REVISTA COLOMBIANA DE COMPUTACIÓN
Volumen 13, número 1
Págs. 73 - 94

Propuesta de Procedimiento para realizar pruebas de Caja Blanca a las aplicaciones que se desarrollan en lenguaje Python

Eduardo Salazar Martínez[1]

Fecha de Recibido: 19/09/2011 Fecha de Aprobación: 22/01/2012

Resumen

Uno de los mayores problemas que se afrontan en la esfera de la informática es la calidad de software. El proceso de pruebas al software (también conocido como beta testing) es uno de los aspectos fundamentales para medir el estado de calidad de un sistema informático e introducirlo satisfactoriamente en el mercado mundial. El objetivo del presente trabajo de diploma, es elaborar la propuesta de un procedimiento para realizar pruebas, aplicando el método de Caja Blanca, a las aplicaciones que se desarrollan con lenguaje Python en el Centro de Desarrollo de la Facultad Regional Granma de la Universidad de las Ciencias Informáticas.

En esta investigación se hizo un análisis de las principales bibliografías especializadas en el tema, profundizando en los diferentes métodos de pruebas que existen, fundamentalmente en las técnicas encaminadas a la revisión del código fuente de un sistema informático.

El trabajo propone un procedimiento para realizar pruebas de Caja Blanca a los sistemas que se desarrollan en Python. En el mismo se exponen las actividades a seguir por el Grupo de Calidad de la Facultad Regional Granma, reflejando cada uno de los artefactos de entrada y salida que se generan, indicando cómo se utilizan y se completan.

Para confirmar la validez del trabajo realizado se aplicó el procedimiento al Sistema de Gestión de Información para la Empresa de Acueducto y Alcantarillado de Granma. De acuerdo a lo planteado en la propuesta se realizaron sus actividades y se evidenciaron los resultados en cada uno de los artefactos involucrados.

Palabras clave: *Artefactos, Centro de Desarrollo, Código Fuente, Grupo de Calidad, Procedimiento, Pruebas de Caja Blanca.*

Abstract

Software quality is one of the bigest issues in Informatics. The process of software testing (also known as beta testing) is a fundamental aspect to measure the quality state of an informatics system so as to successfully introduce it into the market. The objective of this research is to present a proposal of procedure to apply tests as the beta testing (White Box) method to software developed in Python language at the University of Informatics Sciences' Development Center from Granma Regional Faculty.

To conduct this research, an analysis of the main specialized publications on the topic was carried out to deepen on the different testing methods available, mainly about the techniques to check the source code of software.

This research proposes a procedure to conduct software testing to the systems developed using Python language. The main activities to follow by the Quality Group from the Granma Regional Faculty show the entry and exit artifacts generated as well as indicate how they are used and complemented.

The procedure was applied to the Information Management System of the Aqueduct Enterprise in Granma so as to confirm the validity of the proposal. According to what it is stated in the proposal, all the activities were carried out and the results were evidenced in each of the artifacts involved.

1 Eduardo Salazar Martínez: Ingeniero en Ciencias Informáticas. Profesor. Facultad Regional Granma de la Universidad de las Ciencias Informáticas ‡, esalazar@grm.uci.cu

1. Estado del Arte

Con el crecimiento acelerado de las tecnologías y la informática, la producción de software desempeña un papel importante, provocando a su vez una competencia en los sistemas, donde la calidad es fundamental para conseguir rentabilidad en la producción. La necesidad de realizar pruebas de calidad converge hacia el aseguramiento de la eficiencia del producto antes de salir al mercado.

Muchas empresas existentes dedicadas a la producción de software y gestión de la calidad, gozan de un alto prestigio y cuentan con sus propias estrategias de pruebas y herramientas de apoyo. Otras contratan a empresas e instituciones que se especializan en este proceso, entre ellas podemos mencionar a **GreenSQA** (Green Software Quality Assurance), con la misión de contribuir a la madurez de las empresas e industrias mediante el uso de servicios específicos de pruebas de software e implementación de sistemas de gestión de calidad, garantizando así procesos ágiles, confiables y eficientes.

Otro ejemplo es la empresa **SQS S.A** (Software Quality System S.A), que lleva acabo procesos de pruebas tanto en sus propias instalaciones como en las de sus clientes, asegurando la reducción de costes y el aumento de la calidad. Para ofrecer un mejor servicio a sus clientes, la compañía ha decidido organizar estos servicios de pruebas llevados a cabo en sus instalaciones bajo el nombre de **SOQUS, el Laboratorio de Testing de SQS.**

En este caso la **RCCS** (Red Colombiana de Calidad de Software), es un instrumento de gestión de conocimiento fundamentado en un modelo de ingeniería, que tiene como objetivo gestionar programas de apoyo a la implementación de modelos de software para fortalecer la industria nacional.

También al incremento de las tecnologías se vincula el desarrollo de aplicaciones, y en este caso nos interesan a aquellas que son creadas con la meta de poder automatizar el proceso de pruebas de código que se realizan sobre el software. Dentro del almacén de programas con este objetivo podemos encontrar las que están relacionadas con las pruebas de Caja Blanca. A continuación se mencionan algunas de estas herramientas.

JTest: Es el primer sistema automático de búsqueda de errores en el código de programación para programadores de Java. Esta nueva tecnología desarrollada por la empresa ***ParaSoft***, utiliza la Test Generation Technolgy (Tecnologías de Generación de Pruebas) para analizar programas en Java. Se trata de una herramienta que permite realizar análisis de código, pruebas unitarias automáticas y cobertura de código, así como generación dinámica de pruebas funcionales.

En el ámbito de los análisis dinámicos, **JTest** es capaz de generar

automáticamente todas las pruebas unitarias que sean necesarias, teniendo en cuenta los parámetros de cobertura de código e intentando encontrar pruebas que deriven en errores de ejecución. Genera pruebas funcionales filtradas por las acciones y los datos, incluyendo peticiones *HTTP2* y *JDBC3*.

