Universidad de las Ciencias Informáticas

Facultad 7

Título: Propuesta de Procedimiento y Herramientas para realizar Pruebas de Caja Blanca en el Área Temática APS

Trabajo de Diploma para optar por el Título de
Ingeniero en Ciencias Informáticas

Autora: Dayanis Isaac Morales

.

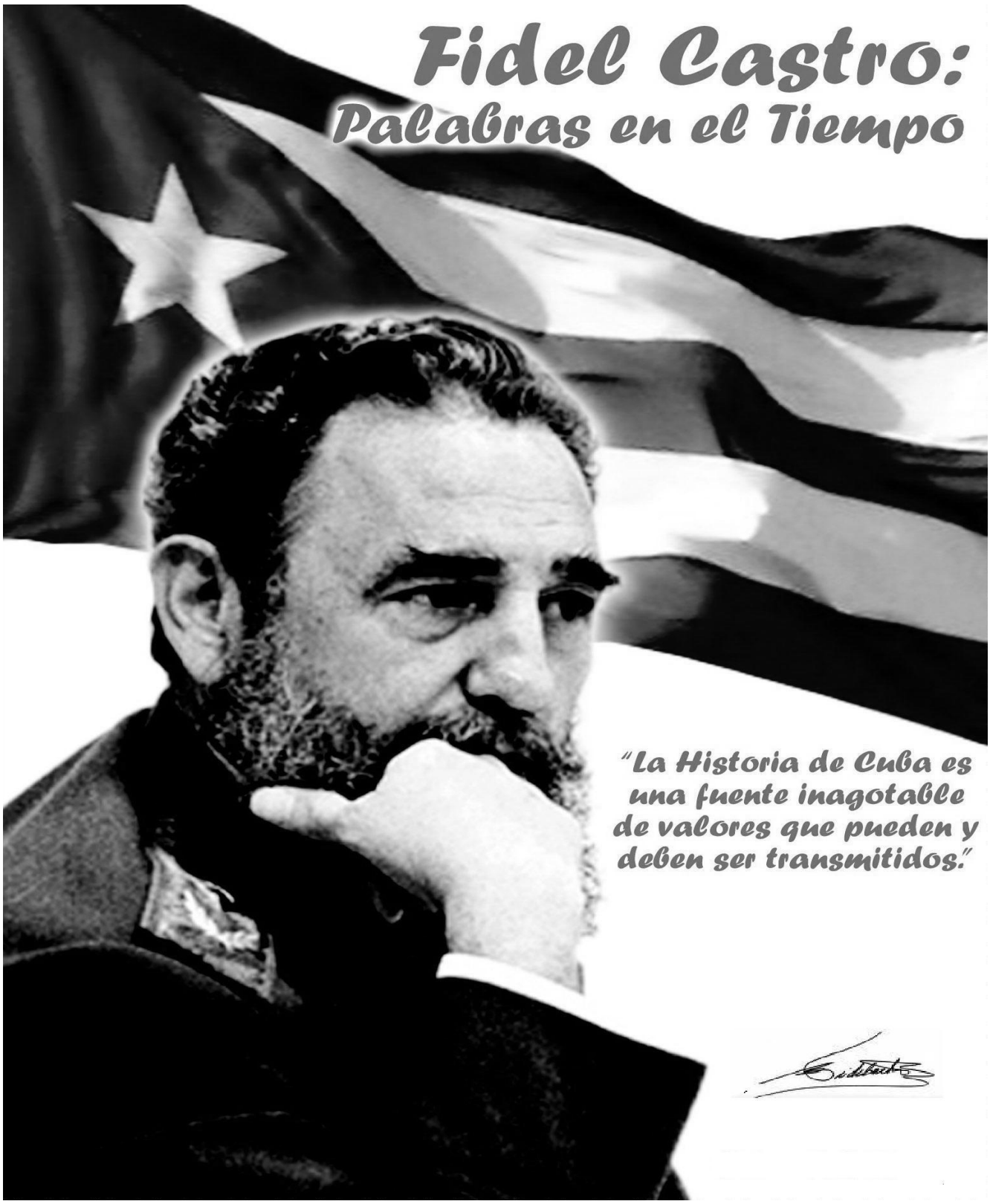
Tutores: Ing. Yoan Manuel Cabrera Arribas

Ing. Yoelvis Osés Sosa

Ciudad de La Habana Junio del 2009

"Año del 50 Aniversario del triunfo de la Revolución"

Fidel Castro:
Palabras en el Tiempo



"La Historia de Cuba es una fuente inagotable de valores que pueden y deben ser transmitidos."

Declaración de Autoría

Declaro que soy la autora de este trabajo y autorizo a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

Para que así conste firmo la presente a los 22 días del mes de junio del año 2009.

Autora: _____

Dayanis Isaac Morales

Tutores: _____ _____

Ing. Yoan Manuel Cabrera Arribas **Ing. Yoelvis Osés Sosa**

Datos de Contacto

Ing. Yoan Manuel Cabrera Arribas (ycabreraar@uci.cu): Graduado de Ingeniero en Ciencias Informáticas, Profesor Recién graduado, imparte la asignatura de Segundo Perfil en la Facultad 7, es líder de un equipo de desarrollo del Área Temática APS. Ha presentado trabajos en el Evento Uciencia 2007, 2008 y en la XIII Convención y Feria Internacional Informática 2009, posee publicaciones en la Serie Interna de la UCI.

Yoelvis Osés Sosa (yoses@uci.cu): Graduado de Ingeniero en Ciencias Informáticas, Profesor Recién graduado, imparte las asignaturas de Historia de la Informática y Software Libre en la Facultad 7, Asesor Comercial del Polo productivo de Informática para la Salud. Presentó un trabajo en el Evento Uciencia 2008 y en la XIII Convención y Feria Internacional Informática 2009, posee publicaciones en las memorias de dichos eventos.

Agradecimientos

A la Revolución por darme la posibilidad de estudiar en esta Universidad de Excelencia.

Al Comandante Fidel por haber pensado en nosotros como la generación del futuro.

A mis padres Odalis y Pedro por todo el esfuerzo que han hecho para lograr mis sueños.

A mis tutores Yoan y Yoelvis que me han sabido guiar por el camino correcto.

A mi oponente Ariel Pons por su cooperación para realizar este trabajo.

A Franklin por su amor, su amistad, su compañía y su apoyo durante estos 5 años.

A Daily, Yamilet, Yuliet e Irina que a pesar de las diferencias siempre están a mi lado.

A Rosabel y Fabio por su amistad incondicional.

A todos aquellos que de una forma u otra me ayudaron en el desarrollo de este trabajo.

Por su dedicación y esmero,

¡Gracias!

Dayanis Isaac Morales

Dedicatoria

Dedico este trabajo a las personas más importantes de mi vida:

A mi abuelo José Grabiel Morales, por la educación que me dió.

A mi mamá por ser mi luz, mi guía, mi todo.

A mi papá que se ha esmerado en darme lo mejor.

A Franklin por ser tan especial conmigo.

A toda mi familia, y mis amigos.

Dayanis Isaac Morales

Resumen

La calidad del software constituye un objetivo principal de los desarrolladores. El software es sometido a pruebas para detectar posibles defectos. Entre ellas las pruebas estructurales o de Caja Blanca, se encargan de verificar que los algoritmos utilizados en cada uno de los procesos funcionen correctamente.

El presente trabajo propone una secuencia de acciones para realizar las Pruebas de Caja Blanca en el Área Temática APS. Se toma como punto de partida el Flujo de Trabajo de Pruebas definido, en base a las actividades que en él se exponen, se indica cada una de las acciones a seguir por el Equipo de Calidad del Área Temática. Refleja cada uno de los artefactos de entrada y salida, indicando cómo se utilizan y completan. Presenta herramientas de prueba que servirán para realizar el proceso de pruebas de forma rápida y factible.

Para confirmar la validez del trabajo realizado se aplicó el procedimiento al Registro de Actividades Diarias de la Atención Primaria de Salud. De acuerdo a lo planteado en la propuesta se realizan cada una de sus actividades y se plasman los resultados en cada una de las plantillas involucradas.

La puesta en práctica del procedimiento propuesto, implica que se examine la estructura interna del software, lo que disminuye el número de errores existentes en los sistemas y garantiza mayor calidad y confiabilidad. La utilización de herramientas de automatización agilizaría el proceso de pruebas.

PALABRAS CLAVE: Pruebas de Caja Blanca, Código, Flujo de trabajo de prueba, Equipo de Calidad.

Contenido

Introducción

Las Tecnologías de la Información y la Comunicación (TIC) se encargan del estudio, desarrollo, implementación, almacenamiento y distribución de la información mediante la utilización de hardware y software como medio de sistema informático.(1) Las utilidades que ellas brindan han provocado gran impacto en la sociedad, pues permite realizar el trabajo de manera ágil y eficiente. Hoy en día, constituyen un motor de avance para el desarrollo de las diferentes ramas de la economía del país, dentro de ellas la salud, un sector privilegiado, que cuenta con la colaboración de la población en general.

La informatización del Sistema Nacional de Salud (SNS) es el conjunto de métodos, técnicas, procederes y actividades gerenciales dirigidas al manejo de la información en esta área, la cual comprende la información sobre el estado de salud de la población, el conocimiento de las ciencias de la salud y la información en general para la toma de decisiones, clínico-epidemiológicas, operativas y estratégicas. (2) Esta debe incrementar así, la efectividad y eficiencia en el entorno de salud asistencial, teniendo como eslabón principal, la población, por ser el elemento beneficiado al obtener organización, calidad y consistencia de la información.

