## 

**FACULTAD DE CIENCIAS E INGENIERÍA**

**ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMAS E INFORMÁTICA**

**TEMA:**

Pruebas de Software

**CURSO:**

Algorítmica II

**PROFESOR:**

Coronel Castillo Eric Gustavo

**AUTORES:**

Clemente Mondragón Josías Código: 16101030

Holguín Cueva Pedro Código: 16101052

Cruz Acuña Jeferson Código: 16201017

Mendoza Gómez Maycol Código: 17101050

**FECHA DE ENTREGA:**

13-10-2017

**Los Olivos – Perú**

**2017-II**

**INDICE**

1. **RESUMEN**
2. **HISTORIA**
3. **FUNDAMENTOS DE LA INVESTIGACION** 
   1. **Identificación del problema**
   2. **Problema**
   3. **Planteamiento de solución**
   4. **Objetivos**
4. **DESARROLLO DEL TEMA**
   1. **Pruebas de Software**
   2. **Tipos de prueba**
   3. **Niveles de Prueba**
   4. **Herramientas para realizar Pruebas de Software**
      1. **Herramienta** **Open Source (libres)**
      2. **Herramienta de comerciales (de pago)**
   5. **Enfoques de Pruebas**
      1. **Pruebas de Caja Blanca**
      2. **Pruebas de Caja Negra**
   6. **Cobertura de pruebas**
      1. **Criterios de Cobertura**
5. **CASOS**
6. **CONCLUCIONES**
7. **RECOMENDACIONES**
8. **BIBLIOGRAFIA**

**RESUMEN**

El presente trabajo explica sobre las diferentes pruebas que se realizan en un software empezando con los más pequeño como los (módulos o clases) hasta los más grandes como (los sistemas) estas diferentes pruebas se realizan con el objetivo de satisfacer las demandas que hacen los clientes. También se podría definir que las pruebas de software miden la calidad del software programado y la cantidad de códigos mal estructurado o que no están siendo ejecutados, también se resaltara las diferentes herramientas que se utiliza para hacer estas pruebas que se dividen en diferentes tipos de pruebas, la secuencia de niveles que pasa un software y por qué es importante la cobertura en un software para que salga al mercado.

**HISTORIA**

Las pruebas ubicadas dentro de la historia de la Ingeniería del software y siendo organizadas por un criterio de orientación, o paradigma, que permita hacer un balance sobre su evolución pueden subdividirse, por el momento, en cinco intervalos temporales o antecedentes históricos:

**Antes de 1956. Periodo orientado a debugging:**

En este momento todas las pruebas que se realizaban estaban orientadas a la corrección directa del código fuente de los programas. Eran realizadas directamente por los programadores y no estaba clara la diferencia entre: checkout, debugging y testing. El concepto de test o prueba en lo referente al software parte, o es precursor del mismo, A.M. Turing que en 1949 redacta: Checking a Large Routine donde ya se discute el uso de aserciones para realizar pruebas de corrección.

**Entre 1957 y 1978. Periodo orientado a demostración:**

En este momento las pruebas se centran en la realización de checkouts exhaustivos que se focalizan en dos aspectos clave. Por un lado, asegurar que el programa funciona (Debugging) y por otro asegurar que el programa resuelve el problema (Testing), tal y como expone en 1957 C. Baker en su artículo Review of D.D. McCracken’s Digital Computer Programming. Podría afirmarse que en esta fase se utilizan de forma masiva test para garantizar que se cumple con la especificación. Dichos test se realizan al final del desarrollo del software. “La prueba de software puede demostrar la existencia de fallos, pero nunca podría demostrar la ausencia de los mismos.” E.W. Dijkstra.

