Revista de I+D Tecnológico

Portal de revistas: http://revistas.utp.ac.pa

Pruebas de mutación, control sobre variaciones en el código fuente

Mutation tests, control of variations in the source code

Nelson Sánchez Álvarez[1], Neybis Lago Clara[1]

[1] *Centro de Soporte, Universidad de las Ciencias Informáticas, Cuba*

***Autor de correspondencia:** nalvarez@uci.cu

RESUMEN– Las pruebas de *software*, como complemento fundamental dentro del proceso de calidad, se han convertido en un factor importante para las empresas. La calidad es el resultado de una serie de pruebas y revisiones que se le realiza a un programa en específico para comprobar que se cumpla con los requisitos definidos en las primeras etapas de su diseño. En la rama de las ciencias informáticas la calidad se rige por métricas y modelos específicos que brindan un apoyo importante para validar el *software*. La manera de lograr un nivel mayor de calidad requiere de esfuerzo y técnicas que ayuden a este proceso. La presente investigación apoya el proceso de pruebas que se realiza en los productos *software* que se crean en el Departamento de Señales Digitales de la CITEC. Tiene como objetivo principal el desarrollo de pruebas de mutación para de esta manera obtener resultados a través de más de una técnica de prueba de Caja Blanca. Se utilizaron tecnologías de gran potencia para el desarrollo de la herramienta entre las que se mencionan C++ como lenguaje de programación y Qt como *framework* de desarrollo. El *Visual Paradigm* fue la herramienta seleccionada para el modelado de los principales diagramas y representaciones del sistema y la construcción de la solución fue guiada por la metodología ágil SXP. Además, se aplicaron los patrones de diseño y de arquitectura para brindan mayor organización a la estructura de la aplicación.

Palabras clave– *Prueba, mutación, técnicas de pruebas de caja Blanca.*

ABSTRACT– Software testing as an essential complement in the quality process, have become an important factor for companies. Quality is the result of a series of tests and inspections to be performed at a specific program to check compliance with the requirements defined in the early stages of design. In the branch of computer science, the quality is governed by metrics and specific models that provides important support to validate the software. The way to achieve a higher level of quality requires effort and techniques to help this process. This research contributes to raising the quality of the tests performed to software products that are created in the Digital Signals Department of the CITEC. It has as main goal the development of testing structure of control and testing mutation so as to obtain results through more than one technique of White Box testing. Powerful technologies were used for the development of the tool between the mentioned C++ programming language and development framework Qt, The Visual Paradigm tool was selected for modeling the main diagrams and representations of the system and the construction of the solution was guided by the agile methodology SXP. Further references to architecture and design patterns were applied to provide greater organization to the structure of the application.

Keywords– *Test, mutation, technical white box testing.*

1. Introducción

La creación e introducción paulatina de productos informáticos, con el paso de los años, ha ocasionado gran dependencia de los mismos al estar ligados a la mayoría de los procesos sociales. Estos productos, aunque son creados para eliminar en gran medida los errores del trabajo manual, no están exentos de ellos y en muchas ocasiones no corresponden a las reglas establecidas para su elaboración. Las empresas o instituciones que demandan la utilización de este tipo de recursos exigen, por demás, un funcionamiento correcto de los mismos.

Es bajo estas condiciones que la palabra calidad también llega a las ciencias informáticas como una de las disímiles formas de entregar productos con el mínimo de errores posibles, tanto desde el punto de vista sintáctico como desde la lógica que guía su desarrollo. La calidad como un ente individual "es ambiguamente definido y pocas veces comprendido debido a que la calidad no es una sola idea, es un concepto multidimensional. La dimensión de calidad incluye el interés, el punto de vista y los atributos de la entidad y varía según cada persona en particular [1] ."

En el mundo del *software*, se puede decir que la calidad, es el resultado de una serie de pruebas y revisiones que se le realizan a un programa en específico, para comprobar que se cumpla con los requisitos definidos en las primeras etapas de su diseño.

Citación: N. Sanchez y N. Lago, "Pruebas de mutación, control sobre variaciones en el código fuente", Revista de I+D Tecnológico, vol. 15, no. 2, pp. (38-45), 2019.

Tipo de artículo: Original**. Recibido:** 17 de septiembre de 2018. **Recibido con correcciones:** 10 de junio de 2019. **Aceptado:** 26 de julio de 2019.

DOI. https://doi.org/10.33412/idt.v15.2.2228.

