

La Parte 1 es la creación de una entrada de código, donde el usuario rellenará el código a probar y también los botones para controlar cada función. El código de entrada se pasa a Antlr4 para su procesamiento a través de Ajax con el tipo de datos String.

La Parte 2 muestra la imagen CFG generada después del procesamiento.

La Parte 3 es solo una representación aproximada del diseño del proyecto, que es responsable de mostrar la información devuelta por los cálculos de este programa. Esta información se proporciona al json utilizando jsonObject mediante el método put, y luego se pasa a la página web.

3.1.2 CFG

En cuanto a la generación de CFG, para implementar esta parte de la función he tomado como referencia el algoritmo propuesto por Zhang Yan y Lin Ying en el artículo Automatic Generation Algorithm of the Control Flow Graph[11]. El algoritmo propuesto en el artículo se denomina algoritmo PDLToCFG, el cual lo mencionaré como PDLToCFG en las secciones posteriores que hagan referencia al algoritmo.

La estructura de nodos de diseño en PDLToCFG contiene cuatro partes de información: el número de nodo, el tipo de nodo, el número del primer arco con ese nodo como cabeza de arco y el número del primer arco con ese nodo como cola de arco. El arco en el PDLToCFG incluye cinco informaciones: el número de arco; condición de arco: Nulo para la asignación, T representa que la condición es verdadera cuando pasa por este arco, F significa que la condición es falsa cuando pasa por este arco; el siguiente arco que apunta al mismo arco que la cabeza del arco; el siguiente arco que apunta a la cola del arco.

Se crea una pila para registrar como "if", "else", "while", etc. como palabras clave y número de nodo. Luego, durante la conversión del código, el diagrama de flujo de control del programa se obtiene mediante la creación de nodos, la adición de arcos, la actualización del número de nodo actual y las operaciones de salida y entrada de la pila en cada estructura, como se muestra en la Figura 3-2.

El diagrama de flujo de control básico de la estructura IF, tomando como ejemplo el 9, 10, 11 y 12 en la Figura 3-2, 9 es el nodo condicional de IF, 10 y 11 son caminos diferentes hacia el nodo representando la condición verdadera y falsa. El 12 es el nodo de convergencia de la estructura IF, que no tiene ningún código y debe pasar por él en su camino hacia el siguiente nodo. Teniendo en cuenta los requisitos de test posteriores y el cálculo de la cobertura, debido a su propiedad necesariamente transcurrida del nodo sumidero, ya no habrá más nodo sumidero en ninguna estructura de declaración de este proyecto.

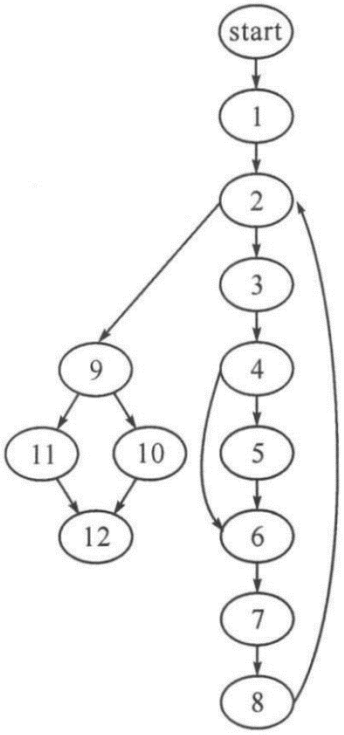


Figura 3-2. CFG obtenido por PDLToCFG

La estructura if y sus estructuras anidadas después de la eliminación del nodo sumidero están vinculadas como la forma que aparece en el Capítulo 4 de GRÁFICOS DE FLUJO DE CONTROL Y COBERTURA DE CÓDIGO de Robert Gold[12]. Este artículo también fusiona las declaraciones de la estructura secuencial, que es útil como referencia en este proyecto. Teniendo en cuenta la fusión de los nodos de estructura secuencial, las estructuras if-branch y sus estructuras anidadas, hay varias rutas que deben conectarse al siguiente nodo de entrada de estructura al mismo tiempo, lo que se convierte en una parte difícil de esta sección y es necesario repensar y construir el algoritmo para generar CFG basado en PDLToCFG.

3.1.2.1 Estructura de datos

En este proyecto, se conservan las tres estructuras de datos básicas originales de PDLToCFG, pero se modifica el contenido de las estructuras, como se muestra en la Figura 3-3, donde node es el número de nodo, type es el tipo de nodo, como "VariableDeclaration", "while", "ifCondition", etc., y value es el contenido del código del nodo. La estructura del arco en este proyecto es solamente para registrar la conectividad de los nodos, esto es también para facilitar el acceso directo a la información cuando se utiliza Graphviz para dibujar. Donde fromNode es el número del

nodo al principio del arco dirigido, toNode es el número del nodo al final del arco y el lable almacena la información.

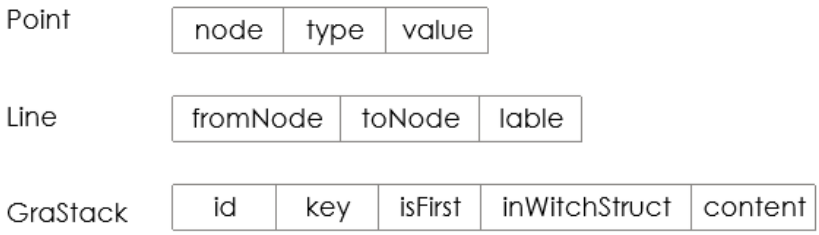


Figura 3-3. Estructura del nodo, el arco y la pila

GraStack se usa para la entrada iterativa de código en if, while, switch y otras estructuras.

Id = Número de nodo

Key = Palabra clave de la estructura

isFirst = Registra si la sentencia es la primera sentencia de la estructura actual

inWitchStruct = Registra el número de nodo de la estructura padre donde se encuentra la estructura actual

content = Almacenar el contenido.

Estas tres estructuras se utilizan para registrar el contenido de value, lable y content respectivamente, aunque no son obligatorios. Se debe determinar qué contenido de código almacenar de acuerdo con la situación real.

3.1.2.2 Implementación

Al implementar el algoritmo, la gramática Antlr4 utilizada en este proyecto agrupa algunas oraciones no estructuradas en "statementWithoutTrailingSubstatement", por lo que utilicé tres estructuras List adicionales: stack, endStack y endAnyStack. Cuando el código que se va a probar entra a cualquier estructura, la estructura de datos GracStack correspondiente se mete en la pila. Cuando Antlr4 sale de la estructura de la rama, inserta la clave de la estructura y el número de nodo de la estructura en endStack. Al final de una iteración completa de cualquier estructura, la clave de la

estructura se registra en endAnyStack, que es utilizada por los nodos subsiguientes cuando se conectan, para determinar si hay una estructura por delante y qué tipo de conexión se necesita. Después de que la conexión se complete, el valor en endAnyStack se borra y se extrae la estructura correspondiente a la pila y endStack. La descripción anterior irá seguida de ejemplos.

El proceso se lleva a cabo de la misma manera que el algoritmo PDLToCFG, utilizando Active para registrar el número de nodo actual y siguiendo el diagrama de flujo en Figura 3-4 cuando Antlr4 pasa al nodo de entrada que debe procesarse. Cuando Antlr4 llega al nodo saliente, realiza el procesamiento out-stack correspondiente como se ha descrito anteriormente. Sin embargo, esto puede variar ligeramente en función de la estructura sintáctica.

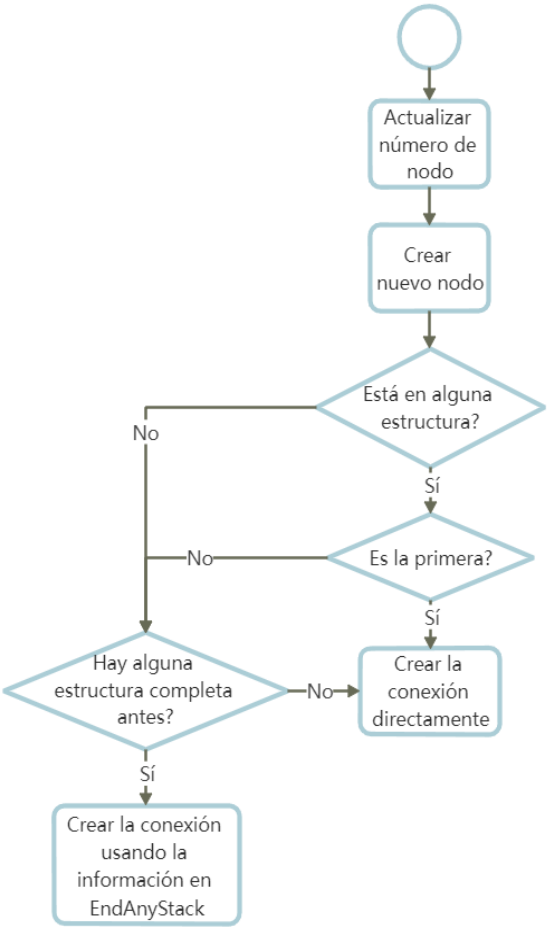


Figura 3-4. Diagrama de flujo

Como ejemplo de una estructura de secuencia y una estructura de if-branch, a continuación, se muestra el pseudocódigo correspondiente:

```
   input: La estructura ctx dada por la declaración que se activa
   Data: e es el primer elemento del stack, active es el número de nodo actual, num es el número de
         nodo
 1 if ctx block not null then
 2 |  type ← Valor key de la e;
 3 end
 4 if type tipo es igual a la estructura especificada then
 5 |  if es la primera expresión de esta estructura then
 6 |  |  num++;
 7 |  |  createNode(num, "Expression", Contenido del código en ctx);
 8 |  |  addEdge(active, num);
 9 |  |  active ← num;
10 |  |  isFirst ← false;
11 |  else
12 |  |  createSimpleNode("Expression", Contenido del código en ctx);
13 |  end
14 else
15 |  // Instrucciones de asignación que no están en estructura;
16 |  if no hay expresión por delante then
17 |  |  num++;
18 |  |  createNode(num, "Expression", Contenido del código en ctx);
19 |  |  addEdge(active, num);
20 |  |  active ← num;
21 |  else
22 |  |  createSimpleNode("Expression", Contenido del código en ctx);
23 |  end
24 end
```

Figura 3-5. enterStatementWithoutTrailingSubstatement

El pseudocódigo de la estructura de secuencia de la Figura 3-5 es un ejemplo de activación "StatementWithoutTrailingSubstatement" en Antlr4. El tipo de declaración de variable tiene una función dedicada responsable de activarla en este proyecto, y el proceso de ejecución es similar a las líneas 17 a 24 del código de la Figura 3-5. Todas las sentencias no estructurales entrarán a la función para ser procesadas. Las líneas 2-14 se tratan de procesar las sentencias simples en la estructura, dependiendo el tipo de estructura en la que se encuentre el código se modificará ligeramente en las líneas 7 a 9, como la primera sentencia debajo de la rama if-else, el "active" en addEdge se reemplazará con la id del primer elemento en la pila a conectar al crear la conexión.

Para la estructura if-branch, la sintaxis utilizada en este proyecto se divide en "If-then" y "if-then-else", y aquí hay un ejemplo de la segunda. Cuando se activa esta estructura, "if-then-else" se divide en "StatementNoShortIf" y "fThenElseStatement", y la Figura 3-6 muestra el pseudocódigo de la rama "ifthen". Dado que la rama ifthen se activa para salir inmediatamente después de esta técnica, no es necesario reescribir la interfaz correspondiente a la rama ifelse.

```
   input: La estructura ctx dada por la declaración que se activa
   Data: e2 es el segundo elemento del stack, active es el número de nodo actual, num es el número
         de nodo
 1 num++;
 2 createNode (num, "IfCondition", "If");
 3 if esta en la estructura then
 4     if es la primera expresión de esta estructura then
 5         isFirst ← false;
 6         stack push nuevo Grastack (num, "Ifthen", "Condition");
 7         // Si key del e2 es Ifelse, el active debe reemplazarse con el id
           del segundo elemento;
 8         addEdge (active, num);
 9     else
10         stack push nuevo Grastack (num, "Ifthen", "Condition");
11         if endAnyStruct está vacío then
12             addEdge (active, num);
13         else
14             while endAnyStruct no está vacío do
15                 addEdge (el id del primer elemento del endstack, num);
16                 pop los elementos correspondientes en endstack y endAnyStruct;
17             end
18             addEdge (active, num);
19         end
20     end
21 else
22     stack push nuevo Grastack (num, "Ifthen", "Condition");
23     addEdge (active, num);
24     Repita el código en las líneas 11-19;
25 end
26 active ← num;
```

Figura 3-6. enterStatementNoShortIf

input: La estructura ctx dada por la declaración que se activa
Data: e es el primer elemento del $stack$, $active$ es el número de nodo actual
1 endstack push nuevo `Grastack` (active, "IFTHENELSE");
2 stack push nuevo `Grastack` (el id del e, "Ifelse", "Ifelse");

Figura 3-7. exitStatementNoShortIf

A diferencia de lo que ocurre al entrar en una rama, las funciones de las dos ramas que salen de if deben reescribirse. La Figura 3-7 muestra la salida de rama "ifthen", y la rama "ifelse" no se muestra aquí, señala el final de un ifthenelse completo, por lo que "endAnyStruct" hace push a "IFTHENELSE" para indicar que la estructura está completa, mientras que stack debe hacer pop dos veces para empujar las dos ramas de if.