**Entre 1979 y 1982. Periodo orientado a destrucción:**

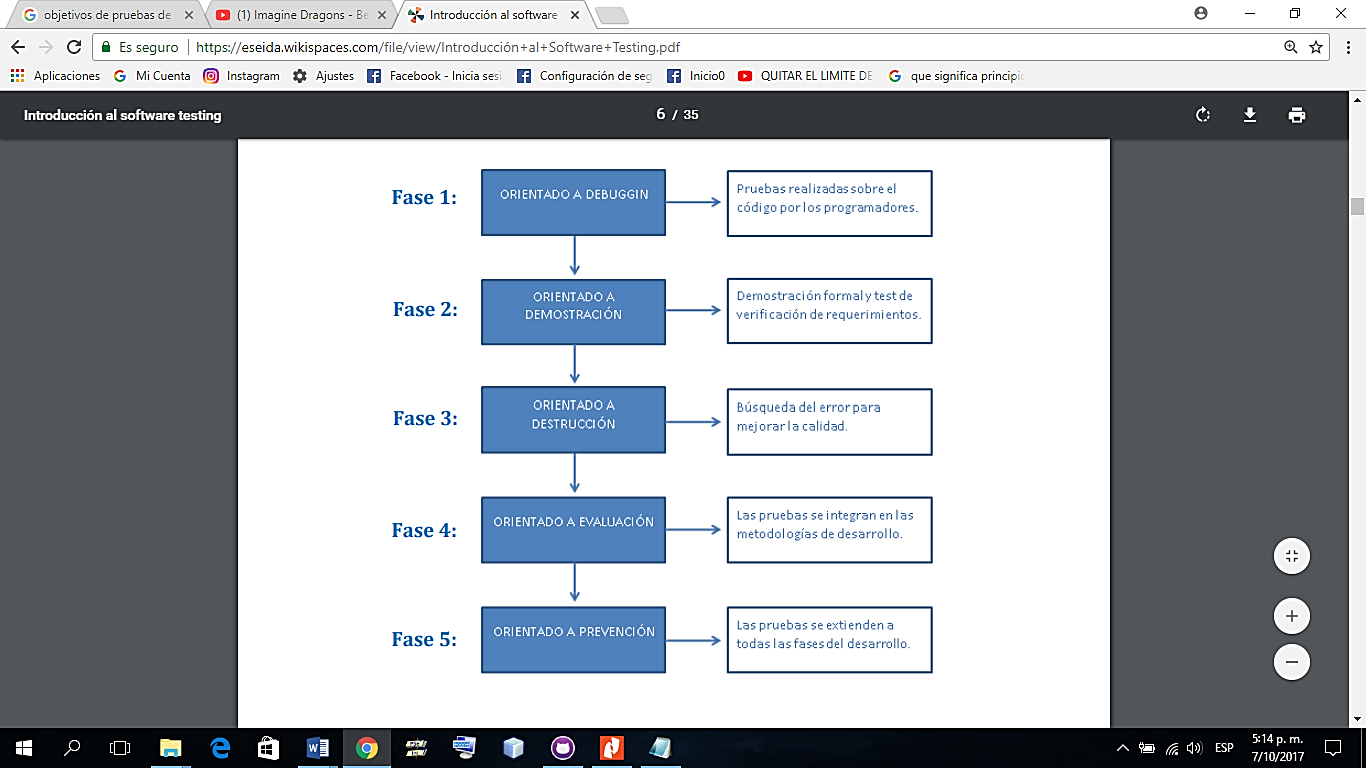
En 1979 G.J. Myers, publica The Art of Software Testing, donde expone que el “Testing es el proceso de ejecutar un programa con la intención de encontrar errores.” Esto cambia por completo el paradigma de las pruebas ya que se pasa de intentar demostrar que un programa es correcto mediante pruebas y demostraciones teóricas basadas en matemáticas a intentar hacer fallar el programa. El objetivo no cambia, intentar que el programa no tenga fallos, pero si la forma de buscarlos y garantizarlos. En este momento si una prueba produce un fallo se considera que la prueba ha tenido éxito. A efectos de concepto, los tests se convierten en Casos de Prueba que se aplican a los productos desarrollados para encontrar errores y corregirlos. “Las pruebas de software son el proceso de ejecutar un programa con la intención de encontrar errores” G.J. Myers.

**Entre 1983 y 1984. Periodo orientado a evaluación:**

En 1983 Guideline for Lifecycle Validation, Verification and Testing of Computer Software. NBS FIPS propone y describe una metodología que integra análisis, revisión y testing en el ciclo de vida del desarrollo de software. Las pruebas de Software empiezan a integrarse en las diferentes metodologías de desarrollo de software. “El objetivo general de las pruebas de software es confirmar la calidad de los sistemas software ejercitando sistemáticamente el software en unas circunstancias cuidadosamente controladas” E.F. Miller.