El Ministerio de Salud Pública (MINSAP) es el órgano rector del SNS. Encargado de dirigir, ejecutar y controlar la aplicación de la política del Estado y del Gobierno en cuanto a la Salud Pública, el desarrollo de las Ciencias Médicas y la Industria Médico Farmacéutica. (3)

Este ha definido a la informatización como una de sus prioridades y ha convocando para ello, a un grupo de instituciones propias del sector, del Ministerio de Informática y Comunicaciones (MIC) y de otros organismos de la administración central del estado, para definir de conjunto la estrategia a desarrollar. En algunos casos se ha tomado como punto de partida sistemas ya desarrollados en el país en el marco de aquella primera estrategia de desarrollo. (4)

Al calor de la Batalla de Ideas surge la Universidad de las Ciencias Informáticas (UCI). Su principal objetivo es construir software para las diferentes esferas de la sociedad, con la calidad necesaria, lo que conllevará al perfeccionamiento empresarial y el avance tecnológico del país. Es una Universidad innovadora, de excelencia científica, académica y productiva que forma de manera continua profesionales integrales comprometidos con la Patria, con el soporte de la informatización del país y con la competitividad internacional de la Industria Cubana del Software.

Dentro de la UCI, la facultad 7 tiene el perfil de Salud. Su misión es la proyección, planificación y ejecución de los diferentes proyectos encaminados a la informatización de este sector. Se definen varias Áreas Temáticas para desarrollar aplicaciones que permitan perfeccionar la calidad de los servicios brindados a la sociedad, faciliten las funciones del personal de salud y que colaboren con la gestión administrativa, asistencial, docente y de investigación, contando con los datos generados en los distintos niveles de salud.

Dentro de estas áreas se encuentra Atención Primaria de la Salud (APS) que se encarga de desarrollar software para dicho nivel asistencial e involucra un grupo de estudiantes, profesores de la UCI y expertos funcionales del MINSAP.

Obtener un software con calidad implica utilizar metodologías o procedimientos en todo el ciclo de desarrollo del mismo que permitan uniformar la filosofía de trabajo, para lograr mayor confiabilidad, mantenibilidad y facilidad de prueba, a la vez que eleven la productividad. (5)

Para controlar la calidad en un proyecto, se planifica un conjunto de actividades antes de comenzar a desarrollar las aplicaciones. Una de ellas es la prueba del software.

Una prueba no es más que ejecutar el software con determinados datos de entrada y producir resultados que luego serán comparados con los teóricos, con el objetivo de encontrar errores. (6) Las pruebas de software se realizan durante todo el ciclo de vida de éste, pero requieren mayor esfuerzo durante la fase de construcción.

Dentro de este flujo de trabajo se realiza una secuencia de actividades desarrolladas por los trabajadores involucrados. Aquí se definen un conjunto de métodos, niveles, estrategias y tipos de pruebas enfocadas a garantizar la eficiencia del producto final.

Dentro de los métodos de prueba que se realizan están las Pruebas de Caja Blanca. Estas son pruebas diseñadas después que existe un código fuente, comprobando los caminos lógicos y proponiendo casos de prueba que examinen que están correctas todas las condiciones y/o bucles para determinar si el estado real coincide con el esperado o afirmado. (7) Estas requieren del conocimiento de la estructura interna del programa y son derivadas a partir de las especificaciones internas de diseño o el código.

Las Pruebas de Caja Blanca se consideran como unas de las más importantes que se aplican al software, pues logran como resultado que disminuya el número de errores existentes en los sistemas, lo que garantiza mayor calidad y confiabilidad.

Las investigaciones realizadas en el Área Temática APS mostraron como resultado que no está definido un procedimiento que permita examinar la estructura interna del programa. No se verifica que los algoritmos utilizados en cada uno de los procesos del sistema funcionen correctamente. Una vez conocido el funcionamiento del producto no se asegura que las operaciones se ajusten a las especificaciones, ni cumpla con los estándares definidos con el proyecto. No se cuenta con herramientas que permitan agilizar este proceso, por tanto las Pruebas de Caja Blanca no se están llevando a cabo, provocando inseguridad en la calidad de los productos elaborados.

Por todo lo que hasta aquí se ha expuesto el esfuerzo estará encaminado en resolver el siguiente **problema**: ¿Cómo llevar a cabo el proceso de Pruebas de Caja Blanca en el Área Temática APS?

Se define como **objeto de estudio** el flujo de trabajo de Pruebas en el Área Temática APS. El **campo de acción** se enfoca en las Pruebas de Caja Blanca en el Área Temática APS.

El **objetivo general** de la investigación es proponer un procedimiento y herramientas para realizar Pruebas de Caja Blanca en el Área Temática APS.

Para cumplir el objetivo del trabajo se proponen las siguientes **tareas de investigación**:

- ✓ Analizar la arquitectura del Área Temática APS.
- ✓ Analizar el flujo de trabajo de pruebas del Área Temática APS.
- ✓ Elaborar un marco teórico sobre las pruebas analizando las mejores prácticas aplicadas en este flujo de trabajo, además de los niveles, tipos, estrategias, métodos y técnicas de prueba.
- ✓ Analizar herramientas de automatización de Pruebas de Caja Blanca que se ajusten al desarrollo de software del Área Temática.
- ✓ Proponer un procedimiento de Pruebas de Caja Blanca.
- ✓ Ejecutar el procedimiento de Pruebas de Caja Blanca propuesto al Registro de Actividades Diarias de la Atención Primaria de Salud (RAD).
- ✓ Evaluar el procedimiento en base a los resultados obtenidos.

Este trabajo ha sido estructurado en tres capítulos, de la siguiente manera:

Capítulo 1

En este capítulo se elabora un marco teórico referido a los conceptos fundamentales sobre Pruebas, haciendo énfasis en las Pruebas de Caja Blanca. Explica como se comportan estas pruebas en el Área Temática, así como en la Universidad. Relaciona un conjunto de herramientas para la automatización de las pruebas.

Capítulo 2

Describe la propuesta de un procedimiento de Pruebas de Caja Blanca que será aplicado en el Área Temática. Muestra las actividades, trabajadores y artefactos involucrados en la realización de las Pruebas de Caja Blancas. Además propone herramientas de automatización que permiten agilizar el proceso de pruebas.

Capítulo 3

Muestra la ejecución del procedimiento propuesto al Registro de Actividades Diarias de la Atención Primaria de Salud, para de esta forma evaluar su eficiencia, calidad y objetividad.

Capítulo 1: Fundamentación Teórica

En este capítulo se realiza un estudio de las mejores prácticas aplicadas en el flujo de trabajo de prueba definido por la metodología RUP, además de los niveles, tipos, estrategias y métodos de pruebas, definiendo los conceptos fundamentales relacionados con el tema a desarrollar y enfocando los referentes a las Pruebas de Caja Blanca. También se relacionan varias herramientas de automatización, creadas para agilizar dicho proceso. Estas serán seleccionadas de acuerdo a un análisis previo de la arquitectura definida, con el fin de mejorar el proceso de pruebas que en la actualidad lleva a cabo el Área Temática APS.

Las aplicaciones desarrolladas en la Universidad son sometidas a un conjunto de pruebas que permiten al equipo de trabajo conocer la eficiencia del producto. Estas son un elemento crítico para garantizar la calidad del software y representa una revisión de las especificaciones, del diseño y de la codificación.

Para obtener un buen producto se debe controlar la calidad.

1.1. Calidad

La calidad del software es el grado con el que un sistema, componente o proceso cumple los requerimientos especificados y las necesidades o expectativas del cliente o usuario. (8)

Es un aspecto muy importante porque si el usuario no queda satisfecho ninguna otra cosa importa. Al controlar la calidad se realizan una serie de inspecciones, revisiones y pruebas para asegurar que el producto cumple con los requisitos que le han sido asignados. Las actividades pueden ser manuales o de forma automática.

1.2. Pruebas

Las Pruebas de software son los procesos que permiten verificar y revelar la calidad o el buen funcionamiento de un producto software. Estas se integran dentro de las diferentes fases del Ciclo del software dentro de la Ingeniería de Software. Así se ejecuta un programa y mediante técnicas experimentales se trata de descubrir los errores que podría tener. Para determinar el nivel de calidad

se deben efectuar unas medidas o pruebas que permitan comprobar el grado de cumplimiento respecto a las especificaciones iniciales del sistema. (9)

Probar es ejecutar el software con determinados valores de entrada y producir resultados que luego serán comparados con los teóricos con el objetivo de encontrar errores. Para comenzar el proceso de pruebas lo primero es trazar una estrategia que ayude al equipo de trabajo a definir cómo y cuándo se realizarán estas actividades.

1.3. Estrategia de Pruebas

Una estrategia integra las técnicas de diseño de casos de prueba en una serie de pasos bien planificados que llevan a la construcción correcta del software con el objetivo de planificar las pruebas necesarias en cada iteración, diseñarlas e implementarlas creando los casos de prueba que especifican qué probar, cómo realizarlas y creando, si es posible, componentes ejecutables para automatizarlas, además de realizar diferentes pruebas y manejar los resultados de cada una sistemáticamente.

La estrategia de prueba describe el enfoque y los objetivos generales de las actividades de prueba**.**

Considerando el proceso desde el punto de vista procedimental, la prueba, en el contexto de la ingeniería del software, realmente es una serie de cuatro pasos que se llevan a cabo secuencialmente. Inicialmente, la prueba se centra en cada módulo individualmente, asegurando que funcionan adecuadamente como una unidad. De ahí el nombre de *prueba de unidad.* Luego se deben ensamblar o integrar los módulos para formar el paquete de software completo, esta es la *prueba de integración*. Después de que el software se ha integrado (construido), se dirigen un conjunto de *pruebas de alto nivel.* Se deben comprobar los criterios de validación (establecidos durante el análisis de requisitos). La *prueba de validación* proporciona una seguridad final de que el software satisface todos los requisitos funcionales, de comportamiento y de rendimiento. (10)

La estrategia encierra los niveles de prueba que van a ser probados. La Prueba es aplicada con diferentes objetivos, en diferentes escenarios o niveles de trabajo.

1.4. Niveles de Prueba

1.4.1. Pruebas de unidad

Las pruebas de unidad o pruebas unitarias centran el proceso de verificación en la menor unidad del diseño del software: el componente software o módulo. (11)

Una prueba de unidad pretende probar cada función en un archivo de programa simple. Esto quiere decir que un módulo que tiene una prueba de unidad se puede probar independientemente del resto del sistema.