Copyright: 2019 N. Sanchez y N. Lago. This is an open access article under the CC BY-NC-SA 4.0 license (https://creativecommons.org/licenses/by-nc-sa/4.0/).

En la rama de las Ciencias Informáticas la calidad se rige por métricas y modelos específicos que brindan un apoyo paso a paso de que es importante validar en dicho *software*.

Alguno de los métodos más utilizados es la realización de pruebas de *software* para mejorar la calidad de los productos informáticos. Según la IEEE (*Institute of Electricalan Electronics Engineers*, por sus siglas en inglés) las pruebas se definen como "una actividad en la cual un sistema o componente es ejecutado bajo condiciones específicas, se observan o almacenan los resultados y se realiza una evaluación de algún aspecto del sistema o componente" [16]. Son varias los métodos de pruebas que existen y con ellos es posible probar el *software* en diferentes aspectos. Desde el punto de vista funcional se pueden mencionar las pruebas de caja blanca y las de caja negra.

Las pruebas de caja negra "también conocidas como Pruebas de Comportamiento, son pruebas que se basan en la especificación del programa o componente a ser probado para elaborar los casos de prueba" [2]. Las pruebas de caja blanca, por su parte, son "un tipo de prueba de *software* que se realiza sobre las funciones internas de un módulo" [2]. Su utilización tiene gran importancia debido a que los métodos de caja blanca "permiten derivar casos de prueba que garanticen que todas las rutas independientes dentro del módulo se ejecuten al menos una vez. Ejecuta los lados verdadero y falso de todas las decisiones lógicas dentro del código. Posibilita ejecutar todos los ciclos dentro y en sus límites operacionales y ejercita las estructuras de datos internas para asegurar su validez" [3].

De acuerdo con la importancia de la calidad en desarrollo de *software*, durante cada una de sus etapas y en el producto final, se determinó evaluar el proceso de pruebas que se realiza en el Facultad de Ciencias y Tecnologías Computacionales. Se tomó como muestra el departamento Señales Digitales, encargado del desarrollo de sistemas para el control de cámaras y procesamiento de imágenes, entre otras actividades.

Al analizar el modo en que se ejecutaba el proceso de pruebas, se observó que solamente se realizan pruebas de Caja Negra y se manifestó que no existía una herramienta que posibilitara la realización de técnicas de prueba de Caja Blanca. La ausencia de este tipo de pruebas provoca que los desarrolladores y probadores de software deban realizar estas técnicas de forma manual, lo que ocasiones conlleva a que ni siquiera se realicen. El hecho de que este tipo de pruebas no sea realizado de manera correcta influye en que los productos que se desarrollan en el departamento no sean lo suficientemente correcto desde el punto de vista del código fuente.

Luego de varios encuentros con los equipos de desarrollo del Departamento Señales Digitales se comprobó que muchos de los productos *software* que se desarrollan en el centro muestran errores luego de desplegados. Además, la modificación de cualquier operador dentro del código fuente muestra resultados que difieren, en gran medida, de los obtenidos con la versión anterior del código ejecutado.

Todo esto influye en la satisfacción del cliente, pues en algunos casos los resultados que obtienen a través del software no son certeros o de confianza y provoca que tengan que recurrir a los antiguos métodos no automatizados para realizar su trabajo. Por lo tanto, se afecta su trabajo y el de los desarrolladores al tener que realizar cambios al producto durante la fase de soporte para mejorar su funcionamiento.

A partir de la problemática detectada, en el desarrollo de productos de *software*, surge la necesidad de realizar pruebas de mutación que permitan detectar las variaciones en el código fuente, para evitar errores posteriores en su ejecución, o incumplimiento de los requisitos funcionales solicitados por el cliente. Se propone en la presente investigación automatizar las pruebas de mutación, garantizando mayor fiabilidad en los resultados y disminuir el tiempo en que sea realizan las pruebas antes mencionadas.

2. Conceptos fundamentales

Pruebas de *software*

Las pruebas de *software*, desde un enfoque general, pueden definirse como "una etapa del desarrollo de software que incluye procesos que permiten verificar y revelar la calidad de un *software*. Básicamente es una fase en el desarrollo de software que consiste en probar las aplicaciones construidas [4]."

Al decir de Pressman, lo principales objetivos de las pruebas de software son:

- ✓ La prueba es el proceso de ejecución de un programa con la intención de descubrir un error.

