

Universidad de Alcalá  
Escuela Politécnica Superior

GRADO EN INGENIERÍA EN SISTEMAS DE  
INFORMACIÓN



Trabajo Fin de Grado

**Aplicación de herramientas y análisis de datos para el  
aseguramiento de calidad del software**



ESCUELA POLITECNICA  
SUPERIOR

**Autor:** Inés López Baldominos

**Tutor/es:** Luis Fernández Sanz

2020

UNIVERSIDAD DE ALCALÁ

Escuela Politécnica Superior

**GRADO EN INGENIERÍA EN SISTEMAS DE INFORMACIÓN**

Trabajo Fin de Grado

Aplicación de herramientas y análisis de datos para el aseguramiento  
de calidad del software

**Autor:** Inés López Baldominos

**Tutor/es:** Luis Fernández Sanz

TRIBUNAL:

**Presidente:**

**Vocal 1º:**

**Vocal 2º:**

**FECHA:** 14/07/20

## **Agradecimientos**

A mi tutor, Luis Fernández, por guiarme durante este trabajo y durante el último año y medio. Por enseñarme el camino de la docencia y la investigación en el que espero poder dedicarle muchos más agradecimientos.

A mis compañeros por los ánimos y las ayudas que nos han llevado a todos a acabar. La alianza de la tranquilidad y los conejillos de indias. Dani sin ti no habría aprobado estructuras de datos, gracias.

A mis padres y hermanos de los que aprendo cada día y han hecho que sea como soy.

A mi novio por su infinita paciencia y convencerme de que puedo con todo y más. Por esperarme.

A mi yo de hace 4 años, por decidir seguir.

A mis abuelos.

# Índice de contenidos

Índice de contenidos.....	4
Índice de figuras.....	6
Índice de tablas .....	7
Índice de gráficos.....	8
Resumen .....	9
Palabras clave .....	9
Abstract .....	9
Keywords.....	9
1. Introducción.....	10
1.1. Presentación.....	10
1.2. Objetivos .....	10
1.3. Glosario de acrónimos y abreviaturas .....	11
1.4. Estructura de la memoria.....	11
2. Utilización de herramientas de ayuda al aseguramiento de la calidad del software .....	13
2.1. Conceptos teóricos .....	13
2.1.1. Técnicas para el diseño de casos de prueba .....	13
2.1.2. Importancia de la planificación en el proceso de pruebas.....	14
2.2. Análisis de las herramientas disponibles.....	15
2.2.1. Herramienta para generar diagramas de flujo - Code2flow .....	15
2.2.2. Herramienta de métricas de complejidad – Lizard .....	16
2.2.3. Herramienta para medir cobertura – Expcov demo .....	16
2.2.4. Herramienta para depuración – OnlineGDB .....	18
2.2.5. Herramienta para seguimiento de proyectos e incidencias – Klaros .....	18
2.3. Descripción del ejercicio práctico.....	19
2.4. Diseño de las clases de equivalencia para el enfoque de caja negra.....	20
2.5. Diseño de las suites de prueba según los criterios de cobertura .....	21
2.6. Aplicación de las suites de pruebas y análisis de resultados .....	27
2.6.1. Versión A .....	27
2.6.2. Versión B.....	35
2.6.3. Resumen de errores encontrados .....	45
2.7. Implementación de mejoras. Versión C del programa.....	45
2.8. Desarrollo adicional. Java .....	49
3. Análisis de datos de defectos e incidencias de proyectos reales .....	50

3.1.	Conceptos teóricos .....	50
3.2.	Análisis de datos .....	51
3.2.1.	Datos de defectos del sistema ACME.....	51
3.2.2.	Datos de defectos del sistema FOX .....	53
3.2.3.	Datos de la matriz de regresión del sistema FOX.....	58
3.3.	Comparativa de proyectos .....	59
4.	Conclusiones y trabajos futuros.....	60
4.1.	Conclusiones.....	60
4.2.	Trabajos futuros .....	61
5.	Referencias .....	61
Anexo I	Código de los programas .....	62
	Versión A. ....	62
	Versión B. ....	63
	Versión C. ....	64
	Adaptación a Java. ....	65
Anexo II	Límites numéricos de los tipos de datos.....	67
	Límites en C .....	67
	Límites en Java.....	68

## Índice de figuras

Figura 1- Ejemplo de utilización de la herramienta Code2flow .....	15
Figura 2- Ejemplo de utilización de Lizard.....	16
Figura 3- Ejemplo de utilización de la herramienta Expcov demo.....	17
Figura 4- Ejemplo de un informe de cobertura generado con Expcov .....	17
Figura 5- Resultados del informe de cobertura de decisiones .....	17
Figura 6- Ejemplo de uso de OnlineGDB .....	18
Figura 7- Ejemplo de uso de la plataforma Klaros .....	19
Figura 8 - Análisis de complejidad para la versión A del código.....	22
Figura 9 - Resultados de complejidad para la versión B del código.....	22
Figura 10 - Diagrama de flujo de versión A del flujo atendiendo al criterio de decisiones....	23
Figura 11 - Diagrama de flujo de versión A del flujo atendiendo al criterio de condiciones .	24
Figura 12- Informe de resultados de cobertura del programa A para la suite de cobertura de funciones.....	27
Figura 13- Salida del programa A para la suite de cobertura de funciones.....	27
Figura 14 - Informe de resultados de cobertura del programa A para la suite de cobertura de sentencias y condiciones .....	28
Figura 15 - Salida del programa A para la suite de cobertura de sentencias y decisiones .....	28
Figura 16 - Salida del programa A para la suite de cobertura de condiciones .....	29
Figura 17 - Salida del programa A para la suite de cobertura funcional.....	31
Figura 18- Salida del programa A para el caso de prueba "Número no entero de n" .....	32
Figura 19 - Salida del programa A para el caso de prueba "Caracter no numérico en n" .....	32
Figura 20 - Salida del programa A para el caso de prueba "Menos de 3 lados" .....	32
Figura 21 - Salida del programa A para el caso de prueba "Más de 3 lados" .....	32
Figura 22 - Salida del programa A para el caso de prueba "Lado no entero" .....	32
Figura 23 - Salida del programa A para el caso de prueba "Lado no número" .....	33
Figura 24 - Resultado de introducir un valor decimal para el lado en el programa A.....	34
Figura 25 - Informe de resultados de cobertura del programa B para la suite de cobertura de funciones.....	35
Figura 26 - Salida del programa B para la suite de cobertura de funciones .....	35
Figura 27 - Informe de resultados de cobertura del programa B para la suite de cobertura de sentencias y condiciones .....	35
Figura 28 - Salida del programa B para la suite de cobertura de sentencias y decisiones .....	36
Figura 29 - Salida del programa B para la suite de cobertura de condiciones .....	38
Figura 30 - Salida del programa B para la suite de cobertura funcional.....	41
Figura 31 - Pruebas para analizar el valor máximo de un valor tipo float en Codelite.....	43
Figura 32 – Pruebas para delimitar el límite máximo de float en C. I.....	44
Figura 33 - Pruebas para delimitar el límite máximo de float en C. II .....	44
Figura 34 - Pruebas para delimitar el número máximo de decimales en C.....	45
Figura 35 - Salida del programa C para la suite de cobertura de condiciones .....	47
Figura 36 - Pruebas para testar el límite máximo de float en Java .....	49
Figura 37 - Valores límite para las variables en C.....	67
Figura 38 - Valores límite para las variables en Java .....	68

## Índice de tablas

Tabla 1- Diseño de pruebas de caja blanca .....	21
Tabla 2- Suites de prueba según criterios de cobertura.....	26
Tabla 3 - Resultados de la ejecución de la suite de cobertura de funciones para el programa A .....	27
Tabla 4 - Resultados de la ejecución de la suite de cobertura de sentencias y decisiones para el programa A.....	28
Tabla 5 - Resultados de la ejecución de la suite de cobertura de condiciones para el programa A.....	30
Tabla 6 - Resultados de la ejecución de la suite de cobertura funcional para el programa A .	34
Tabla 7- Casos de prueba añadidos a la suite para la cobertura de condiciones .....	37
Tabla 8 - Resultados de la ejecución de la suite de cobertura de condiciones para el programa B.....	40
Tabla 9 - Resultados de la ejecución de la suite de cobertura funcional para el programa B .	42
Tabla 10 - Resultados de la ejecución de la suite de cobertura de condiciones para el programa C.....	49
Tabla 11 - Tiempo medio de resolución de incidencias según gravedad y prioridad en el sistema ACME .....	52
Tabla 12 - Tiempo medio de resolución de incidencias según gravedad en el sistema FOX .	54
Tabla 13 - Datos comunes de los proyectos ACME y FOX.....	59

## Índice de gráficos

Gráfico 1 - Comparativa entre los valores introducidos y los valores asignados a las variables .....	42
Gráfico 2 - Comparativa entre la entrada y la salida obtenida .....	43
Gráfico 3 - Distribución de las incidencias según gravedad y prioridad en el sistema ACME .....	52
Gráfico 4 - Tiempo medio de resolución de incidencias según gravedad en el sistema ACME .....	53
Gráfico 5 - Tiempo medio de resolución de incidencias según gravedad y prioridad en el sistema ACME.....	53
Gráfico 6 - Distribución de las incidencias según gravedad en el sistema FOX.....	54
Gráfico 7 - Tiempo medio de resolución de incidencias según gravedad en el sistema FOX	54
Gráfico 8 - Porcentaje de las asignaciones que recibe cada miembro del equipo en el sistema FOX.....	55
Gráfico 9 - Distribución de las asignaciones de cada miembro según el tipo de gravedad en el sistema FOX .....	55
Gráfico 10 - Tiempo medio de resolución de las incidencias por cada miembro del equipo en el sistema FOX.....	56
Gráfico 11 - Tiempo medio de resolución según el número de asignaciones de cada persona en el sistema FOX.....	56
Gráfico 12 - Diagrama de cajas para la variable Tiempo medio de resolución.....	57
Gráfico 13 - Diagrama de cajas para la variable Número de asignaciones .....	57
Gráfico 14 - Tiempo medio de resolución según el número de asignaciones de cada persona en el sistema FOX (sin valores atípicos).....	58
Gráfico 15 - Número de fallos acumulados en las distintas versiones del sistema FOX .....	59



## Resumen

Las técnicas de aseguramiento de calidad del software incluyen pruebas, revisiones e inspecciones y las métricas como las más habituales en los proyectos de desarrollo. En este trabajo se explora la aplicación eficaz de estas técnicas mediante herramientas habituales o fácilmente disponibles para estudiantes de informática a la vez que se aplican a ejemplos concretos de software para ejemplificar las buenas prácticas de trabajo y analizar los datos obtenidos para facilitar decisiones en proyectos y para mejorar los resultados de calidad.

## Palabras clave

Calidad del Software, pruebas, herramientas online, análisis

## Abstract

Software quality assurance techniques include testing, reviews and inspections and metrics as the most commonly used in development projects. This paper explores the effective application of these techniques using tools that are familiar or readily available to computer science students while applying them to concrete software examples to exemplify good work practices and to analyse the data obtained in order to facilitate project decisions and to improve quality results.

## Keywords

Software quality, testing, online tools, analysis

# 1. Introducción

## 1.1. Presentación

El software es esencial en la sociedad actual, además, crece cada vez más rápido en complejidad y tamaño. Por eso es imprescindible asegurar la calidad del programa desarrollado. Un software de calidad no cumple únicamente con la funcionalidad indicada, sino que, además, lo hace de forma eficiente y permite introducir cambios con poca dificultad haciendo que el coste de mantenimiento sea lo más bajo posible.

Para garantizar la calidad del software hay que establecer controles, pruebas y métricas, con los que poder establecer de forma objetiva si se cumplen con los estándares. La importancia de estos controles es innegable, pero además éstos han de hacerse de forma continua durante todo el ciclo de vida del producto. Probar, y fallar cuanto antes, tiene un gran impacto económico y de recursos en el desarrollo del software.

Las buenas prácticas para el diseño y codificación de aplicaciones es un asunto a tener en cuenta también desde el punto de vista de los estudiantes de informática. En este trabajo se llevarán a cabo pruebas utilizando herramientas fácilmente disponibles para este colectivo y un estudio de casos reales de proyectos para aportar soluciones enfocadas al aseguramiento de la calidad del software.

## 1.2. Objetivos

El objetivo del presente trabajo es doble. Por una parte, se crearán ejemplos de utilización de herramientas de ayuda al aseguramiento de la calidad del software que dan soporte a buenas prácticas para el desarrollo de código y el diseño detallado de las aplicaciones. Por otra parte, se realizarán ejemplos de análisis básicos de datos obtenidos de proyectos reales, generados a través del registro de actividades de calidad de software, para evaluar la calidad y sugerir decisiones aplicables en los proyectos de desarrollo.

En el caso de los ejemplos de utilización de herramientas, se pretende que el objetivo considere las siguientes condiciones:

- Se preferirían las herramientas disponibles como servicios online que eviten la instalación de software o entornos de desarrollo. También se utilizarán las herramientas de desarrollo habituales en los grados de informática.
- Los ejemplos se plantearán desde una perspectiva de aplicación por parte de estudiantes de grados de informática de últimos cursos, que ya cuentan con conocimientos de desarrollo e ingeniería del software, aparte de otros conocimientos habituales desarrollados en los cursos anteriores.
- Los ejemplos serán aplicables sobre código o diseño detallado del software sin necesidad de trabajar sobre modelos de mayor nivel modelados con herramientas CASE u otros entornos de desarrollo. Estos ejemplos podrán ser obviamente limitados para ser fácilmente entendibles y manejables por estudiantes.

- Los ejemplos contemplarán no solo la aplicación de las herramientas sino también la explicación de sus bases técnicas, las soluciones sugeridas para cada aspecto de los mismos y las sugerencias de aplicabilidad en entornos reales de las prácticas desarrolladas.
- De forma orientativa, se considerarán como tipos de herramientas iniciales a explorar las relacionadas con soporte a pruebas de software (especialmente las de cobertura y gestión de las mismas), las de análisis estático de código para mejora del mismo y recuperación de diseño y las herramientas de soporte a la medición del software.

Respecto del objetivo de análisis de datos obtenidos de proyectos reales, se consideran las siguientes condiciones:

- Los datos serán básicamente los obtenidos de herramientas de registros y seguimiento de defectos e incidencias (*defect-tracking* o *bug-tracking*) con los datos habitualmente registrados en las mismas.
- El análisis básico de los datos estará orientado a mostrar cómo se pueden explotar dichos datos para obtener conclusiones prácticas aplicables a los proyectos de software, explicando los métodos de análisis y aportando ejemplos de decisiones y conclusiones que podrían obtenerse en situaciones similares.

Por último, este trabajo está acotado por las siguientes limitaciones que se han considerado apropiadas para limitar el alcance del mismo a los objetivos de un trabajo de fin de grado:

- No se considerarán ejemplos de gran tamaño o complejidad o directamente extraídos de proyectos reales que no puedan ser abordables por estudiantes dentro de las limitaciones habituales de trabajo y tiempo del curso académico.
- Los ejemplos se limitarán a una serie predefinida de herramientas con las características anteriormente indicadas y no pretenden tener carácter exhaustivo en técnicas de aseguramiento de calidad.

### 1.3. Glosario de acrónimos y abreviaturas

**CASE:** Computer Aided Software Engineering (Ingeniería del Software Asistida por Ordenador).

**NLOC:** Number of lines of code (Número de líneas de código)

**SRGM:** Software reliability growth model (Modelo de crecimiento de la fiabilidad del software)

### 1.4. Estructura de la memoria

La memoria de este trabajo se divide en dos partes, en cada una se desarrolla uno de los objetivos principales.

En el apartado 2 se desarrolla el trabajo realizado con las distintas herramientas que ayudan al aseguramiento de la calidad del software. Dentro de este apartado encontramos los siguientes puntos:

- 2.1. Conceptos teóricos básicos para el desarrollo y seguimiento correcto del trabajo.
- 2.2. Conjunto de herramientas de ayuda al aseguramiento de la calidad del software disponibles online y sin necesidad de registro. Para cada una de ellas se indica el enlace en el que está disponible, se describe su funcionalidad y se muestran capturas de pantalla que ejemplifican su uso.
- 2.3. Descripción del ejercicio práctico. Se explica la especificación del ejercicio que se va a utilizar como ejemplo para utilizar las herramientas analizadas en el punto anterior.
- 2.4. Diseño de las clases de equivalencia atendiendo al enfoque de caja blanca, es decir según la funcionalidad pedida en la especificación del programa.
- 2.5. Diseño de los casos de prueba, con entradas concretas y salidas esperadas necesarios para cumplir con los distintos criterios de cobertura.
- 2.6. Ejecución de las pruebas en las dos versiones del código y análisis de los resultados en función de cada uno de los criterios de cobertura considerados (cobertura de funciones, de sentencias y decisiones, de condiciones y funcional).
- 2.7. En este punto se presenta una nueva versión del programa en el que se ha solucionado uno de los errores presentes en las versiones anteriores y se vuelve a ejecutar la suite de pruebas correspondiente.
- 2.8. Como propuesta adicional a lo planteado en el trabajo se inicia una nueva versión del programa en Java y se analizan los límites aplicables a las variables tipo *float* en este lenguaje.

En el apartado 3 se desarrolla en un análisis con los datos de defectos e incidencias en proyectos reales. Dentro de este apartado encontramos los siguientes puntos:

- 3.1. Conceptos teóricos básicos para el desarrollo y seguimiento correcto del trabajo.
- 3.2. Análisis de datos y resultados de cada uno de los 3 registros disponibles.
- 3.3. Comparación de los proyectos en función de los datos comunes disponibles.

Finalmente, en el trabajo se incluye un apartado de conclusiones y trabajos futuros, así como un apartado con las referencias consultadas para su correcto desarrollo.

## 2. Utilización de herramientas de ayuda al aseguramiento de la calidad del software

### 2.1. Conceptos teóricos

Las pruebas del software se han vuelto, a la vez, más fáciles y más difíciles que nunca, según Myers [1]. La dificultad viene dada por la gran variedad de lenguajes de programación, sistemas operativos y plataformas disponibles hoy en día. Además, por supuesto, del impacto (en términos económicos y en número de usuarios) que puede suponer que un defecto del código llegue a un cliente cuyo negocio está cada vez más estrechamente relacionado con las aplicaciones en su día a día. Por el contrario, las buenas prácticas de programación, la reutilización de código ya probado y las numerosas herramientas disponibles para apoyar el proceso de pruebas en cada etapa hacen que esta tarea sea cada vez más fácil y efectiva.

En el diccionario IEEE 610 [2] encontramos las definiciones de los términos necesarios para la correcta comprensión del trabajo:

- Aseguramiento de calidad (*quality assurance*): “pauta planificada y sistemática de todas las acciones necesarias para proporcionar la confianza adecuada de que un elemento o producto se ajusta a las normas los requisitos establecidos”.
- Prueba (*test*): “actividad en la cual un sistema o uno de sus componentes se ejecuta en circunstancias previamente especificadas, los resultados se observan y registran y se realiza una evaluación de algún aspecto”.
- Casos de prueba (*test case*): “conjunto de entradas, condiciones de ejecución, y resultados esperados desarrollados para un objetivo particular como, por ejemplo, ejercitar un camino concreto de un programa o verificar el cumplimiento de un determinado requisito”.
- Pruebas estructurales: “pruebas que tienen en cuenta el funcionamiento interno de un sistema o componente”.
- Pruebas funcionales: “pruebas que ignoran el funcionamiento interno de un sistema o componente y se centran únicamente en los resultados generados en respuesta a entradas y condiciones de ejecución seleccionadas”.
- Error: “paso, proceso o definición de datos incorrecto”.
- Fallo: “incapacidad de un sistema o componente para realizar sus funciones requeridas dentro de los requisitos de rendimiento especificados”.

Como se extrae de estas definiciones el objetivo de las pruebas es encontrar defectos para así garantizar el aseguramiento de la calidad. Por lo tanto, descubrir un defecto es tener éxito. Dada la imposibilidad de realizar pruebas totalmente exhaustivas es imprescindible definir casos de prueba que sean representativos tanto de datos de entrada válidos y esperados como los no válidos e inesperados. Así, el objetivo de las pruebas es tanto comprobar que el software hace lo que debe hacer y no hace lo que no debe.