3.2 Cálculo de cobertura

3.2.1 Requisitos de test

En el cálculo de los requisitos de test, en este proyecto es el cálculo de los caminos primarios, he optado por seguir usando el método de cálculo que había aprendido en clase: encontrar todos los caminos simples y luego filtrar los caminos primarios de ellos. El pseudocódigo se muestra en la Figura 3-8, donde lineList es la conexión del nodo, es decir, el arco dirigido, como se menciona en 3.1.2, y se obtiene directamente al iterar el código que se va a probar.

Cada vez que se completa la extensión, la ruta desde el nodo inicial hasta llegar al nodo final y la formación de una ruta circular, estos dos como ruta no puede continuar extendiéndose directamente al camino primario. El resto de la ruta elegible para almacenamiento en "simplesPaths", finalmente se hace la colación para eliminar la parte como una sub-ruta con los caminos primarios. En el Capítulo 4 se pueden ver ejemplos de resultados de salida.

```
input  : lineList
output: trList
Data: simplePaths, loopList, aux1 almacena la ruta que se puede extender y aux2 es para rutas
      extendidas
1  do
2      extender aux2 con lineList;
3      if empezar con "start" y terminar con "end" then
4          agregar a trList;
5      else
6          if es un bucle then
7              agregar a loopList;
8              agregar a trList;
9          else
10             if la ruta contiene elementos de looplist then
11                 quitar de la aux2;
12             end
13             if no terminar con "end" then
14                 agregar a aux1;
15             end
16             agregar a simplePaths;
17         end
18     end
19 while existen caminos que se pueden extender;
20 for i ← longitud de simplePaths to 1 do
21     if simplePaths_i no es un subcamino a ningún elemento en trList then
22         agregar la ruta a trList;
23     end
24 end
25 output trList;
```

Figura 3-8. Obtener los requisitos de test

3.2.2 Cobertura

El núcleo del cálculo de la parte de cobertura de requisitos de test es la ejecución dinámica y reflexión de Java. Se agrega un nuevo StringBuild a Antlr4 al atravesar el código para registrarlo. Al activar la creación de nodos, StringBuild almacena el código, así como arrayList.add (correspondiente al número de nodo). Hay que tener en cuenta que el código devuelto después de la iteración no se puede agregar directamente, sino que requiere un pequeño procesamiento, como por ejemplo for(int i = 0; i<x; i++) en el que int i = 0 se extraerá como un nodo de tipo forInit, e i++ se utiliza como un nodo

forUpdate, que necesita ser modificado para tener en cuenta que el código no se escribe repetidamente.

Después de completar la iteración, se crea un nuevo código de función a probar. Cuando se ejecuta el nuevo código, los nodos de código cubiertos por los casos de prueba se agregan a arrayList, con el efecto que se muestra en la Figura 3-9, y finalmente se devuelven en forma de array como ruta de ejecución. Las rutas de ejecución obtenidas se comparan con las rutas de los requisitos de test obtenidos de la forma mencionada en el capítulo 3.2.1 para calcular la cobertura.

```
public static ArrayList runCode() {
    moverUno(new int[0]);
    return arrayList;
}

public static int[] moverUno(int[] var0) {
    arrayList.add(1);
    if (var0 == null) {
        arrayList.add(2);
        throw new NullPointerException("NullPointerException");
    } else {
        arrayList.add(3);
        int[] var1 = new int[var0.length];
        var1[0] = -1;
        arrayList.add(4);
```

Figura 3-9. Código regenerado(parcial)

Dado que los códigos de prueba de los dos softwares se generan en diferentes ubicaciones y la estructura del código de los archivos generados también difiere, los archivos deben leerse por separado. El archivo de prueba generado que se muestra en el Capítulo 4 muestra que el archivo contiene múltiples casos de prueba, por lo que es necesario dividir el archivo de prueba durante la ejecución. Primero por el "@Test" común para el corte inicial, luego se realizan otras divisiones de código según la llamada a la función a probar en cada Unit test, la cual se ejecuta a su vez para obtener la ruta de ejecución.

3.3 Otros

Acerca de la versión de cliente reemplazada, debido a la incapacidad de coexistir con entornos de desarrollo posteriores, esta versión del código se cargó en GitHub como V0.1[4]. El efecto de la interfaz se muestra en la Figura 3-10, la información de texto devuelta por el proyecto se presenta en pestañas, y la visualización de CFG en el lado derecho de la interfaz es diferente de la versión web en que ajusta la posición del panel y el tamaño de las imágenes a través de evento de desplazamiento y evento de arrastre del mouse, que se realiza la función de acercar y alejar la rueda de desplazamiento, así como hacer clic y arrastrar, que es más flexible que la versión web.

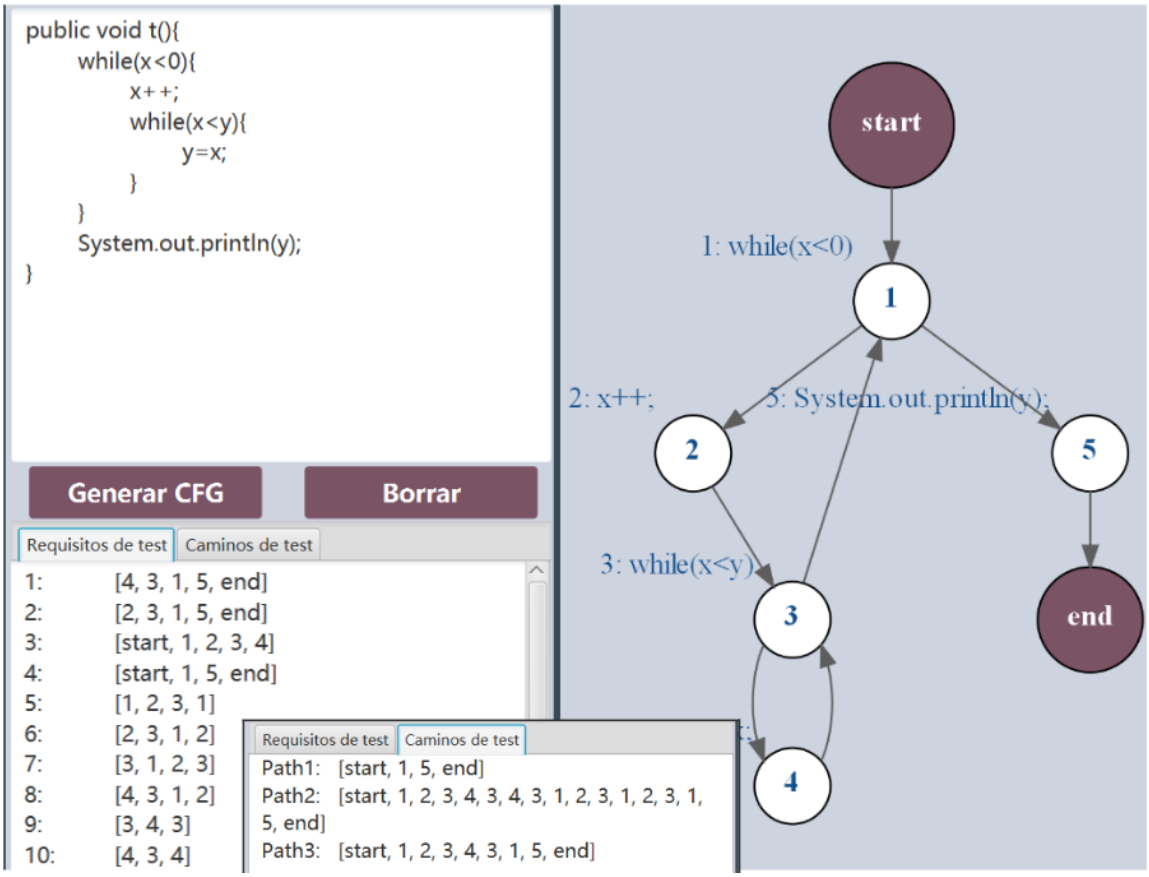


Figura 3-10. Versión del cliente y caminos de test1

[4] https://github.com/luojuee/toolTFM/releases

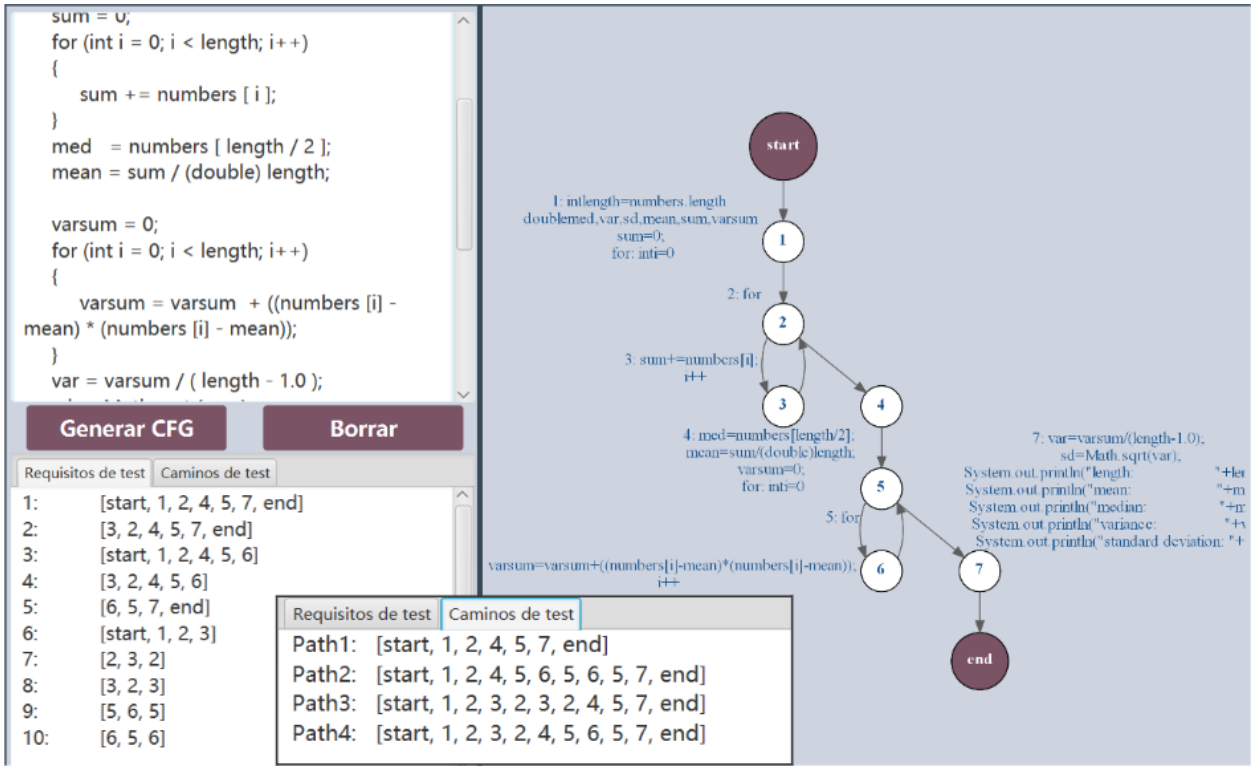


Figura 3-11. Versión del cliente y caminos de test 2

Otra parte no utilizada en esta visión de proyecto: la generación de caminos de prueba cubriendo los caminos primarios. Esta función se refiere al algoritmo propuesto en el artículo Minimum Number of Test Paths for Prime Path and other Structural Coverage Criteria[13] y da una implementación en la que el proceso principal es fusionar las rutas de demanda usando Superstring y unirlas en un nuevo gráfico, luego reduciendo los anillos en el nuevo gráfico. A continuación, se usa el algoritmo de flujo de red Ford-Fulkerson para encontrar las rutas de prueba. Los resultados de las dos ejecuciones de ejemplo se muestran en Figura 3-10 y Figura 3-11 que se publican por separado, ya que el conmutador de pestañas no puede mostrar tanto los requisitos de test como las rutas de prueba en el mismo gráfico.

Capítulo 4 - Experimentos

En este capítulo se muestran los resultados finales del proyecto, con una simple distinción según el código a probar. La forma de utilizar esta herramienta y el código que no se puede mostrar en su totalidad se adjuntan en los apéndices al final de este documento, y las partes que no se pueden ver completamente en las capturas de pantalla también se muestran por separado.