- ✓ Un buen caso de prueba es aquel que tiene una alta probabilidad de mostrar un error no descubierto hasta entonces.
- ✓ Una prueba tiene éxito si descubre un error no detectado [5].

El proceso de verificación del *software* es de carácter destructivo y describe que un caso de prueba se considera exitoso si detecta defectos no descubiertos aún. La prueba demuestra hasta qué punto el programa tiene un buen funcionamiento, pero sin embargo no garantiza la ausencia de defectos en un *software*. Es por este motivo que la calidad de un *software* no depende solamente de un plan de pruebas bien estructurado, puesto que de esta forma no se asegura que el producto se encuentre libre de defectos, aunque es considerado como un factor importante y que influye en gran medida que este parámetro sea cumplido exitosamente.

2.1 Métodos de pruebas

Entre los métodos de pruebas las más utilizadas por ser más generales y poder aplicarse a cualquier *software* son [6]:

Pruebas de caja negra: se centran en los requisitos funcionales del *software* y los casos de prueba diseñados pretenden demostrar que las funciones del *software* son operativas, que la entrada se acepta de forma adecuada y que se produce una salida correcta.

Pruebas de caja blanca: es un método de diseño de casos de prueba que usa la estructura de control del diseño procedimental para derivar los casos de prueba.

Pruebas de estructuras de control

"La estructura de control es un bloque de código que permite tomar decisiones de manera dinámica, sobre un código existente. Existen diferentes variantes de las pruebas de estructura de control donde cada una de ellas amplía la cobertura de la prueba y mejora la calidad del resultado obtenido. Dentro de las pruebas de estructura de control se encuentran" [13]

- ✓ Pruebas de condición.
- ✓ Prueba de flujo de datos.
- ✓ Pruebas de bucles.

2.2 Pruebas de Mutación

Las técnicas de mutación se basan en la modelización de las faltas típicas que se comenten al hacer un programa, mediante lo que se conocen como operadores de mutación (dependientes del lenguaje de programación). Cada operador de mutación se aplica sobre el programa, dando lugar a una serie de mutantes. Una vez que se tiene generado el conjunto de mutantes, se generan casos de pruebas que ejerciten la parte mutada del mismo. Tras generar casos de pruebas para cubrir todos los mutantes, teóricamente se tienen cubiertas todas las posibles faltas cometidas (en la práctica, solo las faltas modeladas por los operadores de mutación) [7]. La eficiencia de esta técnica depende de la cantidad de mutantes que se construyan en busca de abarcar la mayor cantidad de código posible.

Tipos de pruebas de mutación

Mutación (*Standard* o fuerte): Se basa en operadores básicos dependiendo del lenguaje de programación que se utilice.

Mutación *abs/ror*: Utiliza solamente los operadores *abs* y *ror* para generar los mutantes. El operador *abs* reemplaza el valor de cada variable *xporabs*(x),-*abs*(x) y *zpush*(x). El operador *zpush* hace que el mutante muera inmediatamente si su argumento es cero, lo que requiere que los datos de prueba fuercen a que toda expresión adquiera el valor cero. El operador *ror* genera mutantes reemplazando cada operador relacional por otros operadores relacionales.

Mutación 10%: En este caso, se seleccionan aleatoriamente el diez por ciento de los mutantes generados para cada tipo de mutación.

Mutación selectiva: En la mutación selectiva se descartan los mutantes que fueron generados con los operadores de mutación que generan más mutantes.

Mutación débil: En la mutación débil, los mutantes son evaluados antes de finalizar la ejecución del programa mutante. Es decir que la comparación entre el programa original y el mutante se realiza en un estado intermedio del mismo, lo que permite que se reduzcan los tiempos de prueba.

Los pasos principales para elaborar las pruebas de mutación son [7]:

1. Análisis del programa: operadores de mutación.
2. Generación de mutantes.
3. Ejecución contra casos de prueba.

Las pruebas de mutación cuentan con una gran diversificación de lenguajes y un aumento dentro del paradigma orientado a objetos. Desde otro punto de vista y como parte de sus desventajas se encuentra la necesidad de la aparición de nuevos operadores de prueba.