Insure++: Es un entorno automatizado en una aplicación de herramientas de prueba C/C++ que detecta errores difíciles de localizar; como corrupción de la memoria, asignación de errores de memoria, errores de iniciación de variables, definición de conflictos entre variables, indicador de errores, errores de biblioteca, errores lógicos y errores en algoritmos. Sin embargo tiene licencia la cual hay que pagar un monto significativo cada cierto período de tiempo.

BullseyeCoverage: 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. Sin embargo la licencia se debe comprar cada cierto período de tiempo, según el cliente determine, oscilando de 500 euros a 1000 euros.

LDRA: Es una herramienta de control de calidad que ofrece un potente código fuente de pruebas y análisis para la validación y verificación de aplicaciones de software. Es importante cuando el software informático requiere ser fiable, robusto y libre de errores. Se trata de un potente y plenamente integrado suite de herramientas, que permite al software avanzado de técnicas de análisis que puedan aplicarse en las etapas claves del desarrollo del ciclo de vida. Sin embargo sus herramientas no contienen la comprobación de estándares de codificación y para su empleo hay que comprarla a precios estimados.

Logiscope TestChecker: Aplicación para la representación gráfica de cobertura del código fuente. Evalúa el nivel de cobertura del código, el usuario debe de comprar la licencia por período de tiempos, no realiza evaluaciones al código de C Sharp y no cuenta con la verificación de estándares.

CMT++: Es una herramienta para la medida de la complejidad para C/C ++, la misma es fácil de utilizar para ambos lenguajes. También el código en ensamblador se puede medir con esta herramienta. CMT++ está destinado para organizaciones desarrolladoras de software que se esfuerzan por un proceso de desarrollo productivo resultante en productos de alta calidad.

Ayuda a estimar el mantenimiento general del código base y a localizar fácilmente las partes complejas de este. Se pueden evaluar por separado: poniendo especial atención a las pruebas, o tal vez rediseñándolas. Se puede utilizar CMT++ también para medir la cantidad de código

que se tiene: líneas físicas, líneas de comentarios, líneas de programa, declaraciones. Es una herramienta que a pesar de sus características no es gratuita y la documentación para su aprendizaje se encuentra en otros idiomas exceptuando el castellano.

CTC++: Con esta herramienta ocurre lo mismos inconvenientes que con CMT++, aunque no se debe de dejar de mencionar que Testwell CTC++ (Test Analizador de Cobertura para C y C++) es una herramienta de cobertura código/prueba potente y fácil de usar, la cual muestra las partes del código que han sido ejecutadas (probadas). La herramienta analiza todos los niveles de cobertura requeridos en proyectos "críticos" y ayuda a garantizar una mayor calidad del código. Testwell CTC++ puede utilizarse para obtener las certificaciones en la industria automotriz, aérea y médica.

Visto el estudio y análisis en el marco internacional de las empresas desarrolladoras de software y gestión de la calidad, y ejemplos de aplicaciones existentes que se suelen emplear para garantizar la mayor calidad posible de los sistemas sometiéndolos a procesos de pruebas, la idea de alcanzar una empresa o sistema que asegure la calidad en el desarrollo de software se ha venido fortaleciendo en Cuba de forma continua. Existen empresas en el territorio que se inclinan hacia el mismo objetivo, como **CITMATEL** (Empresa de Tecnologías de la Información y Servicios Telemáticos Avanzados) que se distingue entre las entidades cubanas por su excelencia y creciente proyección hacia el mercado externo e interno, con una amplia diversidad de productos y servicios integrales de alto valor agregado. Entre las principales líneas de trabajo se destaca el desarrollo de sistemas dirigidos a automatizar la gestión en empresas de todo tipo.

Otro ejemplo es **ALBET** (Albet Ingeniería y Sistemas), cuyo origen y desarrollo se vincula estrechamente a la Universidad de Ciencias Informáticas (UCI), modelo de universidad productiva que agrupa una multitud de profesionales, técnicos y estudiantes. No podemos dejar de mencionar que la UCI cuenta también con un Centro para la Certificación de la Calidad de Software CALISOFT, unido al Departamento de Producción de Software. El centro cuenta con trabajos de diploma investigativos que se han realizado referentes a los temas sobre procedimientos generales de pruebas de Caja Blanca y procesos de pruebas referidos al método de Caja Negra. Existen en algunos proyectos productivos en la UCI, la utilización de herramientas que aplican diferentes pruebas de Caja Negra, probando la funcionalidad del software y en la minoría (prácticamente ninguno) se aplica el método de Caja Blanca de forma manual a un pequeño fragmento de código. El Grupo de Calidad de la Facultad Regional de la UCI en Granma, también participa en la realización de pruebas de software, aplicando generalmente pruebas de Caja Negra y de Carga. Por consiguiente existe la carencia de procedimientos de pruebas de Caja Blanca o empleo de aplicaciones de apoyo que permitan la automatización de estos proceso en la universidad.

2. Calidad de Software

La calidad de software es un problema actual que afecta tanto a los productores de software como a los clientes. Con el aumento de la informatización a escala mundial la demanda de software crece exponencialmente y los desarrolladores le han brindado poco interés a la calidad de sus productos. Sucede que muchas veces los clientes reciben el software cuando se han violado las etapas de pruebas.

La calidad del software puede definirse de muchas maneras. Una de las más limitadas, conocida como "calidad pequeña", define la calidad como la ausencia de defectos [4]. Para evaluarla de esta forma se emplean procedimientos estadísticos a partir de las tendencias de aparición de fallas durante la prueba de software.