Utilizando la descripción del diseño se van probando los caminos de control importantes con el fin de descubrir errores dentro del ámbito del módulo. Esto ayuda a que el módulo se haga independiente.

Se prueba la interfaz del módulo para asegurar que la información fluye de forma adecuada hacia y desde la unidad del programa que está siendo probado. Se examinan las estructuras de datos locales para asegurar que los datos que se mantienen temporalmente conservan su integridad durante todos los pasos de ejecución del algoritmo. Se prueban las condiciones límite para asegurar que el módulo funciona correctamente en los límites establecidos como restricciones de procesamiento. Se ejercitan todos los caminos independientes de la estructura de control con el fin de asegurar que todas las sentencias del módulo se ejecutan por lo menos una vez. Y finalmente se prueban todos los caminos de manejo de errores.

Durante la prueba de unidad, la comprobación selectiva de los caminos de ejecución es una tarea esencial. Se deben diseñar casos de prueba para detectar errores debido a cálculos incorrectos, comparaciones incorrectas o flujos de control inapropiados.

Una vez probado el nivel de unidad se pasa a la integración de los módulos.

1.4.2. Prueba de integración

Pruebas integrales o pruebas de integración son aquellas que se realizan en el ámbito del desarrollo de software una vez que se han aprobado las pruebas unitarias. El objetivo es construir una estructura de programa que esté de acuerdo con lo que dicta el diseño y con los módulos probados en la prueba de unidad.

Las pruebas de integración en los proyectos de desarrollo de software, no solo se presentan en la integración entre módulos de un mismo producto sino que se están planteando en proyectos que

ofrecen soluciones conformadas por varios productos de software, lo cual le da una nueva dimensión al proceso de pruebas de integración (12).

1.4.3. Prueba del sistema

Las pruebas del sistema se realizan cuando el software está funcionando como un todo. Es la actividad de prueba dirigida a verificar el programa final, después que todos los componentes de software y hardware han sido integrados. En un ciclo iterativo estas pruebas ocurren más temprano, tan pronto como subconjuntos bien formados de comportamiento de caso de uso son implementados. Verifica que cada elemento encaja de forma adecuada y que se alcanza la funcionalidad y el rendimiento total del sistema.

1.4.4. Pruebas de aceptación

Las pruebas de aceptación se realizan cuando el producto está listo para implantarse en el entorno del cliente. El usuario debe ser el que realice las pruebas, ayudado por personas del equipo de pruebas, siendo deseable, que sea el mismo usuario quien aporte los casos de prueba.

Es muy recomendable que las pruebas de aceptación se realicen en el entorno en que se va a explotar el sistema (incluido el personal que lo maneje). En caso de un producto de interés general, se realizan pruebas con varios usuarios que reportarán sus valoraciones sobre el producto.

La estrategia delimita también el tipo de prueba a ser ejecutadas.

1.5. Tipos de Pruebas

Para los diferentes niveles se ejecutan varios tipos de prueba entre los que se encuentran (13):

Pruebas de Función: Se enfocan a validar funcionalidades específicas provistas por servicios requeridos, métodos, o casos de uso. Estas pruebas se implementan y ejecutan a nivel de unidades, unidades integradas, aplicaciones y sistemas.

Pruebas de Seguridad: Pruebas enfocadas en asegurar que los usuarios están restringidos a funciones específicas o su acceso está limitado únicamente a los datos que están autorizados a acceder. Que solo aquellos usuarios autorizados a acceder al sistema son capaces de ejecutar las funciones del sistema.

Pruebas de Volumen: Pruebas enfocadas en la verificación de la habilidad de manejar grandes cantidades de datos, bien sea como entrada, salida o residente en una base de datos. Tiene como objetivo verificar que el sistema soporta los volúmenes máximos definidos en la cuantificación de requerimientos: Capacidad de Almacenamiento, Capacidad de Procesamiento, entre otras.

Pruebas de Usabilidad: Prueba enfocada a factores humanos, estéticos, consistencia en la interfaz de usuario, ayuda sensitiva al contexto y en línea, asistentes de documentación de usuarios y materiales de entrenamiento. Permiten detectar zonas no cubiertas (no ejecutadas), porcentaje o total de utilización, tiempo de ejecución (por zona). Puede contrastarse este perfil contra el diseño.

Pruebas de Integridad: Se enfocan en probar la robustez (resistencia a fallas) y el uso adecuado del lenguaje, sintaxis y uso de recursos. Este tipo de prueba puede aplicarse tanto a unidades como a integración de unidades.

Pruebas de Estructura: Estas pruebas se enfocan en hallar problemas de adherencia del elemento objetivo a su diseño y formación. Típicamente, estas pruebas se realizan sobre aplicaciones Web, asegurando que todos los enlaces están conectados, los controles adecuados se muestran, y no hay contenido inaccesible.

Pruebas de Estrés: Este tipo de prueba se enfoca a evaluar el comportamiento del sistema bajo condiciones anormales. Estrés del sistema se refiere a extrema carga, memoria insuficiente, no disponibilidad de servicios y hardware o recursos compartidos limitados. Este tipo de prueba permite comprender mejor cómo y qué áreas del sistema colapsarán, de este modo es posible planificar contingencias y actualizar el mantenimiento y planear y asignar recursos de antemano.

Benchmark: Compara el rendimiento de un elemento nuevo o desconocido a uno de carga de trabajo de referencia conocido.

Pruebas de Contención: Valida que el elemento que se prueba funciona adecuadamente cuando muchos actores solicitan el mismo recurso.

Pruebas de Carga: Valida y evalúa la aceptabilidad de un elemento de un sistema sobre diferentes cargas de trabajo mientras el sistema permanece constante. Generalmente se incluye simulación de cargas de trabajo promedio y pico que puedan ocurrir dentro de la tolerancia operacional normal.

Pruebas de Perfil de Desempeño: Monitorea el perfil en el tiempo incluyendo flujo de ejecución, acceso a datos, llamadas a funciones para identificar cuellos de botella y procesos ineficientes.

Pruebas de Configuración: Se enfocan en evaluar aquellos elementos configurados para diferentes hardwares y/o configuraciones de software. Pueden implementarse como pruebas de rendimiento del sistema.

Pruebas de Instalación: Se enfoca en evaluar que el elemento a probar se instala como se indica, en diferentes hardware y /o configuraciones de sistemas de software y bajo diferentes condiciones (tales como espacio insuficiente en disco, interrupción de electricidad). Este tipo de prueba se aplica y ejecuta sobre aplicaciones y sistemas.

1.6. Métodos de Prueba

Cualquier proceso de ingeniería puede ser probado de dos formas:

- ✓ Se pueden llevar a cabo pruebas que demuestren que cada función es completamente operativa.
- ✓ Se pueden desarrollar pruebas que aseguren que la operación interna se ajusta a las especificaciones y que todos los componentes internos se han comprobado de forma adecuada.

La primera aproximación se denomina Prueba de la Caja Negra (PCN) y la segunda Prueba de la Caja Blanca (PCB). Ambas constituyen métodos de prueba pues ayudan a definir conjuntos de casos de prueba, aplicando un cierto criterio los casos de prueba quedarán determinados por los valores a asignar a las entradas en su ejecución.

1.6.1. Pruebas de Caja Blanca

Estas pruebas del software se diseñan luego de existir un código fuente. Su objetivo es verificar que el algoritmo utilizado en cada uno de los procesos del sistema funcione correctamente. Se van comprobando los caminos lógicos del software y se proponen casos de prueba que examinen si están correctas todas las condiciones, para determinar si el estado real coincide con el esperado.

Mediante los métodos de Caja Blanca, el ingeniero de software puede obtener casos de prueba que garanticen que se ejerciten al menos una vez todos los caminos independientes de cada módulo, que se ejerciten todas las decisiones lógicas en sus vertientes verdaderas y falsas, que se ejecuten todos los bucles en sus límites y con sus límites operacionales y que se ejerciten las estructuras internas de datos para asegurar su validez. (14)

El **Caso de Prueba** específica la forma de probar un sistema incluyendo las entradas, salidas y resultados esperados, así como bajo que condiciones debe probarse el sistema.

Para diseñar los casos de Pruebas de Caja Blanca existen varias técnicas, como son:

- La Prueba del Camino Básico

Esta es una técnica de prueba propuesta por Tom McCabe (1976) que permite tener la medida de la complejidad lógica de un diseño procedimental y usarla como guía para definir un conjunto básico de caminos de ejecución. Los casos de prueba obtenidos garantizan que durante la prueba se ejecuta por lo menos una vez cada sentencia del programa.

Se explicará mediante un ejemplo cómo se obtienen los casos de prueba en una serie de 4 pasos.
Paso 1: Obtener el Grafo de Flujo, a partir del diseño o del código del módulo.

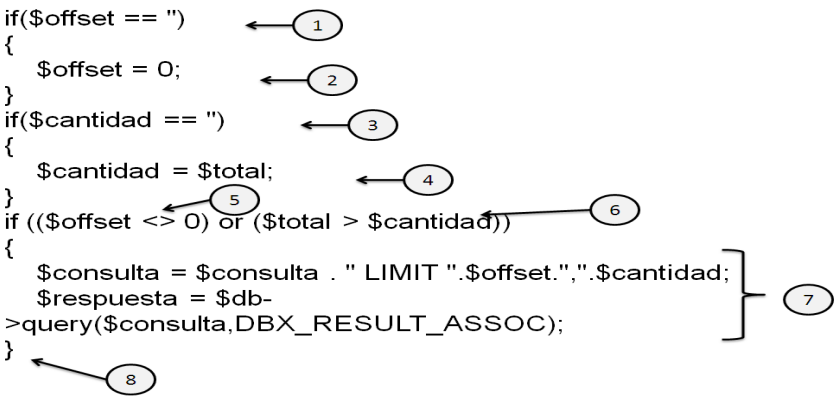


Figura 1.1: Código.