2.3 Herramientas para la realización de pruebas. Estudio del arte

La necesidad de realizar pruebas de calidad converge hacia el aseguramiento de la eficiencia del producto antes de salir al mercado. Es por ello que es imprescindible evaluar desde todos los ámbitos posibles un *software* y realizar todo tipo de pruebas, ya sean las pruebas de caja blanca o pruebas de caja negra. Las primeras posibilitan medir la eficiencia de los códigos convirtiendo la automatización de las mismas en un reto actual para las empresas de *software* en el mundo.

Program Exploration

Se trata de una herramienta desarrollada por un equipo de *Microsoft Research*[1], cuya última actualización data de febrero del 2015. Tiene la capacidad de explorar el código fuente de las aplicaciones, encontrar los grafos de caminos, seleccionar el subconjunto mínimo suficiente de caminos para probar todas las sentencias de código y, finalmente, genera las entradas representativas necesarias al programa para recorrer todos estos caminos.

Durante el estudio de la herramienta se observó que se especializa solamente en pruebas del camino básico y no posee ninguna funcionalidad para pruebas de mutación. Además, la herramienta no ha sido actualizada por lo que no incluye nuevas técnicas de pruebas o mejoras en las que realiza, por lo que sería necesario utilizar más de una herramienta. De esta forma el proceso de pruebas sería más costoso, llevaría más tiempo y esfuerzo por parte del equipo de desarrollo.

JTest

Es una herramienta que permite realizar análisis de código, pruebas unitarias automáticas y cobertura de código, así como generación dinámica de pruebas funcionales. "El uso de *JTest* es realmente sencillo, y se centra en una potente interfaz de usuario. Para comenzar el proceso de test, tan solo es necesario seleccionar la clase deseada y pulsar un botón. Los resultados del test se presentan de forma clara y organizada, utilizando una estructura tipo árbol. Dicha estructura incluye todos los detalles del test (datos de entrada, salidas generadas, errores encontrados, líneas del código fuente, entre otros) y facilita enlaces directos con el código fuente para su edición. Hay cuatro tipos de procesos de test disponibles: Caja Blanca, Caja Negra, Regresión y Cobertura [8]."

La herramienta *JTest* no posee ningún método para realizar pruebas de mutación, por lo que no es viables su utilización como solución a la problemática detectada. Tampoco puede ser ajustada a dichas pruebas pues es *software* privativo, por lo que su código fuente no está disponible para su modificación.

Bullseye Coverage

"Es un analizador de código de cobertura para C++ y C que indica cómo gran parte del código fuente se pone a prueba. Puede usar esta información para rápidamente centrar su esfuerzo de ensayo y determinar las áreas que necesitan ser revisadas. El código cobertura de análisis es útil durante la unidad de verificación, integración de pruebas y la liberación final. Permite crear código más fiable y ahorrar tiempo [14].

Según la bibliografía consultada, la licencia del *software* debe ser renovada cada cierto tiempo, según la necesidad del cliente, oscilando de 500 euros a 1000 euros. Además de costoso, es un *software* privativo por lo que sus funcionalidades no pueden ser adaptadas para solucionar la problemática de la investigación.

En Cuba se aboga por la soberanía tecnológica y el desarrollo de *software* libre, logrando disminuir costos en materia de automatización de procesos y evitando la dependencia de terceros. El estudio del arte realizado demostró que las herramientas estudiadas no cumplían con las necesidades planteadas o, en algunos casos, son distribuidas bajo licencia privativa, por lo que se propone realizar una herramienta propia del país, que ajuste la realización de pruebas de mutación a las características de la industria de desarrollo de *software* en Cuba.

2.4 Propuesta de solución

Se propone como objetivo de la investigación, en primer orden, implementar una herramienta de pruebas de mutación de acuerdo a las tecnologías definidas para el Departamento Señales Digitales. Además, se automatizará esta técnica, posibilitando probar un *software* desde distintas perspectivas, generando tantos mutantes como sean necesarios para evaluar la modificación de los operadores del código, según los diferentes escenarios en los que pueda utilizarse el producto desarrollado.

[1] Es una división de Microsoft para investigar los diversos temas de ciencia de equipo y problemas.

Todo esto permitirá que el trabajo del grupo calidad, a través del uso de la herramienta, sea más fácil y organizado.

Diagrama de componentes

La herramienta está compuesta por dos paquetes principales, que representan a su vez cada una de las capas en que está estructurada. La capa de presentación incluye el componente llamado principal, a través del cual se maneja la entrada y salida de los datos necesarios para el funcionamiento del sistema.