#### 2.1.1. Técnicas para el diseño de casos de prueba

Ya que no es viable probar todas las posibilidades de funcionamiento del software, la idea fundamental para el diseño casos de prueba consiste en elegir las entradas que, por sus

características, se consideren representativas del resto y así conseguir un equilibrio entre aseguramiento de la calidad e impacto económico de las pruebas.

El enfoque funcional o de caja negra consiste en estudiar la especificación de las funciones, la entrada y la salida para derivar los casos a probar y las clases de equivalencia representativas. En los márgenes de las clases de equivalencia es donde se acumulan más errores, por eso es necesario que los casos de prueba exploren los valores límite.

El enfoque estructural o de caja blanca consiste en centrarse en la implementación del programa para elegir los casos de prueba. En este caso, la prueba ideal consistiría en probar todos los posibles caminos de ejecución, a través de las instrucciones del código, que puedan trazarse.

Estos enfoques no son excluyentes entre sí, se pueden combinar para conseguir una detección de defectos más eficaz. Esta es la aproximación que se va a considerar para el desarrollo del trabajo.

Con cualquiera de los enfoques mencionados, es imprescindible establecer criterios de cobertura. Tomamos como referencia la clasificación de criterios de cobertura lógica de Myers [1] que se muestran a continuación ordenados de menor a mayor nivel de exigencia.

- Cobertura de sentencias: se trata de generar los casos de prueba necesarios para que cada sentencia o instrucción del programa se ejecute, al menos, una vez.
- Cobertura de decisiones: escribir casos suficientes para que cada decisión tenga, por lo menos una vez, un resultado verdadero y, al menos una vez, uno falso. En general, una ejecución de pruebas que cumple la cobertura de decisiones cumple también la cobertura de sentencias.
- Cobertura de condiciones: se trata de diseñar tantos casos como sea necesario para que cada condición de cada decisión adopte el valor verdadero al menos una vez y falso otra.

Como medida de cobertura para las pruebas de caja negra, utilizamos el criterio de cobertura funcional, es decir, si se ha diseñado un caso de prueba para cada requisito de la especificación.

Tanto la prueba exhaustiva de caja blanca como de caja negra son impracticables. Conviene emplear lo mejor de todas las técnicas para obtener pruebas más eficaces.

### 2.1.2. Importancia de la planificación en el proceso de pruebas

El proceso de prueba comienza con la generación de un plan de pruebas en base a la documentación sobre el proyecto y la documentación sobre el software a probar. A partir de dicho plan, se entra en detalle diseñando pruebas específicas. La documentación del diseño de pruebas es necesaria para una buena organización y para asegurar su reutilización. El estándar IEEE std 829 [3] explica cómo debería ser el proceso exhaustivo de documentación para cada fase del proceso de pruebas de software y sistemas. Y, aunque, su aplicación íntegra se limita a desarrollos críticos sí que hay unos mínimos que es necesario desarrollar:

- plan de pruebas
- diseño de pruebas
- especificación de los casos de prueba
- especificación del procedimiento a seguir

La filosofía más adecuada para las pruebas consiste en planificarlas y diseñarlas de forma sistemática para poder detectar el máximo número y variedad de defectos con el mínimo consumo de tiempo y esfuerzo. El objetivo es gestionar adecuadamente el conocimiento generado y permitir la reutilización de las pruebas tanto por el propio equipo de testing como por otros equipos.

## 2.2. Análisis de las herramientas disponibles

Existen herramientas relacionadas con la calidad del software disponibles online para las que no hace falta ningún tipo de registro o instalación, lo que facilita su utilización por parte de los estudiantes. Entre todas las disponibles, se han seleccionado y utilizado para realizar este trabajo las siguientes:

### 2.2.1. Herramienta para generar diagramas de flujo - Code2flow

Disponible en: <https://app.code2flow.com/>

Esta herramienta muestra el diagrama de flujo asociado a un fragmento de código a partir de dicho fragmento de código. En la Figura 1 se muestra un ejemplo de utilización de la herramienta. Consta de un panel de edición en el que se puede incluir código en distintos lenguajes. En el panel derecho se muestra el diagrama de flujo, no se pueden hacer modificaciones directamente en este panel.

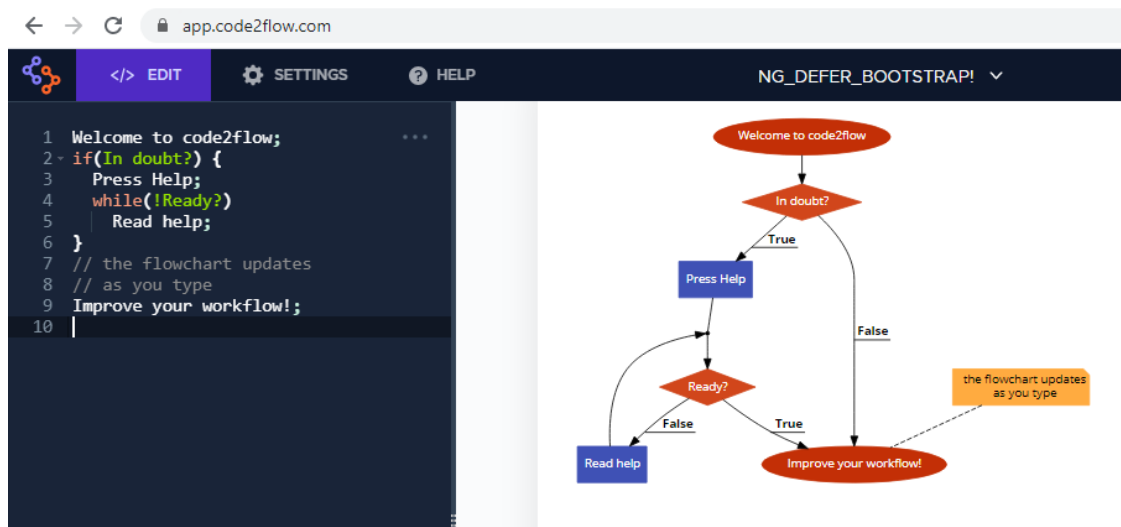


Figura 1- Ejemplo de utilización de la herramienta Code2flow

Es importante tener en cuenta que la aplicación considera como un nodo único las decisiones que implican varias condiciones. A la hora de establecer los criterios de cobertura hay que tener en cuenta esta particularidad.

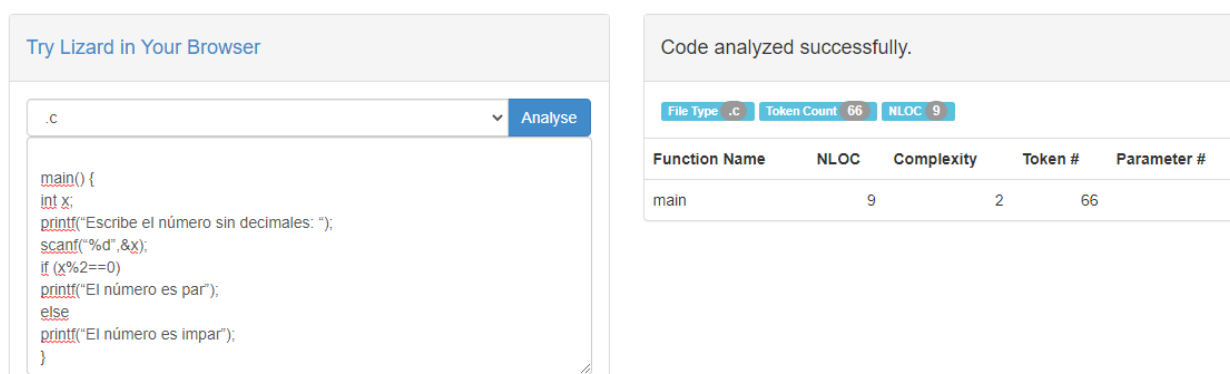
La versión disponible online de forma gratuita tiene ciertas limitaciones en cuanto a las funcionalidades que ofrece el producto. La primera y más importante a tener en cuenta es el máximo de 25 nodos por diagrama. Se puede ampliar el límite a 50 nodos creando una cuenta gratuita. Otras restricciones de la aplicación en su versión gratuita son, por ejemplo, limitaciones para la edición y personalización de los diagramas o no tener la posibilidad de guardar el trabajo para continuar en otro momento.

## 2.2.2. Herramienta de métricas de complejidad – Lizard

Disponible en: <http://www.lizard.ws/>

Lizard es una herramienta de código abierto que permite el análisis estático básico del código de forma automática para lenguajes de programación imperativa como Java, C/C++, JavaScript, Python, Ruby, Swift, Objective C, entre otros. Las métricas que ofrece son:

- NLOC: Líneas de código sin comentarios ni líneas en blanco
- Complexity: complejidad ciclomática. Definida según McCabe [4]
- Token: número de ocurrencias totales de los operandos y operadores del programa, se puede considerar equivalente a la métrica de longitud de Halstead.
- Parameter; número de parámetros de las funciones



The screenshot shows the Lizard web application interface. On the left, there is a text area with a file type dropdown set to '.c' and an 'Analyse' button. The code in the text area is a C program that reads an integer and checks if it is even or odd. On the right, the results are displayed in a table. The table has columns for Function Name, NLOC, Complexity, Token #, and Parameter #. The results for the 'main' function are: NLOC: 9, Complexity: 2, Token #: 66, and Parameter #: 0.

Function Name	NLOC	Complexity	Token #	Parameter #
main	9	2	66	0

Figura 2- Ejemplo de utilización de Lizard

Lizard no cuenta las sentencias correspondientes a importaciones de librerías para calcular estas métricas.

Esta aplicación se puede utilizar desde los navegadores habituales, aunque no con todas las funcionalidades que sí están disponibles si se instala la herramienta. Con la instalación, por ejemplo, sería posible además establecer límites y alarmas para los valores de NLOC, complejidad y parámetros o el número de máximos alarmas que el usuario está dispuesto a ignorar.

Tal y como se indica en la documentación de la herramienta<sup>1</sup>, el objetivo no es dar valores de complejidad ciclomática precisos, sino una primera aproximación para que el usuario pueda conocer el estado, en términos de complejidad, de una aplicación o programa. No se describe la metodología utilizada para calcular la complejidad, pero los resultados coinciden con la complejidad de McCabe, dividiendo las decisiones compuestas en una condición por nodo.

## 2.2.3. Herramienta para medir cobertura – Expcov demo

Disponible en: <https://www.expcov.com/cgi-bin/expcov/demo.cgi>

---

<sup>1</sup> La documentación está disponible para su consulta en: <https://pypi.org/project/lizard/>



Esta herramienta genera un informe de cobertura dado el código del programa y las entradas de los casos que se pretende probar. Permite analizar código en C, C++ y Objective C. En el informe se analiza la cobertura según tres aspectos: funciones, sentencias y decisiones.

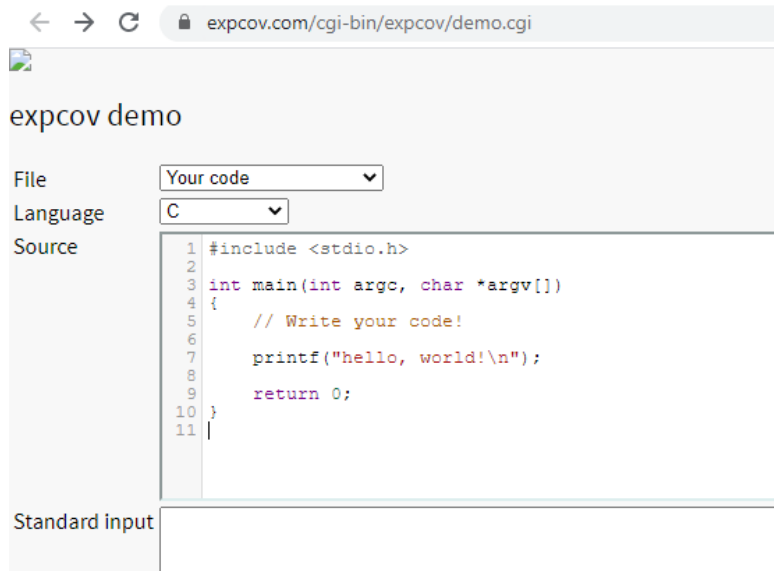


Figura 3- Ejemplo de utilización de la herramienta Expcov demo

En el informe se muestran las estadísticas de los tres aspectos de cobertura analizados. Tal y como se puede ver en la Figura 4, el informe indica el tanto por ciento de cobertura conseguido y el número absoluto de funciones o expresiones analizadas con respecto del total.

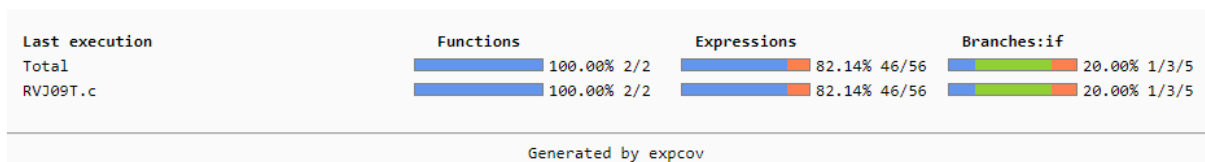


Figura 4- Ejemplo de un informe de cobertura generado con Expcov

En el caso de las decisiones el informe no es tan sencillo de interpretar. Muestra el porcentaje de decisiones que se han analizado completamente (tanto con valor True como False). El resto de valores se refieren a decisiones cubiertas totalmente, decisiones no cubiertas en ninguna rama y decisiones totales según e muestra en la Figura 5.

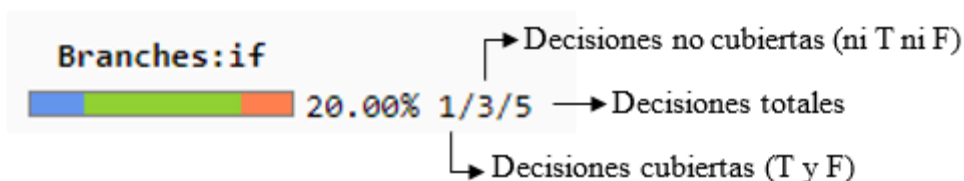


Figura 5- Resultados del informe de cobertura de decisiones

## 2.2.4. Herramienta para depuración – OnlineGDB

Disponible en: <https://www.onlinegdb.com/>

Es una herramienta online que permite compilar y depurar código para los lenguajes de programación más habituales como C, C++, Python, Java, PHP, y también Ruby, Perl, C#, VB, Swift, Pascal, Fortran, Haskell, Objective-C, Assembly, HTML, CSS, JS, SQLite, Prolog.

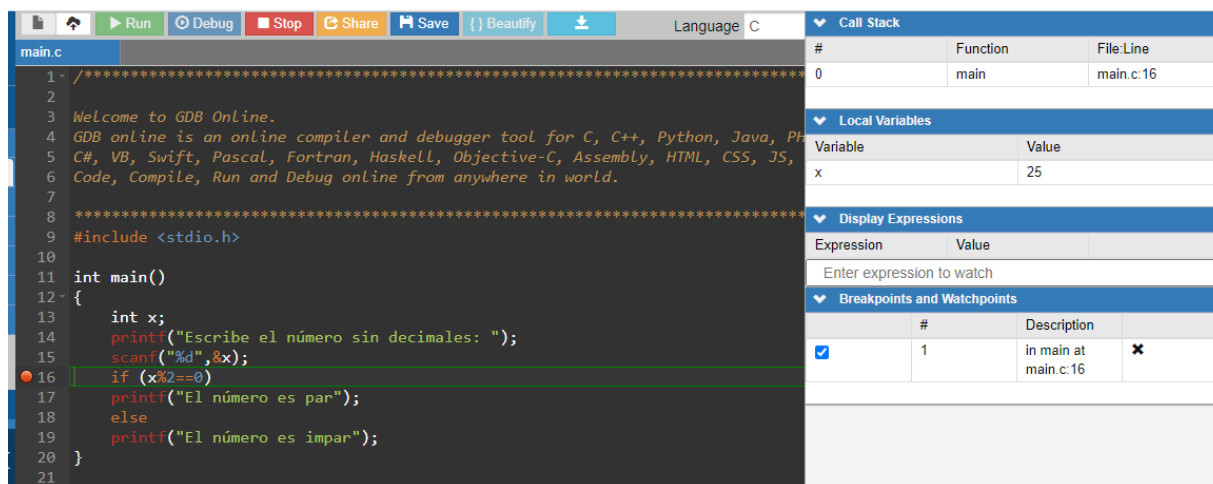


Figura 6- Ejemplo de uso de OnlineGDB

Ofrece las mismas funcionalidades que cualquier depurador de los entornos tradicionales de desarrollo, se establecen puntos de ruptura (*breakpoints*) y en cada punto muestra los valores de las variables locales y las expresiones que se quiera evaluar.

## 2.2.5. Herramienta para seguimiento de proyectos e incidencias – Klaros

Disponible en: <https://www.klaros-testmanagement.com/demo/pages/login.seam>

Klaros es una herramienta de seguimiento de proyectos y gestión de pruebas que dispone de una versión de prueba online. Las funcionalidades que ofrece y su funcionamiento es similar a otras herramientas gestión más conocidas como Jira o Bugzilla, con las que incluso se puede integrar.

The screenshot shows the Klaros web interface. At the top, there's a navigation bar with 'Define', 'Plan', 'Execute', 'Evaluate', and 'Configure' tabs. The user is logged in as 'Felix Mustermann (Administrator)'. The main content area displays 'P99999 - Finance Tracker Sticky Demo Project' with tabs for 'Properties', 'User Defined (15)', 'Copy Objects', 'Access', 'Integration', 'Results', and 'Changes'. The 'Results' tab is active, showing 'Test Case Results (5176)', 'Test Suite Results (70)', and 'Test Runs (355)'. A table lists individual test runs with columns for ID, Start, Iteration, Test Case/Suite, Test Environment, System under Test, Job, Executor, Results, and Action. The results column uses colored icons (green for pass, red for fail) to indicate the status of each run.

ID	Start	Iteration	Test Case/Suite	Test Environment	System under Test	Job	Executor	Results	Action
TRU0000544	6/23/20 4:53:28 PM		TS00063	Vivaldi(Linux)	SUT		Felix Mustermann	5	1 fail, 4 pass
TRU0000532	6/4/20 2:25:13 PM		TC00153	Windows 10	5.1 Open		Felix Mustermann	1	1 fail
TRU0000531	6/4/20 2:24:48 PM		TC00153	Thin Client	FinanceTracker 1.1.1		Felix Mustermann	1	1 pass
TRU0000530	6/4/20 2:23:58 PM		TC00152	Thin Client	SUT		Felix Mustermann	1	1 fail
TRU0000524	5/20/20 9:02:41 AM		TC00153	Thin Client	5.1 Open		Felix Mustermann	1	1 fail
TRU0000523	5/20/20 9:01:49 AM		TC00153	Thin Client	5.1 Open		Felix Mustermann	1	1 fail
TRU0000521	5/12/20 5:25:39 PM		TC00124	Android Smartwatch	FinanceTracker 1.0.0		Felix Mustermann	1	1 fail
TRU0000520	5/5/20 7:14:16 AM		TS00063	Thin Client	SUT		Felix Mustermann	5	1 fail, 2 pass, 2 other
TRU0000518	4/29/20 4:28:05 PM		TC00134	redas	test065		Felix Mustermann	1	1 fail
TRU0000517	4/17/20 12:00:00 PM		TS00064	Thin Client	5.1 Open		Felix Mustermann	1	1 fail
TRU0000516	4/15/20 11:38:33 AM		TS00063	Thin Client	SUT		Felix Mustermann	5	5 pass
TRU0000514	4/8/20 12:38:49 PM		TS00063	Thin Client	SUT		Max Mustermann	5	4 pass, 1 fail
TRU0000510	3/27/20 11:10:13 AM		TC00153	Thin Client	5.1 Open		Felix Mustermann	1	1 pass
TRU0000507	3/25/20 4:19:22 AM		TS00046	neuer Test	FinanceTracker 1.0.0	JOB00486	Felix Mustermann	2	1 fail, 1 pass
TRU0000506	3/18/20 7:37:54 AM		TS00059	ERack123	test065		Felix Mustermann	1	1 pass

Figura 7- Ejemplo de uso de la plataforma Klaros

Con respecto a las pruebas, Klaros permite crear casos de prueba y suite de pruebas para ejecutar los test y obtener fácilmente informes de resultados. Además, la trazabilidad existente entre requisitos y pruebas ejecutadas permite conocer cuál es la cobertura de requisitos en cada fase del proyecto.