Para realizar la notación hay que tener en cuenta varias reglas:

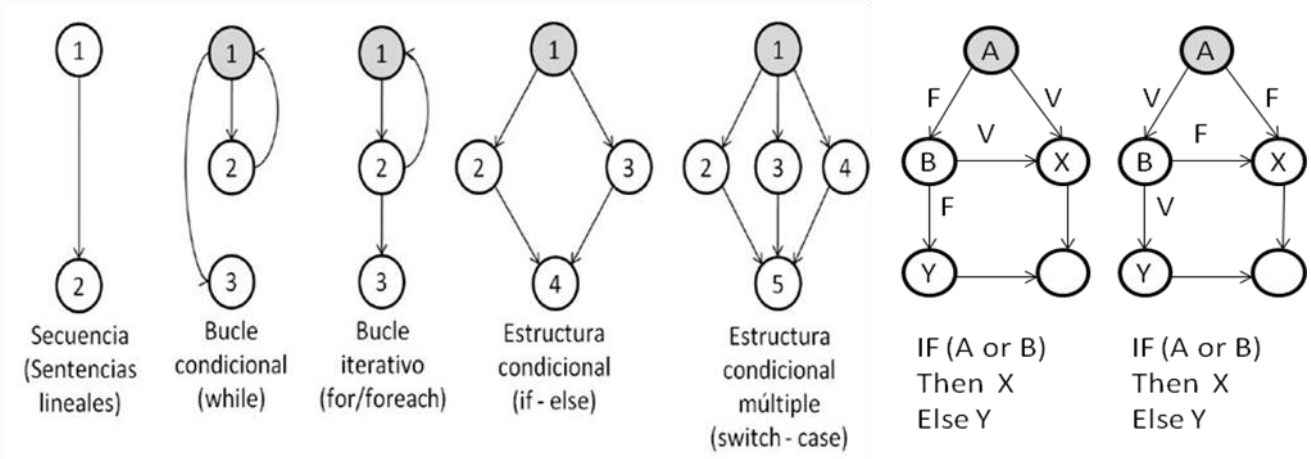


Figura 1.2: Notación de grafo de flujo

- ✓ Cada círculo denominado *Nodo* representa 1 ó más sentencias.
- ✓ Un solo nodo puede corresponder a una secuencia de cuadros de procesos y a un rombo de decisión.
- ✓ Las flechas denominadas *Aristas* representan flujo de control y deben terminar en un nodo.
- ✓ Las áreas delimitadas por aristas y nodos se denominan *Regiones*.
- ✓ Cuando se contabilizan las regiones se incluye el área exterior del grafo.
- ✓ Si en el diseño procedimental se utilizan condiciones compuestas (OR, AND, NAND, NOR), la generación del grafo de flujo tiene que descomponer las condiciones compuestas en condiciones sencillas.
- ✓ Cada nodo que contiene una condición se denomina *Nodo Predicado* y está caracterizado porque 2 o más aristas emergen de él.

A partir del código de la Figura 1.1 se obtendrá el grafo de flujo que por las reglas anteriormente explicadas quedaría de la siguiente forma:

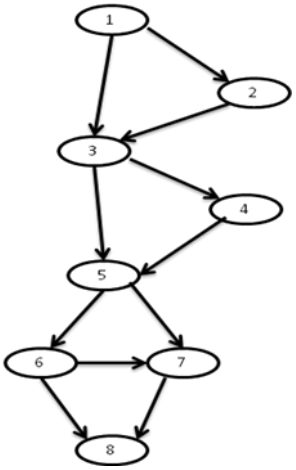


Figura 1.3: Grafo de flujo.

Paso 2: Obtener la Complejidad Ciclomática del grafo de flujo

Una vez representado el grafo de flujo la Complejidad Ciclomática sería:

V (G) = *Número de regiones*. → **5**

V (G) = Aristas - Nodos + 2. → ***11-8 +2 = 5***

V (G) = Nodos predicado + 1. → ***4* + 1 = 5**

Paso 3: Definir el conjunto básico de caminos independientes.

La Complejidad Ciclomática brinda el número de caminos linealmente independientes de la estructura de control del programa. En este ejemplo hay que especificar 5 caminos.

Un **camino independiente** es cualquier camino del programa que introduce, por lo menos, un nuevo conjunto de sentencias simples o una nueva condición. En términos del grafo de flujo, un camino independiente esta constituido por lo menos por una arista que no haya sido recorrida anteriormente a la definición del camino.

Camino 1: 1-3-5-6-7-8.

Camino 2: 1-3-5-7-8.

Camino 3: 1-3-5-6-8.

Camino 4: 1-2-3-5-6-8.

Camino 5: 1-3-4-5-6-7-8.

Paso 4: Determinar los casos de prueba que permitan la ejecución de cada uno de los caminos anteriores.

Para determinar los casos de prueba se debe escoger los datos de forma que las condiciones de los nodos predicados estén adecuadamente establecidas, con el fin de comprobar cada camino. Los casos de prueba que satisfacen el conjunto básico previamente descrito son:
Caso de prueba del camino 1:

Variable offset ≠"" y cantidad < total.

Caso de prueba del camino 2:

Variable offset ≠"" y cantidad < total.

Caso de prueba del camino 3:

Variables offset = 0 y cantidad ≠ "" > total.

Caso de prueba del camino 4:

Variable offset = 0 y cantidad < total.

Caso de prueba del camino 5:

Variable offset ≠ "" y = "".
Una vez ejecutado cada caso de prueba se comparan los resultados obtenidos con los esperados.

- La Prueba de Condición

Es un método de diseño de casos de prueba que ejercita las condiciones lógicas contenidas en el módulo de un programa. Se centra en la prueba de cada una de las condiciones del programa. Su propósito es detectar no solo los errores en las condiciones sino también otros errores en dicho programa.

Las *condiciones simples* son variables lógicas, también llamadas expresiones relacionales. Las expresiones relacionales se expresan la forma E_1 <operador-relacional> E_2 donde E_1 y E_2 son expresiones aritméticas y <operador-relacional> puede ser: "<","<=","=","≠ (desigualdad)", ">", o ">=".

Las condiciones compuestas están formadas por 2 o más condiciones simples, operadores lógicos (AND"&" OR"|") y paréntesis.

Existen varias *estrategias de prueba de condiciones* por ejemplo:

La *prueba de ramificaciones*: Para una condición compuesta C, es necesario ejecutar al menos una vez las ramas verdadera y falsa de C y cada condición simple de C.

La *prueba del dominio*: Requiere la realización de 3 ó 4 pruebas para una expresión relacional para comprobar que el valor de E_1 es mayor, igual o menor que el valor de E_2. Si el <operador-relacional>es incorrecto y E_1 y E_2 son correctos, entonces esas 3 pruebas garantizarán la detección de un error del operador. Para detectar errores en E_1 y E_2, la prueba que haga el valor de E_1 mayor o menor que el valor de E_2 debe hacer que la diferencia entre estos valores sea la más pequeña posible.

Para una expresión lógica con n variables, habrá que realizar las 2^n pruebas posibles (n>0).

BRO (prueba del operador relacional y de ramificación): Estrategia basada en las dos anteriores que garantiza la detección de errores de operadores relacionales y ramificaciones en una condición dada, en la que todas las variables lógicas y operadores aparecen solo una vez y no tienen variable común.

Esta estrategia utiliza restricciones de condición. Una restricción de condición para C con *n* condiciones simples se define cono (D_1, D_2,…, D_n), donde D_i ($0 < i \leq n$) es un símbolo que especifica una restricción sobre el resultado de la i-ésima condición simple de la condición C. Una restricción D se cubre o trata si durante esta ejecución de C, el resultado de cada condición simple de C satisface la restricción correspondiente de D.

Ejemplo 1: Para una variable lógica B, la restricción consiste en que B tiene que ser verdadero o falso. Imagine la expresión C_1: B_1 & B_2 donde B_1y B_2 son variables lógicas. La restricción para C es (D_1, D_2), donde ambas son verdaderas o falsas. El valor (v, f) es una restricción de condición para C_1y se cubre mediante la prueba que hace que el valor de B_1 sea verdadero y B_2 sea falso. La estrategia BRO requiere que el conjunto de restricciones {(v, v), (v, f), (f, v)} sea cubierto mediante las ejecuciones de C_1. Si C_1 es incorrecto por uno o más errores de operador lógico, por lo menos un par del conjunto de restricciones forzará el fallo de C_1.

- La Prueba de Flujo de Datos

Se selecciona caminos de prueba de un programa de acuerdo con la ubicación de las definiciones y los usos de las variables del programa.

A cada sentencia de un programa se le asigna un número único de sentencia y las funciones no modifican sus parámetros o variables globales. Para cada sentencia con S como número de sentencia:

DEF(S) = {X| la sentencia S contiene una definición de X}

USO(S)= {X| la sentencia S contiene un uso de X}

Si la sentencia S es una sentencia IF o de bucle, su conjunto DEF estará vacío y su conjunto USO estará basado en la condición de la sentencia S. La definición de una variable X en una sentencia S se dice que esta viva en una sentencia S' si existe un camino de la sentencia S a la sentencia S' que no contenga otra definición de X.

Una cadena de definición-uso (o cadena DU) de una variable X tiene la forma [X,S,S'], donde S y S' son números de sentencia, X esta en DEF(S) y en USO(S') y la definición de X en la sentencia S está viva en la sentencia S'.

- La Prueba de Bucles

Es una técnica de Prueba de Caja Blanca que se centra exclusivamente en la validez de las construcciones de bucles. Se pueden definir 4 clases diferentes de bucles:

Bucles simples: A los bucles simples de les debe aplicar el siguiente conjunto de de pruebas, donde n es el número máximo de pasos permitidos por el bucle:

- ✓ Pasar por alto totalmente el bucle.
- ✓ Pasar una sola vez por el bucle.
- ✓ Pasar dos veces por el bucle.
- ✓ Hacer m pasos por el bucle con m < n.
- ✓ Hacer n -1, n y n +1 pasos por el bucle.