La capa de lógica del negocio, por su parte, contiene el componente mutación, en el cual se define lo necesario para realizar la técnica de pruebas de mutación. Además, se representa el componente controlador, que realiza las principales funcionalidades de la herramienta y se relaciona con mutación.

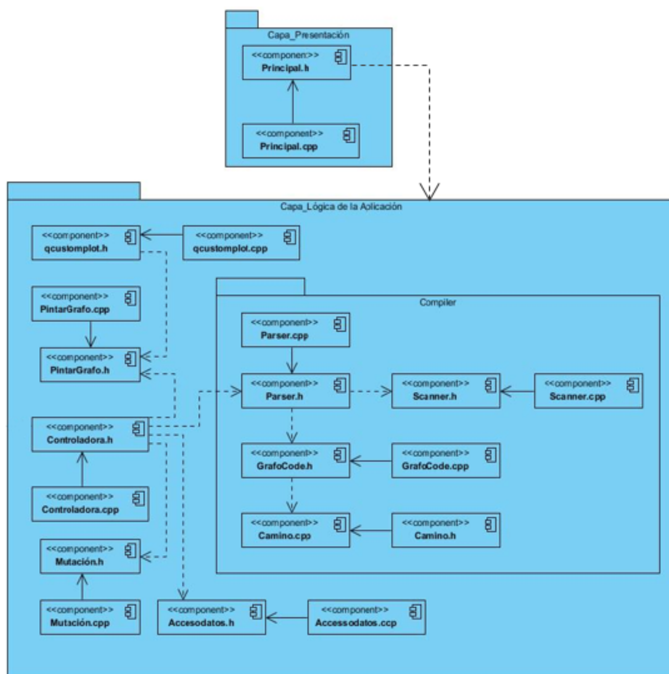

Figura 1. Diagrama de componentes. Imagen propia.

2.5 Lenguaje de programación y entorno de desarrollo integrado

Los lenguajes de programación son idiomas artificiales compuestos por reglas semánticas y sintácticas, escritas mediante símbolos. El objetivo de estos lenguajes es lograr un entendimiento entre el lenguaje natural y el código que entienden las computadoras para implementar programas y sistemas que faciliten el uso de estas últimas [9].

2.5.1 Lenguaje de programación C++

C++ surge como una extensión del lenguaje C. En sus inicios incluía gran parte del lenguaje original agregando, por demás, muchas características sofisticadas como el paradigma de Programación Orientada a Objetos, excepciones, sobrecarga de operadores, templates o plantillas. Al igual que C, C++ es un lenguaje que proporciona gran fortaleza a las aplicaciones que se desarrollan con él y permite la construcción de complejos productos de software.

La llegada de C++ al mundo del *software* ocasionó un salto de avance en la comunidad de los programadores, pues este último conservaba la potencia estructural del lenguaje C, mejorando muchas de las desventajas que este poseía. Los desarrolladores encontraron entonces un lenguaje potente con una curva de aprendizaje más alta que C, de mayor entendimiento y sobre todo que fuera multiplataforma.

Ventajas de C++ [10]

1. Es un lenguaje muy flexible que permite programar con múltiples estilos. Uno de los más empleados es el estructurado "no llevado al extremo" (permitiendo ciertas licencias de ruptura).
2. Un sistema de tipos que impide operaciones sin sentido.
3. Posee un conjunto reducido de palabras clave.
4. Usa un lenguaje procesado para tareas como definir macros e incluir múltiples archivos de código fuente
5. Tipos de datos agregados que permiten que datos relacionados se combinen y se manipulen como un todo.
6. Acceso a memoria de bajo nivel mediante el uso de punteros.

2.5.2 QtCreator

Ofrece gran compatibilidad con varios sistemas operativos, posee riquezas en sus funciones y librerías permitiendo el desarrollo de interfaces amigables y con un alto rendimiento con C++. *Qt* tiene la característica de presentar el código fuente disponible, posee gran

documentación y calidad en cuanto a soporte técnico y una comunidad de desarrollo que le da mantenimiento, ha sido adaptado a las características de los entornos de desarrollo actuales y está pensado para sistemas operativos recientes.

Características principales de *QtCreator*

Permite desarrollar aplicaciones *Qt* de forma rápida y fácilmente con asistentes de proyectos, así como acceder rápidamente a los últimos proyectos y sesiones.