4.1 Código simple

En esta categoría se dividen los códigos relativamente sencillos y cortos, como los del cuadro siguiente. Este tipo de código es simple y muy básico en su estructura, y no hay casos o consideraciones especiales que deban tenerse en cuenta, ni en la primera mitad del proyecto ni posteriormente utilizando la herramienta de generación de casos de prueba automatizados. Por lo tanto, aquí sólo se muestra el while anidado como ejemplo para demostrar los resultados.

```
public int tFor(int y)
{
  int z = 0;
  for (int x = 0; x <
y; x++)
  {
     int aux = y - x;
     z += aux;
  }
  return z;
}
```

```
public int tIfElse(int
x, int y) {
  int z = 0;
  if (x < y){
     z = 1;
  }
  else{
     z = x;
     if (x < 0){
        z++;
     }
  }
  return z;
}
```

```
public int t(int x,
int y) {
    while(x<0){
        x++;
        while(x<y){
            y--;
        }
    }
    return y;
}
```

Tabla 4-1. Código simple

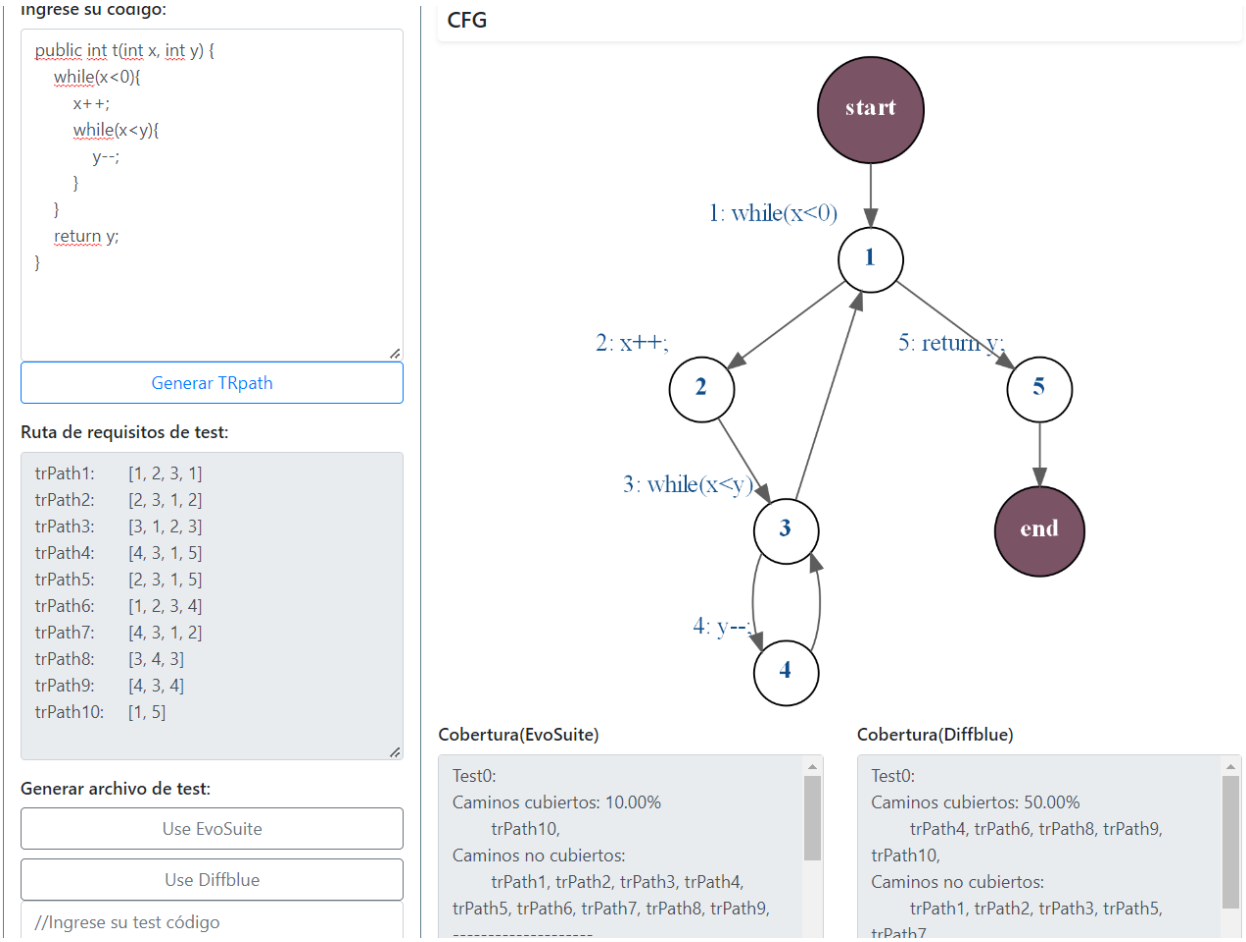


Figura 4-1. el resultado del "while"

- Cobertura y archivo generado por Evosuite：

```
public class Test446_ESTest extends Test446_ESTest_scaffolding {

 @Test(timeout = 4000)
 public void test0()  throws Throwable  {
     int int0 = Test446.t(1085, 1085);
     assertEquals(1085, int0);
     int int1 = Test446.t(1003, 0);
     assertEquals(0, int1);
 }

 @Test(timeout = 4000)
 public void test1()  throws Throwable  {
     Test446.t((-3100), (-3100));
     int int0 = (-683);
     Test446.t((-3100), (-683));
     Test446.t(0, (-683));
     int int1 = 1;
```

```
     Test446.t(1, (-683));
     // Undeclared exception!
     Test446.t((-3099), 0);
 }
}
```

```
Test0:
Caminos cubiertos: 10.00%
        trPath10,
Caminos no cubiertos:
        trPath1, trPath2, trPath3, trPath4, trPath5, trPath6,
trPath7, trPath8, trPath9,
--------------------
Test1:
Caminos cubiertos: 90.00%
        trPath1, trPath2, trPath3, trPath5, trPath6, trPath7,
trPath8, trPath9, trPath10,
Caminos no cubiertos:
        trPath4,
--------------------
```

Tabla 4-2. resultado de "while" - Evosuite

- Cobertura y archivo generado por Diffblue：

```
public class Test446Test {
   /**
    * Method under test: {@link Test446#t(int, int)}
    */
   @Test
   public void testT() {
       // test start with comment
       assertEquals(3, Test446.t(2, 3));
       assertEquals(1, Test446.t(1, 1));
       assertEquals(3, Test446.t(3, 3));
       assertEquals(3, Test446.t(1, 3));
       assertEquals(0, Test446.t(-1, 3));
   }
}
```

```
Test0:
Caminos cubiertos: 50.00%
        trPath4, trPath6, trPath8, trPath9, trPath10,
Caminos no cubiertos:
        trPath1, trPath2, trPath3, trPath5, trPath7,
--------------------
```

Tabla 4-3. resultado de "while" - Diffblue

4.2 Código que lanza la excepción

En esta categoría, cuando se trata de código que contiene Array, y se encuentra con casos en los que los arrays sean Nulos, puede generar excepciones. Por ejemplo, en el caso del moveUno, si el código para hacer test ha tenido en cuenta las excepciones lanzadas y lo tiene controlado, la cobertura se calculará normalmente de acuerdo con la ruta de ejecución del caso de test, véase Test2 en Evosuiten el archivo de generación de Evosuite y Test1 en Diffbule. Si el código no ha tenido en cuenta estas excepciones, el caso de test fallará cuando se ejecute y la cobertura de la prueba mostrará 0.

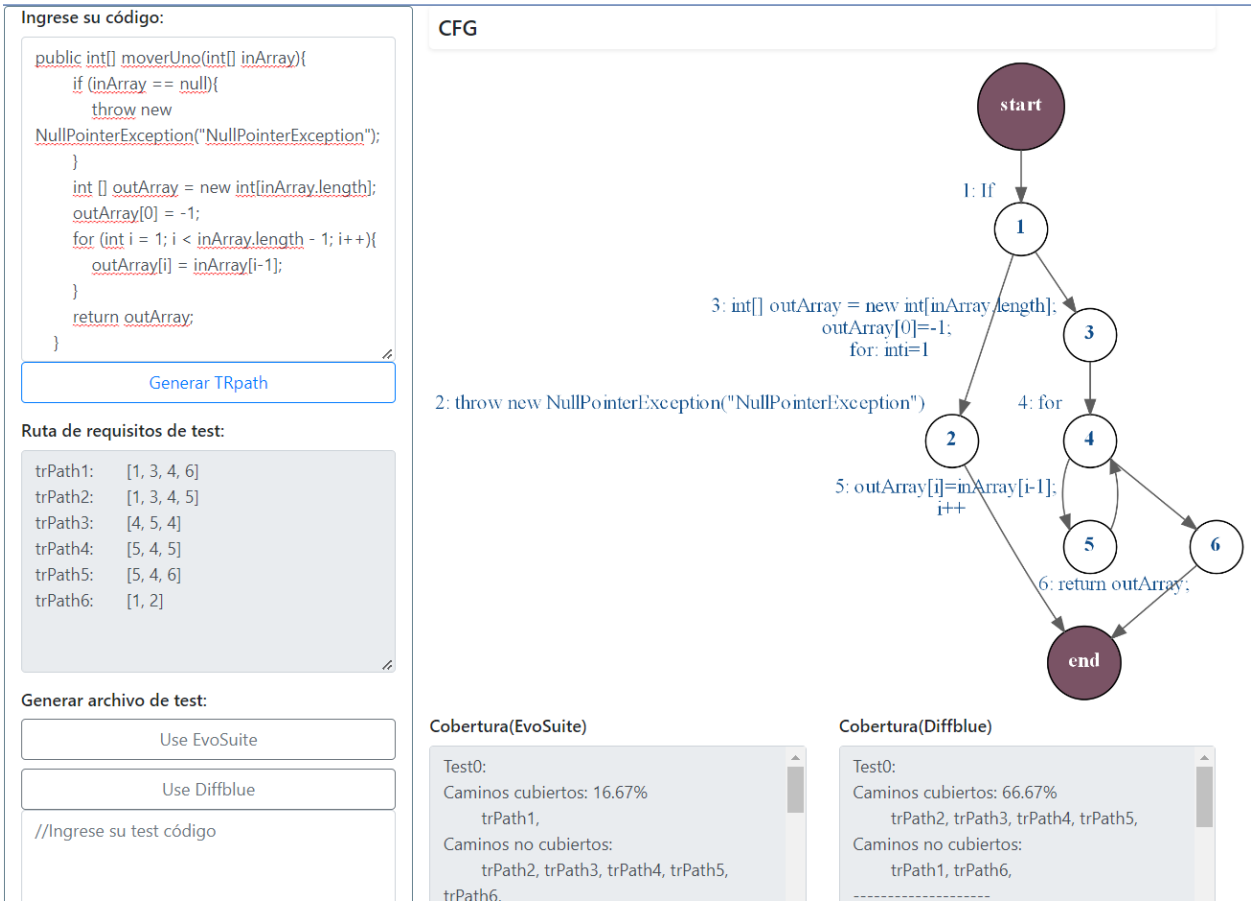


Figura 4-2. el resultado del "moverUno"

- Cobertura y archivo generado por Evosuite：

```
public class Test280_ESTest extends Test280_ESTest_scaffolding {
```

```java
 @Test(timeout = 4000)
 public void test0()  throws Throwable  {
     Test280 test280_0 = new Test280();
     int[] intArray0 = new int[1];
     int[] intArray1 = test280_0.moverUno(intArray0);
     assertArrayEquals(new int[] {(-1)}, intArray1);
 }

 @Test(timeout = 4000)
 public void test1()  throws Throwable  {
     Test280 test280_0 = new Test280();
     int[] intArray0 = new int[0];
     // Undeclared exception!
     try {
       test280_0.moverUno(intArray0);
       fail("Expecting exception: ArrayIndexOutOfBoundsException");

     } catch(ArrayIndexOutOfBoundsException e) {
        //
        // 0
        //
        verifyException("Test280", e);
     }
 }

 @Test(timeout = 4000)
 public void test2()  throws Throwable  {
     Test280 test280_0 = new Test280();
     // Undeclared exception!
     try {
       test280_0.moverUno((int[]) null);
       fail("Expecting exception: NullPointerException");

     } catch(NullPointerException e) {
        //
        // NullPointerException
        //
        verifyException("Test280", e);
     }
 }

 @Test(timeout = 4000)
 public void test3()  throws Throwable  {
     Test280 test280_0 = new Test280();
     int[] intArray0 = new int[5];
     int[] intArray1 = test280_0.moverUno(intArray0);
     assertArrayEquals(new int[] {(-1), 0, 0, 0, 0}, intArray1);
 }
}
```

```
Test0:
Caminos cubiertos: 16.67%
```

```
        trPath1,
Caminos no cubiertos:
        trPath2, trPath3, trPath4, trPath5, trPath6,
--------------------
Test1:
Caminos cubiertos: 0.00%

Caminos no cubiertos:
        trPath1, trPath2, trPath3, trPath4, trPath5, trPath6,
--------------------
Test2:
Caminos cubiertos: 16.67%
        trPath6,
Caminos no cubiertos:
        trPath1, trPath2, trPath3, trPath4, trPath5,
--------------------
Test3:
Caminos cubiertos: 66.67%
        trPath2, trPath3, trPath4, trPath5,
Caminos no cubiertos:
        trPath1, trPath6,
--------------------
```

Tabla 4-4. resultado de "moverUno" - Evosuite

- Cobertura y archivo generado por Diffblue：

```
public class Test280Test {
   /**
    * Method under test: {@link Test280#moverUno(int[])}
    */
   @Test
   public void testMoverUno() {
       int[] actualMoverUnoResult = (new Test280()).moverUno(new
int[]{1, 1, 1, 1});
       assertEquals(4, actualMoverUnoResult.length);
       assertEquals(-1, actualMoverUnoResult[0]);
       assertEquals(1, actualMoverUnoResult[1]);
       assertEquals(1, actualMoverUnoResult[2]);
       assertEquals(0, actualMoverUnoResult[3]);
   }

   /**
    * Method under test: {@link Test280#moverUno(int[])}
    */
   @Test
   @Ignore("TODO: Complete this test")
   public void testMoverUno2() {
       // TODO: Complete this test.
       //   Reason: R013 No inputs found that don't throw a trivial
```

```
exception.
        //   Diffblue Cover tried to run the arrange/act section, but the
method under
        //   test threw
        //   java.lang.NullPointerException: NullPointerException
        //       at Test280.moverUno(Test280.java:6)
        //   In order to prevent moverUno(int[])
        //   from throwing NullPointerException, add constructors or
factory
        //   methods that make it easier to construct fully initialized
objects used in
        //   moverUno(int[]).
        //   See https://diff.blue/R013 to resolve this issue.

        (new Test280()).moverUno(null);
    }

    /**
     * Method under test: {@link Test280#moverUno(int[])}
     */
    @Test
    @Ignore("TODO: Complete this test")
    public void testMoverUno3() {
        // TODO: Complete this test.
        //   Reason: R013 No inputs found that don't throw a trivial
exception.
        //   Diffblue Cover tried to run the arrange/act section, but the
method under
        //   test threw
        //   java.lang.ArrayIndexOutOfBoundsException: 0
        //       at Test280.moverUno(Test280.java:9)
        //   In order to prevent moverUno(int[])
        //   from throwing ArrayIndexOutOfBoundsException, add
constructors or factory
        //   methods that make it easier to construct fully initialized
objects used in
        //   moverUno(int[]).
        //   See https://diff.blue/R013 to resolve this issue.

        (new Test280()).moverUno(new int[]{});
    }
}
```

```
Test0:
Caminos cubiertos: 66.67%
        trPath2, trPath3, trPath4, trPath5,
Caminos no cubiertos:
        trPath1, trPath6,
--------------------
Test1:
Caminos cubiertos: 16.67%
```

```
        trPath6,
Caminos no cubiertos:
        trPath1, trPath2, trPath3, trPath4, trPath5,
--------------------
Test2:
Caminos cubiertos: 0.00%

Caminos no cubiertos:
        trPath1, trPath2, trPath3, trPath4, trPath5, trPath6,
--------------------
```