Se puede acceder a la versión demo del programa utilizando las siguientes cuentas predefinidas:

- Usuario: admin      Contraseña: admin
- Usuario: manager      Contraseña: manager
- Usuario: tester      Contraseña: tester

### 2.3. Descripción del ejercicio práctico

Para crear ejemplos de utilización de herramientas de ayuda al aseguramiento de la calidad del software que se han descrito en el apartado anterior se utiliza como base el ejercicio práctico propuesto por Myers [1]. En la propuesta de Myers, se pide estudiar los casos de prueba necesarios para asegurar el correcto funcionamiento de un programa que lee primero el número de ejecuciones que se quieren realizar y luego para cada ejecución tres datos que representan los lados de un triángulo, mostrando como resultado el tipo de triángulo que forman dichos lados según su longitud.

En este trabajo se toma como base este ejercicio y se añade que además el programa clasifique el triángulo según los ángulos que determinan los lados. Además, se estudiarán los casos de prueba necesarios para versiones distintas del programa desarrollado en C<sup>2</sup>:

Versión A.      La implementación se realiza con números enteros para los lados. El cálculo del tipo de triángulo según ángulos se realiza según el signo del resultado de

<sup>2</sup> El código fuente de cada una de las versiones se puede encontrar como anexo al final del presente trabajo.

aplicar el teorema del coseno. Es decir, si es 0 el ángulo es recto, si es positivo el ángulo es agudo y si es negativo el ángulo es obtuso.

Versión B. En esta versión los datos para la longitud de los lados son números reales y se implementan como *float*. El resto de las condiciones de la implementación son las mismas que en la versión A.

Para cada una de estas versiones del programa se realiza un estudio de los casos de prueba, se utilizan las distintas herramientas disponibles para analizar la complejidad de las funciones y conseguir asegurar la calidad del programa en función de criterios de cobertura de funciones, caminos críticos, decisiones y condiciones.

## 2.4. Diseño de las clases de equivalencia para el enfoque de caja negra

Con el objetivo de que el proceso de aseguramiento de la calidad sea exhaustivo y riguroso se diseñan y detallan en la Tabla 1 los casos de prueba necesarios basados en las especificaciones del problema para cada una de las versiones propuestas.

Para poder hacer referencia a los distintos casos posteriormente, todos se numeran. Aquellos a los que se añade una letra (por ejemplo, 2A) solo se aplican a la versión del programa que identifica dicha letra. En aquellos en los que no se especifica, se aplican a ambas versiones.

<b>Condición de la especificación</b>	<b>Caso de prueba válido</b>		<b>Caso de prueba no válido</b>	
<i>3 longitudes (A) enteras / (B) reales de lados</i>	3 datos	( 1 )	< 3 datos	( 3 )
			> 3 datos	( 4 )
	Enteros	( 2A )	Nº no entero	( 5A )
	Reales	( 2B )	No número	( 6 )
	Mayor que 0	( 7 )	Lado $\leq 0$	( 8 )
<i>Formación de un triángulo</i>	$A < B + C$	( 9 )	$A \geq B + C$	( 10 )
	$B < A + C$	( 11 )	$B \geq A + C$	( 12 )
	$C < A + B$	( 13 )	$C \geq A + B$	( 14 )
<i>Tipo de triángulo *</i>	Escaleno	( 15 )		
	rectángulo			
	Escaleno	( 16 )		
	acutángulo			
	Escaleno	( 17 )		
	obtusángulo			
	Isósceles	( 18 )		
	acutángulo			
Isósceles	( 19 )			
obtusángulo				
Equilátero	( 20 )			
acutángulo				
<i>Número de ejecuciones (n)</i>	1 dato	( 21 )	No entero	( 22 )

	Entero	( 23 )	No número	( 24 )
<i>Valores máximos de la representación del tipo de dato</i>			Lado > Máximo int	( 25A )
			Máximo entero alcanzado en operaciones internas	( 26A )
			Lado > Máximo float	( 25B )
			Máximo real alcanzado en operaciones internas	( 26B )
			Lado > Máxima representación de decimales	( 27B )
<i>Ángulos límite</i>	Agudo casi recto	( 28B )		
	Obtuso casi recto	( 29B )		

Tabla 1- Diseño de pruebas de caja blanca

## 2.5. Diseño de las suites de prueba según los criterios de cobertura

Además de probar los casos descritos en el apartado anterior es importante saber qué nivel de cobertura se está alcanzando con cada suite de pruebas. Para ello se tienen en cuenta los siguientes criterios de cobertura (ordenados de menor a mayor exigencia de comprobación y detección de defectos) [5]:

- Cobertura de funciones, con la herramienta Expcov.
- Cobertura de sentencias, con la herramienta Expcov.
- Cobertura de decisiones, con la herramienta Expcov.
- Cobertura de condiciones, con la herramienta OnlineGDB.
- Cobertura funcional, basada en los casos de prueba de caja blanca. Se utiliza la herramienta OnlineGDB.

Antes de entrar a detallar los casos de prueba, utilizando las herramientas Code2flow y Lizard (para generar diagramas de flujo y medir la complejidad, respectivamente) se puede establecer qué función necesita un mayor número de casos de prueba por su complejidad y qué casos de prueba son necesarios para cumplir con la cobertura de decisiones y condiciones.

En concreto, para el ejemplo desarrollado en este trabajo en ambas versiones, la función que puede tener una mayor de defectos y, por lo tanto, en la que es necesario testear con más profundidad y detalle, es *checktriangle* con complejidad 11. Un valor que supone una variación del 450% con respecto a la complejidad de la función *main*, a pesar de que la longitud solo ha variado en 140%. Los resultados de Lizard para las versiones A y B se muestran en la Figura 8 y la Figura 9, respectivamente.

Code analyzed successfully.

File Type **.C** Token Count **292** NLOC **51**

Function Name	NLOC	Complexity	Token #	Parameter #
checktriangle	36	11	174	
main	15	2	116	

Figura 8 - Análisis de complejidad para la versión A del código

Code analyzed successfully.

File Type **.C** Token Count **297** NLOC **54**

Function Name	NLOC	Complexity	Token #	Parameter #
checktriangle	36	11	174	
main	16	2	117	

Figura 9 - Resultados de complejidad para la versión B del código

El diagrama de flujo del código, aunque no es obligatorio, es de gran ayuda a la hora de establecer los casos de prueba necesarios para conseguir una cobertura de decisiones y condiciones completa y asegurar que cada una de ellas toma se evalúa tanto en True como en False tras la ejecución de las pruebas [6].

En la Figura 10, se muestra el diagrama de flujo obtenido con Code2flow para la versión A del programa.

Como esta herramienta no descompone en condiciones las decisiones compuestas, se desarrolla de forma manual un diagrama de flujo (ver Figura 11) y así poder establecer fácilmente los casos de prueba necesarios para cumplir con la cobertura de condiciones.

Hay que aclarar que las diferencias entre las versiones A y B del programa no se traducen en diferencias en condiciones ni decisiones en los diagramas de flujo (únicamente se modifica un tipo de dato) y, por eso, se ha optado por incluir, solamente, los diagramas de la versión A en este trabajo.

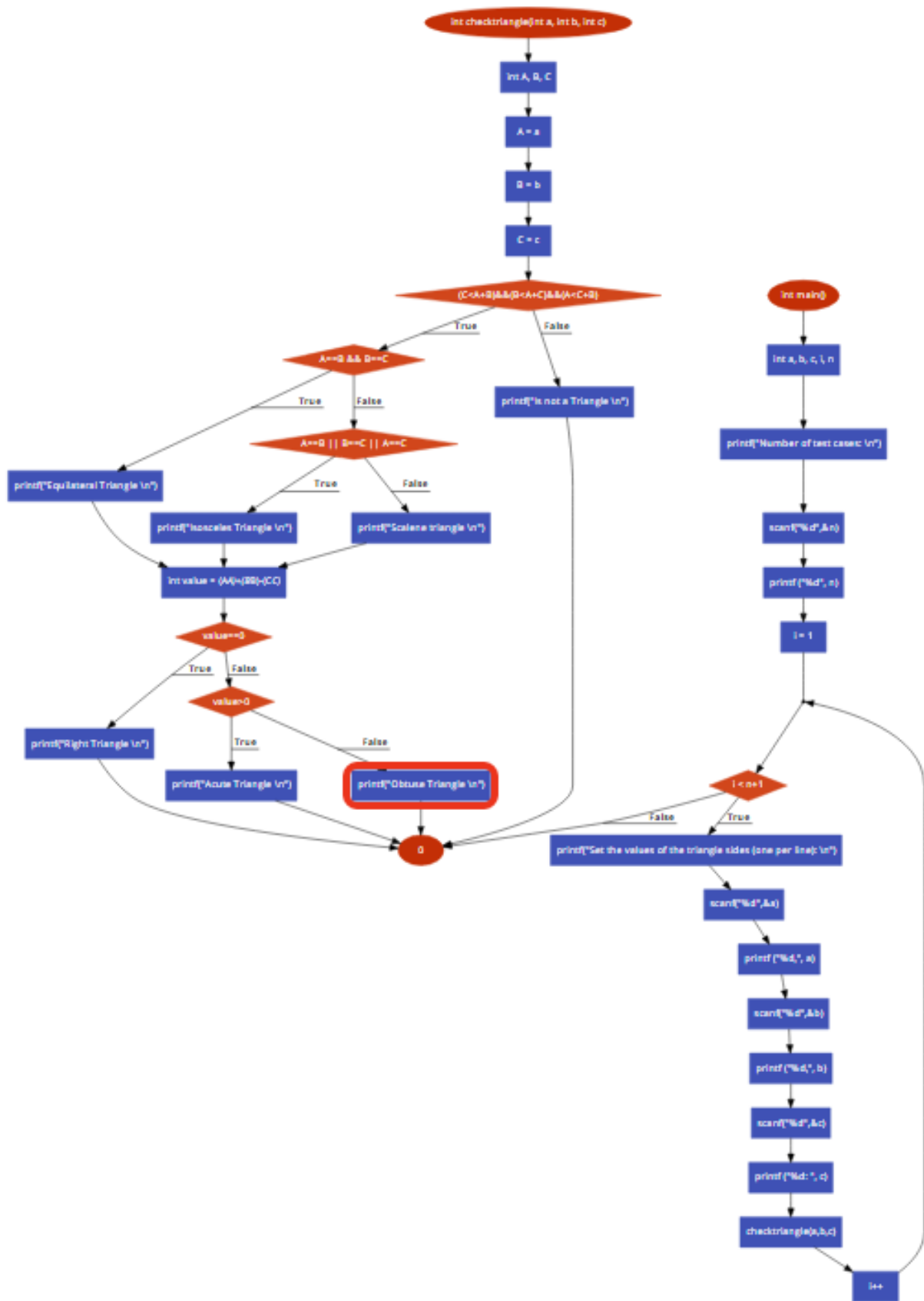


Figura 10 - Diagrama de flujo de versión A del flujo atendiendo al criterio de decisiones

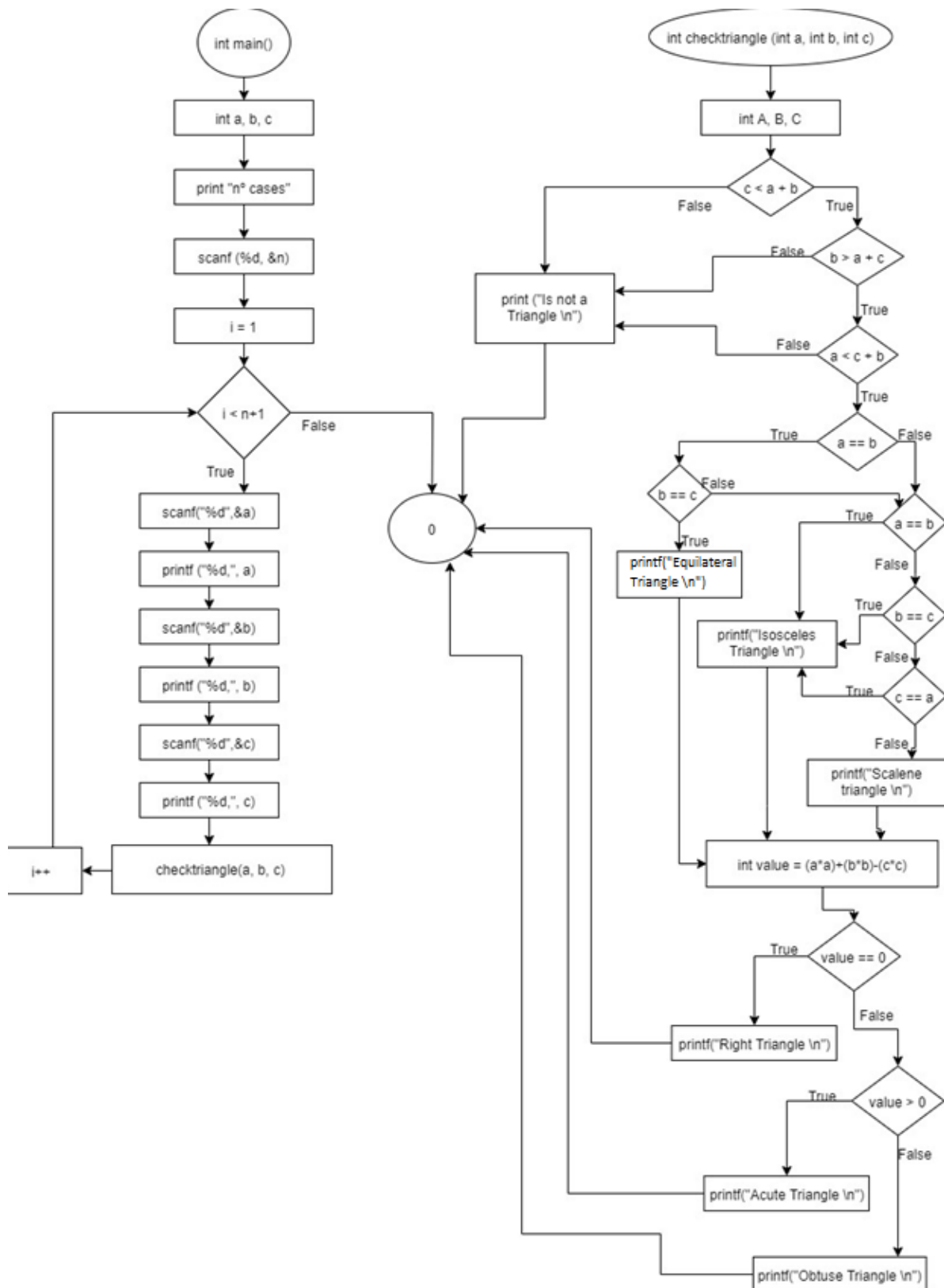


Figura 11 - Diagrama de flujo de versión A del flujo atendiendo al criterio de condiciones

Finalmente, tras evaluar todas estas consideraciones se establecen las entradas concretas necesarias para cumplir con los distintos criterios de cobertura tal y como se puede ver en la Tabla 2, a continuación.



<i>Casos de prueba</i>	<i>Valor de entrada</i>	<i>Cob. de funciones</i>	<i>Cob. de sentencias</i>	<i>Cob. de decisiones</i>	<i>Cob. de condiciones</i>	<i>Cobertura funcional</i>	<i>Salida esperada</i>
n entero	1	x	x	x	x	(21) (23)	Introduce valores para los lados
n no entero	2,5					(22)	Error: Formato no válido
n no número	z					(24)	Error: Formato no válido
Escaleno rectángulo válido	(3, 4, 5)	x	x	x	x	(1) (2A) (2B) (7) (9) (11) (13) (15)	Triángulo escaleno rectángulo
Escaleno acutángulo válido	(4, 5, 6)				x	(1) (2A) (2B) (7) (9) (11) (13) (16)	Triangulo escaleno acutángulo
Escaleno obtusángulo válido	(10, 6, 5)				x	(1) (2A) (2B) (7) (9) (11) (13) (17)	Triangulo escaleno obtusángulo
Isósceles acutángulo válido						(1) (2A) (2B) (7) (9) (11) (13) (18)	Triangulo isósceles acutángulo
A==B	(9, 9, 2)				x		
B==C	(2, 9, 9)				x		
A==C	(9, 2, 9)				x		
Isósceles obtusángulo válido						(1) (2A) (2B) (7) (9) (11) (13) (19)	Triángulo isósceles obtusángulo
A==B	(5, 5, 9)		x	x	x		
B==C	(9, 5, 5)				x		
A==C	(5, 9, 5)				x		
Equilátero acutángulo válido	(5, 5, 5)		x	x	x	(1) (2A) (2B) (7) (9) (11) (13) (20)	Triángulo equilátero acutángulo
$A \geq B + C$	(3, 2, 1)		x	x	x	(10)	No es un triángulo
$B \geq A + C$	(2, 3, 1)				x	(12)	No es un triángulo
$C \geq A + B$	(1, 2, 3)				x	(14)	No es un triángulo
Lado cero	(0, 1, 3)					(8)	Error: Lado no válido
Lado negativo	(-1, 3, 4)					(8)	Error: Lado no válido
Punto singular	(0, 0, 0)					(8)	Error: Lado no válido
Menos de 3 datos	(3,5)					(3)	Error: Formato de no válido
Más de 3 datos	(3, 5, 4, 4)					(4)	Error: Formato de no válido
No entero	(5.5, 4, 7)					(5A)	Error: Formato de lado no válido
No número	(1, *, 3)					(6)	Error: Formato de lado no válido

Máximo entero. Límite válido	(2147483647, 2147483647, 2147483647)	(25A)	Triángulo equilátero acutángulo
Máximo entero. Límite no válido	(2147483648, 2147483648, 2147483648)	(25A)	Error: Valores por encima del límite
Máximo en operaciones internas con enteros. Límite válido	(46340, 46340, 46340)	(26A)	Triángulo equilátero acutángulo
Máximo en operaciones internas con enteros. Límite válido	(46341, 46341, 46341)	(26A)	Error: valores no válidos
Máximo real. Límite válido	3.402823E+38	(25B)	Triángulo equilátero acutángulo
Máximo real. Límite no válido	3.402823E+38	(25B)	Error: Valores por encima del límite
Máximo en operaciones internas con reales. Límite válido.	1.844674E+19	(26B)	Triángulo equilátero acutángulo
Máximo en operaciones internas con reales. Límite no válido.	1.844674E+19	(26B)	Error: valores no válidos
Máximo número de decimales. Límite válido. Seis cifras decimales.	(30, 25 5.000001)	(27B)	Triángulo escaleno acutángulo
Máximo número de decimales. Límite no válido. Siete cifras decimales.	(30, 25, 5.0000009)	(27B)	Error: Valores por encima del límite
Límite. Escaleno casi rectángulo (obtusángulo)	(3, 4, 5.000001)	(28B)	Triángulo escaleno obtusángulo
Límite. Escaleno casi rectángulo (acutángulo)	(3, 4, 4.999999)	(29B)	Triángulo escaleno acutángulo

Tabla 2- Suites de prueba según criterios de cobertura

Los límites de los distintos tipos de datos en lenguaje C se obtienen accediendo a las respectivas constantes de macro. La explicación detallada y el código utilizado se pueden consultar en el Anexo II Límites numéricos .

## 2.6. Aplicación de las suites de pruebas y análisis de resultados

### 2.6.1. Versión A

#### Cobertura de funciones

Se ejecutan los test diseñados para conseguir la cobertura de funciones con ayuda de la herramienta Expcov, como se muestra en la Figura 12.