Bucles anidados: A los bucles anidados se les debe aplicar el siguiente conjunto de de pruebas:

- ✓ Comenzar por el bucle más interior. Establecer o configurar los demás bucles con sus valores mínimos.
- ✓ Llevar a cabo la prueba de bucles simples para el bucle más interior, mientras se mantienen los parámetros de iteración. Añadir otra prueba para valores fuera de rango o excluidos.
- ✓ Progresar hacia afuera, llevando a cabo pruebas para el siguiente bucle, pero manteniendo todos los bucles externos en sus valores mínimos y los demás bucles anidados en sus valores típicos.
- ✓ Continuar hasta que se hayan probado todos los bucles.

Bucles Concatenados: Se pueden probar mediante el enfoque anteriormente definido para los bucles simples, mientras cada uno de los bucles sea más independiente del resto. Sin embargo, si hay 2 bucles concatenados y se usa el controlador del bucle 1 como valor inicial del bucle 2, entonces los bucles no son independientes. Cuando los bucles no son independientes, se recomienda usar el enfoque aplicado a los bucles anidados.

Bucles no estructurados: Siempre que sea posible, esta clase de bucles se deben rediseñar para que se ajusten a las condiciones de programación estructurada.

1.7. Antecedentes

La UCI se creó para desarrollar software para las diferentes ramas de la economía. Asegurar la calidad de los productos elaborados es una tarea primordial. Con este propósito varias facultades han desarrollado estrategias para probar que las aplicaciones desarrolladas por ella cuenten con toda la calidad que requieren. Las pruebas al código son un reto que poco a poco se trata de vencer. A continuación los diferentes trabajos de diploma que se han realizado en la UCI con este fin, que ayudarán de alguna forma a la elaboración del procedimiento.

1.7.1. Estrategia para PCB y PCN

Estrategia para la aplicación de Pruebas de Caja Blanca y Pruebas de Caja Negra al proyecto Registros y Notarias. (15)

Trabajo de diploma desarrollado por estudiantes de la Facultad #3. Propone una estrategia para aplicar Pruebas de Caja Blanca y Caja Negra al proyecto Registros y Notarias.

La estrategia para las Pruebas de Caja Blanca propone a RUP como metodología pues tiene un proceso de pruebas bien definido, así como un conjunto de artefactos. Expone una serie de roles que son responsables de diseñar, aplicar y dar seguimiento a todas las pruebas a realizar. La herramienta propuesta es la NUnit que es un framework que se puede ejecutar desde la consola o a través de una interfaz gráfica y se puede integrar con el Visual Studio en cualquiera de sus versiones.

El Área Temática desarrolla sus aplicaciones en el lenguaje PHP, por lo que la herramienta no se ajusta ya que se usa para aplicaciones que se desarrollan en la plataforma .NET.

1.7.2. Procedimiento General de PCB

Procedimiento general de Pruebas de Caja Blanca aplicando la técnica del Camino Básico. (16)

Trabajo de diploma desarrollado por estudiantes de la Facultad #7. Propone un procedimiento para aplicar Pruebas de Caja Blanca aplicando la técnica del Camino Básico.

El procedimiento presenta un conjunto de actividades a llevar acabo durante el proceso de realización de las pruebas y las diferentes plantillas para registrar los resultados de estas.

Las actividades y plantillas propuestas pueden utilizarse en el procedimiento que se va a elaborar ya que está bien explícito como se va a proceder, pero no propone una herramienta para la automatización de pruebas.

1.7.3. Herramienta para PCB

Implementación de herramientas para viabilizar el proceso de Pruebas de Caja Blanca. (17)

Trabajo de diploma desarrollado por estudiantes de la Facultad #7. Se encarga de la implementación de una herramienta para realizar Pruebas de Caja Blanca usando la técnica de Camino Básico.

La herramienta es una aplicación de escritorio que utilizando un analizador léxico, lee código fuente y de ahí, aplicando la técnica del Camino Básico obtiene casos de prueba.

Esta herramienta solo lee el código C# y el Área Temática utiliza el lenguaje PHP.

1.8. Flujo de Trabajo de Pruebas en APS

El flujo de trabajo puesto en práctica en APS cuenta con 6 actividades que garantizarán la eficiencia de las pruebas llevadas a cabo. (18)

Se *Planifican las Pruebas,* aquí se describe la estrategia de pruebas. Luego se *Analizan y Diseñan* donde se lleva a cabo un diseño de las pruebas y los diferentes casos de pruebas a implementar. Se construyen los procedimientos de prueba que especifican como realizar uno o varios casos de prueba, utilizando una serie de técnicas propias del tipo de prueba. Después se *Implementan***,** aquí se automatizan uno o varios procedimientos de prueba que permitan agilizar el proceso de pruebas. Concluida la implementación se *Ejecutan las Pruebas*. Finalmente se *Evalúan* los resultados de la prueba comparándolos con los objetivos trazados en el Plan de Pruebas. Si el resultado de la evaluación es negativo se regresa a planificar pruebas, repitiendo así el ciclo de vida del flujo de trabajo. Si el resultado es positivo, se pasa a *Aceptar las Pruebas*, cumpliendo así los objetivos y por último se termina con el proceso de liberación correspondiente por parte del grupo de desarrollo.

Actualmente no se lleva a cabo la automatización de las pruebas porque aún no se cuenta con herramientas que motoricen estos procesos.

1.9. Automatización de Prueba

La automatización de pruebas es uno de los mayores avances en la programación. Los conjuntos de casos de prueba garantizan que la aplicación hace lo que se supone que debe hacer. Incluso cuando el código interno de la aplicación cambia constantemente, las pruebas automatizadas permiten garantizar que los cambios no introducen incompatibilidades en el funcionamiento de la aplicación.

Se han creado un conjunto de herramientas que ayudan a realizar las pruebas al código y de esa forma disminuir tiempo y esfuerzo.

1.9.1. Symfony Lime

Symfony incluye su propio framework llamado Lime. Se basa en la librería Test::More de Perl y es compatible con TAP, lo que significa que los resultados de las pruebas se muestran con el formato definido en el *"Test Anything Protocol"*, creado para facilitar la lectura de los resultados de las pruebas. Este proporciona el soporte para las pruebas unitarias, es más eficiente que otros frameworks de pruebas de PHP. (19)

1.9.2. PHPUnit

Es una herramienta para la ejecución de pruebas al desarrollo de programas/sistemas desarrollados específicamente en PHP. Se trata de un miembro de la familia xUnit y proporciona un marco que realiza de manera fácil la escritura y la ejecución de las pruebas, así como el análisis de los resultados. Las pruebas son fáciles de aprender, de escribir y de ejecutar. (20)

1.9.3. TestNG

Es un framework para pruebas que trabaja con Java y está basado en JUnit (para Java) y NUnit (para .NET), pero introduciendo nuevas funcionalidades que lo hacen más poderoso y fácil de usar.

Es una herramienta de ejecución (y reporte de resultado) de pruebas unitarias. Esta herramienta ofrece una serie de opciones de configuración que permiten un control exhaustivo sobre la ejecución de las pruebas. En ocasiones se necesita que las pruebas se ejecuten en un cierto orden, primero la carga de contenidos, recuperación, seguido de actualización de los mismos y por último borrado. Estas

operaciones claramente requieren orden en su ejecución. JUnit, herramienta por excelencia para ejecución de pruebas unitarias. (21)

1.9.4. JSF UNIT

Es un framework para realizar pruebas de unidad sobre aplicaciones JSF basado en Cactus y JUnit. El framework, que se distribuye bajo una licencia libre, permite realizar tres tipos diferentes de pruebas: análisis estático de componentes JSF, análisis dentro del contenedor y análisis del ciclo de vida y del rendimiento de las páginas JSF. (22)

1.9.5. JR UNIT

JRunit es un proyecto para la evaluación basada en casos de prueba JUnit, además proporciona una extensión para este. Permite la distribución cliente /servidor basada en pruebas. JRunit no es un reemplazo para el popular framework experimental JUnit.

En lo que respecta al cliente / servidor basado en pruebas, hay ciertas limitaciones de JUnit que lo hacen poco práctico. La mayor limitación es que se ha diseñado para poder ejecutar todas las pruebas dentro de un proceso único dentro de una misma clase. Que también está diseñada con la restricción de que todas las pruebas atómicas y son independientes de cualquier otro código de la prueba fuera de plazo y que el ciclo de vida de estas pruebas son independientes para cada prueba. Esto es para todas las pruebas de unidad de código de bajo nivel. (23)

1.10. Arquitectura de APS

El Área Temática Atención Primaria de la Salud (APS) tiene como objetivo englobar todos los procesos de gestión de la información referente a este sector a través de una plataforma única para la gestión, procesamiento y transmisión de la información clínica en el SNS.
La plataforma fue desarrollada basada en componentes, lo cual posibilita una mayor independencia entre ellos. Se escogió el paradigma Arquitectura Orientada a Servicios (SOA), que define la utilización de los mismos para dar soporte a los requerimientos de software, empleando servicios web XML, específicamente SOAP.

Distribuye la aplicación en *Capas*: Capa *Presentación*, que contiene el diseño de las páginas de la aplicación. A través de la clase *Fachada* se comunica con la capa Negocio que contiene implementadas todas las funcionalidades y a su vez se comunica a través de la clase *PlaserDBZ* con la Capa de *Datos* que contiene la Base de Datos del módulo.

Estas están desplegadas en 3 servidores diferentes y el **Modelo-Vista-Controlador** que separa dentro de las capas definidas toda la lógica de negocio de las interfaces al usuario final, de manera que estas puedan ser cambiantes y adaptables.