"Existen estándares industriales que marcan aceptabilidad cuando se estima el número de defectos residuales en 0.02 defectos por millar de líneas de código y aún menos." [5]

"Otros enfoques de calidad consideran diversos factores, entre ellos la confiabilidad" [6]. Existe una larga tradición de estudio de la confiabilidad que se asocia estadísticamente con el comportamiento del software.

Existen varias formas de definir la confiabilidad, en unos casos se considera tiempo de operación y en otros la variedad de usos propuestos. Una definición más reciente, plantea que: "La confiabilidad es la probabilidad de operación exitosa de un programa dado, en un intervalo de tiempo, en un ambiente específico". [6]

Obteniendo la calidad requerida en el software, se logra reducir su número de errores o eliminarlos completamente, se alcanza una mayor fiabilidad para las funciones que debe realizar el mismo, mayor eficiencia e integridad de los datos así como flexibilidad y reusabilidad.

"La calidad de software es una actividad de protección que se aplica a lo largo de todo el proceso de Ingeniería del Software. Esta engloba los siguientes aspectos:" [6]

- Un enfoque de gestión de calidad.
- Tecnología de Ingeniería del Software efectiva (métodos y herramientas).
- Revisiones técnicas formales que se aplican durante el proceso del software.
- Una estrategia de prueba multiescala.

- El control de la documentación del software y de los cambios realizados.
- Un procedimiento que asegure un ajuste a los estándares de desarrollo del software.
- Mecanismos de medición y de generación de informes.

"La calidad debe ser especificada, planificada, administrada, medida y certificada. Esto implica una visión integral que arroja la comprobación del software, con el fin de lograr un mayor grado de satisfacción y confianza del cliente hacia la organización productora de software. Constituye entonces las pruebas de los software, tarea de alta prioridad para las empresas productoras". [2].

Analizando los concepto expuesto sobre la calidad por varios autores y haciendo una inclinación por este último expresado por Pressman gracias a que se considera como uno de los más completos y acordes al objeto de estudio, también cabe decir que la calidad es considerada una disciplina integral y a la vez una cualidad indisoluble del software para su comprobación, muy ligada a las empresas productoras como tarea fundamental en sus procesos de pruebas.

3. Pruebas de Software

Unas de las vías más importantes para determinar el estado de la calidad de un producto de software es el proceso de pruebas. Estas están dirigidas a componentes del sistema en su totalidad, con el objetivo de medir el grado en que cumple con los requerimientos. En ellas se usan casos de prueba, especificados de forma estructurada mediante técnicas. Sus objetivos, métodos y técnicas usadas se describen en el plan de prueba.

La prueba es una actividad fundamental en muchos procesos de desarrollo, incluyendo el del software. Estas permiten detectar la presencia de errores que pudieran generar las entradas o salidas de datos y comportamientos inapropiados durante su ejecución. Un concepto más específico dado por algunos desarrolladores de software es:

"Cualquier intento de demostrar que el software tiene propiedades por debajo de la calidad requerida". [7].

De acuerdo a la IEEE [8] el concepto de prueba se define 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". [9].

Otro concepto importante a tomar en consideración es el emitido por Pressman en su edición de 1998, que plantea lo siguiente:

"La prueba del software es un elemento crítico para la garantía de calidad del software y representa una revisión de las especificaciones, del diseño y de la codificación". [9].

Teniendo en cuenta las definiciones anteriores se puede concluir que la prueba de software es una actividad en la cual el sistema es ejecutado bajo condiciones específicas para demostrar que no tiene la madurez necesaria para ser implantado. Dentro de las actividades que se practican para obtener un software con la madurez necesaria están:

- Revisiones: consiste en que cada integrante del equipo de desarrollo revisa el producto que va generando.
- Inspecciones: revisión de cada producto por parte de colegas.
- Validaciones: es el cliente quien revisa el producto para decir si cumple con sus necesidades.

Esta definición implica que se considera una prueba exitosa si se demuestran deficiencias en el software. Las fallas pueden ser en el código o en el modelado, en dependencia del tipo de pruebas que se le apliquen al software.

Se distinguen pruebas técnicas y pruebas funcionales. Las pruebas técnicas son la responsabilidad de los ingenieros de software que han desarrollado el producto, pero estos ingenieros en ocasiones deben hacerse cargo de las pruebas funcionales.

En proyectos a gran escala las pruebas funcionales son la responsabilidad de un equipo de pruebas, formado por uno o varios técnicos, un coordinador de pruebas y un gestor de pruebas o de calidad.

3.1. Características generales de la Estrategia de Prueba

Al aplicarles las pruebas al software se deben seguir un conjunto de estrategias para lograr que estas se hagan en el menor tiempo posible y con la calidad requerida, además de garantizar que arrojen los resultados esperados.

Dentro de las características generales de la estrategia de prueba se encuentran. [2]

1. La prueba comienza en el nivel de módulo y trabaja "hacia fuera", hacia la integración completa del sistema completo.

2. En diferentes puntos es adecuada la utilización de técnicas de prueba distintas.

3. La prueba la lleva a cabo el que desarrolla el software y para grandes proyectos, un grupo de prueba independiente.

4. La prueba y la depuración son actividades diferentes, pero la depuración puede entrar en cualquier estrategia de prueba.

Hay dos estrategias generales para la prueba de software: las estrategias de pruebas de especificación (Caja Negra) y pruebas de código (Caja Blanca).

4. Métodos de Pruebas

Existen diversos métodos para realizar las pruebas de software, entre las más importantes se encuentran la prueba de Caja Blanca, prueba de Caja Negra y prueba de la Estructura de Control.

El uso de la prueba de Caja Blanca es mejor para verificar que se recorran todos los caminos y detectar un mayor número de errores. La Caja Negra brinda la posibilidad de cubrir la mayor parte de las combinaciones de entradas y lograr así un juego de pruebas más eficaz.