Cuenta con dos editores visuales, *QtDesigner* para diseñar interfaces de usuario a partir de *QtWidgets*, y Qt *Quick Designer* para el desarrollo de interfaces de usuario animadas con el lenguaje *JavaScript*.

Contiene un sofisticado editor de código que proporciona completamiento de código y ayuda de contexto para el lenguaje C++ y *JavaScript*.

Permite generar, ejecutar e implementar proyectos de *Qt* dirigidos a plataformas móviles y de escritorio, tales como Microsoft Windows, Mac OS X, Linux, Symbian, Android y Maemo.

Posibilita acceder fácilmente a la documentación mediante el uso de la ayuda contextual integrada de *Qt*.

Este se encuentra bajo licencia libre, además de presentar la gran ventaja de ser multiplataforma [11].

3. Pruebas al sistema

Las **pruebas de aceptación** se han asociado con muchas definiciones diferentes. “A grandes rasgos se puede decir que están basadas más en cómo comprobar que el sistema en desarrollo cumple los requisitos del cliente y menos en reducir el número de errores en el código. En otras palabras, las pruebas de aceptación no son acerca de las pruebas de código, sino sobre lo que desea el cliente con el sistema o de su negocio” [12].

4. Pruebas al sistema

Las **pruebas de aceptación** se han asociado con muchas definiciones diferentes. “A grandes rasgos se puede decir que están basadas más en cómo comprobar que el sistema en desarrollo cumple los requisitos del cliente y menos en reducir el número de errores en el código. En otras palabras, las pruebas de aceptación no son acerca de las pruebas de código, sino sobre lo que desea el cliente con el sistema o de su negocio” [12].

Se realizaron 13 Pruebas de aceptación, de acuerdo a la cantidad de Historias de usuarios definidas. Para ello el cliente ejecutó los pasos de acuerdo a lo establecido en la fase de negocio para cada una de las funcionalidades, evaluando los resultados, aceptando los mismos cuando el requisito asociado a la historia de usuario.

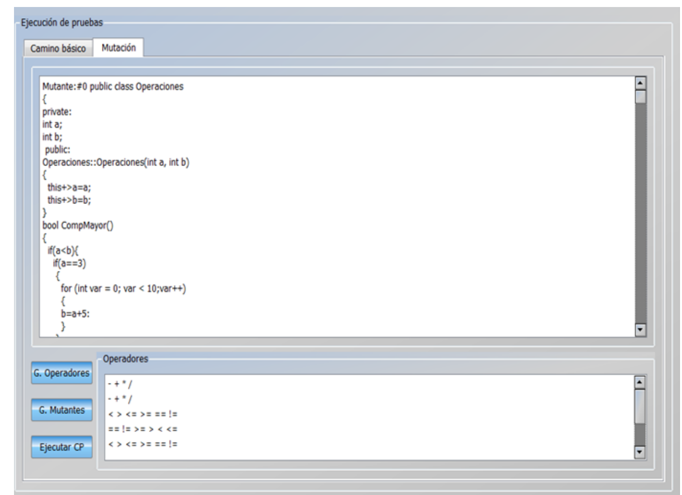

Figura 2. Vista de la solución. Imagen propia.

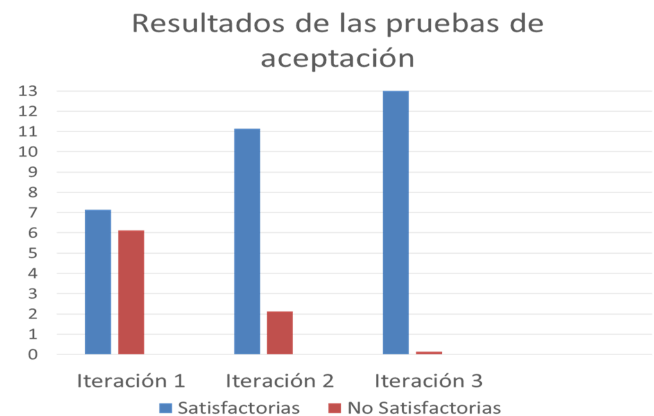

Figura 3. Gráfico con el resultado de pruebas de aceptación. Imagen propia.

En la **primera iteración** resultaron satisfactorias siete de las pruebas realizadas representando el 53, 84 % y seis no satisfactorias representando el 46, 16 %. Las no conformidades encontradas fueron:

La funcionalidad generar mutantes: No generaba la cantidad de mutantes en correspondencia a los operadores identificados en el código evaluado.