Tabla 4-5. resultado de "moverUno" - Diffblue

4.3 Código largo

Por último, el código más largo, que tendrá múltiples ramas, lo que también significa que la necesidad de test aumenta y hay que probar más nodos. Aquí se utiliza Triángulo como función a probar, el código completo está en CHAPTER 1 - Chapter 1 - Apéndice B -

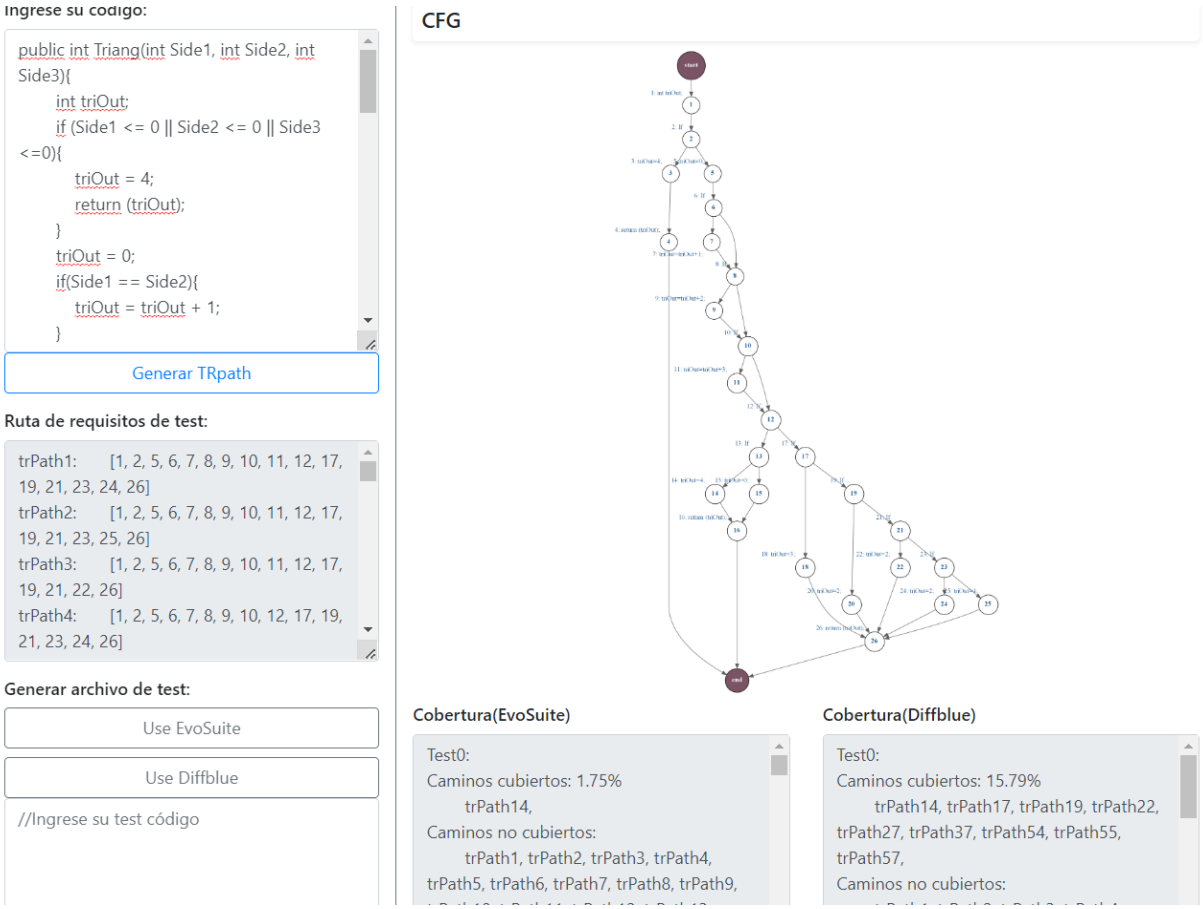


Figura 4-3. el resultado del "Triángulo"

Después de la ejecución del código Triángulo, tanto su ruta de requisitos de prueba como el archivo de prueba generado y sus resultados de cobertura devueltos, son demasiado largos para ser mostrado en esta sección, por lo que he dejado los resultados detallados en el apéndice C

4.4 Diferencias entre Evosuite y DiffBlue

A partir de la cobertura de la salida del conjunto de test anterior, excluyendo el caso de cobertura cero, las pruebas unitarias generadas por Evosuite y DiffBlue no son tan buenas como esperaba en términos de cobertura de los caminos primarios. Parte se debe a que los caminos principales calculados para este proyecto no excluyeron los caminos inalcanzables. La siguiente tabla ofrece una comparación de las dos herramientas para mostrar su similitud o diferencia:

	EvoSuite	Diffblue
Ubicación del test	.evosuite	src/test
El tiempo de generación	Desde unos pocos minutos hasta media hora	Más corto, normalmente menos de un minuto
Cantidad de casos de test	Normal	Mucha
Tipos de casos de test	Validación funcional + verificación de excepciones	Validación funcional
	Normalmente cada prueba constituye una prueba unitaria @Test separada	El mismo tipo de prueba en el mismo @Test unitario
Valores utilizados en los casos de test	Se utilizarán varios valores límite para las pruebas en función del tipo de parámetro	Se adaptan a la función del código
Código a probar sin parámetros ni valor de retorno	Genere casos de prueba sin aserciones	
Casos de test para lanzar excepciones	Todos los tests throws Throwable	Se lanza la misma excepción para el caso de test correspondiente
	Se forman diferentes @Test unitario en función de la excepción	

Tabla 4-6. Comparación de EvoSuite y Diffblue

Con cualquiera de estas dos herramientas, las pruebas que generan son muy buenas en términos de cobertura del código. En cuanto a la cobertura de la ruta principal por parte de las pruebas unitarias, EvoSuite, al igual que los resultados de Triángulo, es muy pobre en términos de cobertura debido al número de pruebas unitarias divididas y a que cada prueba unitaria se trata por separado, en este aspecto DiffBlue es mejor. Sin embargo, desde el punto de vista de todo el archivo de pruebas, es decir, la suma de la base de caminos principales cubierta por todas las pruebas unitarias, EvoSuite es ligeramente mejor que DiffBlue.

Capítulo 5 - Conclusiones y trabajo futuro

Este capítulo es una conclusión de este proyecto, y hace algunas sugerencias de mejoras para un trabajo futuro y ampliaciones basadas en él.

5.1 Conclusiones

Como se ha descrito en los capítulos anteriores, todos los objetivos mencionados en este proyecto se han cumplido. No obstante, en el proceso de conseguirlos, no se puede negar que este proyecto ha dado bastantes rodeos. Antes de pasar a utilizar herramientas de pruebas automatizadas, el desarrollo de este proyecto había sido como establecido en el plan de trabajo, una aplicación cliente desarrollada con JavaFx en un entorno Java 16. Al meter los casos de prueba manualmente se encontró que algunas funciones fueron eliminadas con la actualización de la versión de Java, por lo que el entorno fue bajado a 11, así como el conflicto[5] que hubo con la versión de Java después de usar EvoSuite. Finalmente, he tenido que cambiar la aplicación cliente por la construcción de páginas web con Spring Boot.

La parte de generación de CFG se desarrolló con Antlr4, donde existen varias versiones de java en la librería de sintaxis oficial de Antlr4. Tras prueba y error, se eligió Java 8 y se propuso un algoritmo implementable para generar los vértices y aristas necesarios para dibujar el CFG de forma secuencial a medida que Antlr4 recorre el código de entrada, mientras almacena el código de los nodos y los números de nodo para la posterior ejecución del código de prueba.

En el objetivo final, la diferencia entre EvoSuite y Diffblue en cuanto a la cobertura de los resultados de las pruebas generadas por ellos, como se ha mostrado en la sección anterior, es muy clara. Genera un gran número de casos de prueba, y la mayoría de ellos sólo contienen una prueba. Los dos se complementan muy bien y es muy fácil para el usuario observar la cobertura de la ruta principal y elegir los casos de prueba con los que está satisfecho.

[5] JavaFx requiere Java 11 o superior para ejecutarse, mientras que EvoSuite vincula la versión a jdk1.8, y actualmente EvoSuite soporta la ejecución en plataformas jdk1.9

5.2 Trabajo futuro

Este proyecto tiene un alto factor de escalabilidad, como se mencionó anteriormente Antlr4 proporciona oficialmente una biblioteca de cien lenguajes y permite a los usuarios escribir su propia sintaxis. Este proyecto ya ha propuesto una solución y la ha implementado en Java, y por la misma razón, puede que no sea replicable debido a las diferencias en la estructura del lenguaje, pero creo que la idea se puede aplicar a otros lenguajes, y en el futuro se podrá utilizar sin limitarse a Java.

En segundo lugar, he elegido EvoSuite y Diffblue como las dos únicas herramientas de pruebas automatizadas, pero hay muchas otras herramientas similares en el mercado, como Squaretest y Randoop, que pueden utilizarse para calcular la comparación. Además, estas herramientas también están en una actualización continua y se pueden añadir más para la comparación. En el futuro, es posible que no haya tantos conflictos de versiones, que la compatibilidad mejore y que exista una solución mejor que la que se ofrece en este proyecto.

Por último, este proyecto aportó una solución para generar la ruta de prueba que cubra todo el recorrido de los requisitos de la prueba, pero lo más corta posible. Esta parte del código seguía siendo parcialmente defectuosa y, por tanto, no ha sido utilizado finalmente en este proyecto. Se puede mejorar en una futura versión para utilizar las rutas de prueba obtenidas por este método para generar los casos de prueba correspondientes.

Capítulo 6 - Introduction

The first chapter will introduce this theme as the motivation for the project, presenting the project's objectives and the work plan developed to achieve these.

6.1 Motivation

Software testing, as an important means to ensure software quality and reliability, is one of the most important and indispensable parts of the software development process. With the fast development of computer science and technology, software testing nowadays no longer only means the traditional testing process after the coding completion, it is a continuous process throughout the whole life cycle of each product since its development until it is put into operation. This change is a significant step forward for the information industry: early participation in the development process and the discovery of hidden bugs and vulnerabilities as soon as possible will undoubtedly save the cost of human resources and time required to fix them, and also further reduce the risk of facing difficulties in the final stage of software development due to the accumulation of errors in the early stages.

From the investigation results published in "Software Testing Trends 2022", we can find that API testing and test automation are receiving more and more attention from the profession today. As the mainstream trend of today's software development, microservice architecture, API-oriented development and application, API testing is bound to become one of the most import ant means of verification in the same type of testing. In addition, API testing is also easier to automate from objective conditions.

I was fortunate to register for an optional course on software testing at the undergraduate level and learned some knowledge of the basics related to software testing and gradually became interested in it. During the learning process, I found that when making static analysis of the given program, such as drawing control flow diagrams, testing requirements, main path calculations, and so on, it was done by hand. A slightly longer code often requires double-checking, is quite error-prone, and produces a very large number of drafts. Unfortunately, some of the tools I have found do not always satisfy

the needs and have more or less defects, so I decided to complete a tool that can show the CFG, test requirements, and test case coverage of the program in this project.

6.2 Objectives

The main objective of the project can be divided into three parts: transformation of the code to CFG - > calculation of test requirements - > generation of the test case coverage of the test requirements.

Before starting the first part, it is necessary to simply create an application interface that allows the user to enter the code to be tested, display the textual information returned by the project, and show the generated CFG diagram. Once prepared, we go with the first objective: the technology to iterate through the program under test is initially chosen as Antlr4, which needs to record the nodes and their connections when the program under test encounters the syntax of the specified structure and select the appropriate drawing tool to draw the nodes and edges into a CFG according to the rules. The CFG needs to show some basic information instead of simply 1,2,3,4 to avoid the need for the user to repeatedly compare the input program method.

The second part is basically to transfer the once calculated on paper, step by step derivation is the process into a program, which needs to get the needed part from the information stored in the process of generating the CFG for calculating the test requirement path.

To achieve the third goal, it is necessary to first find a suitable method for not only running the input test cases but also recording the node routes that the test cases run through while executing the user-entered program. Then calculate from the run path to obtain its coverage of the test requirements obtained in the previous objective. After reaching this goal, find a suitable automation test tool to replace the manual input test case module and modify the program to ensure proper operation.

6.3 Relative work

Before starting my project, I looked for a few similar software as a reference. For CFG drawing, I found Eclipse Flow Chart Generator[2], an Eclipse plug-in developed by

a student, which generates a CFG by selecting method and right-clicking to run, calculating the nodes, the number of connections, and the McCabe metric to calculate the cyclomatic complexity.

The CFG nodes generated by this plug-in can be dragged by the user, which is very flexible. However, the weakness is that it converts each line of code it receives into a node, which causes the diagram to become very long and uncompressible when there are multiple successive lines such as declarations, print and other simple statements, generating multiple nodes. It is possible to try to improve them in this development.