Las pruebas mencionadas permiten probar cada una de las condiciones existentes en el programa, identificar claramente las entradas, salidas y estudiar las relaciones que existen entre ellas, permitiendo así maximizar la calidad de las pruebas y en dependencia del resultado se constará con un sistema más estable y confiable.

4.1. Prueba de Especi cación (Caja Negra)

Pruebas de Caja Negra: También suelen ser llamadas funcionales y basadas en especificaciones. En ellas se pretende examinar el programa en busca de que cuente con las funcionalidades que debe tener y como lleva a cabo las mismas, analizando siempre los resultados que devuelve y probando todas las entradas en sus valores válidos e inválidos.

Al ejecutar las pruebas de Caja Negra se desarrollan casos de prueba reales para cada condición o combinación de condiciones y se analizan los resultados que arroja el sistema para cada uno de los casos. En esta estrategia se verifica el programa considerándolo una caja negra. Las pruebas no se hacen en base al código, sino a la interfaz. No importa que se cubran todas las rutas dentro del programa, lo importante es probar todas las entradas en sus valores válidos e inválidos y lograr que el sistema tenga una interfaz amigable.

4.1.1. Limitaciones

Lograr una buena cobertura con pruebas de caja negra es un objetivo deseable; pero no suficiente a todos los efectos. Un programa puede pasar con holgura millones de pruebas de especificación y sin embargo tener defectos internos que surgen en el momento más inoportuno.

Por ejemplo, una computadora que contenga el virus Viernes-13 puede estar pasando pruebas de caja negra durante años y años. Sólo falla si es viernes y es día 13; pero ¿a quién se le iba a ocurrir hacer esa prueba? [11]

Las pruebas de caja negra nos convencen de que un programa realizar bien sus funcionalidades programadas, pero no de que haga (además) otras cosas menos aceptables.

4.2. Prueba de Código (Caja Blanca)

Pruebas de Caja Blanca: También suelen ser llamadas estructurales o de cobertura lógica. En ellas se pretende investigar sobre la estructura interna del código, exceptuando detalles referidos a datos de entrada o salida, para probar la lógica del programa desde el punto de vista algorítmico. Realizan un seguimiento del código fuente según se va ejecutando los casos de prueba, determinándose de manera concreta las instrucciones, bloques, etc. que han sido ejecutados por los casos de prueba.

En las pruebas de Caja Blanca se desarrollan casos de prueba que produzcan la ejecución de cada posible ruta del programa o módulo, considerándose una ruta como una combinación específica de condiciones manejadas por un programa.

Hay que señalar que no todos los errores de software se pueden descubrir verificando todas las rutas de un programa, hay errores que se descubren al integrar unidades del sistema y pueden existir errores que no tengan relación con el código específicamente.

4.2.1. Características de las pruebas de Caja Blanca

En las pruebas de Caja Blanca, se pretende indagar sobre la estructura interna del código, omitiendo detalles referidos a datos de entrada o salida. Su objetivo principal es probar la lógica del programa desde el punto de vista algorítmico.

Estas se basan en el diseño de Casos de Prueba que usa la estructura de control del diseño procedimental para derivarlos. Mediante las pruebas de Caja Blanca el ingeniero de software puede obtener Casos de Prueba que: [11]

- Garanticen que se ejerciten por lo menos una vez todos los caminos independientes de cada módulo, programa o método.
- Ejerciten todas las decisiones lógicas en las vertientes verdadera y falsa.
- Ejecuten todos los bucles en sus límites operacionales.
- Ejerciten las estructuras internas de datos para asegurar su validez.

Las pruebas de Caja Blanca son consideradas entre las más importantes que se aplican a los sistemas, con la que se obtienen como resultados la disminución en un gran porciento el número de errores existentes en el software y por ende una mayor calidad y confiabilidad en la codificación.

4.2.2. Tipos de pruebas de Caja Blanca

De estructura de datos locales:

Se centran en el estudio de las variables del programa. Busca que toda variable esté declarada y que no existan con el mismo nombre, ni declaradas local y globalmente, que haya referencias a todas las variables y para cada variable, analiza su comportamiento en comparaciones.

De cobertura lógica:

- De Cobertura de Sentencias: Comprueba que todas las sentencias se ejecuten al menos una vez.
- De Cobertura de Decisión: Ejecuta casos de prueba de modo que cada decisión se pruebe al menos una vez a Verdadero (True) y otra a Falso (False).
- De Cobertura de Condición: Ejecuta un caso de prueba a True y otro a False por cada condición, teniendo en cuenta que una decisión puede estar formada por varias condiciones.
- De Cobertura de Condición/Decisión: Se realizan las pruebas de cobertura de condición y las de decisión a la vez.
- De Condición Múltiple: Cada decisión multicondición se traduce a una condición simple, aplicando posteriormente la cobertura de decisión.
- De Cobertura de Caminos: Se escriben casos de prueba suficientes para que se ejecuten todos los caminos de un programa. Entendiéndose camino como una secuencia de sentencias encadenadas desde la entrada del programa hasta su salida. [11]

4.2.3. Prueba del Camino Básico

Buscando una mejor comprensión de los contenidos, se hace importante definir primeramente algunos conceptos fundamentales:

Camino: "Secuencia de todas las instrucciones de un programa de principio a fin". [2] Un camino se puede definir como la ruta de secuencias que se siguen dentro del código de fuente de un programa, un ejemplo es, desde la entrada de valores al sistema hasta la devolución de resultados que arroja, respetando la estructura de código.

Camino Básico: Es una técnica de prueba de Caja Blanca que permite obtener una medida de complejidad lógica para generar un conjunto básico de caminos que se ejecutan por lo menos una vez durante la ejecución del programa. [12]

La prueba del camino básico es una técnica de pruebas de Caja Blanca propuesta por Tom MacCabe. Esta técnica permite obtener una medida de la complejidad lógica de un diseño y usar esta medida como guía para la definición de un conjunto básico. La idea es derivar casos de prueba a partir de un conjunto dado de caminos independientes por los cuales puede circular el flujo de control.