Conclusiones

En el presente capítulo se realizó un análisis sobre las pruebas, lo que evidenció que la fase de pruebas acarrea mucho tiempo en el desarrollo del software; se concluye que no se debe esperar al final para realizar las pruebas sino según se va implementando. Las pruebas pueden encontrar fallos pero no aseguran que éstos no existen. Diseñar casos de prueba que encuentren errores antes no encontrados ayudará a mejorar la calidad del sistema. Evaluar las posibilidades de automatización de las pruebas de caja blanca, estudiar herramientas existentes y adaptarlas a las necesidades propias de cada Área Temática ahorrará tiempo y esfuerzo al equipo de desarrollo.

Capítulo 2: Propuesta de Procedimiento

En el presente capítulo se describe la propuesta de procedimiento para realizar Pruebas de Caja Blanca en el Área Temática APS. En el capítulo anterior se elaboró un marco teórico sobre las pruebas que permitió conocer los diferentes métodos, técnicas, tipos y niveles de prueba que aseguran la calidad del software, centrándose en las Pruebas de Caja Blanca, y aportando la comprensión necesaria para cumplir el objetivo trazado. El procedimiento está basado en el Flujo de Trabajo que rige el Área Temática. Se centra en definir qué trabajador realizará las pruebas; cómo, cuándo y dónde lo hará. Se realiza el estudio de varias herramientas de automatización con el fin de especificar cuales pueden ser utilizadas en APS.

2.1. Objetivos

- ✓ Escribir un procedimiento para la planificación, diseño, implementación y ejecución de las Pruebas de Caja Blanca en el Área Temática APS.
- ✓ Especificar los trabajadores que intervendrán, las actividades que se realizarán y artefactos que generará el procedimiento.
- ✓ Proponer una o varias herramientas que automaticen el trabajo de las pruebas.

2.2. Alcance

El procedimiento comprende aquellos productos de software que se encuentren en desarrollo en el Área Temática APS.

2.3. Definiciones, Acrónimos y Abreviaturas

APS: Atención Primaria de Salud.

Artefacto: Productos tangibles del proyecto, que son producidos, modificados y usados por las actividades.

Actividad: Tareas que tienen un propósito y se asigna a un rol.

Flujo de trabajo: Secuencia de actividades para producir determinados artefactos.

Trabajador: Roles que definen el comportamiento y responsabilidad de los individuos.

2.4. Roles y Responsabilidades

2.4.1. Administrador de Pruebas

Responsable del éxito que tendrá la prueba que dirige y organiza. Responsable de elaborar el Plan de Pruebas y el Cronograma de Pruebas.

2.4.2. Analista de Pruebas

Es responsable de identificar y definir las pruebas requeridas en una iteración, supervisando el progreso de las pruebas, así como el resultado para cada ciclo de prueba**.**

2.4.3. Diseñador de Pruebas

Se centra en identificar las técnicas apropiadas, herramientas y pautas para llevar a cabo las pruebas y usar los recursos correspondientes para el esfuerzo de las mismas. Especifica el enfoque, así como las funcionalidades que serán probadas en la presente iteración.

2.4.4. Probador

Ejecuta o produce las principales pruebas del software y es quien obtiene directamente los resultados.

2.5. Técnicas de prueba.

Para realizar las Pruebas de Caja Blanca se tendrán en cuenta dos técnicas de prueba: las pruebas de Condición y la prueba de Camino Básico ambas explicadas anteriormente en el epígrafe 1.6 del Capítulo 1. La *Técnica de Pruebas de Condición* se realizará de forma manual con el objetivo de verificar la integridad de las condiciones. Esto facilitará el trabajo para después aplicar la *Técnica del Camino Básico* que se encargará de obtener los casos de prueba. Esta técnica garantizará que durante la prueba se ejecute por lo menos una vez cada sentencia del programa.

2.6. Cronograma de Actividades

2.6.1. Planificar Prueba

En la actividad Planificar Prueba (Ver Figura 2.1) se reúnen los trabajadores involucrados **Administrador de Pruebas, Analista de Pruebas** y **Diseñador de Pruebas** para planificar las Pruebas de Caja Blanca para el Área Temática. Se describe la estrategia de prueba de la iteración

definiendo claramente el método apropiado para ejecutarlas, los objetivos, las posibles pruebas a realizar, las pautas a seguir y el enfoque de las mismas.

El objetivo de esta actividad es realizar el Plan de prueba que describe los recursos y la planificación de las mismas La actividad tiene como entrada el Plan de Desarrollo del Software que muestra la planificación de las actividades que realizarán los diferentes Grupos de Trabajo para el desarrollo de las aplicaciones.

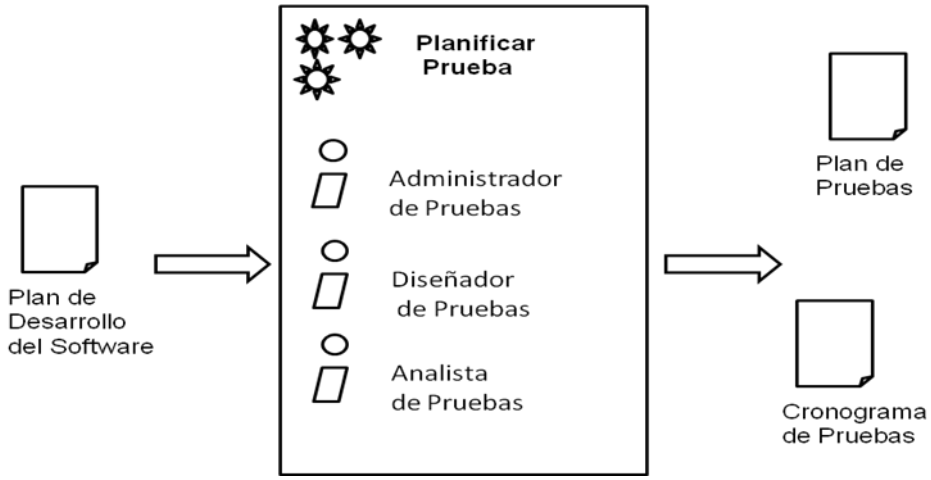


Figura 2.1: Entradas y salidas de la actividad Planificar Prueba.

El **Analista de Pruebas** identifica los objetivos y los tipos de prueba que realizarán. El **Diseñador de Prueba** especifica el enfoque, así como las funcionalidades que serán probadas en la presente iteración.

- Objetivos de las pruebas

✓ ***Verificar que las operaciones cumplan con los estándares definidos.***

Para cumplir este objetivo el Probador verificará los estándares de codificación, en la actividad Ejecutar Pruebas de acuerdo a lo planteado en las plantillas relacionadas con el tema que están bien definidas en el repositorio del Área Temática. Los defectos detectados al módulo se plasmarán en la plantilla Registro de No Conformidades (Ver Anexo E).

Para verificar los estándares se tendrán en cuenta las diferentes plantillas que se citan a continuación:

- APS_Estándares_Caracteres ExtrañosUso de los arreglos.
- APS_Estándares_Codificación

- APS_Estándares_Programar_CP

- ✓ ***Verificar que los algoritmos utilizados funcionen correctamente.***

La funcionalidad de los algoritmos se verificará mediante dos técnicas de prueba: la Prueba de Condición y la Prueba de Camino Básico. Ambas se realizaran de forma manual. En caso de poseer un componente de pruebas u otras herramientas que permitan agilizar este proceso, se hará uso de ellos.

La planificación quedará plasmada en la **Plantilla Plan de Pruebas** (Ver Anexo A) que elaborará el **Administrador de Pruebas** y contiene los siguientes epígrafes más significativos:

- ✓ **Introducción**: Breve descripción del módulo que se va a probar. Incluye el alcance, las **Definiciones, Acrónimos y Abreviaturas, además de las referencias.**
- ✓ **Organización del Equipo de Pruebas**: Descripción del equipo de probadores, por quienes está compuesto, responsabilidad de cada miembro.
- ✓ **Especificaciones del Software y Hardware**: Describe las características que debe tener el hardware y el software utilizado para ejecutar la aplicación.
- ✓ **Descripción del Plan de Pruebas**: Describe cómo serán planificadas las pruebas así como, las funcionalidades a probar.
- ✓ **Casos de Prueba**: Describe dónde estarán ubicadas las plantillas que contienen los diseños de casos de prueba especificando el caso de uso al que hacen referencia.
- ✓ **Estrategia de Prueba:** Describe cómo los objetivos de la prueba serán alcanzados para cada uno de los tipos de pruebas que hacen parte del plan. Además del Objetivo, la Técnica, el Entorno de Prueba, el Proceso, los Casos de Prueba y las Herramientas.
- ✓ **Recursos Requeridos**: Identifica los roles y las responsabilidades que serán requeridas para la ejecución del Plan de Pruebas.
- ✓ **Calendario y Plazos:** Documenta el plazo en el cual la aplicación a probar estará disponible para pruebas y el tiempo estimado para ejecutar los casos de prueba.
- ✓ **Definición de los Entregables:** Lista de entregables asociado a las pruebas.

- ✓ **Seguimiento y Reporte de Defectos:** Documenta el instrumento y el proceso usado para registrar y rastrear los defectos.
- ✓ **Aprobación del Plan:** Obtiene las firmas de aprobación en todas las páginas del mismo.
- ✓ **Documentación de los Resultados:** Cuando el esfuerzo de prueba esté terminado, se deben documentar los resultados y mediciones.

El **Administrador** también elaborará un Cronograma de Pruebas (Ver Anexo B). Esta será una plantilla nueva que se creará. Enunciará cuándo se planificarán, diseñarán, implementarán y ejecutarán las pruebas al código de acuerdo con el Plan de Desarrollo del Software.

2.6.2. Analizar y Diseñar Prueba

La actividad Analizar y Diseñar Prueba (Ver Figura 2.2) tiene como entradas los artefactos Plan de Pruebas y Cronograma de Pruebas además del Código fuente que saldrá según el Cronograma.