For coverage calculation, Cobertura[3] is an open-source coverage tool for Java, based on Jcoverage. It measures code coverage by checking the basic code and observing what code is executed and what code is not executed while the test package is running. Cobertura can optimize code by flagging useless, unexecuted code in addition to finding untested code and discovering bugs. All of this is helpful for me to have the ideas of development.

6.4 Work plan

I chose to develop the project in a waterfall model to ensure its success. The development was divided into four phases, as shown in the gantt chart: Investigation, Desarrollo, Test y modification, and writing paper.

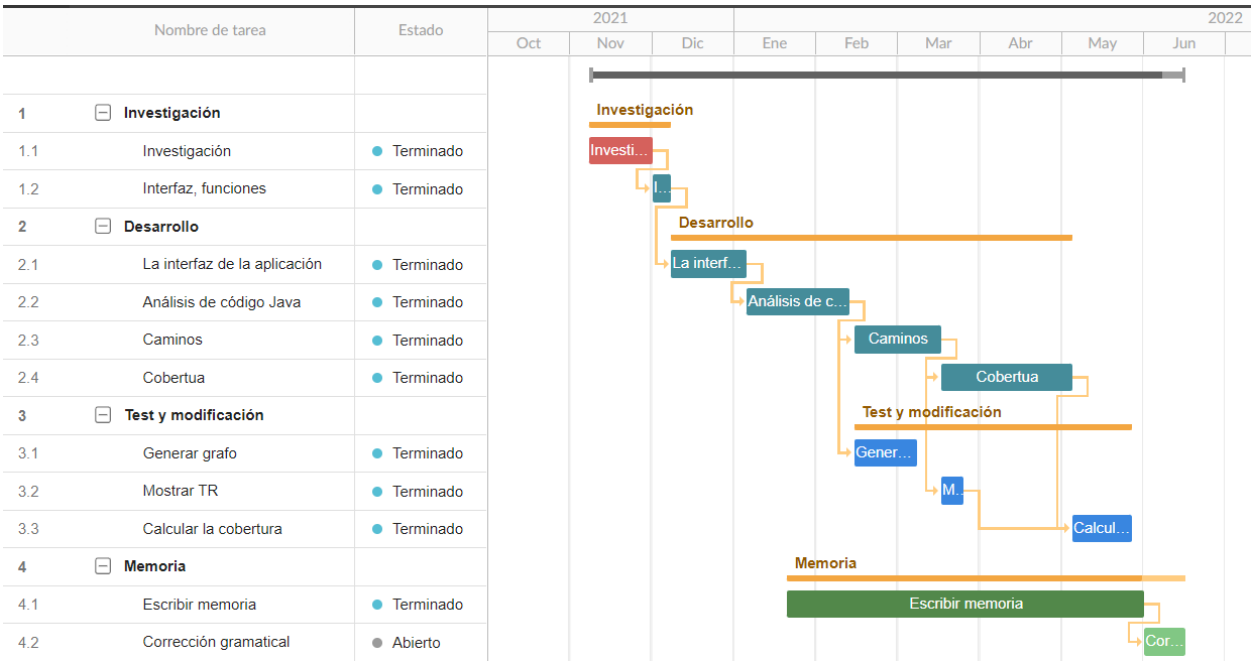


Figura 6-1. Gantt chart

During the investigation phase, I first did a detailed analysis of what I wanted to do for the project. I determined which techniques were needed for the project and which techniques were available for initial study for about 1 month. Of course, part of this time was included to review the knowledge in the software testing course that had been studied. According to the organization and learning in the preparation phase, determine the main techniques to be used and prepare 1-2 alternative options for the objectives mentioned in chapter 6.2. Once the solution has been finalized, create a simple design of the program interface and functions.

Upon analysis of the development purpose and requirements, it was discovered that each module had a clear functional division, and the first goal was the foundation and core to ensure the correct presentation of the whole project, and that the rest of the goals needed to be developed and improved on this basis, so the incremental model was finally adopted for this project. To reduce development risks, after each module is completed, it will be tested on time, and the program will be corrected until it is error-free

before proceeding to the next stage of development and testing. In the meantime, the code for this project will be upload to my Github repository[6].

Finally, the paper will be written throughout the whole software development process until the end, and the development status and program writing ideas will be recorded at any time. In this way, it is possible to make sure that the details of the previous work are not forgotten in the event of sudden changes in one's development. As well, it can facilitate the following to complete the text. Once the paper is finished, it will be left for part of the time for language and syntax corrections.

[6] https://github.com/luojuee/toolTFM

Capítulo 7 - Conclusions and future work

This chapter is a conclusion of this project and makes some suggestions for future improvements and extensions based on it.

7.1 Conclusions

As described in previous chapters, the purposes mentioned in this project have all been achieved. But during the process of reaching them, it is undeniable that this project has taken quite a few roundabouts. Before the project entered into the use of the automated testing tools, this project development was kept as set in the work plan: a client application developed in the Java16 environment using JavaFx. When the manual input of test cases began, it was found that some of the functions were canceled with the increase of Java version, and for this reason the environment was reduced to 11, as well as the Java version conflict appeared again after using EvoSuite. Eventually the client had to be abandoned in favor of building web pages by using Spring Boot.

The CFG generation part was developed based on Antlr4, where multiple java versions exist in the official Antlr4 syntax library, and Java8 was selected by trial and error, and proposed an implementable method to sequentially generate the vertices and edges needed to draw the CFG when Anltr4 traversed the input code, and at the same time store the node code and node number for later test code execution.

On the final objective, the difference between EvoSuite and Diffblue in terms of the coverage results of the tests generated is obvious, as shown in the previous chapter, Diffblue takes a short time to run until the results are generated but a method usually generates only one test case, which contains a large number of tests. In contrast, EvoSuite takes a long time, usually a method generates many test cases, and most of them contain only one test. The two complement each other very well, and it is very easy for users to pick out the test cases they are satisfied with after observing the prime path coverage.

7.2 Trabajo futuro

This project has a great scope for expansion. As mentioned before, Antlr4 officially provides a library of a hundred languages and also supports users to write their own syntax. This project has already proposed a solution and implemented it in Java, and by the same token, this approach may not be applied due to the difference in language structure, but I think this idea can be applied to other languages so that it cannot be limited to Java in the future.

Secondly, this time, only EvoSuite and Diffblue were chosen as the two automated testing tools, there are many similar tools on the market, such as Squaretest, Randoop, etc., which can be used for calculation and comparison. Furthermore, these tools are also being updated, and more such tools can be added for comparison. In the future, there may not be so many version conflicts, and compatibility will be improved, and there should exist a better solution than the one offered in this project.

Finally, this project actually provides a solution to generate the test path that covers the entire route of the test requirements, but as short as possible. This part of the code is still partially flawed so it was not applied in this project in the end. It can be improved in a future version by using the test paths obtained by this method to generate test cases corresponding to them.

Bibliografía

[1] *Software testing trends for 2022.* Blog.csdn.net. (2022). Retrieved 26 July 2022, from https://blog.csdn.net/KerryZhu/article/details/122712121.

[2] *Eclipse Control Flow Graph Generator.* Eclipsefcg.sourceforge.net. (2022). Retrieved 28 July 2022, from http://eclipsefcg.sourceforge.net/index.html.

[3] *Cobertura*. Cobertura.github.io. (2022). Retrieved 28 July 2022, from http://cobertura.github.io/cobertura/.

[4] *IntelliJ IDEA: The Capable & Ergonomic Java IDE by JetBrains.* (2022). Retrieved 28 July 2022, from https://www.jetbrains.com/idea/.

[5] *JavaFX*. Openjfx.io. (2022). Retrieved 28 July 2022, from https://openjfx.io/.

[6] Phillip Webb, S. (2022). *Spring Boot Reference Documentation.* Docs.spring.io. Retrieved 28 July 2022, from https://docs.spring.io/spring-boot/docs/2.7.2/reference/htmlsingle/.

[7] Parr, T. (2013). *The definitive ANTLR 4 reference.* The Definitive ANTLR 4 Reference, 1-326.

[8] Gansner, E., Koutsofios, E., & North, S. (2006). *Drawing graphs with dot.*

[9] About | EvoSuite. Evosuite.org. (2022). Retrieved 1 August 2022, from https://www.evosuite.org/evosuite/.

[10] Diffblue Accelerate Java Shift Left. Diffblue. (2022). Retrieved 1 August 2022, from https://www.diffblue.com/.

[11] Zhang Yan & Lin Ying. (2010). Automatic Generation Algorithm of the Control Flow Graph[J]. Computer and Digital Engineering, 38(2), 28-30. https://doi.org/10.3969/j.issn.1672-9722.2010.02.009

[12] Gold, R. (2010). Control flow graphs and code coverage. International Journal of Applied Mathematics and Computer Science, 20(4), 739-749. https://doi.org/10.2478/v10006-010-0056-9

[13] Dwarakanath, A., Jankiti, A. (2014). Minimum Number of Test Paths for Prime Path and Other Structural Coverage Criteria. In: Merayo, M.G., de Oca, E.M. (eds) Testing Software and Systems. ICTSS 2014. Lecture Notes in Computer Science, vol 8763. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-662-44857-1_5

Apéndices

Apéndice A - El uso de herramientas

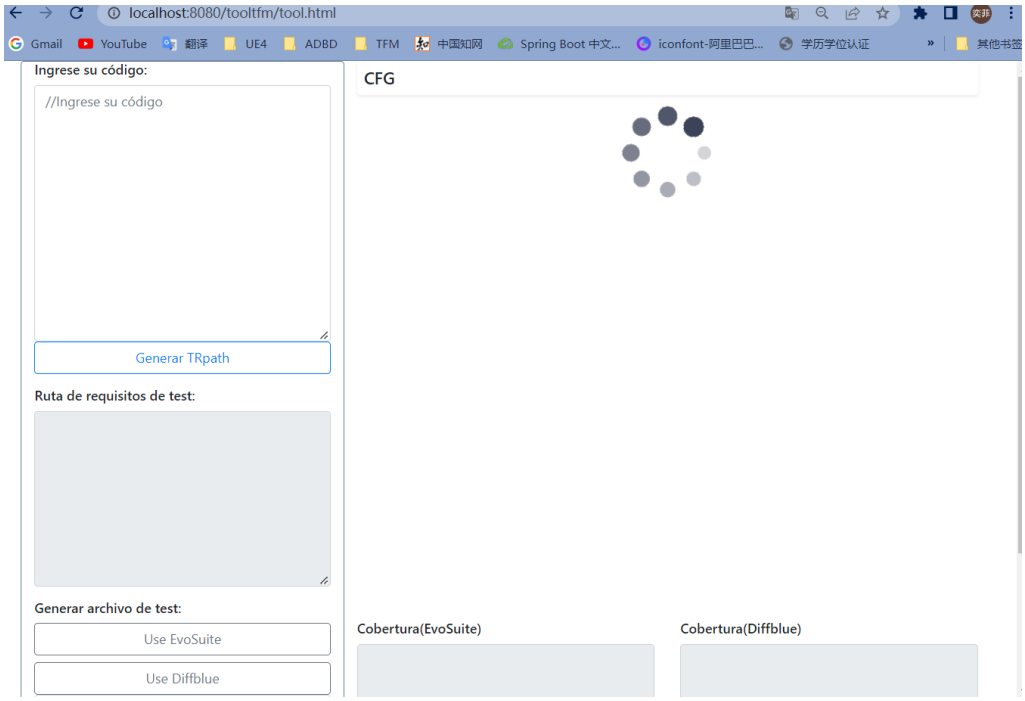
El uso de esta herramienta actualmente requiere que el usuario tenga un entorno de Java 8 en el ordenador, y que tenga instalado Dot en el equipo que se usa para la herramienta GraphViz. El usuario necesita descargar evosuite-1.0.6.jar, no necesita instalarlo, pero tiene que saber la ubicación de la descarga y la ruta a JAVA_HOME en su equipo que se usa para EvoSuit. Los plug-ins EvoSuit, Diffblue y Junit deben instalarse en IDEA.

```
/**
 * Where is your dot program located? It will be called externally.
 */
private static String DOT = "[redacted]/bin/dot";
```

Apéndice A Figura 1. modificación de ruta

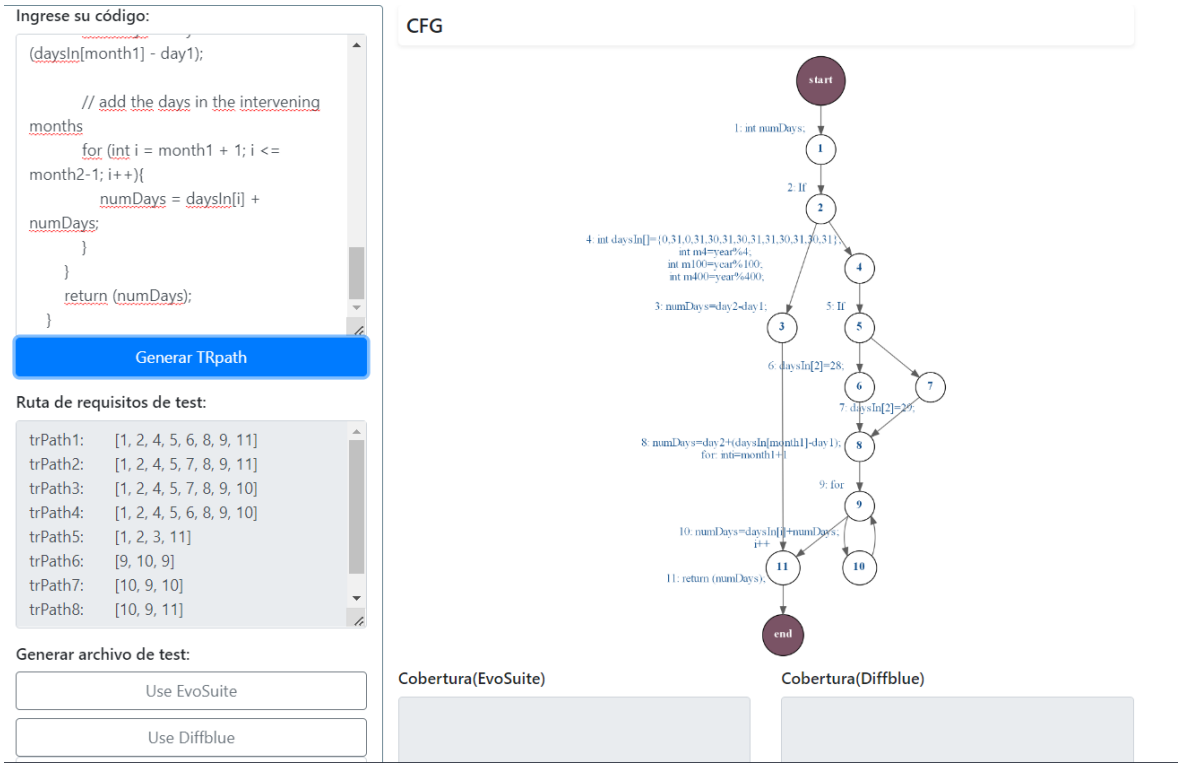
Busque el archivo GraphViz.java en el paquete "modelo" del proyecto y cambie "DOT" a la ruta de instalación. Del mismo modo, busque" javacDir" en TestRun.java bajo el paquete " reflex" y diríjase a la ruta donde está instalado javac en su ordenador.