Camino independiente: "Es cualquier camino del programa que incluye nuevas instrucciones de un proceso o una nueva condición". [12]

El conjunto de caminos independientes se obtiene construyendo el Grafo de Flujo asociado y se calcula su complejidad ciclomática. Por último se diseñan los casos de prueba y se ejecutan los mismos.

Complejidad: Es proporcional al número de errores en un segmento de código. "Entre más complejo, más susceptible a errores". [13] Se relaciona con el esfuerzo requerido para probar. "Entre más complejo, mayor atención para probar". [13]

Complejidad ciclomática: Es la "medida de la complejidad lógica de un módulo "G" y el esfuerzo mínimo necesario para calificarlo. Es el número de rutas lineales independientes de un módulo "G", por lo tanto es el número mínimo de rutas que deben probarse". [13]

Esta técnica ofrece una gran ventaja con respecto a las otras técnicas, ya que el número mínimo requerido de pruebas se sabe por adelantado y por tanto el proceso de prueba se puede planear y supervisar en mayor detalle.

Los pasos a seguir para aplicar esta técnica son:

1) Representar el programa en un grafo de flujo.

2) Calcular la complejidad ciclomática.

3) Determinar el conjunto básico de caminos independientes.

4) Derivar los casos de prueba que fuerzan la ejecución de cada camino.

A continuación, se detallan cada uno de estos pasos.

1) Representación de un grafo de flujo:

El grafo de flujo se utiliza para representar el flujo de control lógico de un programa. Este emplea los tres elementos siguientes:

- Nodos: Representan cero, una o varias sentencias en secuencia. Cada uno comprende como máximo una sentencia de decisión (bifurcación).

- Aristas: Líneas que unen dos nodos.

- Regiones: Áreas delimitadas por aristas y nodos. Cuando se contabilizan las regiones de un programa debe incluirse el área externa como una región más.

Así, cada construcción lógica de un programa tiene una representación. La Figura 1 muestra un grafo de flujo del diagrama de módulos correspondiente. Nótese cómo la estructura principal corresponde a un while y dentro del bucle se encuentran anidados dos constructores if. [14]

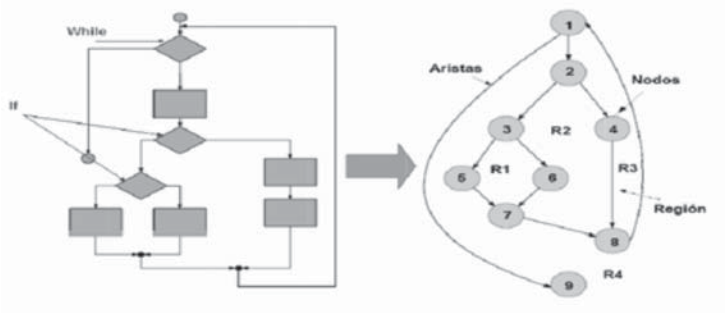

Figura 1. Ejemplo de grafo de flujo correspondiente a un diagrama de módulos

2) Calcular la complejidad ciclomática:

La complejidad ciclomática es una métrica del software que proporciona una medida cuantitativa de la complejidad lógica de un programa. Se basa en la representación gráfica del flujo de control del programa, el análisis desprende una medida cuantitativa de la dificultad de prueba y

una indicación de la fiabilidad final. Cuando se utiliza en el contexto del método de prueba del camino básico, el valor calculado como complejidad ciclomática define el número de pruebas que se deben realizar para asegurar que se ejecute cada sentencia al menos una vez.

Métrica: "Medida estadística que se aplica a todos los aspectos de calidad de software, los cuales deben ser medidos desde diferentes puntos de vista". [13] La métrica se define como término para valorar en una medida la calidad de software considerando diferentes puntos necesario a evaluar.

Es una de las métricas de software más ampliamente aceptada, ya que ha sido creada para ser independiente del lenguaje. Se han realizado investigaciones y ensayos, en los cuales se ha medido un gran número de programas, a modo de establecer rangos de complejidad que ayuden al ingeniero de software a determinar la estabilidad y riesgo inherente de un programa. La medida resultante puede ser utilizada en el desarrollo, mantenimiento y reingeniería para estimar el riesgo, costo y estabilidad.

Estos estudios indicaron la existencia de distintas relaciones entre la métrica de McCabe y el número de errores existentes en el código fuente, así como el tiempo requerido para encontrar y corregir esos errores. "Se puede comparar la complejidad ciclomática contra un conjunto de valores límites como se observa en la Tabla 1". [13].

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	Programa no testeable, muy alto riesgo.

Tabla 1. Complejidad ciclomática vs Evaluación de riesgo

El ámbito más común de utilización para probar módulos unitarios es la prueba estructural (Caja Blanca). McCabe también expone que se puede utilizar la complejidad ciclomática para dar una indicación cuantitativa del tamaño máximo de un módulo. A partir del análisis de muchos proyectos se encontró que un valor 10 es un límite superior práctico para el tamaño de un módulo.

Cuando la complejidad supera este valor se hace muy difícil probarlo, entenderlo y modificarlo. La limitación deliberada de la complejidad en todas las fases del desarrollo ayuda a evitar los problemas asociados a proyectos de alta complejidad. El límite propuesto por McCabe sin embargo es fuente de polémicas. Algunas organizaciones han utilizado el valor 15 con bastante éxito. No obstante, un valor superior a 10

debería ser reservado para proyectos que tengan ventajas operacionales con respecto a proyectos típicos.