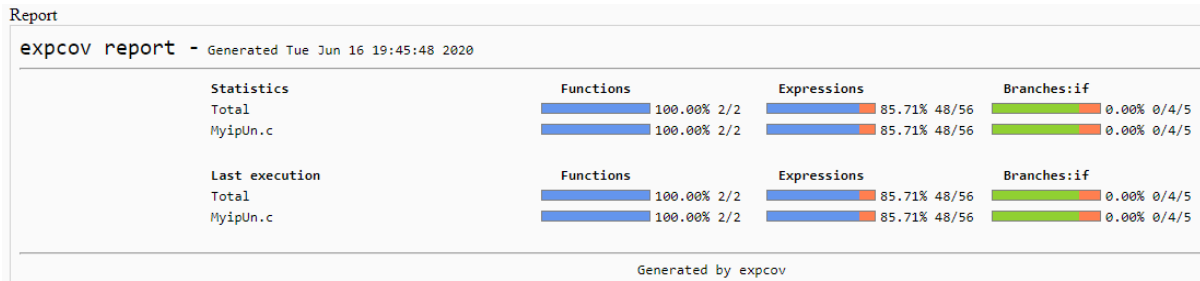


Figura 12- Informe de resultados de cobertura del programa A para la suite de cobertura de funciones

Se recoge la salida obtenida (ver Figura 13) para compararla con los resultados esperados (ver Tabla 3) y comprobar el resultado de los test.

```
Standard output
Number of test cases:
1Set the values of the triangle sides (one per line):
3,4,5: Scalene triangle
Right Triangle
```

Figura 13- Salida del programa A para la suite de cobertura de funciones

Casos de prueba	Valor de entrada	Salida esperada	Salida obtenida	Resultado del test
<b>N entero</b>	1	Introduce valores para los lados	Set the values of the triangle sides	Pasa
<b>Escaleno rectángulo válido</b>	(3, 4, 5)	Triángulo escaleno rectángulo	Scalene triangle Right Triangle	Pasa

Tabla 3 - Resultados de la ejecución de la suite de cobertura de funciones para el programa A

#### Cobertura de sentencias y decisiones

En este caso la cobertura de sentencias y de decisiones van de la mano y si se evalúan todas las decisiones con valor V y F se ejecutan todas las sentencias del programa. En la Figura 14 se muestra el informe generado en Expcov. Con esta suite de test se consigue la cobertura del 100% de sentencias y de condiciones.

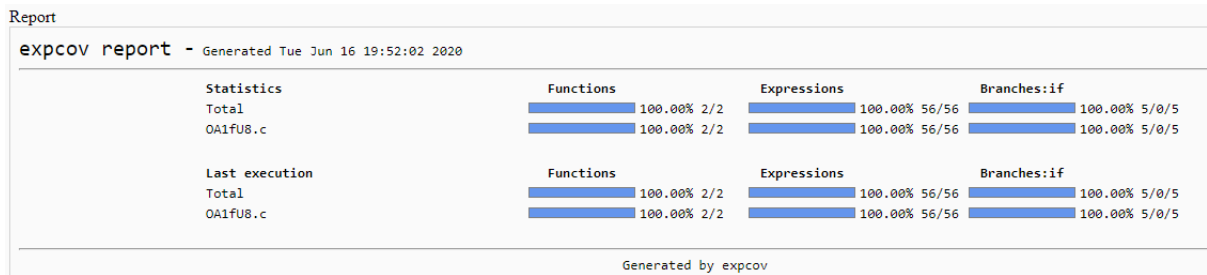


Figura 14 - Informe de resultados de cobertura del programa A para la suite de cobertura de sentencias y condiciones

La salida obtenida en la ejecución de las pruebas (ver Figura 15) se compara con la salida prevista al diseñar los test (ver Tabla 4).

```
Standard output
Number of test cases:
4Set the values of the triangle sides (one per line):
3,4,5: Scalene triangle
Right Triangle
Set the values of the triangle sides (one per line):
5,5,9: Isosceles Triangle
Obtuse Triangle
Set the values of the triangle sides (one per line):
5,5,5: Equilateral Triangle
Acute Triangle
Set the values of the triangle sides (one per line):
3,2,1: Is not a Triangle
```

Figura 15 - Salida del programa A para la suite de cobertura de sentencias y decisiones

Casos de prueba	Valor de entrada	Salida esperada	Salida obtenida	Resultado del test
<b>N entero</b>	4	Introduce valores para los lados	Set the values of the triangle sides	Pasa
<b>Escaleno rectángulo válido</b>	(3, 4, 5)	Triángulo escaleno rectángulo	Scalene triangle Right Triangle	Pasa
<b>Isósceles obtusángulo válido (A=B)</b>	(5, 5, 9)	Triángulo isósceles obtusángulo	Isosceles Triangle Obtuse Triangle	Pasa
<b>Equilátero acutángulo válido</b>	(5, 5, 5)	Triángulo equilátero acutángulo	Equilateral Triangle Acute Triangle	Pasa
<b>A ≥ B + C</b>	(3, 2, 1)	No es un triángulo	Is not a Triangle	Pasa

Tabla 4 - Resultados de la ejecución de la suite de cobertura de sentencias y decisiones para el programa A

### Cobertura de condiciones

Para conseguir la cobertura de condiciones se ejecutan los test en la herramienta Online GDB. Ya que no hay una herramienta para medir la cobertura de condiciones disponible online, se utiliza un depurador que permite identificar fácilmente el valor de las condiciones a medida que se ejecuta el código. Se recoge la salida de la Figura 16 y se compara con los resultados esperados en la Tabla 5 para determinar si se puede detectar algún defecto.



```
input                                stdout
Compiled Successfully. memory: 1536 time: 0.01 exit code: 0
Number of test cases:
14Set the values of the triangle sides (one per line):
3,4,5: Scalene triangle
Right Triangle
Set the values of the triangle sides (one per line):
4,5,6: Scalene triangle
Acute Triangle
Set the values of the triangle sides (one per line):
10,6,5: Scalene triangle
Acute Triangle
Set the values of the triangle sides (one per line):
9,9,2: Isosceles Triangle
Acute Triangle
Set the values of the triangle sides (one per line):
2,9,9: Isosceles Triangle
Acute Triangle
Set the values of the triangle sides (one per line):
9,2,9: Isosceles Triangle
Acute Triangle
Set the values of the triangle sides (one per line):
5,5,9: Isosceles Triangle
Obtuse Triangle
Set the values of the triangle sides (one per line):
9,5,5: Isosceles Triangle
Acute Triangle
Set the values of the triangle sides (one per line):
5,9,5: Isosceles Triangle
Acute Triangle
Set the values of the triangle sides (one per line):
5,5,5: Equilateral Triangle
Acute Triangle
Set the values of the triangle sides (one per line):
3,2,1: Is not a Triangle
Set the values of the triangle sides (one per line):
2,3,1: Is not a Triangle
Set the values of the triangle sides (one per line):
1,2,3: Is not a Triangle
Set the values of the triangle sides (one per line):
0,1,3: Is not a Triangle
```

Figura 16 - Salida del programa A para la suite de cobertura de condiciones

Casos de prueba	Valor de entrada	Salida esperada	Salida obtenida	Resultado del test
<b>N entero</b>	13	Introduce valores para los lados	Set the values of the triangle sides	Pasa
<b>Escaleno rectángulo válido</b>	(3, 4, 5)	Triángulo escaleno rectángulo	Scalene triangle Right Triangle	Pasa
<b>Escaleno acutángulo válido</b>	(4, 5, 6)	Triángulo escaleno acutángulo	Scalene triangle Acute Triangle	Pasa
<b>Escaleno obtusángulo válido</b>	(10, 6, 5)	Triángulo escaleno obtusángulo	Scalene triangle Acute Triangle	<b>Falla</b>
<b>Isósceles acutángulo válido (A=B)</b>	(9, 9, 2)	Triángulo isósceles acutángulo	Isosceles Triangle Acute Triangle	Pasa
<b>Isósceles acutángulo válido (B=C)</b>	(2, 9, 9)	Triángulo isósceles acutángulo	Isosceles Triangle Acute Triangle	Pasa
<b>Isósceles acutángulo válido (A=B)</b>	(9, 2, 9)	Triángulo isósceles acutángulo	Isosceles Triangle Acute Triangle	Pasa
<b>Isósceles obtusángulo válido (A=B)</b>	(5, 5, 9)	Triángulo isósceles obtusángulo	Isosceles Triangle Obtuse Triangle	Pasa
<b>Isósceles obtusángulo válido (B=C)</b>	(9, 5, 5)	Triángulo isósceles obtusángulo	Isosceles Triangle Acute Triangle	<b>Falla</b>
<b>Isósceles obtusángulo válido (A=C)</b>	(5, 9, 5)	Triángulo isósceles obtusángulo	Isosceles Triangle Acute Triangle	<b>Falla</b>
<b>Equilátero acutángulo válido</b>	(5, 5, 5)	Triángulo equilátero acutángulo	Equilateral Triangle Acute Triangle	Pasa
<b>A ≥ B + C</b>	(3, 2, 1)	No es un triángulo	Is not a Triangle	Pasa
<b>B ≥ A + C</b>	(2, 3, 1)	No es un triángulo	Is not a Triangle	Pasa
<b>C ≥ A + B</b>	(1, 2, 3)	No es un triángulo	Is not a Triangle	Pasa

Tabla 5 - Resultados de la ejecución de la suite de cobertura de condiciones para el programa A

En la ejecución de esta suite de pruebas sí hay test que fallan porque no se obtienen los resultados esperados. En concreto, el fallo se traduce en la identificación incorrecta de ángulos obtusos como agudos en función del orden en que se hayan introducido las longitudes de los lados. El defecto del código se muestra como fallo siempre que el lado mayor no se introduce en último lugar.

### Cobertura funcional

Para ejecutar estos casos de prueba también se utiliza la herramienta Online GDB. Estos test permitirán confirmar los fallos mostrados en el apartado anterior. La salida obtenida se muestra en la Figura 17 a excepción de las pruebas de casos no válidos para el formato de número de ejecuciones del programa (n) (ver Figura 18 y Figura 19) que no se pueden ejecutar como lote ya que en el programa no se manejan excepciones y finaliza la ejecución del programa.

También se ejecutan por separado las pruebas para un número de lados distinto de tres (ver Figura 20 y Figura 21) y para formatos de lado no válidos (ver Figura 22y Figura 23).

```
Compiled Successfully. memory: 1620 time: 0.61 exit code: 0
Number of test cases:
19Set the values of the triangle sides (one per line):
3,4,5: Scalene triangle
Right Triangle
Set the values of the triangle sides (one per line):
4,5,6: Scalene triangle
Acute Triangle
Set the values of the triangle sides (one per line):
10,6,5: Scalene triangle
Acute Triangle
Set the values of the triangle sides (one per line):
9,9,2: Isosceles Triangle
Acute Triangle
Set the values of the triangle sides (one per line):
2,9,9: Isosceles Triangle
Acute Triangle
Set the values of the triangle sides (one per line):
9,2,9: Isosceles Triangle
Acute Triangle
Set the values of the triangle sides (one per line):
5,5,9: Isosceles Triangle
Obtuse Triangle
Set the values of the triangle sides (one per line):
9,5,5: Isosceles Triangle
Acute Triangle
Set the values of the triangle sides (one per line):
5,9,5: Isosceles Triangle
Acute Triangle
Set the values of the triangle sides (one per line):
5,5,5: Equilateral Triangle
Acute Triangle
Set the values of the triangle sides (one per line):
3,2,1: Is not a Triangle
Set the values of the triangle sides (one per line):
2,3,1: Is not a Triangle
Set the values of the triangle sides (one per line):
1,2,3: Is not a Triangle
Set the values of the triangle sides (one per line):
0,1,3: Is not a Triangle

Set the values of the triangle sides (one per line):
-1,3,4: Is not a Triangle
Set the values of the triangle sides (one per line):
2147483647,2147483647,2147483647: Is not a Triangle
Set the values of the triangle sides (one per line):
-2147483648,-2147483648,-2147483648: Equilateral Triangle
Right Triangle
Set the values of the triangle sides (one per line):
46340,46340,46340: Equilateral Triangle
Acute Triangle
Set the values of the triangle sides (one per line):
46341,46341,46341: Equilateral Triangle
Obtuse Triangle
```

Figura 17 - Salida del programa A para la suite de cobertura funcional

```
input
Compiled Successfully. memory: 1588 time: 0.01 exit code: 0

Number of test cases:
2Set the values of the triangle sides (one per line):
0,-1380022176,32767: Is not a Triangle
Set the values of the triangle sides (one per line):
0,-1380022176,32767: Is not a Triangle
```

Figura 18- Salida del programa A para el caso de prueba "Número no entero de n"

```
input
Compiled Successfully. memory: 1520 time: 0.01 exit code: 0

Number of test cases:
0
```

Figura 19- Salida del programa A para el caso de prueba "Caracter no numérico en n"

```
input
Compiled Successfully. memory: 1664 time: 1.12 exit code: 0

Number of test cases:
1Set the values of the triangle sides (one per line):
3,5,32767: Is not a Triangle
```

Figura 20- Salida del programa A para el caso de prueba "Menos de 3 lados"

```
input
Compiled Successfully. memory: 1552 time: 0 exit code: 0

Number of test cases:
1Set the values of the triangle sides (one per line):
3,5,4: Scalene triangle
Acute Triangle
```

Figura 21- Salida del programa A para el caso de prueba "Más de 3 lados"

```
input
Compiled Successfully. memory: 1600 time: 0.02 exit code: 0

Number of test cases:
1Set the values of the triangle sides (one per line):
5,-1206943952,32767: Is not a Triangle
```

Figura 22- Salida del programa A para el caso de prueba "Lado no entero"



```
Compiled Successfully. memory: 1572 time: 0.09 exit code: 0
```

```
Number of test cases:
1Set the values of the triangle sides (one per line):
1,-1535975808,32765: Is not a Triangle
```

Figura 23 - Salida del programa A para el caso de prueba "Lado no número"

La comparativa entre salidas esperadas y obtenidas para cada caso de prueba y el resultado del test se muestra en la Tabla 6.

Casos de prueba	Valor de entrada	Salida esperada	Salida obtenida	Resultado del test
N no entero	2,5	Error: Formato no válido	Is not a triangle	Falla
N no número	z	Error: Formato no válido	0	Falla
N entero	19	Introduce valores para los lados	Set the values of the triangle sides	Pasa
Escaleno rectángulo válido	(3, 4, 5)	Triángulo escaleno rectángulo	Scalene triangle Right Triangle	Pasa
Escaleno acutángulo válido	(4, 5, 6)	Triangulo escaleno acutángulo	Scalene triangle Acute Triangle	Pasa
Escaleno obtusángulo válido	(10, 6, 5)	Triangulo escaleno obtusángulo	Scalene triangle Acute Triangle	Falla
Isósceles acutángulo válido (A=B)	(9, 9, 2)	Triángulo isósceles acutángulo	Isosceles Triangle Acute Triangle	Pasa
Isósceles acutángulo válido (B=C)	(2, 9, 9)	Triángulo isósceles acutángulo	Isosceles Triangle Acute Triangle	Pasa
Isósceles acutángulo válido (A=B)	(9, 2, 9)	Triángulo isósceles acutángulo	Isosceles Triangle Acute Triangle	Pasa
Isósceles obtusángulo válido (A=B)	(5, 5, 9)	Triángulo isósceles obtusángulo	Isosceles Triangle Obtuse Triangle	Pasa
Isósceles obtusángulo válido (B=C)	(9, 5, 5)	Triángulo isósceles obtusángulo	Isosceles Triangle Acute Triangle	Falla
Isósceles obtusángulo válido (A=C)	(5, 9, 5)	Triángulo isósceles obtusángulo	Isosceles Triangle Acute Triangle	Falla
Equilátero acutángulo válido	(5, 5, 5)	Triángulo equilátero acutángulo	Equilateral Triangle Acute Triangle	Pasa
$A \geq B + C$	(3, 2, 1)	No es un triángulo	Is not a Triangle	Pasa
$B \geq A + C$	(2, 3, 1)	No es un triángulo	Is not a Triangle	Pasa

Casos de prueba	Valor de entrada	Salida esperada	Salida obtenida	Resultado del test
$C \geq A + B$	(1, 2, 3)	No es un triángulo	Is not a Triangle	Pasa
Lado cero	(0, 1, 3)	Error: Formato de lado no válido	Is not a Triangle	Pasa*
Lado negativo	(-1, 3, 4)	Error: Formato de lado no válido	Is not a Triangle	Pasa*
Punto singular	(0, 0, 0)	Error: Formato de lado no válido	Is not a Triangle	Pasa*
Menos de 3 datos	(3, 5)	Error: Formato de lado no válido	Is not a Triangle	Falla
Más de 3 datos	(3, 5, 4, 4)	Error: Formato de lado no válido	Scalene triangle Acute Triangle	Falla
Lado No entero	(5'5, 4, 3)	Error: Formato de lado no válido	Is not a Triangle	Falla
Lado no número	(1, *, 3)	Error: Formato de lado no válido	Is not a Triangle	Falla

Tabla 6 - Resultados de la ejecución de la suite de cobertura funcional para el programa A

Con las pruebas para el criterio de cobertura funcional se detectan los fallos ya comentados en el apartado anterior sobre identificación de ángulos, pero, además, la ejecución de estas pruebas pone de manifiesto falta de control en el formato de los datos introducidos. En todos los casos cuando el dato introducido no tiene el formato adecuado (entero en este caso), el compilador toma como dato para la variable valores residuales de memoria.

Los resultados marcados con asterisco (\*) se ha considerado que son positivos porque no implican resultados erróneos. Sin embargo, sería necesario incluir control de las entradas para mejorar la calidad del software.

De todos los test, es interesante destacar que al introducir uno de los lados como número decimal ese valor se trunca y el programa completa todas las ejecuciones pendientes con valores residuales. Se puede comprobar en la Figura 24, en este caso se quiere realizar 3 ejecuciones de programa para los triángulos de lados (5.5, 3, 4), (1, 1, 1) y (9, 9, 2).

```
Compiled Successfully. memory: 1520 time: 0.03 exit code: 0

Number of test cases:
4Set the values of the triangle sides (one per line):
5,1286946672,32765: Is not a Triangle
Set the values of the triangle sides (one per line):
5,1286946672,32765: Is not a Triangle
Set the values of the triangle sides (one per line):
5,1286946672,32765: Is not a Triangle
Set the values of the triangle sides (one per line):
5,1286946672,32765: Is not a Triangle
```

Figura 24 - Resultado de introducir un valor decimal para el lado en el programa A

## 2.6.2. Versión B

La versión B del programa considera las longitudes de los triángulos como *float*. La lógica de la implementación es la misma y por lo tanto es de esperar que se obtengan los mismos resultados de los test que son comunes a ambas versiones, los test para la cobertura de funciones, sentencias y decisiones y condiciones.

En las respectivas secciones de este documento se muestran imágenes de los resultados obtenidos con las distintas herramientas, pero, para la comparación de los resultados con la salida esperada se remite a la tabla correspondiente del programa versión A si los resultados son los mismos.

### Cobertura de funciones

La ejecución de la suite diseñada para este caso muestra, como se puede ver en la Figura 25, una cobertura de funciones del 100% en expcov.

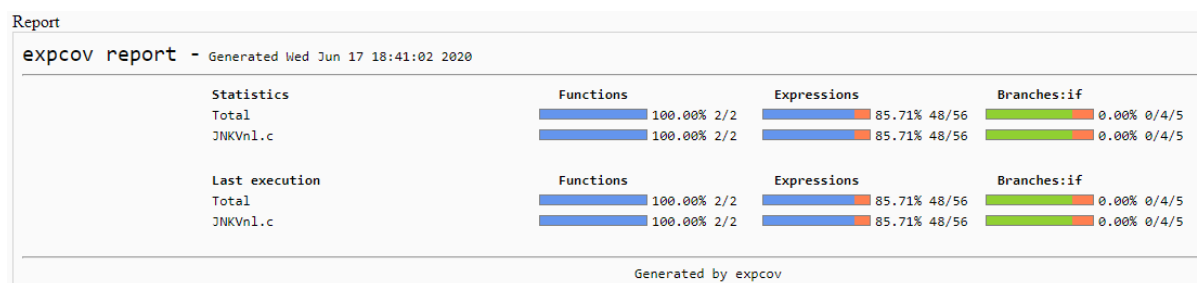


Figura 25 - Informe de resultados de cobertura del programa B para la suite de cobertura de funciones

La salida que se obtiene (ver Figura 26) coincide con la esperada, tal y como se puede comprobar en la Tabla 3 - Resultados de la ejecución de la suite de cobertura de funciones para el programa A.

### Standard output

```
Number of test cases:
1Set the values of the triangle sides (one per line):
3.000000,4.000000,5.000000: Scalene triangle
Right Triangle
```

Figura 26 - Salida del programa B para la suite de cobertura de funciones

### Cobertura de sentencias y decisiones

Con los mismos valores de entrada utilizados para la cobertura de sentencias y decisiones en el programa versión A, se consigue una cobertura del 100% en el programa B (ver Figura 27).