Después de completar los cambios ejecute este proyecto, esta vez el código de entrada que ejecutamos es código 3 del Apéndice B, a continuación, mostramos la interfaz inicial:



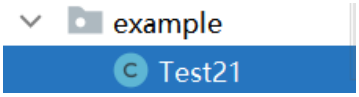
Apéndice A Figura 2. La interfaz inicial

Ingrese el código cal y haga clic en Generar TRpath para generar los requisitos de test (caminos primarios) y CFG.



Apéndice A Figura 3. generar los requisitos de test (caminos primarios) y CFG

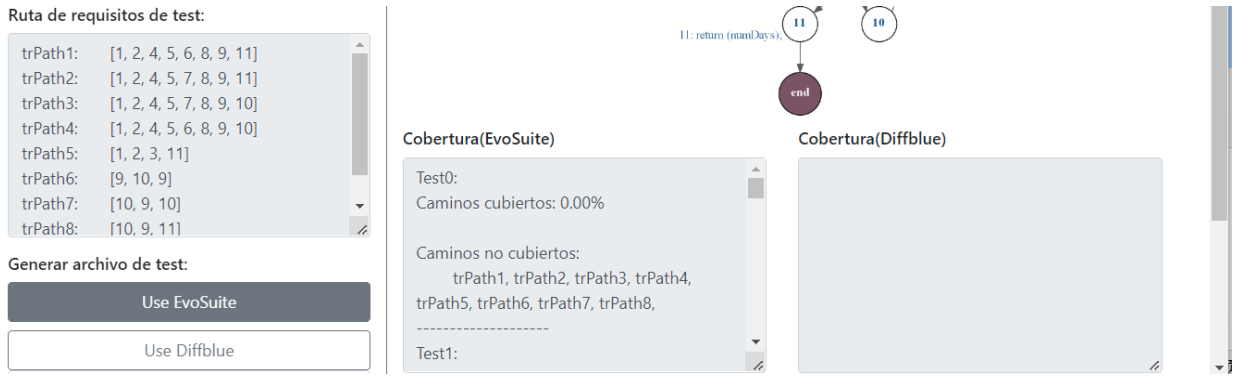
En este punto del ejemplo del proyecto, aparecerá un archivo con el nombre generado aleatoriamente empezando por "Test" en la carpeta "ejemplo" del proyecto. Haga clic con el botón derecho y seleccione Run EvoSuite en la parte inferior para generar automáticamente el archivo del caso de prueba. Tenga en cuenta que el uso de EvoSuit para generar casos de prueba requiere un tiempo de espera muy largo, que va desde unos pocos minutos hasta media hora, y si el proyecto es demasiado grande, puede tardar hasta una hora o más. Una vez finalizado, se creará una nueva carpeta evo, donde se puede encontrar el archivo de prueba recién generado Test21_ESTest.



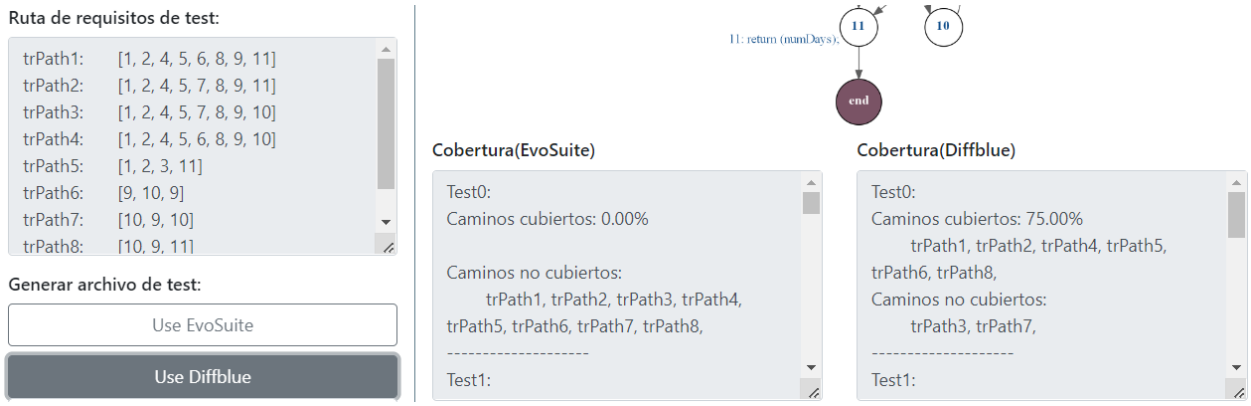
Apéndice A Figura 4. Nuevo file del código de entrada

Haga clic con el botón derecho en Test21 y seleccione "Write Tests" o abra el archivo directamente y haga clic en el ícono pequeño situado en el lado izquierdo del método de "cal" para usar Diffblue y generar automáticamente el archivo del caso de prueba. En poco tiempo aparecerá el archivo Test21Test correspondiente en la carpeta de Test del proyecto.

Vuelva a la interfaz web, seleccione "Use EvoSuite" para extraer el archivo de prueba y calcular la cobertura de los requisitos de test. De la misma manera, seleccione "Use Diffblue" para analizar el archivo correspondiente y mostrar la cobertura en el lado derecho.



Apéndice A Figura 5. Use EvoSuite y calcula la cobertura



Apéndice A Figura 6. Use Diffblue y calcula la cobertura

Apéndice B - Código para probar del capítulo 4

Esta parte proporciona el código original utilizado en algunos de los ejemplos mencionados en el texto principal. Parte del código utilizado para las pruebas proviene del material del curso, parte me lo proporcionó mi tutora y parte se generó durante la realización de algunas tareas.

Código de test 1：computeStats

```
public static void computeStats (int [ ] numbers){
    int length = numbers.length;
    double med, var, sd, mean, sum, varsum;

    sum = 0;
    for (int i = 0; i < length; i++)
    {
         sum += numbers [ i ];
    }
    med   = numbers [ length / 2 ];
    mean = sum / (double) length;

    varsum = 0;
    for (int i = 0; i < length; i++)
    {
         varsum = varsum  + ((numbers [i] - mean) * (numbers [i] - mean));
    }
    var = varsum / ( length - 1.0 );
    sd  = Math.sqrt ( var );

    System.out.println ("length:                    " + length);
    System.out.println ("mean:                     " + mean);
    System.out.println ("median:                   " + med);
    System.out.println ("variance:                 " + var);
    System.out.println ("standard deviation: " + sd);
}
```

Apéndice B Tabla 1. Código - computeStats

Código de test 2：moveUno

```
public int[] moverUno(int[] inArray){
   if (inArray == null){
       throw new NullPointerException("NullPointerException");
   }
   int [] outArray = new int[inArray.length];
   outArray[0] = -1;
   for (int i = 1; i < inArray.length - 1; i++){
       outArray[i] = inArray[i-1];
   }
   return outArray;
}
```

Apéndice B Tabla 2. Código - moveUno

Código de test 3：cal

```
public int cal (int month1, int day1, int month2, int day2, int year)
{
   //*********************************************************
   // Calculate the number of Days between the two given days in
   // the same year.
   // preconditions : day1 and day2 must be in same year
   //                 1 <= month1, month2 <= 12
   //                 1 <= day1, day2 <= 31
   //                 month1 <= month2
   //                 The range for year: 1 ... 10000
   //*********************************************************
   int numDays;
   if (month2 == month1){ // in the same month
       numDays  = day2 - day1;
   }
   else
   {
       // Skip month 0.
       int daysIn[] = {0, 31, 0, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
       // Are we in a leap year?
       int m4 = year % 4;
       int m100 = year % 100;
       int m400 = year % 400;
        if ((m4 != 0)) || ((m100) == 0) && ((m400= 0))){
            daysIn[2] = 28;
        }
        else {
            daysIn[2] = 29;
        }
        // start with days in the two months
        numDays = day2 + (daysIn[month1] - day1)
```

```
        // add the days in the intervening months
        for (int i = month1 + 1; i <= month2-1; i++){
            numDays = daysIn[i] + numDays;
        }
    }
    return (numDays);
}
```

Apéndice B Tabla 3. Código - cal

Código de test 4：Triángulo

```
public int Triang(int Side1, int Side2, int Side3){
   int triOut;
   if (Side1 <= 0 || Side2 <= 0 || Side3 <=0){
       triOut = 4;
       return (triOut);
   }
   triOut = 0;
   if(Side1 == Side2){
       triOut = triOut + 1;
   }
   if (Side1 == Side3){
       triOut = triOut + 2;
   }
   if (Side2 == Side3){
       triOut = triOut + 3;
   }

   if (triOut == 0){
       if (Side1+Side2 <= Side3 || Side2+Side3 <= Side1 || Side1+Side3 <=
Side2){
           triOut = 4;
       }
       else{
           triOut = 0;
       }
       return (triOut);
   }

   if (triOut > 3){
       triOut = 3;
   }
   else if (triOut == 1 && Side1+Side2 > Side3){
       triOut = 2;
   }
```

```
    else if (triOut == 2 && Side1+Side3 > Side2){
        triOut = 2;
    }
    else if (triOut == 3 && Side2+Side3 > Side1){
        triOut = 2;
    }
    else {
        triOut = 4;
    }
    return (triOut);
}
```