La complejidad ciclomática puede ser aplicada en varias áreas incluyendo: [15]

- Análisis de Riesgo en desarrollo de código: Mientras el código está en desarrollo, su complejidad puede ser medida para estimar el riesgo inherente.
- Análisis de riesgo de cambio durante la fase de mantenimiento: La complejidad del código tiende a incrementarse a medida que es mantenido durante el tiempo. Midiendo la complejidad antes y después de un cambio propuesto, puede ayudar a decidir cómo minimizar el riesgo del cambio.
- Planificación de Pruebas: El análisis matemático ha demostrado que la complejidad ciclomática indica el número exacto de casos de prueba necesarios para probar cada punto de decisión en un programa.
- Reingeniería: Provee conocimiento de la estructura del código operacional de un sistema. El riesgo involucrado en la reingeniería de una pieza de código está relacionado con su complejidad.

¿Por qué es tan importante?

Permite apreciar la calidad del diseño de software de una manera rápida y con independencia del tamaño de la aplicación. Es una medida esencial cuando se necesita "tomar la temperatura" de un diseño de software de un tamaño considerable (más si se está realizando una auditoría externa y no se conocía de antes la aplicación), y que se puede obtener fácilmente de manera automatizada. También se ha utilizado para planificar proyectos de mejora de grandes productos de software, para priorizar las partes del diseño.

Por otro lado, en muchas ocasiones, es base para calcular el valor de las mejoras del diseño, o el valor que aporta introducir un patrón o buena práctica. Además, da el número de casos de prueba unitarios básicos para obtener una cobertura del 100 porciento, y un aproximado al grado de comprensibilidad.

¿Cómo sería su cálculo?

"Existen varias formas de calcular la complejidad ciclomática (CC) de un programa a partir de un grafo de flujo: [14]

- CC = arcos – nodos + 2
- CC = número de nodos de decisión + 1
- CC = número de regiones".

Esta complejidad ciclomática determina el número de casos de prueba

que deben ejecutarse para garantizar que todas las sentencias de un programa se han ejecutado al menos una vez, y que cada condición se habrá ejecutado en sus vertientes verdadera y falsa. A continuación se expone cómo se identifican estos caminos.

3) Determinar el conjunto de caminos básicos independientes:

Camino linealmente independiente de otros: Introduce por lo menos un nuevo conjunto de sentencias de proceso o una nueva condición.

Un camino independiente es cualquier camino del programa que introduce, por lo menos, un nuevo conjunto de sentencias de proceso o una condición, respecto a los caminos existentes. En términos del diagrama de flujo, un camino independiente está constituido por lo menos por una arista que no haya sido recorrida anteriormente a la definición del camino. En la identificación de los distintos caminos de un programa para probar se debe tener en cuenta que cada nuevo camino debe tener el mínimo número de sentencias nuevas o condiciones nuevas respecto a los que ya existen. De esta manera se intenta que el proceso de depuración sea más sencillo.

El conjunto de caminos independientes de un grafo no es único. No obstante, a continuación, se muestran algunas heurísticas para identificar estos caminos: [14]

- Elegir un camino principal que represente una función válida que no sea un tratamiento de error. Debe intentar elegirse el camino que atraviese el máximo número de decisiones en el grafo.

- Identificar el segundo camino mediante la localización de la primera decisión en el camino de la línea básica alternando su resultado mientras se mantiene el máximo de número de decisiones originales del camino inicial.

- Identificar un tercer camino, colocando la primera decisión en su valor original a la vez que se altera la segunda decisión del camino básico, mientras se intenta mantener el resto de decisiones originales.

- Continuar el proceso hasta haber conseguido tratar todas las decisiones, intentando mantener como en su origen el resto de ellas.

Heurística: La búsqueda a ciegas es aquella donde no existe ninguna informaci6n para decidir que nodo expandir, no se conoce la cantidad de pasos o el costo del camino desde el estado actual hasta el objetivo. También se denomina búsqueda no informada. En el otro caso, cuando existe informaci6n para decidir, la búsqueda se denomina informada o heurística. [16]

Este método permite obtener de la complejidad ciclomática, caminos

independientes cubriendo el criterio de cobertura de decisión y sentencia.

Por ejemplo, para la el grafo de la Figura 1 los cuatro posibles caminos independientes generados serían:

Camino 1: 1-9
Camino 2: 1-2-4-8-1-9
Camino 3: 1-2-3-5-7-8-1-9
Camino 4: 1-2-3-6-7-8-1-9

4) Derivar los casos de prueba:

El último paso es construir los casos de prueba que fuerzan la ejecución de cada camino. Una forma de representar el conjunto de casos de prueba sería llenar la "Tabla 1.2". [14] que se muestra a continuación.

Número del Camino	Caso de Prueba	Resultado Esperado

Tabla 1.2. Posible representación de casos de prueba para pruebas estructurales.

5. Plan de Pruebas.

"El propósito del plan de pruebas es dejar de forma explícita el alcance, el enfoque, los recursos requeridos, el calendario, los responsables y el manejo de riesgos de un proceso de pruebas". [17]

Está constituido por un conjunto de pruebas. Cada prueba debe:

- Dejar claro qué tipo de propiedades se quieren probar (corrección, robustez, fiabilidad, amigabilidad, etc.).
- Dejar claro cómo se mide el resultado.
- Especificar en qué consiste la prueba (hasta el último detalle de cómo se ejecuta).
- Definir cuál es el resultado que se espera (identificación, tolerancia,...). ¿Cómo se decide que el resultado es acorde con lo esperado?

Las pruebas carecen de utilidad, tanto, sí no se sabe exactamente lo que

se quiere probar, sí no se está claro cómo se prueba, o si el análisis del resultado se hace a simple vista.