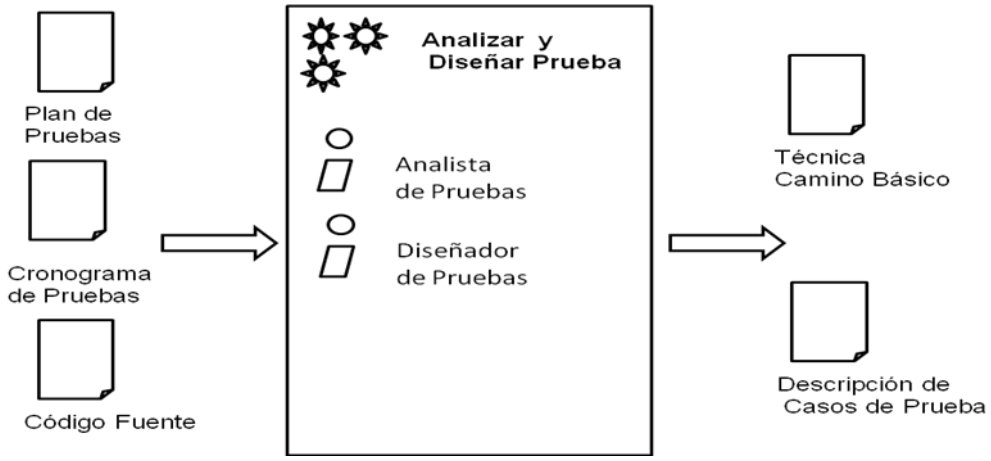


Figura 2.2: Entradas y salidas de la actividad Analizar y Diseñar Pruebas.

El **Analista de Pruebas** verifica la documentación, si no está completa la devuelve al desarrollador, en caso contrario el **Diseñador de Prueba**, que es el otro trabajador involucrado aplicará las técnicas de prueba escogidas guiándose por los siguientes epígrafes del Plan de Prueba:

- ✓ Alcance.
- ✓ Estrategia de prueba.
- ✓ Tipos de pruebas.

- ✓ Recursos.

Todo quedará reflejado en la Plantilla **Técnica Camino Básico** (Ver Anexo C). Esta es una plantilla nueva que se creará. Mostrará el flujo de las operaciones que ejecutará el Diseñador utilizando dicha técnica por cada segmento de código que se pruebe.

Partiendo de ésta se obtendrán los casos de prueba que serán descritos en la Plantilla **Descripción de Casos de Prueba** (Ver Anexo D) que constituye el principal artefacto de esta actividad. Se escogerá los caminos que van a ser automatizados y luego construirá los procedimientos para ejecutar las pruebas.

En la descripción de los casos de Prueba deben quedar definidos:

- ✓ Descripción General.
- ✓ Nombre de la Prueba.
- ✓ Descripción.
- ✓ Condiciones de ejecución.
- ✓ Entrada.
- ✓ Resultado esperado.
- ✓ Evaluación de la Prueba.

2.6.3. Implementar Prueba

El propósito de la actividad Implementar Prueba (Ver Figura 2.3) consiste en automatizar uno o varios procedimientos de pruebas. Tiene como entradas los artefactos Plan de Pruebas, las Plantillas Técnica Camino Básico y Descripción de Casos de Pruebas con las cuales el **Analista de Pruebas** selecciona los casos de prueba que requieren automatización y le informa al **Diseñador de Pruebas**. Este último se encargará de implementar un componente de pruebas, que introduciéndole los valores de entrada definidos en la Descripción de Casos de Prueba se encargará de verificar que se ejecuten los caminos obtenidos en el grafo de flujo derivado al aplicar la técnica del Camino Básico.

El **Componente de Pruebas** es una pequeña aplicación que proporciona entradas de pruebas, controlando y monitoreando los componentes a probar y de ser posible informa el resultado de las

pruebas. Muestra una interfaz sencilla, cuenta con los mismos campos de entrada que la aplicación original así como el código interno, con el fin de poder introducir los datos para realizar las pruebas. Al ser ejecutado el componente mostrará los mensajes correspondientes en cada caso.

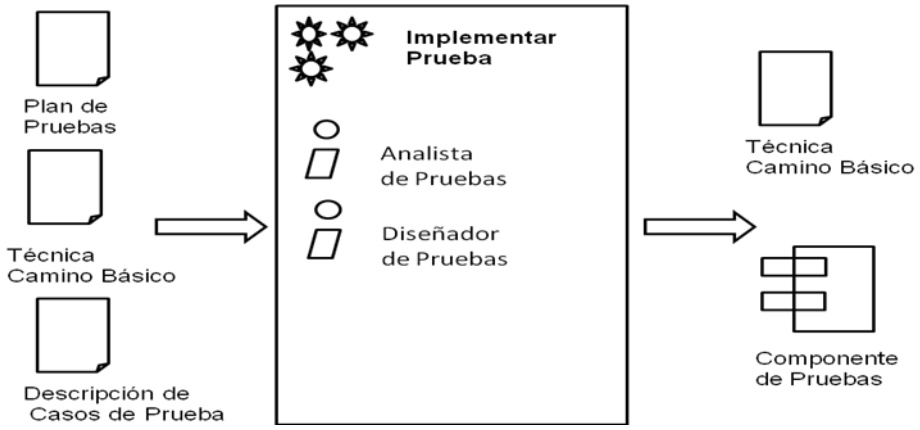


Figura 2.3: Entradas y salidas de la actividad Implementar Prueba.

2.6.4. Ejecutar las Pruebas

La actividad Ejecutar Prueba (Ver Figura 2.4) se centra en la ejecución de los casos de pruebas, a partir de las descripciones de estos, apoyándose de los componentes de pruebas, en el caso que sea necesario.

Tiene como entrada el Plan de Prueba, el Componente de Prueba y la Descripción de Casos de Prueba de Caja Blanca. El trabajador involucrado es el **Probador**, que ejecutará la prueba y reflejará en la Plantilla **Registro de No conformidades** (Ver Anexo E) todas las anomalías encontradas.

Las no conformidades encontradas deben ser resueltas en un plazo de 5 días. Luego se envía al equipo de Pruebas nuevamente para una nueva revisión. Este proceso puede repetirse hasta 3 veces.

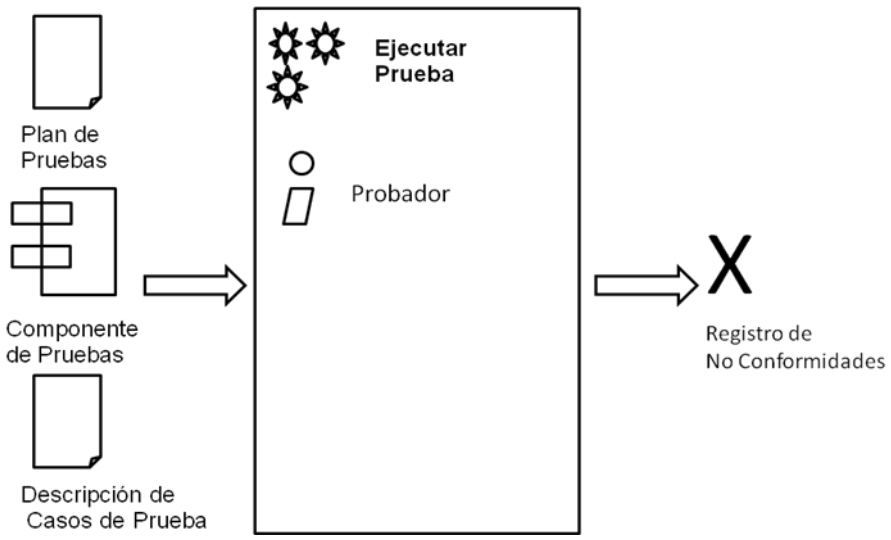


Figura 2.4: Entradas y salidas de la actividad Ejecutar Prueba.

2.7. Documentos de referencia.

- ✓ Procedimiento General de Pruebas de Caja Blanca aplicando la técnica del Camino Básico
- ✓ Proceso de Pruebas del Registro de Áreas de Salud de la Atención Primaria del Sistema de Información para la Salud.

2.8. Modelos

- ✓ Plan de Pruebas v 1.0. Facultad 7.UCI.2007.
- ✓ Registro de No Conformidades v 1.0. Facultad 7.UCI.2007.

2.9. Distribución y archivo

Todo esto se hará en formato digital y se ubicará en el repositorio del Área Temática(\\10.36.5.35\repositorio) en caso de ser posible se debe guardar además una copia impresa. Estos documentos sólo pueden ser modificados por el personal autorizado del Equipo de Calidad del Área Temática.

Técnica de Camino Básico: <Modulo>_TCB_<Sección>.

Ej: RAD_TCB_TipoActiv. (Ver Anexo G)

Descripción de Casos de Prueba: <Modulo>_ECPCB_<Funcionalidad>.

Ej.: RAD_DCPCB_InsertarTipoActiv. (Ver Anexo H)

2.10. Plantillas asociadas al Procedimiento

Plantillas tomadas de otros Trabajos de Diploma:

- ✓ Descripción de Casos de Prueba y Descripción de Casos de Prueba tomados del Trabajo de Diploma Proceso de Pruebas del Registro de Áreas de Salud de la Atención Primaria del Sistema de Información para la Salud (24).
- ✓ Formato para el Procedimiento tomado de: Procedimiento General PCB aplicando la Técnica de Camino Básico (25).

Plantillas estándares del Expediente de Proyecto:

- ✓ Plan de Pruebas
- ✓ Registro de No Conformidades

Plantillas creadas producto del presente trabajo:

- ✓ Cronograma de Pruebas
- ✓ Técnica Camino Básico

2.11. Herramientas

Para garantizar mayor calidad tecnológica se proponen 2 herramientas: Lime para las aplicaciones desarrolladas en el lenguaje PHP puesto que se consideró utilizar PHPUnit pero tiene la limitación de que requiere de PHP 5 o superior y los desarrollos utilizan PHP 4, y TestNG para las aplicaciones implementadas en el lenguaje Java teniendo en cuenta que actualmente el Área Temática está en proceso de migración hacia la tecnología Java. Ambas aludidas en los epígrafes 1.9.1 y 1.9.3 del Capítulo 1 respectivamente.