**Entre 1985 en adelante. Periodo orientado a prevención:**

En 1985 H. Hetzel, y D. Gelperin, implementan STEP (Systematic Test and Evaluation Process) sobre los estándares formales IEEE 829-1983 y 828-1998. Este hecho consigue que las pruebas del software cobren una mayor importancia en el ciclo de desarrollo de software con lo que se abre la posibilidad de integrar pruebas en las diferentes fases del mismo. Durante esta época las pruebas se diversifican para cubrir todas las fases del desarrollo, poder comprobar todos los tipos de artefactos, prototipos, modelos, módulos, subsistemas y sistemas que componen los productos software del momento. Poco a poco las pruebas se considerarán una parte clave de todo el ciclo de desarrollo. Será en 1999 cuando se establezcan las bases de la Programación Extrema (Kent Beck) de las cuales derivará el Test Driven Development, una metodología de desarrollo basada en pruebas. “Hacer pruebas significa comparar los resultados actuales con un estándar” Hutcheson, M.L.



**FUNDAMENTOS DE LA INVESTIGACION**

**Identificación del problema**

Hoy en día hay diversos factores que afectan al usuario al hacer uso de un software, debido a que la mayoría de estos no pasan por un control de calidad o también llamado “Pruebas de software”.

Para ello el programador debería de hacer un Testing de su software para así evitar futuros problemas con el usuario y por consecuente tener pérdida de tiempo y dinero.

**Problema**

¿Por qué es necesario implementar las pruebas de software?

**Planteamiento de solución**

* Hacer los diferentes *niveles de pruebas* al software programado para finalizar con un producto de buen rendimiento.
* Las pruebas de caja blanca son importante a la hora de verificar y evaluar un software puesto que evalúa principalmente el procedimiento del software.
* Si el sistema, software tiene una programación de código muy amplio se recomienda que se utilice las pruebas de caja negra para que sea evaluada de forma más rápida y sencilla.

**Objetivos**

* Aumentar la confianza en el nivel de calidad.
* Reducir costos en mantenimientos.
* Evitar redundancias
* Mejorar la rentabilidad del software.
* Mejorar la calidad del producto final de software.

**DESARROLLO DE TEMA**

**PRUEBAS DE SOFTWARE**

Es un proceso de la Ingeniería por el cual se controla que un sistema cumpla con sus requisitos funcionales y funcione correctamente adecuándolo a unos estándares de calidad y fiabilidad, es una actividad más en el proceso de “control de calidad”.

Las pruebas son el único instrumento que puede determinar la calidad de un producto software, es decir, es el único método por el que se puede asegurar que un sistema software cumple con los requerimientos (Varela, 2012).

Se puede determinar una dependencia directa entre la calidad de un sistema y el valor del mismo, por lo tanto, las pruebas dotan de valor a los productos software.

**TIPOS DE PRUEBAS**

Existen múltiples criterios y formas de clasificación para las pruebas dependiendo del ámbito concreto en el que se realice dicha clasificación:

**Validación vs Verificación:**

* La Validación es un proceso subjetivo donde se valora si es correcto lo que hace un sistema, software, es decir, se centra en el resultado.
* La Verificación es un proceso objetivo que se centra en lo que se supone que debe hacer un programa, comprobando que todo se ejecuta de forma correcta y que no se ejecutan funciones que puedan degradar el funcionamiento.

**Formales vs No Formales:**

* Las pruebas Formales necesitan una especificación formal del problema y por ello suelen ser muy potentes. Dicha especificación es costosa y compleja y suele requerir personal cualificado. Este tipo de pruebas suele aplicarse en ámbitos muy concretos, es decir, zonas críticas del sistema.
* Las pruebas No-Formales se basan en especificaciones realizadas en lenguaje natural (Historias de usuario), son menos potentes pero su uso es generalizado.