Estas mismas ideas se suelen agrupar diciendo que un caso de prueba consta de tres bloques de información:

1. El propósito de la prueba.

2. Los pasos de ejecución de la prueba.

3. El resultado que se espera.

Todos y cada uno de esos puntos deben quedar perfectamente documentados. El plan de pruebas señala el enfoque, los recursos y el esquema de actividades de prueba, así como los elementos a probar, las características, las actividades de prueba, el personal responsable y los riesgos.

6. El Proceso de Pruebas

El proceso de pruebas de un software consta de varias etapas dentro de ellas las más importantes son:

1. Inspección del análisis: Se verifica si se cometieron errores o falla en la etapa de análisis.

2. Inspección del diseño: Se comprueba el diseño y se trata de hallarle defectos.

3. Inspección del código: Se observa el entendimiento y facilidad del código.

4. Pruebas unitarias: Se debe probar cada método de las clases implementadas por separado.

5. Pruebas de integración: Se prueban todas las clases, verificando que compaginen entre sí.

6. Pruebas de validación de requerimientos: Validan que se cumple con todos los requerimientos exigidos por el cliente.

7. Pruebas de sistema: Ejecutar el programa para verificar si cumple con los requisitos exigidos.

6.1. Pruebas de Unidad

Se comprueban los módulos cada uno por separado, buscando errores en el funcionamiento de ese módulo como sistema independiente. Estas pruebas deben ser hechas por el diseñador y el programador del módulo.

6.2. Pruebas de Sistema

Su objetivo es la comprobación del sistema global, se realizan pruebas de tres tipos distintos:

- Seguridad: protección en aplicaciones especialmente sensibles a entradas no deseadas.
- Resistencia: se prueba la robustez del sistema frente al mal uso de la aplicación por parte de ciertos usuarios.
- Rendimiento: Eficiencia medida en velocidad de proceso y recursos consumidos.

6.3. Pruebas de Integridad

Las pruebas de integración se llevan a cabo durante la construcción del sistema, involucran a un número creciente de módulos y terminan probando el sistema como conjunto. Estas pruebas se pueden plantear desde un punto de vista estructural o funcional.

Las pruebas estructurales de integración son similares a las pruebas de caja blanca; pero trabajan a un nivel conceptual superior. En lugar de referirse a sentencias del lenguaje, se refiere a llamadas entre módulos. Se trata de identificar todos los posibles esquemas de llamadas y ejercitarlos para lograr una buena cobertura de segmentos o de ramas.

Las pruebas funcionales de integración son similares a las pruebas de caja negra. Aquí trataremos de encontrar fallos en la respuesta de un módulo cuando su operación depende de los servicios prestados por otro(s) módulo(s). Según nos vamos acercando al sistema total, estas pruebas se van basando más y más en la especificación de requisitos del usuario.

Las pruebas finales de integración cubren todo el sistema y pretenden cubrir plenamente la especificación de requisitos del usuario. Además, a estas alturas ya suele estar disponible el manual de usuario, que también se utiliza para realizar pruebas hasta lograr una cobertura aceptable.

6.4. Pruebas de Aceptación

El uso de cualquier producto de software tiene que estar justificado por las ventajas que ofrece. Sin embargo, antes de su puesta en marcha es muy difícil determinar si sus ventajas realmente justifican su uso. El mejor instrumento para esta determinación es la llamada "prueba de aceptación". En esta prueba se evalúa el grado de calidad del software con relación a todos los aspectos relevantes para que el uso del producto se justifique.

Eliminar la influencia de conflictos de intereses, y para que sea lo más objetiva posible, la prueba de aceptación no debería ser responsabilidad de los ingenieros de software que han desarrollado el producto.

En la preparación, ejecución y evaluación de las pruebas de aceptación no se requiere de conocimientos informáticos. Sin embargo, un conocimiento amplio de métodos y técnicas de prueba y de la gestión de la calidad en general facilita esta labor.

La persona adecuada (o el equipo adecuado) para llevar a cabo la prueba de aceptación disponen de estos conocimientos y además son capaces de interpretar los requerimientos especificados por los futuros usuarios del sistema de software en cuestión.

Estas pruebas las realiza el cliente. Son básicamente pruebas funcionales sobre el sistema completo, y buscan una cobertura de la especificación de requisitos y del manual del usuario. Estas pruebas no se realizan durante el desarrollo, pues sería impresentable al cliente; sino que se realizan sobre el producto terminado e integrado o pudiera ser una versión del producto o una iteración funcional pactada previamente con el cliente.

La experiencia muestra que aún después del más cuidadoso proceso de pruebas por parte del desarrollador, quedan una serie de errores que sólo aparecen cuando el cliente comienza a usarlo.

Sea como sea, el cliente siempre tiene razón. Decir que los requisitos no estaban claros, o que el manual es ambiguo puede salvar la cara; pero ciertamente no deja satisfecho al cliente.

Una prueba de aceptación puede ir desde un informal caso de prueba hasta la ejecución sistemática de una serie de pruebas bien planificadas. De hecho, las pruebas de aceptación pueden tener lugar a lo largo de semanas o meses, descubriendo así errores latentes o escondidos que pueden ir degradando el funcionamiento del sistema. Estas pruebas son muy importantes, ya que definen las nuevas fases del proyecto como el despliegue y mantenimiento.

Se emplean dos técnicas para las pruebas de aceptación:

• La prueba alfa.

Se lleva a cabo, por un cliente, en el lugar de desarrollo y en un entorno controlado. Se usa el software de forma natural con el desarrollador como observador del usuario. Para que tengan validez, se debe primero crear un ambiente con las mismas condiciones que se encontrarán en las instalaciones del cliente. Una vez logrado esto, se procede a realizar las pruebas y a documentar los resultados. [3]

• La prueba beta.

Se lleva a cabo por los usuarios finales del software y se realiza en el entorno de los clientes. A diferencia de la prueba alfa, el desarrollador no está presente normalmente. Así, la prueba beta es una evaluación "en tiempo real" del software en un entorno no planificado por el desarrollador. El cliente registra todos los problemas (reales o imaginarios) que encuentra durante la prueba beta e informa a intervalos regulares al desarrollador.