En la **segunda iteración** resultaron 11 pruebas evaluadas de bien, lo que representó el 84,62 % y de mal dos pruebas, representando el 15,38% del total. Las no conformidades encontradas fueron:

La funcionalidad generar mutantes: Generaba una cantidad de mutantes sin correspondencia a los operadores identificados en el código evaluado.

Las deficiencias encontradas se continuaron corrigiéndose, resultando en una **tercera iteración** el 100 % de las pruebas satisfactorias.

Pruebas unitarias

Cuando los desarrolladores o probadores ejecutan casos de pruebas sobre un sistema completo, puede ocurrir que los errores detectados no puedan localizarse fácilmente, haciendo complicado este proceso. En específico, cuando se utiliza una metodología ágil, el desarrollo se realiza en menos tiempo, lo que, en ocasiones, puede traer como consecuencia mayor cantidad de fallos. Por estos motivos es que la metodología SXP sugiere que se realicen pruebas a fragmentos de código aislados para descartar posibles errores en áreas más pequeñas. De esta manera la integración de esos bloques de código será más fácil y el proceso de pruebas del sistema más satisfactorio.

El tipo de prueba recomendada, en este caso por la metodología utilizada, son las pruebas unitarias descritas como: "Un procedimiento usado para probar que un módulo o método funciona apropiadamente y en forma independiente. A través de ellas se verifica si el bloque de código probado se ejecuta dentro de los parámetros y especificaciones concretas. Permiten, además, detectar efectivamente la inyección de defectos durante fases sucesivas de desarrollo o mantenimiento" [12].

Pruebas del camino básico

Las pruebas unitarias pueden ser aplicadas por varias técnicas de prueba y a través de diferentes vías. En el caso de la presente investigación se realizan las pruebas del camino básico de manera manual, por ser un *software* pequeño. Este tipo de pruebas tiene como característica esencial que genera un grafo de flujo a través del cual se puede calcular la complejidad ciclomática que posee la función analizada, certificando la calidad que posee el código. Este factor puede ser evaluado a través de la existencia de un rango numérico para el valor de la complejidad ciclomática que determina la evaluación de riesgo del código en cuestión para su ejecución. El valor de la complejidad de un grafo de flujo se calcula mediante la fórmula matemática $CC = A - V + 2$, donde:

- ✓ CC significa complejidad ciclomática.
- ✓ A se refiere al conjunto de arista del grafo.
- ✓ V se refiere al conjunto de vértices del grafo.

La siguiente tabla muestra el análisis de la complejidad ciclomática en un *software* [15]:

Tabla 1. Análisis de riesgo de la complejidad ciclomática

Complejidad ciclomática	Evaluación del riesgo
1 – 10	Programa simple, sin mucho riesgo.
11 – 20	Más complejo, riesgo moderado.
21 – 50	Complejo, programa de alto riesgo.
50 en adelante.	Programa no testeable, muy alto riesgo.

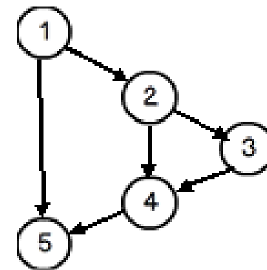

Figura 1. Grafo de flujo del método cargar clase.

- ✓ Complejidad ciclomática del método Cargar clase:

$CC = A - V + 2$
$= 6 - 5 + 2$
$= 3$

- ✓ Evaluación de riesgo: Sin mucho riesgo.

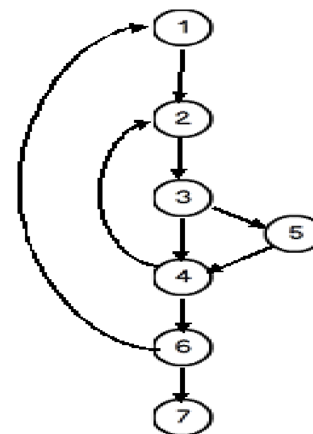

Figura 2. Grafo de flujo del método generar mutantes.

- ✓ Complejidad ciclomática del método generar mutantes:

$$CC = A - V + 2$$
$$= 9 - 7 + 2$$
$$= 4$$

- ✓ Evaluación de riesgo: Sin mucho riesgo.