Apéndice B Tabla 4. Código – Triángulo

Apéndice C - El resultado del uso código

Triángulo

- Ruta de requisitos de test:

```
trPath1:    [1, 2, 5, 6, 7, 8, 9, 10, 11, 12, 17, 19, 21, 23, 24, 26]
trPath2:    [1, 2, 5, 6, 7, 8, 9, 10, 11, 12, 17, 19, 21, 23, 25, 26]
trPath3:    [1, 2, 5, 6, 7, 8, 9, 10, 11, 12, 17, 19, 21, 22, 26]
trPath4:    [1, 2, 5, 6, 7, 8, 9, 10, 12, 17, 19, 21, 23, 24, 26]
trPath5:    [1, 2, 5, 6, 7, 8, 9, 10, 12, 17, 19, 21, 23, 25, 26]
trPath6:    [1, 2, 5, 6, 7, 8, 10, 11, 12, 17, 19, 21, 23, 24, 26]
trPath7:    [1, 2, 5, 6, 7, 8, 10, 11, 12, 17, 19, 21, 23, 25, 26]
trPath8:    [1, 2, 5, 6, 8, 9, 10, 11, 12, 17, 19, 21, 23, 24, 26]
trPath9:    [1, 2, 5, 6, 8, 9, 10, 11, 12, 17, 19, 21, 23, 25, 26]
trPath10:   [1, 2, 5, 6, 7, 8, 9, 10, 11, 12, 17, 19, 20, 26]
trPath11:   [1, 2, 5, 6, 7, 8, 9, 10, 12, 17, 19, 21, 22, 26]
trPath12:   [1, 2, 5, 6, 7, 8, 10, 11, 12, 17, 19, 21, 22, 26]
trPath13:   [1, 2, 5, 6, 7, 8, 10, 12, 17, 19, 21, 23, 24, 26]
trPath14:   [1, 2, 5, 6, 7, 8, 10, 12, 17, 19, 21, 23, 25, 26]
trPath15:   [1, 2, 5, 6, 8, 9, 10, 11, 12, 17, 19, 21, 22, 26]
trPath16:   [1, 2, 5, 6, 8, 9, 10, 12, 17, 19, 21, 23, 24, 26]
trPath17:   [1, 2, 5, 6, 8, 9, 10, 12, 17, 19, 21, 23, 25, 26]
trPath18:   [1, 2, 5, 6, 8, 10, 11, 12, 17, 19, 21, 23, 24, 26]
trPath19:   [1, 2, 5, 6, 8, 10, 11, 12, 17, 19, 21, 23, 25, 26]
trPath20:   [1, 2, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 16]
trPath21:   [1, 2, 5, 6, 7, 8, 9, 10, 11, 12, 13, 15, 16]
trPath22:   [1, 2, 5, 6, 7, 8, 9, 10, 11, 12, 17, 18, 26]
trPath23:   [1, 2, 5, 6, 7, 8, 9, 10, 12, 17, 19, 20, 26]
trPath24:   [1, 2, 5, 6, 7, 8, 10, 11, 12, 17, 19, 20, 26]
trPath25:   [1, 2, 5, 6, 7, 8, 10, 12, 17, 19, 21, 22, 26]
trPath26:   [1, 2, 5, 6, 8, 9, 10, 11, 12, 17, 19, 20, 26]
trPath27:   [1, 2, 5, 6, 8, 9, 10, 12, 17, 19, 21, 22, 26]
trPath28:   [1, 2, 5, 6, 8, 10, 11, 12, 17, 19, 21, 22, 26]
trPath29:   [1, 2, 5, 6, 8, 10, 12, 17, 19, 21, 23, 24, 26]
trPath30:   [1, 2, 5, 6, 8, 10, 12, 17, 19, 21, 23, 25, 26]
trPath31:   [1, 2, 5, 6, 7, 8, 9, 10, 12, 13, 14, 16]
trPath32:   [1, 2, 5, 6, 7, 8, 9, 10, 12, 13, 15, 16]
trPath33:   [1, 2, 5, 6, 7, 8, 9, 10, 12, 17, 18, 26]
trPath34:   [1, 2, 5, 6, 7, 8, 10, 11, 12, 13, 14, 16]
trPath35:   [1, 2, 5, 6, 7, 8, 10, 11, 12, 13, 15, 16]
trPath36:   [1, 2, 5, 6, 7, 8, 10, 11, 12, 17, 18, 26]
trPath37:   [1, 2, 5, 6, 7, 8, 10, 12, 17, 19, 20, 26]
trPath38:   [1, 2, 5, 6, 8, 9, 10, 11, 12, 13, 14, 16]
trPath39:   [1, 2, 5, 6, 8, 9, 10, 11, 12, 13, 15, 16]
trPath40:   [1, 2, 5, 6, 8, 9, 10, 11, 12, 17, 18, 26]
trPath41:   [1, 2, 5, 6, 8, 9, 10, 12, 17, 19, 20, 26]
trPath42:   [1, 2, 5, 6, 8, 10, 11, 12, 17, 19, 20, 26]
trPath43:   [1, 2, 5, 6, 8, 10, 12, 17, 19, 21, 22, 26]
```

```
trPath44:   [1, 2, 5, 6, 7, 8, 10, 12, 13, 14, 16]
trPath45:   [1, 2, 5, 6, 7, 8, 10, 12, 13, 15, 16]
trPath46:   [1, 2, 5, 6, 7, 8, 10, 12, 17, 18, 26]
trPath47:   [1, 2, 5, 6, 8, 9, 10, 12, 13, 14, 16]
trPath48:   [1, 2, 5, 6, 8, 9, 10, 12, 13, 15, 16]
trPath49:   [1, 2, 5, 6, 8, 9, 10, 12, 17, 18, 26]
trPath50:   [1, 2, 5, 6, 8, 10, 11, 12, 13, 14, 16]
trPath51:   [1, 2, 5, 6, 8, 10, 11, 12, 13, 15, 16]
trPath52:   [1, 2, 5, 6, 8, 10, 11, 12, 17, 18, 26]
trPath53:   [1, 2, 5, 6, 8, 10, 12, 17, 19, 20, 26]
trPath54:   [1, 2, 5, 6, 8, 10, 12, 13, 14, 16]
trPath55:   [1, 2, 5, 6, 8, 10, 12, 13, 15, 16]
trPath56:   [1, 2, 5, 6, 8, 10, 12, 17, 18, 26]
trPath57:   [1, 2, 3, 4]
```

Apéndice C Tabla 1. Ruta de requisitos de test - "Triángulo"