Como resultado de los problemas informados durante la prueba beta, el desarrollador del software lleva a cabo modificaciones y así prepara una versión del producto de software para todos los cliente donde se despliegue el producto. [3]

7. Flujo Actual de los Procesos

El proceso de control de calidad antes, durante y después de la implementación de un producto de software es una tarea bastante complicada incluso para los expertos en el tema, ya que nunca se tiene la última palabra a cerca de estos factores y cada situación que se presenta puede resultar novedosa y problemática simultáneamente.

En la Facultad Regional de la Universidad de las Ciencias Informáticas en Granma (FRG-UCI) se producen sistemas destinados al Sector de Cultura, en su mayoría con dimensiones grandes y compuestos por diferentes líneas de trabajo como los Sistemas de Gestión de Información, Realidad Virtual y Desarrollo de Aplicaciones para dispositivos móviles.

El proceso de pruebas en estos sistemas se realiza mensualmente por el Grupo de Calidad (GC) de la FRG-UCI, y una vez estén terminados los productos, el personal implementador también forma parte de este proceso. Las pruebas que se realizan con mayor frecuencia son las Pruebas de Estrés, Pruebas de Funcionalidades y a la Documentación por lo general. El jefe del equipo de prueba recibe todo el sistema y la documentación a probar, por medios de la herramienta de control

de versiones Bazaar, y a medidas que se realizan las revisiones, manualmente se van conformando los registros de no conformidades guiado por una plantilla, estos últimos los recibe el líder de proyecto para hacer cambios a favor de superar la calidad de la aplicación y nuevamente someterla a otra revisión, y así se repite el proceso hasta que el producto quede con el mínimo número de errores posible.

Las pruebas de Caja Blanca todavía no se comprenden dentro de las que se practican el GC a los productos en desarrollo. Cada proyecto revisa el código fuente usando la técnica más conveniente o que mejor conozca, y los pocos que documentan los resultados de las pruebas no lo hacen debidamente.

8. Conclusiones

Los conceptos principales sobre la calidad y pruebas de software logran describir aspectos relacionados con los procesos de pruebas de sistemas identificando las estrategias de pruebas. Las búsquedas de los tipos de pruebas ayudan a resumir sus características y a describir los flujos actuales del proceso de calidad.

Referencias

[1] DIAZ Yanersy, MOLINA Yenisel. Diseño de una aplicación para el Seguimiento de Errores de los productos software de la Facultad 7. UCI. 2007. Cuidad de la Habana.

[2] ROGER S. Pressman: Software Engineering: A Practitioner's Approach (European Adaptation), McGrawHill. 2000. ISBN: 0077096770.

[3] MARQUEZ ALPIZAR Yaimí, VALDEZ HECHAVARRIA Yenni. Procedimiento General de Pruebas de Caja Blanca aplicando la técnica de Camino Básico. UCI, 2007. Cuidad de La Habana. Cap. 1. 20, 21 p.

[4] KAN, S.H. 1995: Metrics and models in software quality engineering, AddisonWesley, Reading, Ma., USA, 1995.

[5] YAMAURA, Tsuneo 1998: How to design practical test cases, IEEE Software. Vol. 15, n6, november/december 1998, pp 3036.

[6] MEYER, B. 1997: Object oriented software construction, 2nd. De., Upper Saddle River, Prentice Hall, 1997.

[7] CIG_LABS. The Home of Groundbreaking Software Quality Management Research, [en línea]. [consultado 10/02/2012].

Disponible: www.cigitallabs.com/reso/definitions/software_testing.html.

[8] IEEE Std 1995, Metrics, IEEE, 1991.

[9] ROGER S. Pressman, R. Can Internetbased Applications Be Engineered? in IEEE Software, September/October IEEE Press, 104110, 1998.

[10] FERNANDEZ PEÑA J. M. IPN México. Pruebas de integración para componentes de software, Marzo 2002.

[11] DARIAS PEREZ Darling, Análisis y Diseño de Componentes para Pruebas de Caja Blanca. UCI, 2008. Cuidad de La Habana, Cap. 1. 14, 15 p.

[12] POLO USAOLA, Dr. Macario. Curso de doctorado sobre Proceso software y gestión del conocimiento. Pruebas del Software. Departamento de Tecnologías y Sistemas de Información. Ciudad Real. 2006. (2008). 46 p.

[13] T. J. McCabe, Structured testing: a testing methodology the cyclomatic complexity metric, Technical Report NIST 500-225, 1996.

[14] JURISTO Natalia, MORENO Ana M., VEGAS Sira. Técnicas de Evaluación de Software. 2006. (2008). 131 p. / UNIVERSIDAD SIMON BOLIVAR. Prueba repetible y mantenible. [en línea]. Bitácora de Prueba. Ingeniería de Software 3. Enero-Marzo 2001. [en línea]. [consultado 12/02/2012]. Disponible: http://www.tuobra.unam.mx/publicadas/040803214240.html.

[15] MARCELO RIZZO, Francisco. Reportes Técnicos, Complejidad Ciclomática. ITBA. CAPIS. 8 p.

[16] ESTEBAN BELLO Rafael, ZENAIDA GARCIA LORENZO Zoila, M. GARCIA MORENO María, REYNOSO LOBATO Antonio, Aplicaciones de la Inteligencia Artificial. Primera edición. 2002, p. 13.

[17] TERUEL Alejandro. [en línea]. [citado 16/02/2012]. Disponible: http://www.ldc.usb.ve/~teruel/ci4713/clases2001/pruebasRep.html#bitacora.

[18] FERNANDEZ, Giovanny. Estándar codificación DOTNET. España. 2005. (2008). 20 p.