**Estáticas vs Dinámicas:**

* Las pruebas Estáticas no requieren ejecución del programa, normalmente se basan en la revisión del código fuente y en la creación de trazas de ejecución. Suelen ser manuales.
* Las pruebas Dinámicas requieren la ejecución del programa, ejecución directa o indirecta si la realiza un humano o un programa (JUnit, por ejemplo).

**Caja blanca vs Caja negra:**

* Las pruebas de Caja Blanca comprueban el cómo un sistema se ejecuta, es decir, se centran en el mecanismo interno del programa comprobando paso a paso cada una de las acciones del mismo.

* Las pruebas de Caja Negra ignoran el interior centrándose únicamente en las entradas y salidas del sistema.

**Manuales vs Automáticas:**

* Las pruebas Manuales son ejecutadas por un usuario. La fiabilidad siempre está en duda ya que al ser realizadas por usuarios estos siempre pueden cometer errores.
* Las pruebas Automáticas se implementan y son ejecutadas automáticamente. Son más fiables, aunque no siempre son posibles.

**NIVELES DE PRUEBA**

**Pruebas Unitarias**

Podemos definir una Prueba de Unidad como un método de evaluación de una parte de un sistema, es decir, de un módulo de nuestro producto software.

Normalmente las Pruebas de Unidad están constituidas como una mezcla de Caja Negra y Caja Blanca, es decir, combinan la validación de entradas y salidas con la verificación por cobertura del código.

1. Analizar la lógica del módulo mediante Caja Blanca.
2. Comprobar el cumplimiento de los requerimientos mediante Caja Negra.

**Pruebas de Integración**

Son aquellas que se realizan en el ámbito del desarrollo de software una vez que se han aprobado las pruebas unitarias y lo que prueban es que todos los elementos unitarios que componen el software funcionen juntos correctamente probándolos en grupo. En otras palabras, las pruebas de Integración son pruebas que comprueban el funcionamiento entre módulos. Existen dos estrategias o acercamientos, uno incremental y otro no incremental.

**Pruebas de Sistema**

Evalúan el *sistema en su conjunto*. El objetivo de este tipo de pruebas es comprobar que se cumplen los requisitos funcionales y las especificaciones técnicas del software. Suele ser habitual que se prueba el sistema en un entorno similar al de producción. Estas pruebas son aplicables a prototipos.

**Pruebas de Carga/Estrés**

Este tipo de pruebas evalúa los requisitos no funcionales.

* Las Pruebas de Estrés comprueban el funcionamiento del sistema bajo condiciones no normales como pueden ser la ausencia de red, la pérdida de un servidor, etc.
* Las Pruebas de Carga someten el sistema a cargas de trabajo extremas determinando la capacidad de resistencia límite del programa, número de usuarios simultáneos, capacidad de cálculo máximo, número de conexiones, etc.

**Pruebas de Regresión**

Las Pruebas de Regresión sólo son aplicables cuando existen versiones previas del sistema. Consisten en comprobar que las versiones anteriores, o el funcionamiento de las versiones anteriores, siguen estando soportadas por el sistema.

**Pruebas de Aceptación**

Las Pruebas de Aceptación evalúan que el sistema cumple con los requisitos del cliente, para ello se cuenta con su participación y por norma general la superación de este tipo de pruebas significa que el cliente acepta el sistema y éste queda validado.

**HERRAMIENTAS PARA REALIZAR PRUEBAS DE SOFTWARE**

El control de la calidad de software lleva consigo *aplicativos* que permiten realizar pruebas autónomas y masivas permitiendo así la verificación desde el punto de vista estático y de caja blanca, es decir pruebas donde se analiza el software sin ejecutar el software mediante el código fuente del mismo. Podemos encontrar herramientas **Open Source (libres)** o **Comerciales (de pago)**. Estas herramientas podrán ser utilizadas para diferentes tipos de pruebas como:

* Herramientas de gestión de pruebas
* Herramientas para pruebas funcionales
* Herramientas para pruebas de carga y rendimiento

|  |  |  |
| --- | --- | --- |
| HERRAMIENTAS | Open Source (libre) | Comerciales (de pago) |
| Herramientas de gestión de pruebas | * Bugzilla Testopia * FitNesse * qaManager * qaBook * RTH (open Source) * Salome-tmf * Squash TM * Test Environment Toolkit * TestLink * Testitool * XQual Studio * Radi-testdir * Data Generator | * [HP Quality Center](https://es.wikipedia.org/wiki/HP_Quality_Center)/ALM * [Silk Central](https://es.wikipedia.org/wiki/Silk_Central) * QA Complete * qaBook * T-Plan Professional * SMARTS * QAS.Test Case Studio * PractiTest * SpiraTest * TestLog * ApTest Manager * Zephyr |
| Herramientas para pruebas funcionales | * Selenium * Soapui * Watir * WatiN * Capedit * Canoo WebTest * Solex * Imprimatur * SAMIE * ITP * WET * WebInject | * Silk Test * QuickTest Pro * Rational Robot * Sahi * SoapTest * Test Complete * QA Wizard * Squish * vTest * Internet Macros |
| Herramientas para pruebas de carga y rendimiento | * JMeter * Gatling * FunkLoad * FWPTT load Testing * loadUI | * HP LoadRunner * IBM Rational Performance Test (RPT) * LoadStorm * NeoLoad * WebLOAD Professional * Forecast * ANTS – Advanced .NET Testing System * Webserver Stress Tool * Load Impact * Silk Performer |

**ENFOQUES DE PRUEBAS**

**Pruebas de Caja Blanca**

Las pruebas de Caja Blanca tienen en cuenta el funcionamiento interno de un sistema o componente, es decir, comprueban que el software realiza de forma correcta las diferentes operaciones para las que esté programado.

Aplicar Caja Blanca sobre un sistema facilita la optimización del código ya que se puede observar de forma precisa el funcionamiento a nivel de sentencia de nuestro sistema. Por lo tanto, aumenta la cobertura del mismo. Otra ventaja de aplicar Caja Blanca es su dinamismo ya que al estar testeando sobre el código este puede alterarse para realizar comprobaciones más exhaustivas (Varela, 2012).

En contraposición a lo anterior, aplicar Caja Blanca, exige un conocimiento elevado del lenguaje de programación y del código del sistema. Este tipo de pruebas dependen directamente de la experiencia del tester y en algunas ocasiones es imposible cubrir todos los caminos posibles por lo que puede pasar nuestro sistema, es decir, la cobertura no puede llegar a ser del 100%.

**Pruebas de Caja Negra**

Las pruebas de Caja Negra tienen en cuenta únicamente las entradas y salidas del sistema o componente, es decir, ignoran el mecanismo interno, el funcionamiento real, a bajo nivel, del software.

Aplicar Caja Negra sobre un sistema posibilita, de forma rápida y sencilla, validar un sistema. Suele ser muy útil cuando nuestro sistema es muy amplio, es decir, contiene grandes cantidades de código. Caja Negra nos informa sobre si un sistema no se comporta según su especificación, es decir, si los resultados para unos datos concretos no son los esperados (Varela, 2012)

**COBERTURA DE PRUEBAS**

La cobertura es la cantidad de código (medida porcentualmente) que está siendo cubierto por las pruebas. En otras palabras, ejecuto las pruebas de mi software y si hay alguna línea o parte de mi código que nunca fue ejecutada en el contexto de las pruebas, entonces dicha línea no está cubierta.

Si mi código consta de 100 líneas y solo 50 líneas están siendo ejecutadas al correr las pruebas, entonces mi cobertura seria solo del 50%.

**¿Qué beneficios tiene medir la cobertura? ¿Qué ventajas tiene el tener una alta cobertura?**

En forma general seria que aumenta la calidad de mi software. Más específicamente podría decir que si tengo una alta cobertura, significa que gran parte me mi código está siendo probado y por consiguiente podría tener cierta certeza sobre el correcto funcionamiento de mi software. Al mismo tiempo medir la cobertura podría ayudarme a detectar código innecesario en mi aplicación, ya que es código que no se ejecuta.

**Criterios de Cobertura**

De acuerdo con (Cornett 2002), el análisis de cobertura del código es el proceso de:

* Encontrar fragmentos del programa que no son ejecutados por los casos de prueba.
* Crear casos de prueba adicionales que incrementen la cobertura.
* Determinar un valor cuantitativo de la cobertura (que es, de manera indirecta, una medida de la calidad del programa)(Polo Usaola, 2006).