- Cobertura y archivo generado por Evosuite：

```
@Test(timeout = 4000)
public void test00()  throws Throwable  {
   int int0 = Test371.Triang(2, 2, 155);
   assertEquals(4, int0);
}

@Test(timeout = 4000)
public void test01()  throws Throwable  {
   int int0 = Test371.Triang(1, 3, 2);
   assertEquals(4, int0);
}

@Test(timeout = 4000)
public void test02()  throws Throwable  {
   int int0 = Test371.Triang(4, 1, 3);
   assertEquals(4, int0);
}

@Test(timeout = 4000)
public void test03()  throws Throwable  {
   int int0 = Test371.Triang(3, 1, 4);
   assertEquals(4, int0);
}

@Test(timeout = 4000)
public void test04()  throws Throwable  {
   int int0 = Test371.Triang(4, 4, 0);
   assertEquals(4, int0);
```

```
}

@Test(timeout = 4000)
public void test05()  throws Throwable  {
   int int0 = Test371.Triang(323, 0, 2);
   assertEquals(4, int0);
}

@Test(timeout = 4000)
public void test06()  throws Throwable  {
   int int0 = Test371.Triang(0, (-381), (-810));
   assertEquals(4, int0);
}

@Test(timeout = 4000)
public void test07()  throws Throwable  {
   int int0 = Test371.Triang(2, 4, 4);
   assertEquals(2, int0);
}

@Test(timeout = 4000)
public void test08()  throws Throwable  {
   int int0 = Test371.Triang(588, 2, 588);
   assertEquals(2, int0);
}

@Test(timeout = 4000)
public void test09()  throws Throwable  {
   int int0 = Test371.Triang(2, 2, 4);
   assertEquals(4, int0);
}

@Test(timeout = 4000)
public void test10()  throws Throwable  {
   int int0 = Test371.Triang(4, 4, 4);
   assertEquals(3, int0);
}

@Test(timeout = 4000)
public void test11()  throws Throwable  {
   int int0 = Test371.Triang(2, 3, 4);
   assertEquals(0, int0);
}

@Test(timeout = 4000)
```

```
public void test12()  throws Throwable  {
   int int0 = Test371.Triang(1940, 4, 99);
   assertEquals(4, int0);
}

@Test(timeout = 4000)
public void test13()  throws Throwable  {
   int int0 = Test371.Triang(4, 588, 2);
   assertEquals(4, int0);
}

@Test(timeout = 4000)
public void test14()  throws Throwable  {
   int int0 = Test371.Triang(4, 588, 4273);
   assertEquals(4, int0);
}

@Test(timeout = 4000)
public void test15()  throws Throwable  {
   int int0 = Test371.Triang(4273, 588, 588);
   assertEquals(4, int0);
}

@Test(timeout = 4000)
public void test16()  throws Throwable  {
   int int0 = Test371.Triang(588, 588, 4);
   assertEquals(2, int0);
}

@Test(timeout = 4000)
public void test17()  throws Throwable  {
   int int0 = Test371.Triang(1940, 1940, (-105));
   assertEquals(4, int0);
}

@Test(timeout = 4000)
public void test18()  throws Throwable  {
   int int0 = Test371.Triang(4, (-2154), 2517);
   assertEquals(4, int0);
}

@Test(timeout = 4000)
public void test19()  throws Throwable  {
   int int0 = Test371.Triang(4, 1652, 4);
   assertEquals(4, int0);
```

```
}

@Test(timeout = 4000)
public void test20()  throws Throwable  {
   int int0 = Test371.Triang((-1), 1740, (-333));
   assertEquals(4, int0);
}

@Test(timeout = 4000)
public void test21()  throws Throwable  {
   Test371 test371_0 = new Test371();
}
```

```
Test0:
Caminos cubiertos: 1.75%
        trPath14,
Caminos no cubiertos:
        trPath1, trPath2, trPath3, trPath4, trPath5, trPath6,
trPath7, trPath8, trPath9, trPath10, trPath11, trPath12, trPath13,
trPath15, trPath16, trPath17, trPath18, trPath19, trPath20,
trPath21, trPath22, trPath23, trPath24, trPath25, trPath26,
trPath27, trPath28, trPath29, trPath30, trPath31, trPath32,
trPath33, trPath34, trPath35, trPath36, trPath37, trPath38,
trPath39, trPath40, trPath41, trPath42, trPath43, trPath44,
trPath45, trPath46, trPath47, trPath48, trPath49, trPath50,
trPath51, trPath52, trPath53, trPath54, trPath55, trPath56,
trPath57,
--------------------
Test1:
Caminos cubiertos: 1.75%
        trPath54,
Caminos no cubiertos:
        trPath1, trPath2, trPath3, trPath4, trPath5, trPath6,
trPath7, trPath8, trPath9, trPath10, trPath11, trPath12, trPath13,
trPath14, trPath15, trPath16, trPath17, trPath18, trPath19,
trPath20, trPath21, trPath22, trPath23, trPath24, trPath25,
trPath26, trPath27, trPath28, trPath29, trPath30, trPath31,
trPath32, trPath33, trPath34, trPath35, trPath36, trPath37,
trPath38, trPath39, trPath40, trPath41, trPath42, trPath43,
trPath44, trPath45, trPath46, trPath47, trPath48, trPath49,
trPath50, trPath51, trPath52, trPath53, trPath55, trPath56,
trPath57,
--------------------
Test2:
```

```
Caminos cubiertos: 1.75%
        trPath54,
Caminos no cubiertos:
        trPath1, trPath2, trPath3, trPath4, trPath5, trPath6,
trPath7, trPath8, trPath9, trPath10, trPath11, trPath12, trPath13,
trPath14, trPath15, trPath16, trPath17, trPath18, trPath19,
trPath20, trPath21, trPath22, trPath23, trPath24, trPath25,
trPath26, trPath27, trPath28, trPath29, trPath30, trPath31,
trPath32, trPath33, trPath34, trPath35, trPath36, trPath37,
trPath38, trPath39, trPath40, trPath41, trPath42, trPath43,
trPath44, trPath45, trPath46, trPath47, trPath48, trPath49,
trPath50, trPath51, trPath52, trPath53, trPath55, trPath56,
trPath57,
--------------------
Test3:
Caminos cubiertos: 1.75%
        trPath54,
Caminos no cubiertos:
        trPath1, trPath2, trPath3, trPath4, trPath5, trPath6,
trPath7, trPath8, trPath9, trPath10, trPath11, trPath12, trPath13,
trPath14, trPath15, trPath16, trPath17, trPath18, trPath19,
trPath20, trPath21, trPath22, trPath23, trPath24, trPath25,
trPath26, trPath27, trPath28, trPath29, trPath30, trPath31,
trPath32, trPath33, trPath34, trPath35, trPath36, trPath37,
trPath38, trPath39, trPath40, trPath41, trPath42, trPath43,
trPath44, trPath45, trPath46, trPath47, trPath48, trPath49,
trPath50, trPath51, trPath52, trPath53, trPath55, trPath56,
trPath57,
--------------------
Test4:
Caminos cubiertos: 1.75%
        trPath57,
Caminos no cubiertos:
        trPath1, trPath2, trPath3, trPath4, trPath5, trPath6,
trPath7, trPath8, trPath9, trPath10, trPath11, trPath12, trPath13,
trPath14, trPath15, trPath16, trPath17, trPath18, trPath19,
trPath20, trPath21, trPath22, trPath23, trPath24, trPath25,
trPath26, trPath27, trPath28, trPath29, trPath30, trPath31,
trPath32, trPath33, trPath34, trPath35, trPath36, trPath37,
trPath38, trPath39, trPath40, trPath41, trPath42, trPath43,
trPath44, trPath45, trPath46, trPath47, trPath48, trPath49,
trPath50, trPath51, trPath52, trPath53, trPath54, trPath55,
trPath56,
```

```
--------------------
Test5:
Caminos cubiertos: 1.75%
        trPath57,
Caminos no cubiertos:
        trPath1, trPath2, trPath3, trPath4, trPath5, trPath6,
trPath7, trPath8, trPath9, trPath10, trPath11, trPath12, trPath13,
trPath14, trPath15, trPath16, trPath17, trPath18, trPath19,
trPath20, trPath21, trPath22, trPath23, trPath24, trPath25,
trPath26, trPath27, trPath28, trPath29, trPath30, trPath31,
trPath32, trPath33, trPath34, trPath35, trPath36, trPath37,
trPath38, trPath39, trPath40, trPath41, trPath42, trPath43,
trPath44, trPath45, trPath46, trPath47, trPath48, trPath49,
trPath50, trPath51, trPath52, trPath53, trPath54, trPath55,
trPath56,
--------------------
Test6:
Caminos cubiertos: 1.75%
        trPath57,
Caminos no cubiertos:
        trPath1, trPath2, trPath3, trPath4, trPath5, trPath6,
trPath7, trPath8, trPath9, trPath10, trPath11, trPath12, trPath13,
trPath14, trPath15, trPath16, trPath17, trPath18, trPath19,
trPath20, trPath21, trPath22, trPath23, trPath24, trPath25,
trPath26, trPath27, trPath28, trPath29, trPath30, trPath31,
trPath32, trPath33, trPath34, trPath35, trPath36, trPath37,
trPath38, trPath39, trPath40, trPath41, trPath42, trPath43,
trPath44, trPath45, trPath46, trPath47, trPath48, trPath49,
trPath50, trPath51, trPath52, trPath53, trPath54, trPath55,
trPath56,
--------------------
Test7:
Caminos cubiertos: 0.00%

Caminos no cubiertos:
        trPath1, trPath2, trPath3, trPath4, trPath5, trPath6,
trPath7, trPath8, trPath9, trPath10, trPath11, trPath12, trPath13,
trPath14, trPath15, trPath16, trPath17, trPath18, trPath19,
trPath20, trPath21, trPath22, trPath23, trPath24, trPath25,
trPath26, trPath27, trPath28, trPath29, trPath30, trPath31,
trPath32, trPath33, trPath34, trPath35, trPath36, trPath37,
trPath38, trPath39, trPath40, trPath41, trPath42, trPath43,
trPath44, trPath45, trPath46, trPath47, trPath48, trPath49,
```

```
trPath50, trPath51, trPath52, trPath53, trPath54, trPath55,
trPath56, trPath57,
--------------------
Test8:
Caminos cubiertos: 1.75%
        trPath27,
Caminos no cubiertos:
        trPath1, trPath2, trPath3, trPath4, trPath5, trPath6,
trPath7, trPath8, trPath9, trPath10, trPath11, trPath12, trPath13,
trPath14, trPath15, trPath16, trPath17, trPath18, trPath19,
trPath20, trPath21, trPath22, trPath23, trPath24, trPath25,
trPath26, trPath28, trPath29, trPath30, trPath31, trPath32,
trPath33, trPath34, trPath35, trPath36, trPath37, trPath38,
trPath39, trPath40, trPath41, trPath42, trPath43, trPath44,
trPath45, trPath46, trPath47, trPath48, trPath49, trPath50,
trPath51, trPath52, trPath53, trPath54, trPath55, trPath56,
trPath57,
--------------------
Test9:
Caminos cubiertos: 1.75%
        trPath14,
Caminos no cubiertos:
        trPath1, trPath2, trPath3, trPath4, trPath5, trPath6,
trPath7, trPath8, trPath9, trPath10, trPath11, trPath12, trPath13,
trPath15, trPath16, trPath17, trPath18, trPath19, trPath20,
trPath21, trPath22, trPath23, trPath24, trPath25, trPath26,
trPath27, trPath28, trPath29, trPath30, trPath31, trPath32,
trPath33, trPath34, trPath35, trPath36, trPath37, trPath38,
trPath39, trPath40, trPath41, trPath42, trPath43, trPath44,
trPath45, trPath46, trPath47, trPath48, trPath49, trPath50,
trPath51, trPath52, trPath53, trPath54, trPath55, trPath56,
trPath57,
--------------------
Test10:
Caminos cubiertos: 1.75%
        trPath22,
Caminos no cubiertos:
        trPath1, trPath2, trPath3, trPath4, trPath5, trPath6,
trPath7, trPath8, trPath9, trPath10, trPath11, trPath12, trPath13,
trPath14, trPath15, trPath16, trPath17, trPath18, trPath19,
trPath20, trPath21, trPath23, trPath24, trPath25, trPath26,
trPath27, trPath28, trPath29, trPath30, trPath31, trPath32,
trPath33, trPath34, trPath35, trPath36, trPath37, trPath38,
```

```
trPath39, trPath40, trPath41, trPath42, trPath43, trPath44,
trPath45, trPath46, trPath47, trPath48, trPath49, trPath50,
trPath51, trPath52, trPath53, trPath54, trPath55, trPath56,
trPath57,
--------------------
Test11:
Caminos cubiertos: 1.75%
        trPath55,
Caminos no cubiertos:
        trPath1, trPath2, trPath3, trPath4, trPath5, trPath6,
trPath7, trPath8, trPath9, trPath10, trPath11, trPath12, trPath13,
trPath14, trPath15, trPath16, trPath17, trPath18, trPath19,
trPath20, trPath21, trPath22, trPath23, trPath24, trPath25,
trPath26, trPath27, trPath28, trPath29, trPath30, trPath31,
trPath32, trPath33, trPath34, trPath35, trPath36, trPath37,
trPath38, trPath39, trPath40, trPath41, trPath42, trPath43,
trPath44, trPath45, trPath46, trPath47, trPath48, trPath49,
trPath50, trPath51, trPath52, trPath53, trPath54, trPath56,
trPath57,
--------------------
Test12:
Caminos cubiertos: 1.75%
        trPath54,
Caminos no cubiertos:
        trPath1, trPath2, trPath3, trPath4, trPath5, trPath6,
trPath7, trPath8, trPath9, trPath10, trPath11, trPath12, trPath13,
trPath14, trPath15, trPath16, trPath17, trPath18, trPath19,
trPath20, trPath21, trPath22, trPath23, trPath24, trPath25,
trPath26, trPath27, trPath28, trPath29, trPath30, trPath31,
trPath32, trPath33, trPath34, trPath35, trPath36, trPath37,
trPath38, trPath39, trPath40, trPath41, trPath42, trPath43,
trPath44, trPath45, trPath46, trPath47, trPath48, trPath49,
trPath50, trPath51, trPath52, trPath53, trPath55, trPath56,
trPath57,
--------------------
Test13:
Caminos cubiertos: 1.75%
        trPath54,
Caminos no cubiertos:
        trPath1, trPath2, trPath3, trPath4, trPath5, trPath6,
trPath7, trPath8, trPath9, trPath10, trPath11, trPath12, trPath13,
trPath14, trPath15, trPath16, trPath17, trPath18, trPath19,
trPath20, trPath21, trPath22, trPath23, trPath24, trPath25,
```

```
trPath26, trPath27, trPath28, trPath29, trPath30, trPath31,
trPath32, trPath33, trPath34, trPath35, trPath36, trPath37,
trPath38, trPath39, trPath40, trPath41, trPath42, trPath43,
trPath44, trPath45, trPath46, trPath47, trPath48, trPath49,
trPath50, trPath51, trPath52, trPath53, trPath55, trPath56,
trPath57,
--------------------
Test14:
Caminos cubiertos: 1.75%
        trPath54,
Caminos no cubiertos:
        trPath1, trPath2, trPath3, trPath4, trPath5, trPath6,
trPath7, trPath8, trPath9, trPath10, trPath11, trPath12, trPath13,
trPath14, trPath15, trPath16, trPath17, trPath18, trPath19,
trPath20, trPath21, trPath22, trPath23, trPath24, trPath25,
trPath26, trPath27, trPath28, trPath29, trPath30, trPath31,
trPath32, trPath33, trPath34, trPath35, trPath36, trPath37,
trPath38, trPath39, trPath40, trPath41, trPath42, trPath43,
trPath44, trPath45, trPath46, trPath47, trPath48, trPath49,
trPath50, trPath51, trPath52, trPath53, trPath55, trPath56,
trPath57,
--------------------
Test15:
Caminos cubiertos: 1.75%
        trPath19,
Caminos no cubiertos:
        trPath1, trPath2, trPath3, trPath4, trPath5, trPath6,
trPath7, trPath8, trPath9, trPath10, trPath11, trPath12, trPath13,
trPath14, trPath15, trPath16, trPath17, trPath18, trPath20,
trPath21, trPath22, trPath23, trPath24, trPath25, trPath26,
trPath27, trPath28, trPath29, trPath30, trPath31, trPath32,
trPath33, trPath34, trPath35, trPath36, trPath37, trPath38,
trPath39, trPath40, trPath41, trPath42, trPath43, trPath44,
trPath45, trPath46, trPath47, trPath48, trPath49, trPath50,
trPath51, trPath52, trPath53, trPath54, trPath55, trPath56,
trPath57,
--------------------
Test16:
Caminos cubiertos: 1.75%
        trPath37,
Caminos no cubiertos:
        trPath1, trPath2, trPath3, trPath4, trPath5, trPath6,
trPath7, trPath8, trPath9, trPath10, trPath11, trPath12, trPath13,
```

```
trPath14, trPath15, trPath16, trPath17, trPath18, trPath19,
trPath20, trPath21, trPath22, trPath23, trPath24, trPath25,
trPath26, trPath27, trPath28, trPath29, trPath30, trPath31,
trPath32, trPath33, trPath34, trPath35, trPath36, trPath38,
trPath39, trPath40, trPath41, trPath42, trPath43, trPath44,
trPath45, trPath46, trPath47, trPath48, trPath49, trPath50,
trPath51, trPath52, trPath53, trPath54, trPath55, trPath56,
trPath57,
--------------------
Test17:
Caminos cubiertos: 1.75%
        trPath57,
Caminos no cubiertos:
        trPath1, trPath2, trPath3, trPath4, trPath5, trPath6,
trPath7, trPath8, trPath9, trPath10, trPath11, trPath12, trPath13,
trPath14, trPath15, trPath16, trPath17, trPath18, trPath19,
trPath20, trPath21, trPath22, trPath23, trPath24, trPath25,
trPath26, trPath27, trPath28, trPath29, trPath30, trPath31,
trPath32, trPath33, trPath34, trPath35, trPath36, trPath37,
trPath38, trPath39, trPath40, trPath41, trPath42, trPath43,
trPath44, trPath45, trPath46, trPath47, trPath48, trPath49,
trPath50, trPath51, trPath52, trPath53, trPath54, trPath55,
trPath56,
--------------------
Test18:
Caminos cubiertos: 1.75%
        trPath57,
Caminos no cubiertos:
        trPath1, trPath2, trPath3, trPath4, trPath5, trPath6,
trPath7, trPath8, trPath9, trPath10, trPath11, trPath12, trPath13,
trPath14, trPath15, trPath16, trPath17, trPath18, trPath19,
trPath20, trPath21, trPath22, trPath23, trPath24, trPath25,
trPath26, trPath27, trPath28, trPath29, trPath30, trPath31,
trPath32, trPath33, trPath34, trPath35, trPath36, trPath37,
trPath38, trPath39, trPath40, trPath41, trPath42, trPath43,
trPath44, trPath45, trPath46, trPath47, trPath48, trPath49,
trPath50, trPath51, trPath52, trPath53, trPath54, trPath55,
trPath56,
--------------------
Test19:
Caminos cubiertos: 1.75%
        trPath17,
Caminos no cubiertos:
```

```
        trPath1, trPath2, trPath3, trPath4, trPath5, trPath6,
trPath7, trPath8, trPath9, trPath10, trPath11, trPath12, trPath13,
trPath14, trPath15, trPath16, trPath18, trPath19, trPath20,
trPath21, trPath22, trPath23, trPath24, trPath25, trPath26,
trPath27, trPath28, trPath29, trPath30, trPath31, trPath32,
trPath33, trPath34, trPath35, trPath36, trPath37, trPath38,
trPath39, trPath40, trPath41, trPath42, trPath43, trPath44,
trPath45, trPath46, trPath47, trPath48, trPath49, trPath50,
trPath51, trPath52, trPath53, trPath54, trPath55, trPath56,
trPath57,
--------------------
Test20:
Caminos cubiertos: 1.75%
        trPath57,
Caminos no cubiertos:
        trPath1, trPath2, trPath3, trPath4, trPath5, trPath6,
trPath7, trPath8, trPath9, trPath10, trPath11, trPath12, trPath13,
trPath14, trPath15, trPath16, trPath17, trPath18, trPath19,
trPath20, trPath21, trPath22, trPath23, trPath24, trPath25,
trPath26, trPath27, trPath28, trPath29, trPath30, trPath31,
trPath32, trPath33, trPath34, trPath35, trPath36, trPath37,
trPath38, trPath39, trPath40, trPath41, trPath42, trPath43,
trPath44, trPath45, trPath46, trPath47, trPath48, trPath49,
trPath50, trPath51, trPath52, trPath53, trPath54, trPath55,
trPath56,
--------------------
Test21:
Caminos cubiertos: 0.00%

Caminos no cubiertos:
        trPath1, trPath2, trPath3, trPath4, trPath5, trPath6,
trPath7, trPath8, trPath9, trPath10, trPath11, trPath12, trPath13,
trPath14, trPath15, trPath16, trPath17, trPath18, trPath19,
trPath20, trPath21, trPath22, trPath23, trPath24, trPath25,
trPath26, trPath27, trPath28, trPath29, trPath30, trPath31,
trPath32, trPath33, trPath34, trPath35, trPath36, trPath37,
trPath38, trPath39, trPath40, trPath41, trPath42, trPath43,
trPath44, trPath45, trPath46, trPath47, trPath48, trPath49,
trPath50, trPath51, trPath52, trPath53, trPath54, trPath55,
trPath56, trPath57,
--------------------
```

Apéndice C Tabla 2. resultado de "Triángulo" - Evosuite

- Cobertura y archivo generado por Diffblue：

```
public class Test371Test {
    /**
     * Method under test: {@link Test371#Triang(int, int, int)}
     */
    @Test
    public void testTriang() {
        assertEquals(3, Test371.Triang(1, 1, 1));
        assertEquals(4, Test371.Triang(2, 1, 1));
        assertEquals(4, Test371.Triang(3, 1, 1));
        assertEquals(4, Test371.Triang(6, 1, 1));
        assertEquals(4, Test371.Triang(0, 1, 1));
        assertEquals(4, Test371.Triang(1, 2, 1));
        assertEquals(4, Test371.Triang(1, 0, 1));
        assertEquals(4, Test371.Triang(1, 1, 2));
        assertEquals(4, Test371.Triang(1, 1, 0));
        assertEquals(2, Test371.Triang(2, 2, 1));
        assertEquals(4, Test371.Triang(2, 3, 1));
        assertEquals(2, Test371.Triang(2, 1, 2));
        assertEquals(4, Test371.Triang(2, 1, 3));
        assertEquals(4, Test371.Triang(3, 2, 1));
        assertEquals(2, Test371.Triang(1, 2, 2));
        assertEquals(0, Test371.Triang(2, 3, 4));
    }
}
```

```
Test0:
Caminos cubiertos: 15.79%
        trPath14, trPath17, trPath19, trPath22, trPath27, trPath37,
trPath54, trPath55, trPath57,
Caminos no cubiertos:
        trPath1, trPath2, trPath3, trPath4, trPath5, trPath6,
trPath7, trPath8, trPath9, trPath10, trPath11, trPath12, trPath13,
trPath15, trPath16, trPath18, trPath20, trPath21, trPath23, trPath24,
trPath25, trPath26, trPath28, trPath29, trPath30, trPath31, trPath32,
trPath33, trPath34, trPath35, trPath36, trPath38, trPath39, trPath40,
trPath41, trPath42, trPath43, trPath44, trPath45, trPath46, trPath47,
trPath48, trPath49, trPath50, trPath51, trPath52, trPath53, trPath56,
--------------------
```

Apéndice C Tabla 3. resultado de "Triángulo" - Diffblue