Se analizaron dos métodos, correspondientes a las clases o funcionalidades de mayor importancia para el desempeño de la herramienta de pruebas, de acuerdo a lo establecido durante el análisis de requisitos. En una primera iteración resultaron en un rango de testeo posible un método representando el 50 % de pruebas satisfactorias. El método Generar operadores resultó no testeables debido a que se encontraron errores en algunas de las estructuras ***if*** que contenían, lo que representó el 50 % de pruebas no satisfactorias. En una segunda iteración, los códigos no testeables fueron corregidos, encontrándose todos los métodos evaluados sin mucho riesgo o con riesgo moderado, lo que representa el 100% de las pruebas satisfactorias.

5. Conclusiones

Con el objetivo de solucionar la problemática detectada en el departamento señales digitales se desarrolló una herramienta que automatizó las pruebas de mutación en el código fuente. En la elaboración del *software* se tuvieron en cuenta los requisitos planteados por los desarrolladores y líderes de proyecto. El producto final fue probado siguiendo la técnica del camino básico y, algunas clases fueron sometidas al proceso de prueba realizando la técnica de mutación. Esta última fue realizada en dos formas diferentes: de manera manual y utilizando la propia herramienta desarrollada como solución a la problemática de la investigación. Los resultados fueron comparados y repetidos hasta lograr su homogenización, comprobándose la fiabilidad de la herramienta. La herramienta para realizar pruebas de mutación desarrollada se utiliza en el departamento señales digitales para la comprobación del código fuente de las aplicaciones que se desarrollan para varios clientes del país. Con su utilización ha sido posible detectar en más de 25% los fallos en los sistemas y reducir los tiempos de pruebas.

5. Referencias

[1] G, Msc. Alejandro Bedini. 2013. Calidad de Software. Calidad Tradicional y de Software. Chile: s.n., 2013.

[2] Mora, Ing. Jhon Freddy Montes. 2012. Pruebas del software "Caja Blanca y Caja Negra". Colombia: Universidad Nacional Abierta y a Distancia UNAD, 2012.

[3] Rodríguez, Dr. Eduardo. 2011. Importancia de las pruebas de software. México: CINVESTAV, 2011.

[4] Tenorio, Roberto Ruiz. 2010. Las pruebas de software y su importancia en las organizaciones. Xalapa, Veracruz: Universidad Veracruzana., 2010.

[5] Pressman, Roger S. 2002. Ingeniería de Software, un enfoque práctico. 2002.

[6] Oliveros, Juan Gabriel Romeros. 2008. Pruebas de Software - Calidad de Software. s.l.: UNAD, 2008.

[7] Guerini, Ing. Mario Luís. 2007. Revisión de resultados experimentales en Técnicas de Prueba y de Educción de Conocimientos. Argentina: s.n., 2007

[8] Torán., Félix. 2014. Entorno para la depuración avanzada de código Java. España: s.n., 2014.

[9] Clara, Neybis Lago. 2012. Módulo Seguridad Social - Prevención y Atención - Asistencia Social de la Dirección de Trabajo de la Administración Provincial de Artemisa. Artemisa: Universidad de las Ciencias Informáticas, 2012.

[10] Yerovi, Ximena y Poz, Ing. Alejandra. 2013. Ventajas de C++. Ecuador: s.n., 2013.

[11] Vega, Yosami González y Valdés, Yasmani Ledesma. 2013. Sistema para la gestión de la información en el estudio neuroinmunológico de proteínas del líquido cefalorraquídeo. La Habana: Universidad de las Ciencias Informáticas, 2013.

[12] Hall, Ben. 2009. MSDN Megazine. [En línea] marzo de 2009. [Citado el: 28 de abril de 2016.] http://msdn.microsoft.com/es-es/magazine

[13] Moreno, Ing Pilar Alexandra. 2012. Modulo Ingenieria de Software. Programa Ingenieria de Sistemas. s.l. : Escuela de Ciencias Basicas, tecnologias e ingenieria., 2012.

[14] Claramunt, Javier Lambert. 2012. Implementación de un prototipo funcional para automatizar el proceso de pruebas de Caja Blanca del departamento Señales Digitales del Centro GEySED. La Habana : s.n., 2012.

[15] Martínez, Ing. Eduardo Salazar. 2012. Propuesta de procedimiento para realizar pruebas de Caja Blanca a la aplicaciones que se desarrollan en el lenguaje Python. Cuba : Facultad regional de Granma., 2012.

[16] Institute of Electrical and Electronics Engineers. [Online]. Available: *https://www.ieee.org/*