**Cobertura de Sentencias**

Comprueba el número de sentencias ejecutables que se han ejecutado.

**Cobertura de Decisiones**

Comprueba el número de decisiones ejecutadas, considerando que se ha ejecutado una decisión cuando se han recorrido todas sus posible ramas (la que la hace true y la que la hace false, pero también todas las posibles ramas de un switch).

**Cobertura de Condiciones**

Comprueba el número de condiciones ejecutadas, entendiendo que se ha ejecutado una condición cuando se han ejecutado todas sus posibles ramas.

**Cobertura de Condiciones múltiples**

Comprueba el número de condiciones múltiples ejecutadas, considerando que se ha ejecutado una condición múltiple cuando se han ejecutado todas sus correspondientes ramas con todas las posibles variantes de la instrucción condicional.

**Cobertura de Caminos**

Comprueba el número de caminos linealmente independientes que se han ejecutado en el grafo de flujo de la unidad que se está probando. El número de caminos linealmente independientes coincide con la complejidad ciclo matica de McCabe.

**Cobertura de Funciones**

Comprueba el número de funciones y procedimientos que han sido llamados.

**Cobertura de Llamadas**

Comprueba el número de llamadas a funciones y procedimientos que se han ejecutado. No debe confundirse con la cobertura de funciones: en la cobertura de funciones contamos cuántas funciones de las que hay en nuestro pro- Mantenimiento Avanzado de Sistemas de Información – Pruebas del software 10 grama han sido llamadas, mientras que la cobertura de llamadas cuenta cuántas de las llamadas a funciones que hay en el programa se han ejecutado.

**Cubrimiento de Bucles**

Comprueba el número de bucles que han sido ejecutados cero veces (excepto para bucles do. While), una vez y más de una vez.

**Cubrimiento de Carrera**

Comprueba el número de tareas o hilos que han ejecutado simultáneamente el mismo bloque de código.

**Cobertura de operadores relacionales**

Comprueba si se han ejecutado los valores límite en los operadores relacionales (>, =, <=), ya que se asume la hipótesis de que estas situaciones son propensas a errores.

**Cobertura de Tablas**

Comprueba si se ha hecho referencia a todos los elementos de los arrays.

**CASOS**

Hablaremos de casos en la que no se hicieron pruebas de software y causaron grandes consecuencias:

* En el año 2007 el aeropuerto de los ángeles colapso por un problema de software que provoco conflictos en una tarjeta de red y bloqueo la red informática.
* Sobredosis radiológica en el instituto nacional del cáncer Panamá en el año 200, errores en el procedimiento y un fallo de software causan que se apliquen dos iserronias de radiación, 8 personas murieron y 20 tuvieron problemas graves de salud.

**CONCLUSIONES**

En conclusión, las pruebas de software es un proceso largo pero muy necesario a la hora de evaluar un software poniéndolo en diferentes situaciones, verificando que cada unidad (modulo, clase, método) funcionen correctamente relacionados entre sí haciendo seguimiento a los siguientes niveles como (integración, sistema) y ya por ultimo a la aceptación. En otras palabras que tu software pase las diferentes pruebas es una gratificación para el programador puesto que su producto llegara a durar años en el mercado.

**RECOMENDACIONES**

* Siempre testear cada pequeña parte (métodos, módulos) de un software, proyecto, etc.
* Aplicar las diferentes pruebas mencionadas para un mejor rendimiento del software.
* Evitar programar un software, proyecto con “código sucio”.
* No solo estar pendiente de las entradas y salidas, sino también del proceso de la programación que se está llevando a cabo.

**BIBLIOGRAFIA**

Varela, L. G. (2012). *Introduccion la Software testing.* Obtenido de wikispaces.com: https://eseida.wikispaces.com/file/view/Introducci%C3%B3n+al+Software+Testing.pdf

Polo Usaola, M. (2006). Pruebas del Software, *1*, 38. Retrieved from https://pdfs.semanticscholar.org/191e/9b745defff8f26557e9c5e4d6da38d3420bd.pdf