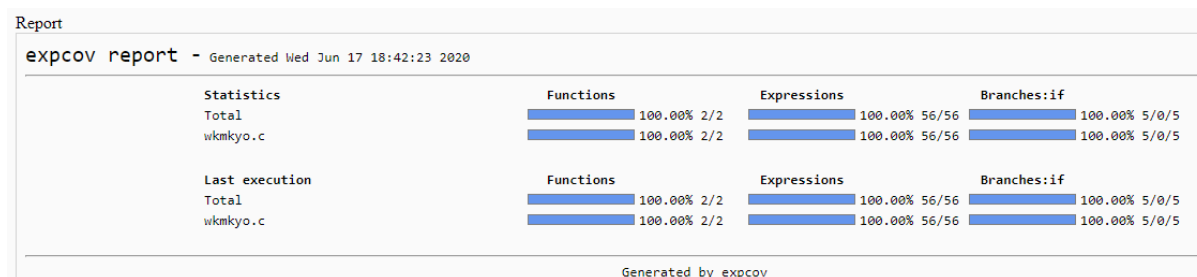


Figura 27 - Informe de resultados de cobertura del programa B para la suite de cobertura de sentencias y condiciones

En la Figura 28 se puede ver la salida obtenida con la ejecución de estas pruebas, que coincide con la salida esperada (y por lo tanto los resultados de los test son positivos) tal y como se detalla en la Tabla 4 - Resultados de la ejecución de la suite de cobertura de sentencias y decisiones para el programa A.

Standard output

```

Number of test cases:
4Set the values of the triangle sides (one per line):
3.000000,4.000000,5.000000: Scalene triangle
Right Triangle
Set the values of the triangle sides (one per line):
5.000000,5.000000,9.000000: Isosceles Triangle
Obtuse Triangle
Set the values of the triangle sides (one per line):
5.000000,5.000000,5.000000: Equilateral Triangle
Acute Triangle
Set the values of the triangle sides (one per line):
3.000000,2.000000,1.000000: Is not a Triangle

```

Figura 28 - Salida del programa B para la suite de cobertura de sentencias y decisiones

*Cobertura de condiciones*

Los test para la cobertura de condiciones fueron los que revelaron los fallos para identificar el tipo de triángulo según sus ángulos en las ejecuciones de pruebas de la versión A. En este caso, para la versión B, en vez de utilizar la misma suite de entradas con la que se obtendrían resultados idénticos, se decide incluir en las entradas todas las posibles permutaciones en el orden de entrada de los lados y así identificar correctamente el defecto del código.

La suite diseñada en la sección 2.5 se complementa con las entradas que se muestran a continuación en la Tabla 7.

Caso de prueba	Valores de entrada	Salida esperada
Triángulo escaleno rectángulo (con permutaciones)	a = 3, b = 4, c = 5 a = 3, b = 5, c = 4 a = 5, b = 3, c = 4 a = 5, b = 4, c = 3 a = 4, b = 3, c = 5 a = 4, b = 5, c = 3	Triángulo escaleno rectángulo
Triángulo escaleno acutángulo (con permutaciones)	a = 4, b = 5, c = 6 a = 4, b = 6, c = 5 a = 5, b = 4, c = 6 a = 5, b = 6, c = 4 a = 6, b = 4, c = 5 a = 6, b = 5, c = 4	Triángulo escaleno acutángulo
Triángulo escaleno obtusángulo (con permutaciones)	a = 5, b = 6, c = 10 a = 5, b = 10, c = 6 a = 6, b = 5, c = 10 a = 6, b = 10, c = 5 a = 10, b = 5, c = 6 a = 10, b = 6, c = 5	Triángulo escaleno obtusángulo
Triángulo isósceles acutángulo (con permutaciones)	a = 2, b = 9, c = 9 a = 9, b = 2, c = 9 a = 9, b = 9, c = 2	Triángulo isósceles acutángulo

Triángulo isósceles obtusángulo (con permutaciones)	a = 5, b = 5, c = 9 a = 5, b = 9, c = 5 a = 9, b = 5, c = 5	Triángulo isósceles obtusángulo
Triángulo equilátero acutángulo	a = 5, b = 5, c = 5	Triángulo equilátero acutángulo

Tabla 7- Casos de prueba añadidos a la suite para la cobertura de condiciones

La salida obtenida en la Figura 29 se compara con la salida esperada en la Tabla 8 para determinar si alguna de las pruebas falla.

```

input
Compiled Successfully. memory: 1536 time: 0.01 exit code: 0
Number of test cases:
28Set the values of the triangle sides (one per line):
3.000000,4.000000,5.000000: Scalene triangle
Right Triangle
Set the values of the triangle sides (one per line):
3.000000,5.000000,4.000000: Scalene triangle
Acute Triangle
Set the values of the triangle sides (one per line):
5.000000,3.000000,4.000000: Scalene triangle
Acute Triangle
Set the values of the triangle sides (one per line):
5.000000,4.000000,3.000000: Scalene triangle
Acute Triangle
Set the values of the triangle sides (one per line):
4.000000,3.000000,5.000000: Scalene triangle
Right Triangle
Set the values of the triangle sides (one per line):
4.000000,5.000000,3.000000: Scalene triangle
Acute Triangle
Set the values of the triangle sides (one per line):
4.000000,5.000000,6.000000: Scalene triangle
Acute Triangle
Set the values of the triangle sides (one per line):
4.000000,6.000000,5.000000: Scalene triangle
Acute Triangle
Set the values of the triangle sides (one per line):
5.000000,4.000000,6.000000: Scalene triangle
Acute Triangle
Set the values of the triangle sides (one per line):
5.000000,6.000000,4.000000: Scalene triangle
Acute Triangle
Set the values of the triangle sides (one per line):
6.000000,4.000000,5.000000: Scalene triangle
Acute Triangle
Set the values of the triangle sides (one per line):
6.000000,5.000000,4.000000: Scalene triangle
Acute Triangle

```

```
Set the values of the triangle sides (one per line):
5.000000,6.000000,10.000000: Scalene triangle
Obtuse Triangle
Set the values of the triangle sides (one per line):
5.000000,10.000000,6.000000: Scalene triangle
Acute Triangle
Set the values of the triangle sides (one per line):
6.000000,5.000000,10.000000: Scalene triangle
Obtuse Triangle
Set the values of the triangle sides (one per line):
6.000000,10.000000,5.000000: Scalene triangle
Acute Triangle
Set the values of the triangle sides (one per line):
10.000000,5.000000,6.000000: Scalene triangle
Acute Triangle
Set the values of the triangle sides (one per line):
10.000000,6.000000,5.000000: Scalene triangle
Acute Triangle
Set the values of the triangle sides (one per line):
2.000000,9.000000,9.000000: Isosceles Triangle
Acute Triangle
Set the values of the triangle sides (one per line):
9.000000,2.000000,9.000000: Isosceles Triangle
Acute Triangle
Set the values of the triangle sides (one per line):
9.000000,9.000000,2.000000: Isosceles Triangle
Acute Triangle
Set the values of the triangle sides (one per line):
5.000000,5.000000,9.000000: Isosceles Triangle
Obtuse Triangle
Set the values of the triangle sides (one per line):
9.000000,5.000000,5.000000: Isosceles Triangle
Acute Triangle
Set the values of the triangle sides (one per line):
5.000000,9.000000,5.000000: Isosceles Triangle
Acute Triangle

Set the values of the triangle sides (one per line):
5.000000,5.000000,5.000000: Equilateral Triangle
Acute Triangle
Set the values of the triangle sides (one per line):
3.000000,2.000000,1.000000: Is not a Triangle
Set the values of the triangle sides (one per line):
2.000000,3.000000,1.000000: Is not a Triangle
Set the values of the triangle sides (one per line):
1.000000,2.000000,3.000000: Is not a Triangle
```

Figura 29 - Salida del programa B para la suite de cobertura de condiciones

Casos de prueba	Valor de entrada	Salida esperada	Salida obtenida	Resultado del test
<b>N entero</b>	28	Introduce valores para los lados	Set the values of the triangle sides	Pasa
<b>Escaleno rectángulo válido</b>	(3, 4, 5)	Triángulo escaleno rectángulo	Scalene triangle Right Triangle	Pasa
	(3, 5, 4)	Triángulo escaleno rectángulo	Scalene triangle Acute Triangle	Falla
	(5, 3, 4)	Triángulo escaleno rectángulo	Scalene triangle Acute Triangle	Falla
	(5, 4, 3)	Triángulo escaleno rectángulo	Scalene triangle Acute Triangle	Falla
	(4, 3, 5)	Triángulo escaleno rectángulo	Scalene triangle Right Triangle	Pasa
	(4, 5, 3)	Triángulo escaleno rectángulo	Scalene triangle Acute Triangle	Falla
<b>Escaleno acutángulo válido</b>	(4, 5, 6)	Triangulo escaleno acutángulo	Scalene triangle Acute Triangle	Pasa
	(4, 6, 5)	Triangulo escaleno acutángulo	Scalene triangle Acute Triangle	Pasa
	(5, 4, 6)	Triangulo escaleno acutángulo	Scalene triangle Acute Triangle	Pasa
	(5, 6, 4)	Triangulo escaleno acutángulo	Scalene triangle Acute Triangle	Pasa
	(6, 4, 5)	Triangulo escaleno acutángulo	Scalene triangle Acute Triangle	Pasa
	(6, 5, 4)	Triangulo escaleno acutángulo	Scalene triangle Acute Triangle	Pasa
<b>Escaleno obtusángulo válido</b>	(5, 6, 10)	Triangulo escaleno obtusángulo	Scalene triangle Obtuse Triangle	Pasa
	(5, 10, 6)	Triangulo escaleno obtusángulo	Scalene triangle Acute Triangle	Falla
	(6, 5, 10)	Triangulo escaleno obtusángulo	Scalene triangle Obtuse Triangle	Pasa
	(6, 10, 5)	Triangulo escaleno obtusángulo	Scalene triangle Acute Triangle	Falla
	(10, 5, 6)	Triangulo escaleno obtusángulo	Scalene triangle Acute Triangle	Falla
	(10, 6, 5)	Triangulo escaleno obtusángulo	Scalene triangle Acute Triangle	Falla
<b>Isósceles acutángulo válido</b>	(2, 9, 9)	Triángulo isósceles acutángulo	Isosceles Triangle Acute Triangle	Pasa
	(9, 2, 9)	Triángulo isósceles acutángulo	Isosceles Triangle Acute Triangle	Pasa
	(9, 9, 2)	Triángulo isósceles acutángulo	Isosceles Triangle Acute Triangle	Pasa

<b>Isósceles obtusángulo válido</b>	(5, 5, 9)	Triángulo isósceles obtusángulo	Isosceles Triangle Obtuse Triangle	Pasa
	(9, 5, 5)	Triángulo isósceles obtusángulo	Isosceles Triangle Acute Triangle	<b>Falla</b>
	(5, 9, 5)	Triángulo isósceles obtusángulo	Isosceles Triangle Acute Triangle	<b>Falla</b>
<b>Equilátero acutángulo válido</b>	(5, 5, 5)	Triángulo equilátero acutángulo	Equilateral Triangle Acute Triangle	Pasa
<b>A ≥ B + C</b>	(3, 2, 1)	No es un triángulo	Is not a Triangle	Pasa
<b>B ≥ A + C</b>	(2, 3, 1)	No es un triángulo	Is not a Triangle	Pasa
<b>C ≥ A + B</b>	(1, 2, 3)	No es un triángulo	Is not a Triangle	Pasa

Tabla 8 - Resultados de la ejecución de la suite de cobertura de condiciones para el programa B

Tras estas pruebas ya se puede determinar que el fallo se produce cuando hay que identificar un triángulo como rectángulo u obtusángulo y el lado mayor no se introduce en último lugar.

#### Cobertura funcional

La identificación de los distintos tipos de triángulo ya se ha comprobado de forma muy exhaustiva en el apartado anterior, por lo tanto, en este apartado el objetivo es localizar defectos con pruebas específicas para los valores límite y las nuevas variables tipo *float* introducidas en la versión B. Esto se puede hacer porque el proceso de pruebas ha sido muy sistemático y detallado a la vez que basado en el diseño de clases de equivalencia, valores límite y cobertura máxima.

Los casos de prueba con los que se van a llevar a cabo los test son los mostrados en la Tabla 2, pero únicamente los que se indica que aplican a la versión B.

A continuación, en la Figura 30 podemos ver la salida obtenida con esta suite de pruebas y en la Tabla 9 la comparativa con la salida esperada y el resultado de cada test.





Máximo número de decimales. Límite válido. 6 cifras decimales.	30 25 5.000001	Triángulo escaleno acutángulo	Is not a triangle	<b>Falla</b>
Máximo número de decimales. Límite no válido. 7 cifras decimales.	30 25 5.0000009	Error: Valores por encima del límite	Is not a triangle	<b>Falla</b>
Límite. Escaleno casi rectángulo (obtusángulo)	(3, 4, 5.000001)	Triángulo escaleno obtusángulo	Scalene triangle Obtuse Triangle	Pasa
Límite. Escaleno casi rectángulo (acutángulo)	(3, 4, 4.999999)	Triángulo escaleno acutángulo	Scalene triangle Acute Triangle	Pasa

Tabla 9 - Resultados de la ejecución de la suite de cobertura funcional para el programa B

Con los resultados de estos test encontramos que los límites máximos para variables tipo *float* no coinciden con los definidos en las variables de macro de C. Incluso con valores dentro del rango teórico los que almacena en las variables no coinciden con los valores introducidos. El valor límite para una variable *float* es mucho más bajo del esperado ya que incluso en los límites establecidos para operaciones internas (en las que se elevan las variables al cuadrado) el valor inicial aparece distorsionado como se puede ver en el Gráfico 1.

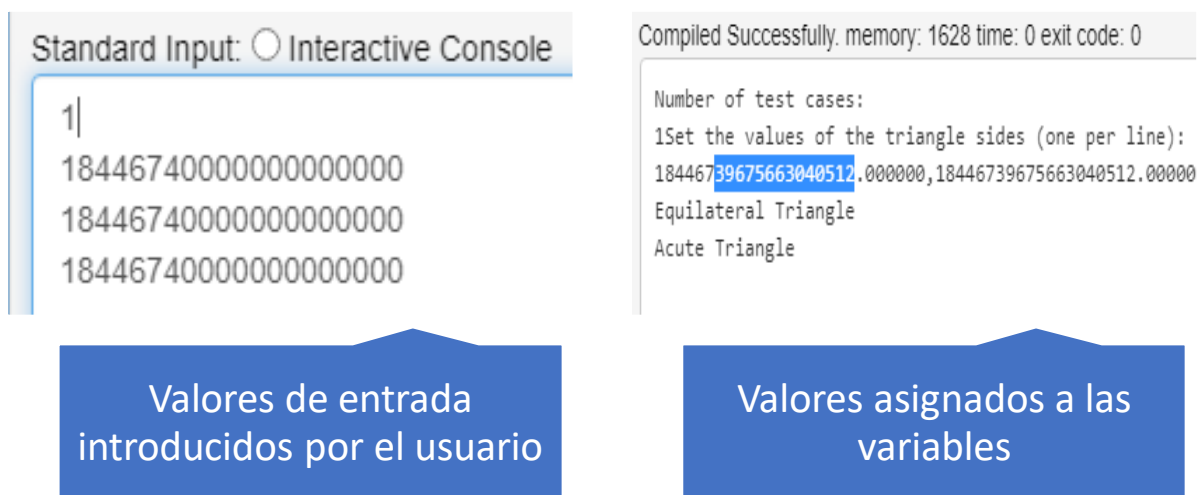


Gráfico 1 - Comparativa entre los valores introducidos y los valores asignados a las variables

Este fallo es muy grave ya que puede llevar a mostrar resultados erróneos, como se ve en el Gráfico 2. En esta aplicación tan trivial, la gravedad podría considerarse relativa, pero para aplicaciones críticas que requieran gran precisión (por ejemplo, en el sector aeroespacial) no hay duda de que la gravedad y prioridad para resolver este defecto sería máxima.



Como no parece depender del entorno, continuamos con las pruebas en la herramienta OnlineGDB. Para acotar el valor máximo se llevan a cabo distintas pruebas, primero introduciendo valores desde  $10^{18}$  hasta  $10^7$  (Figura 32) parece que el límite está en torno a  $10^9$ . En la Figura 33 se intenta acotar algo más. Tras esta primera aproximación, sí que se puede afirmar que no es fiable introducir valores superiores a 99.999.000. No se desarrollan más pruebas de este tipo ya que no entra dentro de los objetivos fijados para el trabajo.

```
Compiled Successfully. memory: 1516 time: 0 exit code: 0

Number of test cases:
4Set the values of the triangle sides (one per line):
9999999980506447872.000000,999999984306749440.000000,99999998430674944.000000:
Is not a Triangle
Set the values of the triangle sides (one per line):
10000000272564224.000000,999999986991104.000000,100000000376832.000000:
Is not a Triangle
Set the values of the triangle sides (one per line):
9999999827968.000000,999999995904.000000,99999997952.000000:
Is not a Triangle
Set the values of the triangle sides (one per line):
100000000000.000000,10000000000.000000,1000000000.000000:
Is not a Triangle
```

Figura 32 – Pruebas para delimitar el límite máximo de float en C. I

```
Compiled Successfully. memory: 1576 time: 0.35 exit code: 0

Number of test cases:
3Set the values of the triangle sides (one per line):
999999872.000000,999998976.000000,999990016.000000:
Scalene triangle
Acute Triangle
Set the values of the triangle sides (one per line):
99999992.000000,99999904.000000,99999000.000000:
Scalene triangle
Acute Triangle
Set the values of the triangle sides (one per line):
99999000.000000,9999900.000000,9999990.000000:
Scalene triangle
Acute Triangle
```

Figura 33 - Pruebas para delimitar el límite máximo de float en C. II

En el caso de las cifras decimales significativas tampoco es válido el valor 6 de la constante de macro. Las pruebas realizadas en la Figura 34 indican que no sería fiable introducir más de 5 cifras decimales.

```
Compiled Successfully. memory: 1624 time: 0 exit code: 0

Number of test cases:
4Set the values of the triangle sides (one per line):
30.000000,25.000000,5.000001:
Is not a Triangle
Set the values of the triangle sides (one per line):
30.000000,25.000000,5.000010:
Scalene triangle
Acute Triangle
Set the values of the triangle sides (one per line):
30.000000,25.000000,5.000100:
Scalene triangle
Acute Triangle
Set the values of the triangle sides (one per line):
30.000000,25.000000,5.001000:
Scalene triangle
Acute Triangle
```

Figura 34 - Pruebas para delimitar el número máximo de decimales en C

### 2.6.3. Resumen de errores encontrados

Tras el proceso de pruebas se han encontrado 3 errores principales:

- Un error de formato en los datos de entrada provoca la finalización de todas las iteraciones pendientes del bucle *for* con los valores de memoria en ese momento.
- Identificación incorrecta del tipo de triángulo según los ángulos siempre que el lado mayor no se introduce en último lugar.
- No se manejan los valores máximos que pueden tomar los datos de entrada. Además, en el caso de las variables tipo *float*, estos valores máximos no coinciden con los teóricos y son bastante menores.

### 2.7. Implementación de mejoras. Versión C del programa

Tras la ejecución de los *tests*, el siguiente punto en el proceso de pruebas es informar al equipo de desarrollo de los resultados y las mejoras que es necesario introducir. En este caso se ha desarrollado una versión C del programa en la que se soluciona el problema de identificación incorrecta del tipo de triángulo según sus ángulos. El código completo se puede encontrar en el Anexo I Código de los programas

Para testear la nueva versión del programa no es necesario repetir todas las suites de pruebas. La buena documentación llevada a cabo durante el proceso de pruebas permite saber qué test están relacionados con qué funcionalidad del programa (ver Tabla 1- Diseño de pruebas de caja blanca y Tabla 2- Suites de prueba según criterios de cobertura). Por lo tanto, para verificar las mejoras introducidas en esta versión se utilizan los casos de prueba para los distintos tipos de triángulos (incluidas las permutaciones de lados) de la Tabla 7- Casos de prueba añadidos a la suite para la cobertura de condiciones.

Las salidas obtenidas con la ejecución de los test se pueden ver en la Figura 35. En la comparación con los resultados esperados de la Tabla 10 comprobamos que todas las pruebas son correctas por lo que el error se considera que está solucionado.

```
Compiled Successfully. memory: 1844 time: 0.01 exit code: 0
```

```
Number of test cases:
28Set the values of the triangle sides (one per line):
3.000000,4.000000,5.000000: Scalene triangle
Right Triangle
Set the values of the triangle sides (one per line):
3.000000,5.000000,4.000000: Scalene triangle
Right Triangle
Set the values of the triangle sides (one per line):
5.000000,3.000000,4.000000: Scalene triangle
Right Triangle
Set the values of the triangle sides (one per line):
5.000000,4.000000,3.000000: Scalene triangle
Right Triangle
Set the values of the triangle sides (one per line):
4.000000,3.000000,5.000000: Scalene triangle
Right Triangle
Set the values of the triangle sides (one per line):
4.000000,5.000000,3.000000: Scalene triangle
Right Triangle
Set the values of the triangle sides (one per line):
4.000000,5.000000,6.000000: Scalene triangle
Acute Triangle
Set the values of the triangle sides (one per line):
4.000000,6.000000,5.000000: Scalene triangle
Acute Triangle
Set the values of the triangle sides (one per line):
5.000000,4.000000,6.000000: Scalene triangle
Acute Triangle
Set the values of the triangle sides (one per line):
5.000000,6.000000,4.000000: Scalene triangle
Acute Triangle
Set the values of the triangle sides (one per line):
6.000000,4.000000,5.000000: Scalene triangle
Acute Triangle
Set the values of the triangle sides (one per line):
6.000000,5.000000,4.000000: Scalene triangle
Acute Triangle
```

```

Set the values of the triangle sides (one per line):
5.000000,6.000000,10.000000: Scalene triangle
Obtuse Triangle
Set the values of the triangle sides (one per line):
5.000000,10.000000,6.000000: Scalene triangle
Obtuse Triangle
Set the values of the triangle sides (one per line):
6.000000,5.000000,10.000000: Scalene triangle
Obtuse Triangle
Set the values of the triangle sides (one per line):
6.000000,10.000000,5.000000: Scalene triangle
Obtuse Triangle
Set the values of the triangle sides (one per line):
10.000000,5.000000,6.000000: Scalene triangle
Obtuse Triangle
Set the values of the triangle sides (one per line):
10.000000,6.000000,5.000000: Scalene triangle
Obtuse Triangle
Set the values of the triangle sides (one per line):
2.000000,9.000000,9.000000: Isosceles Triangle
Acute Triangle
Set the values of the triangle sides (one per line):
9.000000,2.000000,9.000000: Isosceles Triangle
Acute Triangle
Set the values of the triangle sides (one per line):
9.000000,9.000000,2.000000: Isosceles Triangle
Acute Triangle
Set the values of the triangle sides (one per line):
5.000000,5.000000,9.000000: Isosceles Triangle
Obtuse Triangle
Set the values of the triangle sides (one per line):
9.000000,5.000000,5.000000: Isosceles Triangle
Obtuse Triangle
Set the values of the triangle sides (one per line):
5.000000,9.000000,5.000000: Isosceles Triangle
Obtuse Triangle

Set the values of the triangle sides (one per line):
5.000000,5.000000,5.000000: Equilateral Triangle
Acute Triangle

```

Figura 35 - Salida del programa C para la suite de cobertura de condiciones

Casos de prueba	Valor de entrada	Salida esperada	Salida obtenida	Resultado del test
N entero	28	Introduce valores para los lados	Set the values of the triangle sides	Pasa
Escaleno rectángulo válido	(3, 4, 5)	Triángulo escaleno rectángulo	Scalene triangle Right Triangle	Pasa

	(3, 5, 4)	Triángulo escaleno rectángulo	Scalene triangle Right Triangle	Pasa
	(5, 3, 4)	Triángulo escaleno rectángulo	Scalene triangle Right Triangle	Pasa
	(5, 4, 3)	Triángulo escaleno rectángulo	Scalene triangle Right Triangle	Pasa
	(4, 3, 5)	Triángulo escaleno rectángulo	Scalene triangle Right Triangle	Pasa
	(4, 5, 3)	Triángulo escaleno rectángulo	Scalene triangle Right Triangle	Pasa
<b>Escaleno acutángulo válido</b>	(4, 5, 6)	Triangulo escaleno acutángulo	Scalene triangle Acute Triangle	Pasa
	(4, 6, 5)	Triangulo escaleno acutángulo	Scalene triangle Acute Triangle	Pasa
	(5, 4, 6)	Triangulo escaleno acutángulo	Scalene triangle Acute Triangle	Pasa
	(5, 6, 4)	Triangulo escaleno acutángulo	Scalene triangle Acute Triangle	Pasa
	(6, 4, 5)	Triangulo escaleno acutángulo	Scalene triangle Acute Triangle	Pasa
	(6, 5, 4)	Triangulo escaleno acutángulo	Scalene triangle Acute Triangle	Pasa
<b>Escaleno obtusángulo válido</b>	(5, 6, 10)	Triangulo escaleno obtusángulo	Scalene triangle Obtuse Triangle	Pasa
	(5, 10, 6)	Triangulo escaleno obtusángulo	Scalene triangle Obtuse Triangle	Pasa
	(6, 5, 10)	Triangulo escaleno obtusángulo	Scalene triangle Obtuse Triangle	Pasa
	(6, 10, 5)	Triangulo escaleno obtusángulo	Scalene triangle Obtuse Triangle	Pasa
	(10, 5, 6)	Triangulo escaleno obtusángulo	Scalene triangle Obtuse Triangle	Pasa
	(10, 6, 5)	Triangulo escaleno obtusángulo	Scalene triangle Obtuse Triangle	Pasa
<b>Isósceles acutángulo válido</b>	(2, 9, 9)	Triángulo isósceles acutángulo	Isosceles Triangle Acute Triangle	Pasa
	(9, 2, 9)	Triángulo isósceles acutángulo	Isosceles Triangle Acute Triangle	Pasa
	(9, 9, 2)	Triángulo isósceles acutángulo	Isosceles Triangle Acute Triangle	Pasa
<b>Isósceles obtusángulo válido</b>	(5, 5, 9)	Triángulo isósceles obtusángulo	Isosceles Triangle Obtuse Triangle	Pasa
	(9, 5, 5)	Triángulo isósceles obtusángulo	Isosceles Triangle Obtuse Triangle	Pasa
	(5, 9, 5)	Triángulo isósceles obtusángulo	Isosceles Triangle Obtuse Triangle	Pasa
<b>Equilátero acutángulo válido</b>	(5, 5, 5)	Triángulo equilátero acutángulo	Equilateral Triangle Acute Triangle	Pasa





### 3. Análisis de datos de defectos e incidencias de proyectos reales

Tras el diseño y realización de las pruebas en un proyecto se dispone de múltiples datos de los cuales se puede obtener información relativa al funcionamiento del equipo y su productividad y al desarrollo del proyecto en cuestión. Para poder analizar en profundidad datos de *testing*, en esta parte del trabajo se deja de lado los ejemplos sencillos y se procede a trabajar con datos reales de proyectos de desarrollo.

En concreto, se analizan los datos de proyectos reales que se recogieron como parte del proyecto europeo Iceberg donde participó la Universidad de Alcalá desde 2013 a 2017:

- 698 defectos del sistema ACME registrados en Jira entre 2012 y 2014 en una empresa italiana.
- 1898 defectos del sistema FOX registrados en Jira entre 2015 y 2016 en una empresa italiana.
- Una matriz de regresión para 967 casos de prueba y 110 versiones o *release* del programa FOX de la misma empresa.

#### 3.1. Conceptos teóricos

El seguimiento de defectos (*defect tracking*) en ingeniería del software es el proceso de rastrear los defectos de un producto software desde su origen hasta el cierre con el objetivo de hacer nuevas versiones en las que no estén presentes. En los casos en que el proyecto, el equipo o el número de defectos son muy grandes se utiliza un sistema de seguimiento de defectos como Jira para facilitar las tareas de gestión y ayudar a asegurar la calidad del software.

Existen multitud de publicaciones sobre tipologías y métodos para hacer el seguimiento de defecto. Jonathan Strate y Phillip Laplante publicaron en 2013 un estudio [7] donde analizan el estado del arte de la investigación en reporte de defectos de software, analizando 104 referencias entre artículos, congresos, libros y tesis. De sus conclusiones cabe destacar la importancia de utilizar herramientas específicas y métricas para conseguir una mejor productividad en los equipos de calidad y pruebas.

Una medida importante en para determinar la confiabilidad en un sistema es la tasa de fallos. Al inicio de la fase de pase de pruebas los *testers* detectan un mayor número de defectos, sobre todo en equipos con experiencia. A medida que las pruebas llegan a la fase final el número de nuevos defectos encontrados tiende a estabilizarse y solo permanecen en el software los defectos que son más difíciles de encontrar. Este comportamiento se refleja en la llamada curva SRGM (*software reliability growth model*) [8]. Con la elaboración de estos modelos se pretende poder predecir incidencias y fallos futuros en el sistema y cuándo se alcanza un nivel suficiente de confiabilidad como para finalizar la fase de pruebas.

Sin embargo, conseguir modelos predictivos que se ajusten a la realidad no es una tarea sencilla. El proceso es complejo, requiere unificar datos de múltiples fuentes y en muchas empresas no hay concienciación sobre este tema y su importancia. Al final el modelo conseguido solo se ajusta a una empresa, incluso proyecto concreto. Norman Fenton [9] crítica la efectividad de estos modelos y establece como principal causa la dificultad para identificar la relación entre defectos y fallos.

## 3.2. Análisis de datos

### 3.2.1. Datos de defectos del sistema ACME

Este conjunto de datos cuenta con un total de 698 registros de defectos introducidos en Jira entre 2012 y 2014. De las 698 incidencias abiertas se analizan las 692 que ya se han solucionado y cerrado en el sistema. En estos registros se encuentran datos sobre gravedad, prioridad, fecha de creación y fecha de cierre, habituales en las herramientas de *defect-tracking*, y que son los que se van a analizar para extraer información en este trabajo.

Los valores de los indicadores de gravedad y prioridad se determinan en función de criterios internos de la empresa e incluso podrían diferir entre proyectos y/o departamentos. En el análisis hecho en este trabajo no se dispone de información sobre cómo se han categorizado los distintos defectos, pero sí se pueden intuir algunos criterios. Para identificar la gravedad, al registro se le asigna una de las cuatro categorías disponibles: leve, tolerable, bloqueante o crítico o bloqueante; de menor a mayor gravedad. El tipo de incidencia que encaja en cada categoría podría ser el siguiente:

- Leves: no perjudica la realización de ninguna función. Son del tipo fallos de escritura en alguna pantalla.
- Tolerables/Medias: Las funciones se pueden llevar a cabo, pero los usuarios experimentan ciertas limitaciones.
- Críticas: hay funciones que no se pueden realizar.
- Bloqueantes: parada del sistema.

En el caso de la prioridad, las categorías que se pueden asignar son baja, media, alta y muy alta. Dadas estas categorías, y sin conocer los criterios de clasificación exhaustivamente, sorprende que haya incidencias categorizadas como bloqueantes de prioridad baja, por ejemplo. El caso contrario, incidencias leves de prioridad muy alta, podría darse, por ejemplo, si hubiera una falta de ortografía muy llamativa en la pantalla principal.

Este conjunto de datos presenta como limitación principal que no se conoce el recurso al que se ha asignado la resolución de la incidencia. Por lo tanto, el análisis se enfoca en la información que es posible extraer de tres campos; gravedad, prioridad y fecha de inicio y fin, para saber cuál es la distribución del número de incidencias en función de la gravedad y la prioridad y así como el tiempo necesario para resolver cada incidencia según la categoría asignada.

Según la distribución del tipo de incidencia del Gráfico 3, vemos que la mayoría de las incidencias se clasifican en las categorías de menor gravedad. En cuanto a la prioridad, las asignaciones son en su mayoría muy altas o bajas.

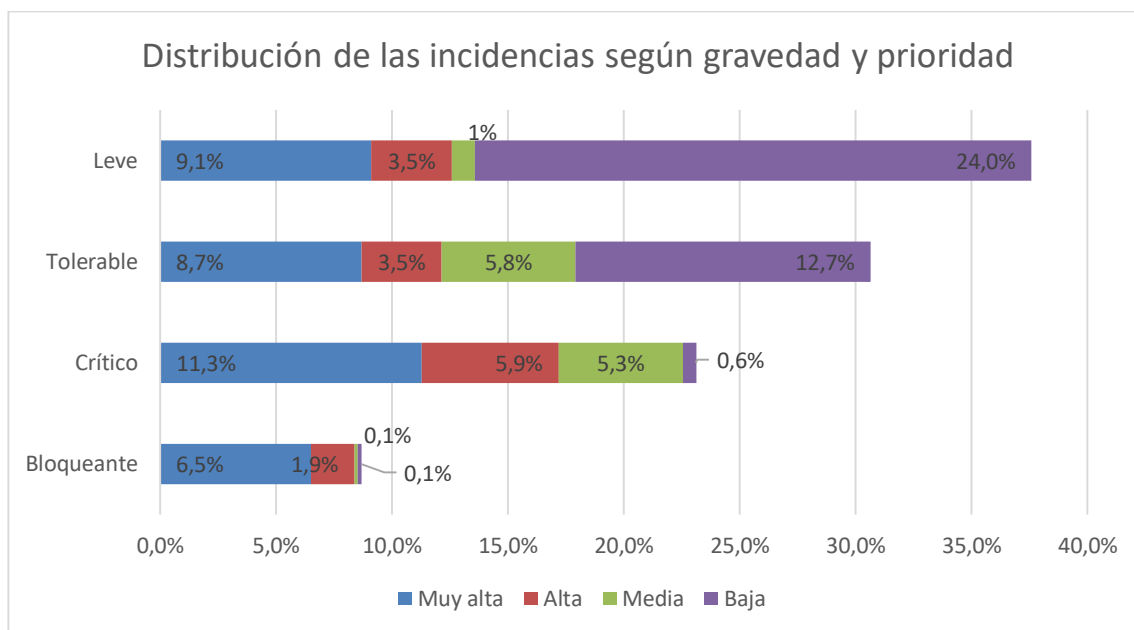


Gráfico 3 - Distribución de las incidencias según gravedad y prioridad en el sistema ACME

Con respecto al tiempo que se tarda en cerrar una incidencia, como se puede ver en el Gráfico 4, con los datos de la Tabla 11, las incidencias con gravedad bloqueante son las que se resuelven más rápido, lo cual tiene sentido ya que es la categoría de mayor gravedad.

Estos valores de duración se han calculado a partir de los días laborables entre las fechas de inicio y finalización de la incidencia, sin embargo, no se puede asumir que todo ese tiempo realmente se haya destinado a la resolución de una incidencia en concreto. Esta es una de las grandes limitaciones que existen a la hora de construir un modelo predictivo y por lo tanto no se pueden sacar conclusiones determinantes. Los datos sí que reflejan que las incidencias bloqueantes son las que tienen un tiempo de resolución más bajo.

	Muy alta	Alta	Media	Baja	Media
<i>Bloqueante</i>	34,00	107,85	26,00	7,00	49,42
<i>Crítico</i>	55,54	76,32	142,08	10,25	79,74
<i>Tolerable</i>	63,08	153,13	30,53	78,77	73,65
<i>Leve</i>	51,17	141,25	86,43	61,33	66,92

*Media de todas las incidencias es de 70,43 días laborables*

Tabla 11 - Tiempo medio de resolución de incidencias según gravedad y prioridad en el sistema ACME

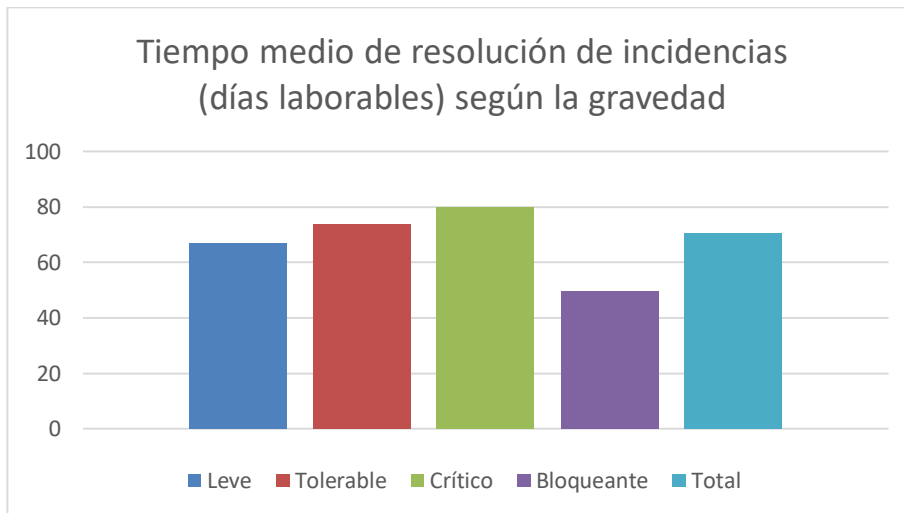


Gráfico 4 - Tiempo medio de resolución de incidencias según gravedad en el sistema ACME

En el Gráfico 5 se muestra la duración media segmentada también por prioridad.

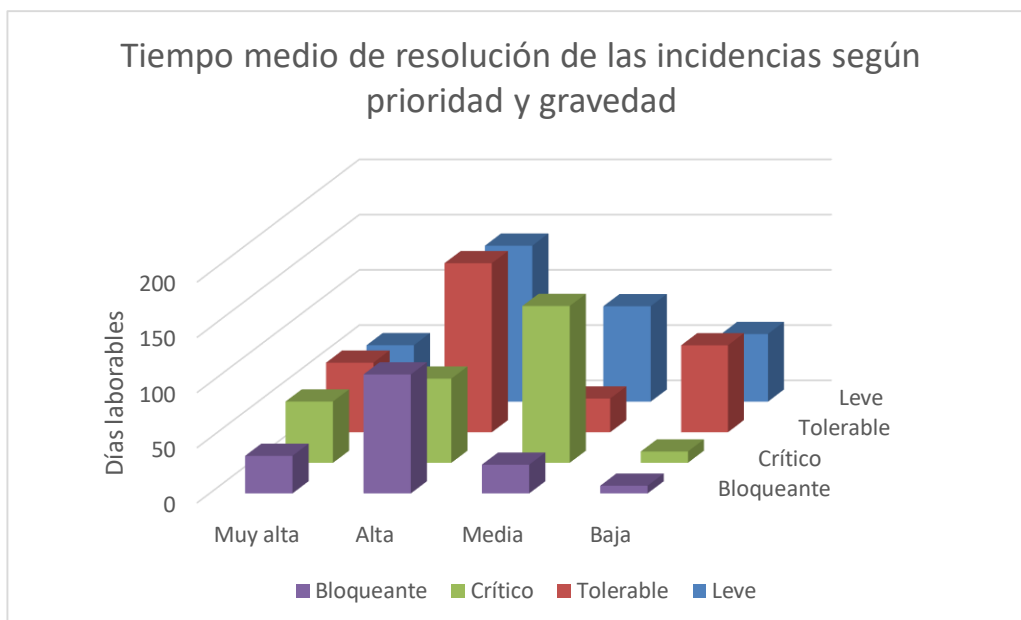


Gráfico 5 - Tiempo medio de resolución de incidencias según gravedad y prioridad en el sistema ACME

### 3.2.2. Datos de defectos del sistema FOX

Este conjunto de datos cuenta con registros de 1898 defectos del sistema FOX registrados en Jira entre 2015 y 2016 en una empresa italiana. El análisis se va a realizar sobre las 1760 incidencias cerradas. Para cada uno de los registros se tienen datos sobre la gravedad de la incidencia, la persona a la que se le asigna su resolución y fechas de creación y resolución. La gravedad se clasifica según cuatro categorías, *low*, *medium*, *major* y *show stopper*.

En este caso, se va a estudiar la distribución de las incidencias según su tipología. La limitación viene dada por disponer solamente del indicador gravedad para categorizar las incidencias. A partir de los datos de las personas del equipo encargadas de resolver incidencias se va a intentar llegar a indicadores de productividad.

Como se muestra en el Gráfico 6 la mayor parte de las incidencias son de tipo *major*.

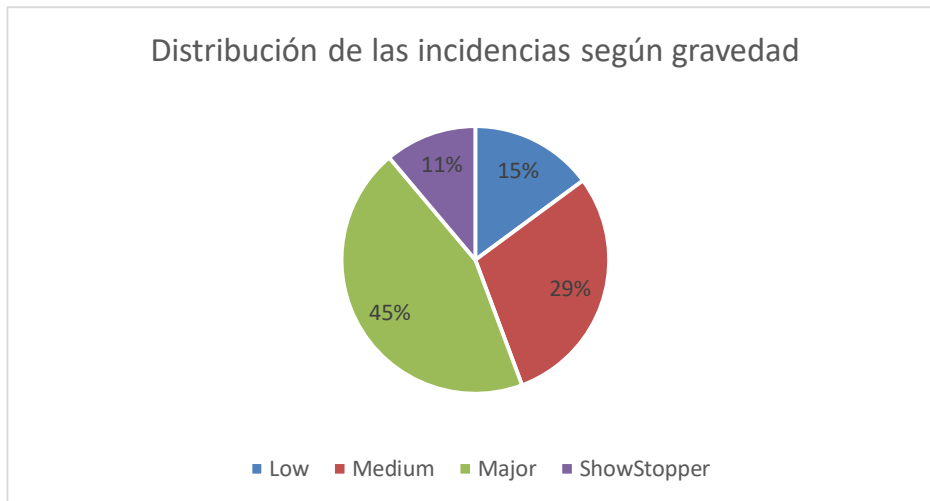


Gráfico 6 - Distribución de las incidencias según gravedad en el sistema FOX

Con los datos relativos a las fechas de inicio y cierre de la incidencia se calcula el valor medio del tiempo de resolución total y segmentado por gravedad (ver Tabla 12). En este caso también se comprueba que aquellas incidencias categorizadas como *showStopper* son las que se resuelven en menos tiempo. En general, aquellos fallos más graves son los que se resuelven en menos tiempo (ver Gráfico 7).

<i>Gravedad</i>	<i>Tiempo medio de resolución</i>
<i>ShowStopper</i>	6,09
<i>Major</i>	9,77
<i>Medium</i>	13,98
<i>Low</i>	11,07

*Media de todas las incidencias es de 10,79 días laborables*

Tabla 12 - Tiempo medio de resolución de incidencias según gravedad en el sistema FOX

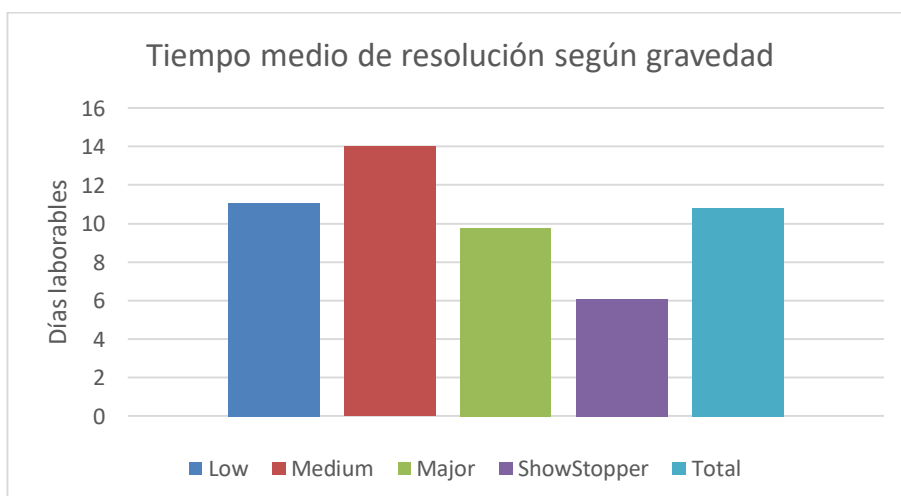


Gráfico 7 - Tiempo medio de resolución de incidencias según gravedad en el sistema FOX

En el Gráfico 8 podemos ver cómo se reparten las asignaciones entre los distintos miembros del equipo. Hay miembros que, tal vez por su experiencia, reciben muchas más asignaciones que el resto. Llama la atención que el 97% de las asignaciones se concentra entre el 52% de los miembros del equipo.

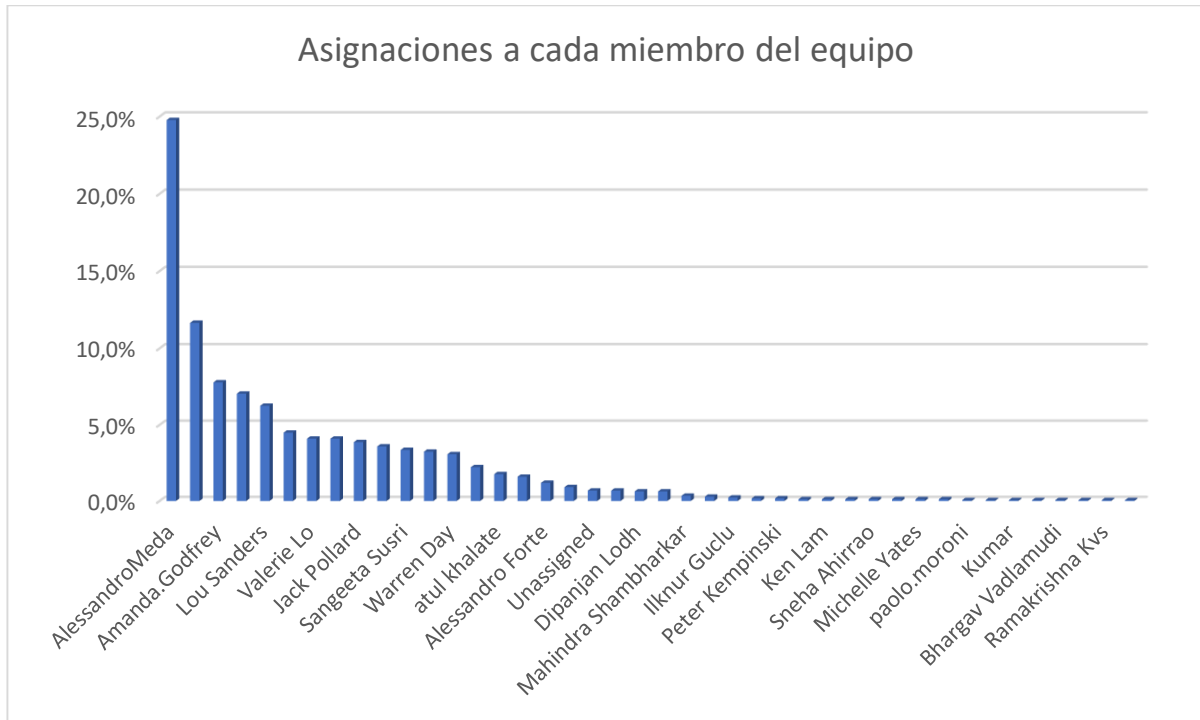


Gráfico 8 - Porcentaje de las asignaciones que recibe cada miembro del equipo en el sistema FOX

En el Gráfico 9 se muestran los distintos tipos de asignaciones que recibe cada miembro del equipo de pruebas. Los de la parte izquierda del gráfico reciben asignaciones de todas las categorías de gravedad. La parte derecha del gráfico muestra a las personas que se pueden considerar más especialistas. En este caso, coinciden, en su mayoría, con el 48% del equipo al que se asigna el 3% de las incidencias.

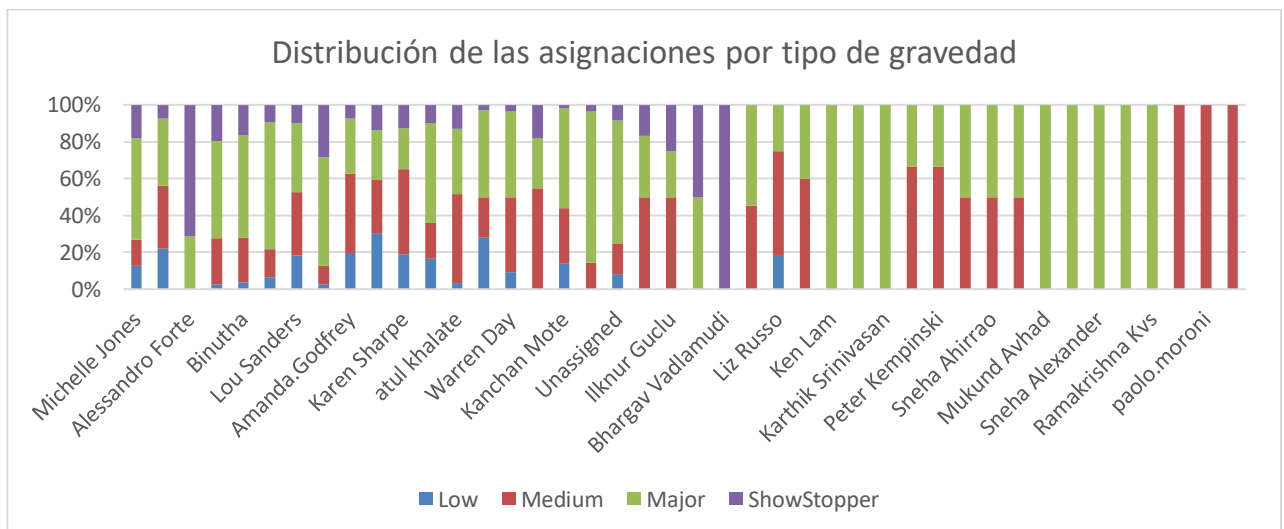


Gráfico 9 - Distribución de las asignaciones de cada miembro según el tipo de gravedad en el sistema FOX

Como cabe esperar tras ver las diferencias en la distribución de asignaciones, los valores de tiempo de resolución medio (medidos en días laborables) también son muy diversos, como se puede ver en el Gráfico 10.

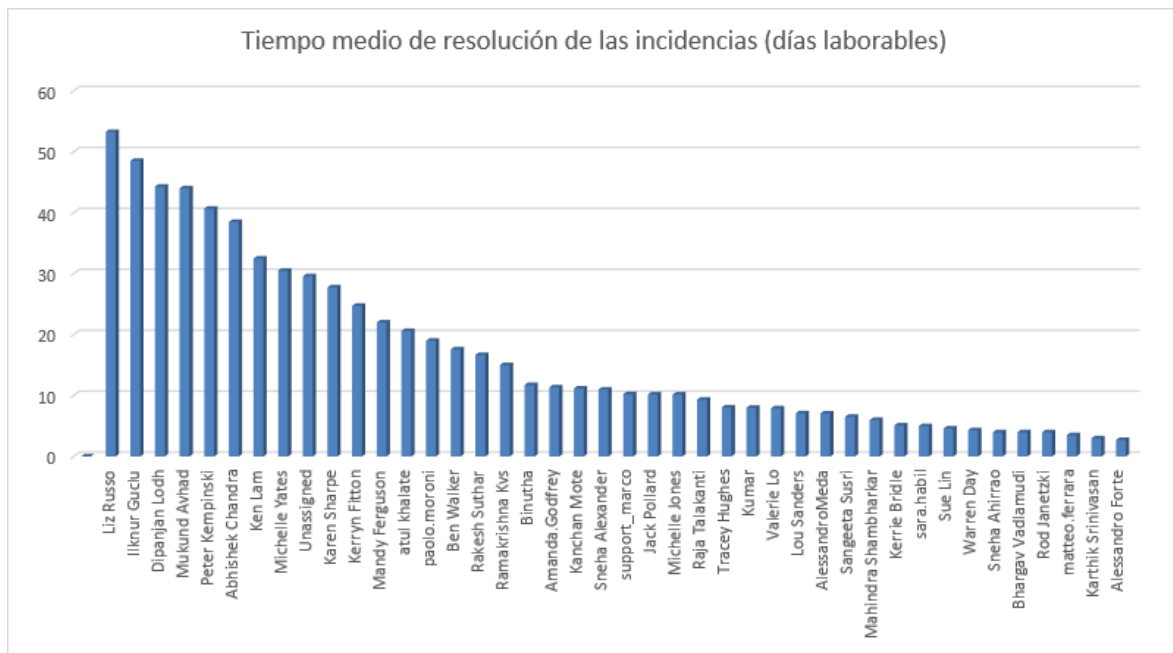


Gráfico 10 - Tiempo medio de resolución de las incidencias por cada miembro del equipo en el sistema FOX

Es interesante representar el número de incidencias asignadas frente al tiempo medio que se tardan en resolver para tener una idea de los miembros del equipo con más productividad. Esta representación también permite conocer si los datos siguen alguna tendencia. En el Gráfico 11 se representan todos los datos y la conclusión, en principio, sería que no hay relación entre las variables. Sin embargo, se aprecia que hay valores atípicos que están distorsionando la tendencia general.

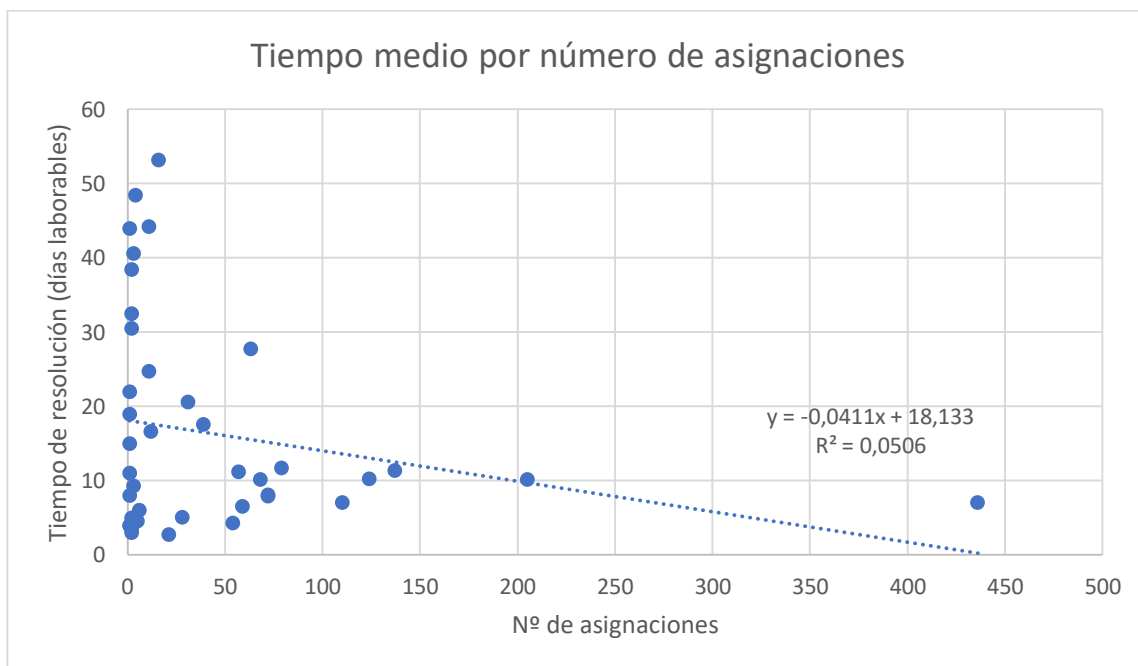


Gráfico 11 - Tiempo medio de resolución según el número de asignaciones de cada persona en el sistema FOX



Para determinar cuáles son estos valores atípicos. Se ha utilizado el test de Turkey, es decir, considerar como valor atípico a aquellos que se encuentren a una distancia superior a 1,5 veces el rango intercuartílico. Estos valores son fácilmente identificables en los diagramas de cajas que se muestran en el Gráfico 12 para el tiempo medio de resolución y en el Gráfico 13 para el número de asignaciones.

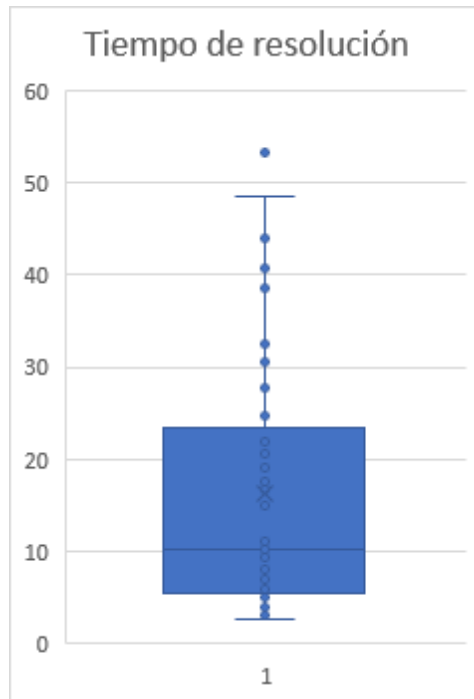


Gráfico 12 - Diagrama de cajas para la variable Tiempo medio de resolución

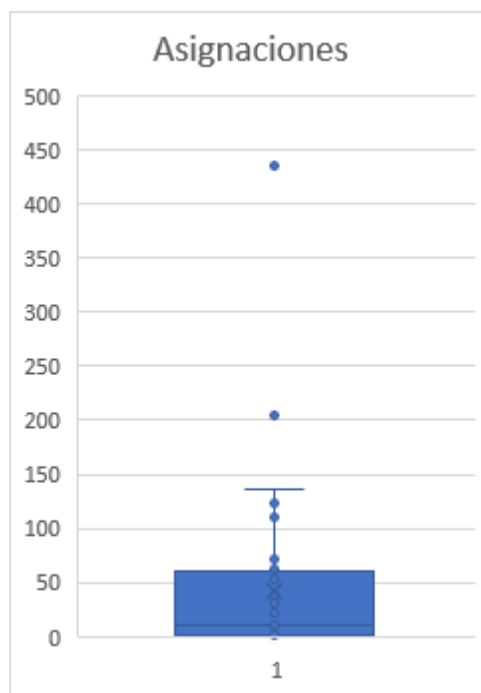


Gráfico 13 - Diagrama de cajas para la variable Número de asignaciones

Además, dadas las características tan particulares de distribución de las incidencias entre los miembros del equipo (un 97% de las incidencias las resuelve un 52% de los miembros del equipo) se toma la decisión de descartar los datos de aquellas personas que han resuelto menos de 10 incidencias por no resultar relevantes y realizar el análisis solo con el 97% de las incidencias restantes.

Con este nuevo conjunto de valores se representa el número de asignaciones frente al tiempo medio de resolución y se analiza de nuevo la tendencia en el Gráfico 14. En este caso los valores siguen una tendencia logarítmica con un coeficiente de determinación ( $R^2$ ) de 0,35. Este valor no lo suficientemente alto y por lo tanto habría que profundizar en el análisis, pero sí que sugiere una relación entre ambas variables. A medida que aumenta el número de asignaciones que recibe una persona, y por lo tanto aumenta su experiencia, el tiempo medio que tarda en cerrar cada incidencia disminuye.

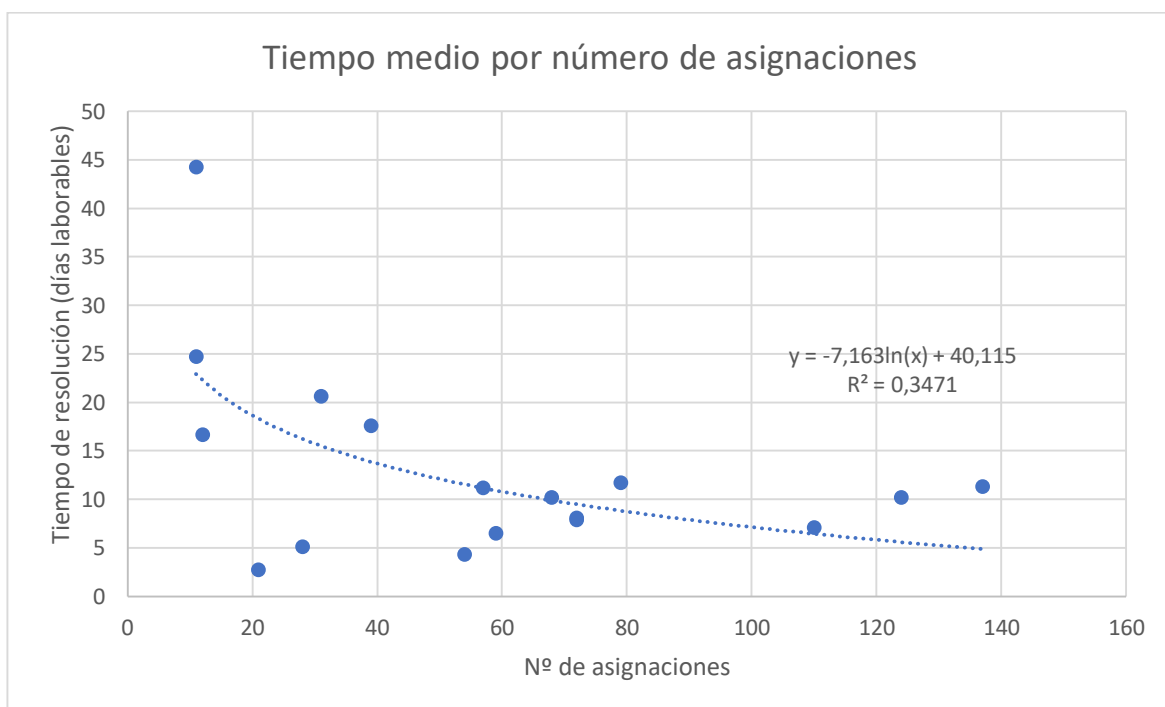


Gráfico 14 - Tiempo medio de resolución según el número de asignaciones de cada persona en el sistema FOX (sin valores atípicos)

### 3.2.3. Datos de la matriz de regresión del sistema FOX

Para el sistema FOX también se disponen de una matriz de regresión con datos de las pruebas realizadas para cada versión del sistema. Se han recogido datos de 967 pruebas distintas (91 de integración y 876 funcionales) que se han realizado a 199 funciones de las 110 versiones distintas del sistema. En total hay registros de 1968 ejecuciones de pruebas.

El Gráfico 15 muestra el número de fallos identificados durante las pruebas a lo largo de las versiones. Esta forma en “S”, es la forma típica de los modelos SRGM. Al inicio de la fase de pruebas o del lanzamiento de una versión nueva, por ejemplo, paso de versión 1.9 a 2.0, la tasa de descubrimiento de fallos es mayor. Según avanza la realización de las pruebas el número de fallos que se encuentran tiende a estabilizarse.

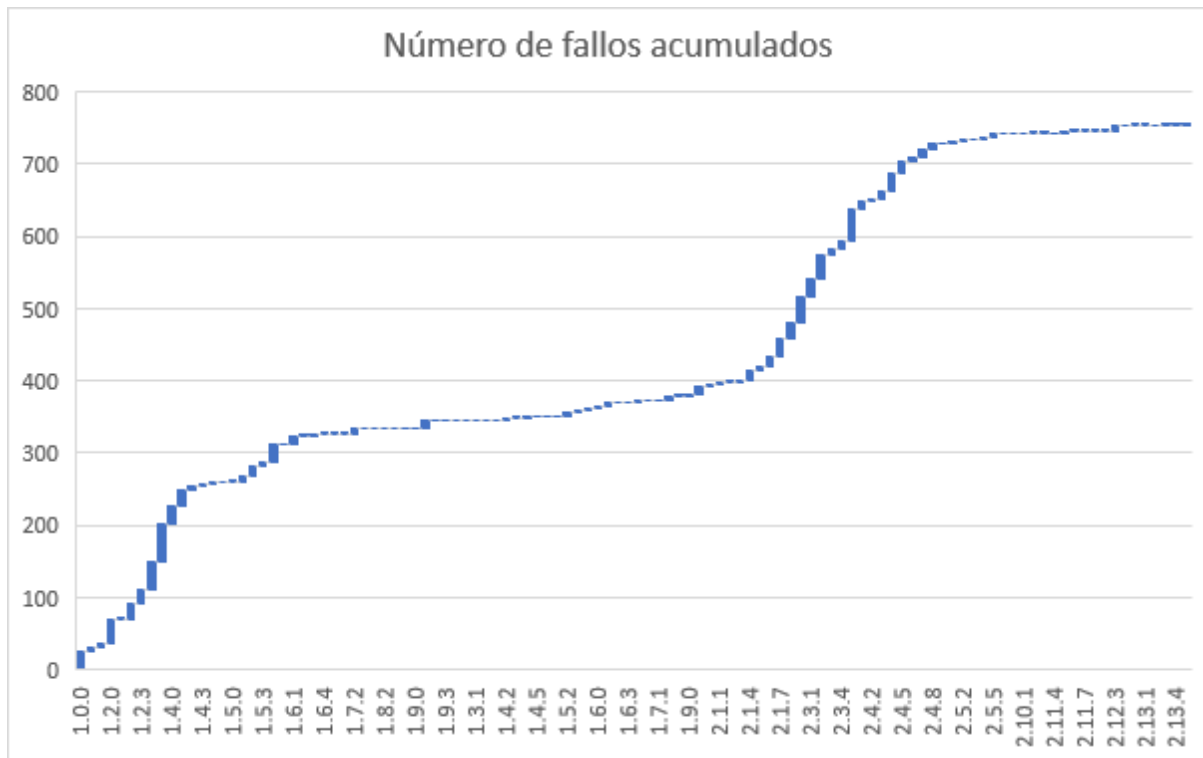


Gráfico 15 - Número de fallos acumulados en las distintas versiones del sistema FOX

### 3.3.Comparativa de proyectos

Como se ha visto hasta ahora los datos disponibles de cada proyecto son diferentes pese a pertenecer a la misma empresa. Con los datos comunes se construye la Tabla 13.

	<i>Sistema ACME</i>	<i>Sistema FOX</i>
<i>Número total de incidencias</i>	692	1760
<i>Bloqueantes</i>	9%	11%
<i>Críticas</i>	23%	45%
<i>Medias</i>	31%	29%
<i>Leves</i>	38%	15%
<i>Tiempo medio de resolución (días laborables)</i>	70,43	10,79
<i>Bloqueantes</i>	49,42	6,09
<i>Críticas</i>	79,74	9,77
<i>Medias</i>	73,65	13,98
<i>Leves</i>	66,92	11,07

Tabla 13 - Datos comunes de los proyectos ACME y FOX

La información disponible es muy limitada. Para un análisis en profundidad haría falta de más información sobre la especificación de cada sistema, el equipo de desarrollo, el equipo de pruebas o los criterios para clasificar las incidencias entre otras fuentes. Con la información disponible sí que podemos concluir que los fallos de tipo bloqueante son los que menos ocurren y también los que se resuelven más rápido, precisamente por su gravedad.

## 4. Conclusiones y trabajos futuros

### 4.1. Conclusiones

Como conclusión principal para el trabajo realizado quiero destacar la importancia de una buena documentación, completa y exhaustiva, del diseño de las pruebas y un proceso sistematizado a la hora de ejecutarlas. Esto ha permitido agilizar la fase de pruebas, poder reutilizar trabajo anterior y además garantizar que los resultados son fiables. Todo este esfuerzo adicional ha resultado primordial para garantizar la calidad del trabajo presentado.

En la fase de pruebas, es también muy importante medir la cobertura que se está consiguiendo para así poder determinar si las pruebas realizadas son suficientes y, por lo tanto, permiten asegurar la calidad del software o no. En el caso estudiado los errores latentes del software no se han manifestado hasta que se ha cumplido el criterio de cobertura más estricto de los planteados.

Me ha resultado interesante y curioso descubrir durante mi trabajo que los valores máximos para los distintos tipos de datos no son tan fijos como podría parecer en un primer momento. En nuestro caso hemos encontrado que difieren incluso de un lenguaje de programación a otro. Este asunto puede parecer trivial en el ejemplo del triángulo que se ha analizado, pero sin duda es muy importante en sistemas de precisión y procesos críticos como podría ser el software que controla los sistemas de un avión o en sistemas espaciales.

En proyectos software reales se utilizan los sistemas de seguimiento de defectos para gestionar las incidencias y los defectos detectados en cualquier fase desarrollo o de explotación. Estos sistemas pueden generar muchos datos y es frecuente que las organizaciones no los aprovechen para sacar más conclusiones que permitan mejorar sus procesos de trabajo o de gestión. Sin embargo, no siempre sirven para extraer conclusiones y tomar decisiones ajustadas por distintas razones. Sobre todo, hace falta que los datos sean más precisos, por ejemplo, conociendo los criterios para categorizar las incidencias o pidiendo *feedback* al *tester* para saber si la categoría asignada era la correcta o simplemente obteniendo mediciones más ajustadas del tiempo realmente dedicado a resolver cada defecto.

También se pueden integrar otras fuentes de datos. En el caso de la matriz de regresión, se podrían incorporar datos de las herramientas de gestión de la configuración para controlar mejor cada elemento de configuración o componente del sistema y los cambios que se han aplicado sobre cada uno. Esto permitiría también controlar mejor si los elementos con más cambios se relacionan con más defectos, precisamente porque los desarrolladores están corrigiendo defectos existentes.

Cuando se integran los datos de las fuentes disponibles es fácil conseguir llegar a un modelo de predicción de fallos. Sin embargo, este modelo será específico para cada departamento o, incluso, para cada proyecto. No es fácil realizar la extrapolación de modelos generales debido a las diferentes formas de trabajar en cada entorno u organización, las características diferentes de los equipos de trabajo o incluso el tipo de proyecto o los requisitos que deben cumplirse, si son más o menos exigentes.

## 4.2. Trabajos futuros

Como idea para continuar este trabajo se proponen las siguientes líneas de investigación:

- Profundizar en la versión en Java para analizar los límites máximos y de precisión de los diferentes tipos de datos, además de comprobar si el seguimiento con las herramientas es realmente equivalente al realizado con el lenguaje C.
- Ampliar las pruebas a otros lenguajes de programación, incluyendo más tipos de datos, así como nuevos ejemplos de código sencillo.
- Profundizar en el uso y posibilidades que ofrecen las herramientas de gestión gratuitas como Klaros. Estudiar las posibilidades que ofrece la integración con los repositorios Github para incorporar más fuentes a los datos de defectos.
- Incorporar al análisis de datos de defectos e incidencias más datos de otros proyectos de desarrollo reales para estudiar los modelos de predicción, tratando de encontrar modelos de regresión o correlación que puedan explicar mejor el comportamiento de los sistemas en pruebas y en explotación.

## 5. Referencias

- [1] Glenford J. Myers, Tom Badgett, Todd M. Thomas, y Corey Sandler, *The art of software testing / Glenford J. Myers; revised and updated by Tom Badgett and Todd Thomas, with Corey Sandler*, 2nd ed. Hoboken, New Jersey : John Wiley & Sons, Inc, 2004.
- [2] IEEE, *IEEE Std 610. Computer Dictionary*. IEEE, 1990.
- [3] IEEE, *IEEE Std 829. Standard for Software and System Test Documentation*. IEEE, 2008.
- [4] T..J. McCabe, *A Complexity Measure*. IEEE Transactions on software engineering, vol. 2, pp. 228-363, 1976.
- [5] Boris Beizer, *Software testing techniques*. New York : Van Nostrand Reinhold, 1990.
- [6] T.J. McCabe, *IEEE Tutorial: Structured testing*. IEEE Computer Society, 1982.
- [7] Jonathan D. Strate y Phillip A. Laplante, «A Literature Review of Research in Software Defect Reporting», *IEEE Trans. Reliab.*, vol. 62, n.º 2, pp. 444-454, jun. 2013, doi: 10.1109/TR.2013.2259204.
- [8] P.K. Kapur, D.N. Goswami, Amit Bardhan, y Ompal Singh, «Flexible software reliability growth model with testing effort dependent learning process», *Appl. Math. Model.*, vol. 32, n.º 7, pp. 1298-1307, 2008, doi: 10.1016/j.apm.2007.04.002.
- [9] Norman E. Fenton y Martin Neil, «A critique of software defect prediction models», vol. 25, n.º 5, pp. 675-689, oct. 1999, doi: 10.1109/32.815326.

## Anexo I Código de los programas

### Versión A.

```
#include <math.h>
#include <stdio.h>

int checktriangle(int a, int b, int c)
{
    int A, B, C;
    A = a;
    B = b;
    C = c;

    if((C<A+B)&&(B<A+C)&&(A<C+B))
    {
        if(A==B && B==C)
        {
            printf("Equilateral Triangle \n");
        }else if(A==B || B==C || A==C)
        {
            printf("Isosceles Triangle \n");
        }else
        {
            printf("Scalene triangle \n");
        }

        int value = (A*A)+(B*B)-(C*C);

        if(value==0)
        {
            printf("Right Triangle \n");
        }else if(value>0)
        {
            printf("Acute Triangle \n");
        }else
        {
            printf("Obtuse Triangle \n");
        }
    }
    else
    {
        printf("Is not a Triangle \n");
    }

    return 0;
}

int main()
{
    int a, b, c, i, n;
    printf("Number of test cases: \n");
    scanf("%d",&n); printf ("%d", n);
    for (i = 1; i < n+1; i++)
    {
        printf("Set the values of the triangle sides (one per line): \n");
        scanf("%d",&a); printf ("%d,", a);
```

```

scanf("%d",&b); printf ("%d", b);
scanf("%d",&c); printf ("%d: ", c);

checktriangle(a,b,c);
};

return 0;
}

```

## Versión B.

```

#include <math.h>
#include <stdio.h>

int checktriangle(float a, float b, float c)
{
    float A, B, C;
    A = a;
    B = b;
    C = c;

    if((C<A+B)&&(b<A+C)&&(A<C+B))
    {
        if(A==B && B==C)
        {
            printf("Equilateral Triangle \n");
        }else if(A==B || B==C || A==C)
        {
            printf("Isosceles Triangle \n");
        }else
        {
            printf("Scalene triangle \n");
        }

        float value = (A*A)+(B*B)-(C*C);

        if(value==0)
        {
            printf("Right Triangle \n");
        }else if(value>0)
        {
            printf("Acute Triangle \n");
        }else
        {
            printf("Obtuse Triangle \n");
        }
    }
    else
    {
        printf("Is not a Triangle \n");
    }

    return 0;
}

int main()
{
    int i, n;

```

```

float a, b, c;
printf("Number of test cases: \n");
scanf("%d",&n); printf ("%d", n);
for (i = 1; i < n+1; i++)
{
printf("Set the values of the triangle sides (one per line): \n");
scanf("%f",&a); printf ("%f,", a);
scanf("%f",&b); printf ("%f,", b);
scanf("%f",&c); printf ("%f: ", c);

checktriangle(a,b,c);
};

return 0;
}

```

### Versión C.

```

#include <math.h>
#include <stdio.h>

int checktriangle(float a, float b, float c)
{
float A, B, C;

if((c<a+b)&&(b<a+c)&&(a<c+b))
{
if(a==b && b==c){
printf("Equilateral Triangle \n");
}else if(a==b || b==c || a==c){
printf("Isosceles Triangle \n");
}else{
printf("Scalene triangle \n");
}

A = (pow(b,2) + pow(c,2) - pow(a,2)) / (2*b*c);
B = (pow(a,2) + pow(c,2) - pow(b,2)) / (2*a*c);
C = (pow(a,2) + pow(b,2) - pow(c,2)) / (2*a*b);

if (A < 0 || B < 0 || C < 0){
printf("Obtuse Triangle \n");
} else if (A == 0 || B == 0 || C == 0){
printf("Right Triangle \n");
} else {
printf("Acute Triangle \n");
}

}
else{
printf("Is not a Triangle \n");
}

return 0;
}

int main()
{

```



```

int i, n;
float a, b, c;
printf("Number of test cases: \n");
scanf("%d",&n); printf ("%d", n);
for (i = 1; i < n+1; i++)
{
printf("Set the values of the triangle sides (one per line): \n");
scanf("%f",&a); printf ("%f,", a);
scanf("%f",&b); printf ("%f,", b);
scanf("%f",&c); printf ("%f: ", c);

checktriangle(a,b,c);
};

return 0;
}

```

### Adaptación a Java.

```

package triangulo;

import java.util.Scanner;

public class Triangulo {

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        int i, n;

        float a = 0, b = 0, c = 0;

        System.out.println("Number of test cases: \n");

        n = sc.nextInt();

        for (i = 0; i < n+1 ; i++){

            System.out.println("Set the values of the triangle sides (one per
line): \n");

            a = sc.nextFloat();

            b = sc.nextFloat();

            c = sc.nextFloat();

        }

        checktriangle (a, b, c);

    }

    public static void checktriangle (float a, float b, float c){

        float A, B, C;

```

```

if ((c<a+b)&&(b<a+c)&&(a<c+b)){
    if (a==b && b==c){
        System.out.println("Equilateral Triangle \n");
    }else if (a==b || b==c || a==c){
        System.out.println("Isosceles Triangle \n");
    }else{
        System.out.println("Scalene Triangle \n");
    }
    A = b*b + c*c - a*a;
    B = a*a + c*c - b*b;
    C = a*a + b*b - c*c;

    if (A < 0 || B < 0 || C < 0){
        System.out.println("Obtuse Triangle \n");
    }else if (A == 0 || B == 0 || C == 0){
        System.out.println("Right Triangle \n");
    }else {
        System.out.println("Acute Triangle");
    }
}else {
    System.out.println("Is not a Triangle");
}
}
}

```

## Anexo II Límites numéricos de los tipos de datos

Para poder establecer cuáles son los valores máximos de los tipos de datos *int* y *float* es necesario conocer el valor de las constantes de macro que determinan estos límites en cada lenguaje de programación.

### Límites en C

En C, estas constantes están incluidas en los archivos de encabezado `limits.h` en el caso del tipo de dato `int` (también hay valores para `char`, `short`, `long` y `long long`) y en el archivo de encabezado `float.h` para el tipo `float` (también se definen aquí los valores para `double` y `long double`)<sup>3</sup>.

A continuación, se presenta el código desarrollado:

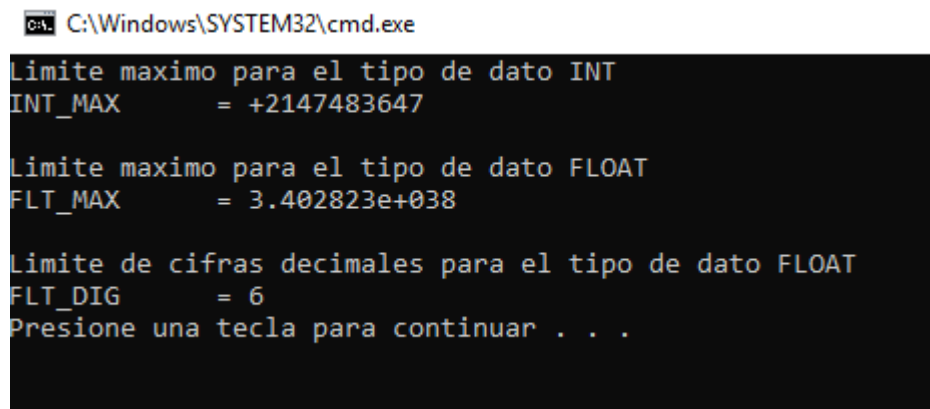
```
#include <stdio.h>
#include <limits.h>
#include <float.h>

int main(int argc, char **argv)
{
    printf("Limite maximo para el tipo de dato INT\n");
    printf("INT_MAX      = %d\n", INT_MAX);

    printf("\nLimite maximo para el tipo de dato FLOAT\n");
    printf("FLT_MAX      = %e\n", FLT_MAX);

    printf("\nLimite de cifras decimales para el tipo de dato FLOAT\n");
    printf("FLT_DIG      = %d\n", FLT_DIG);
    return 0;
}
```

En la Figura 37 se pueden comprobar los valores de las constantes mencionadas y que se utilizan para establecer los límites en las pruebas del presente trabajo.



```
ca. C:\Windows\SYSTEM32\cmd.exe
Limite maximo para el tipo de dato INT
INT_MAX      = +2147483647

Limite maximo para el tipo de dato FLOAT
FLT_MAX      = 3.402823e+038

Limite de cifras decimales para el tipo de dato FLOAT
FLT_DIG      = 6
Presione una tecla para continuar . . .
```

Figura 37 - Valores límite para las variables en C

---

<sup>3</sup> Puede consultar más información sobre las constantes de macro de distintos tipos de datos en C en el enlace a continuación, <https://es.cppreference.com/w/c/types/limits>

## Límites en Java

En Java estos valores vienen determinados por en la estructura de los valores primitivos de datos<sup>4</sup>. Accedemos al límite máximo de un float con el siguiente código:

```
public class Limites {
    public static void main(String[] args) {
        System.out.println("float\t" + Float.MAX_VALUE);
    }
}
```

Obteniendo como resultado el valor mostrado en la Figura 38.

```
run:
float  3.4028235E38
BUILD SUCCESSFUL (total time: 1 second)
```

Figura 38 - Valores límite para las variables en Java

---

<sup>4</sup> La información sobre las constantes del tipo primitivo Float se ha obtenido de la documentación de Java disponible en: <https://docs.oracle.com/en/java/javase/14/docs/api/java.base/java/lang/Float.html>