2.11.1. Symfony Lime v 1.2

Lime proporciona el soporte para las pruebas unitarias, es más eficiente que otros frameworks de pruebas de PHP y tiene las siguientes ventajas:

- ✓ Ejecuta los archivos de prueba en un entorno independiente para evitar interferencias entre las diferentes pruebas. No todos los frameworks de pruebas garantizan un entorno de ejecución "limpio" para cada prueba.
- ✓ Las pruebas de Lime son fáciles de leer y sus resultados también lo son. En los sistemas operativos que lo soportan, los resultados de Lime utilizan diferentes colores para mostrar de forma clara la información más importante.
- ✓ Symfony utiliza Lime para sus propias pruebas y su "prueba de regresión", por lo que el código fuente de Symfony incluye muchos ejemplos reales de pruebas unitarias y funcionales.
- ✓ El núcleo de Lime se valida mediante pruebas unitarias.
- ✓ Está escrito con PHP, es muy rápido y está bien diseñado internamente. Consta únicamente de un archivo, llamado lime.php, y no tiene ninguna dependencia.

Las pruebas unitarias de Symfony (Ver Anexo K)son archivos PHP normales cuyo nombre termina en `Test.php` y que se encuentran en el directorio `test/unit/` de la aplicación. Su sintaxis es sencilla y fácil de leer.

Para las pruebas unitarias el objeto `lime_test` dispone de un gran número de métodos como los que se muestran a continuación:

- ✓ diag ($mensaje): Muestra un comentario, pero no ejecuta ninguna prueba.
- ✓ ok($prueba, $mensaje): Si la condición que se indica es true, la prueba tiene éxito
- ✓ is ($valor1, $valor2, $mensaje): Compara 2 valores y la prueba pasa si los 2 son iguales.
- ✓ isnt ($valor1, $valor2, $mensaje): Compara 2 valores y la prueba pasa si no son iguales.
- ✓ like ($cadena, $expresión Regular, $mensaje): Prueba que una cadena cumpla con el patrón de una expresión regular.
- ✓ unlike ($cadena, $expresión Regular, $mensaje): Prueba que una cadena no cumpla con el patrón de una expresión regular.
- ✓ cmp_ok ($valor1, $operador, $valor2, $mensaje): Compara 2 valores mediante el operador que se indica.

- ✓ isa_ok ($variable, $tipo, $mensaje): Comprueba si la variable que se le pasa es del tipo que se indica.
- ✓ isa_ok ($objeto, $clase, $mensaje): Comprueba si el objeto que se le pasa es de la clase que se indica.
- ✓ can_ok ($objeto, $método, $mensaje): Comprueba si el objeto que se le pasa dispone del método que se indica.
- ✓ is_deeply ($array1, $array2, $mensaje): Comprueba que 2 arrays tengan los mismos valores.
- ✓ include_ok ($archivo, $mensaje): Valida que un archivo existe y que ha sido incluido correctamente.
- ✓ fail (): Provoca que la prueba siempre falle.
- ✓ fass (): Provoca que la prueba siempre se pase.
- ✓ skip ($mensaje, $numeroPruebas): Cuenta como si fueran $numeroPruebas pruebas.
- ✓ todo (): Cuenta como si fuera 1 prueba.

Para ejecutar el conjunto de pruebas, se utiliza la tarea test-unit desde la línea de comandos. El resultado de esta tarea en la línea de comandos es muy explícito, lo que permite localizar fácilmente las pruebas que han fallado y las que se han ejecutado correctamente.

A continuación la tarea Test-Unit:

// Estructura del directorio de pruebas

test/

unit/

miFuncionalTest.php

miSegundoFuncionalTest.php

otro/

nombreTest.php

Para visualizar el resultado de pruebas en la línea de comandos se realiza lo siguiente:

> symfony test-unit miFuncional ## Ejecutar miFuncionalTest.php

> symfony test-unit miFuncional miSegundoFuncional ## Ejecuta las 2 pruebas

> symfony test-unit 'otro/*' ## Ejecuta nombreTest.php

> symfony test-unit '*' ## Ejecuta todas las pruebas (de forma recursiva)

A través de un estudio detallado de la herramienta, cuyos aspectos principales se han mencionado anteriormente, se encuentran los elementos necesarios para analizar y determinar su posible utilización en el Área Temática APS. Entre ellos, existe un grupo de especificaciones que entran en contradicción con la arquitectura de APS (Ver epígrafe 1.10).

Lime concibe las pruebas a los métodos y funciones fuera de su localización natural, lo que deben ser separados a la carpeta “Lib”, y accede desde el archivo de pruebas de forma directa (require_once(dirname(__FILE__).'/../../lib/strtolower.php') no siendo posible especificar la dirección del método a probar a través de la red (\\10.36.5.35\rad_core\CodigoRAD\InsertarActividadesEBS). Para el desarrollo en APS se utiliza el framework PlaSer, y establece una estructura de carpetas (proxpla, web, core). En este último se encuentran los métodos del negocio, que para su correcto funcionamiento no pueden ser sacados de su ambiente.

Otro inconveniente es que los métodos a probar deben recibir los parámetros de entrada de forma directa y devolver los resultados de igual forma, en los registros desarrollados, desde presentación a través de la clase Fachada, son invocados lo métodos del negocio y estos devuelven los resultados a través de un mensaje SOAP.

Por las razones expuestas anteriormente se concluyó que la versión Symfony Lime v.1.2 no se ajusta al desarrollo del Área Temática, razón por la que no puede ser utilizado en la automatización de Pruebas.

De todas formas, el tiempo dedicado no ha sido en vano, el resultado del estudio puede ser utilizado por quienes apuestan por Lime como una guía para decidir si su utilización es factible o no, además de analizar si su arquitectura entra en contradicción con las especificaciones de la herramienta. Quienes superen este paso, encontrarán el material necesario para realizar la automatización de pruebas con Lime.

2.11.2. TestNG

TestNG es un framework para pruebas y que trabaja con Java. Está basado en JUnit (para Java) y NUnit (para .NET), pero introduciendo nuevas funcionalidades que los hacen más poderosos y fáciles de usar, tales como:

- ✓ Anotaciones JDK 5.
- ✓ Configuración flexible de pruebas.
- ✓ Soporte para pruebas para data-driven testing (con @DataProvider).
- ✓ Soporte de pasaje de parámetros.
- ✓ Permite distribución de las pruebas en máquinas esclavas.
- ✓ Modelo de ejecución poderoso (TestSuite nunca más).
- ✓ Soportado por herramientas y plugins importantes y variados como: (Eclipse, IDEA, Maven, etc.).
- ✓ Permite embeber BeanShell para una flexibilidad más amplia.
- ✓ Funciones JDK por defecto de runtime y logging. (sin dependencias)
- ✓ Métodos dependientes para pruebas sobre servidores de aplicación.

TestNG está diseñado para cubrir todas las categorías de las pruebas: unitarias, funcionales, integración, etc.

Escribir una prueba suele ser un proceso de tres pasos:

- ✓ Escribir la lógica de negocio de sus pruebas e insertar anotaciones TestNG en el código.
- ✓ Añadir la información acerca de la prueba (por ejemplo, el nombre de la clase, los grupos que desea ejecutar, etc.) en un archivo build.xml. o en testng.xml
- ✓ Ejecutar TestNG.

Los conceptos utilizados son los siguientes:

- ✓ Una suite está representada por un archivo XML. Puede contener una o más pruebas y se define mediante la etiqueta <suite>.
- ✓ Una prueba está representada por la etiqueta <test> y puede contener una o más clases TestNG.
- ✓ Una clase TestNG es una clase Java que contiene al menos una anotación TestNG. Es representada por la etiqueta <class> y puede contener uno o más métodos de prueba.
- ✓ Un método de prueba es un método de Java anotado por @Test en su fuente de prueba.

¿Cómo se hace una prueba?

- ✓ Se van a crear clases con el nombre de la clase que se va a probar y se le pone la terminación "test".
- ✓ Dentro de la clase habrá métodos de prueba.
- ✓ Tendrá que llevar la anotación test "@test" para indicar que un método es de prueba.
- ✓ Se crea un objeto de la clase a probar, se hace algo y se comprueba que el resultado es correcto.
- ✓ Para lanzar la prueba está el código de eclipse: clic derecho encima de la prueba/run as testNG Test.

A continuación un resumen de las anotaciones disponible en TestNG junto con sus atributos.

Funciones	Configuración de la información para una clase de TestNG:
@BeforeSuite	El método anotado se ejecutará antes que todas las pruebas en esta suite.
@AfterSuite	El método anotado se ejecutará después que todas las pruebas en esta suite.
@BeforeTest	El método anotado se ejecutará antes que la prueba.

@AfterTest	El método anotado se ejecutará después que la prueba.
@BeforeGroups	La lista de grupos que el método de configuración ejecutará antes. Este método funciona poco antes de que se invoque el primer método de prueba que pertenece a alguno de estos grupos
@AfterGroups	La lista de grupos que el método de configuración ejecutará después. Este método funciona poco después de que se invoque el último método de prueba que pertenece a alguno de estos grupos.
@BeforeClass	El método anotado se ejecutará antes del primer método de la clase actual invocada.
@AfterClass	El método anotado se ejecutará después que todos los métodos de la clase actual se hayan ejecutado.
@BeforeMethod	El método anotado se ejecutará antes de cada método de prueba.
@AfterMethod	El método anotado se ejecutará antes de cada método de prueba.
alwaysRun	Para antes de los métodos (beforeSuite, beforeTest, beforeTestClass y beforeTestMethod, pero no beforeGroups): Si se le asigna verdadero, este método de configuración se llevará a cabo independientemente de los que pertenezcan a los grupos. Para después de los métodos (afterSuite, afterClass, ...) Si se le asigna verdadero, este método de configuración se llevará a cabo incluso si uno o más métodos invocado anteriormente falla o se salta.
dependsOnGroups	La lista de grupos de que depende el método.
dependsOnMethods	La lista de métodos de que depende el método.
enabled	Si los métodos de esta clase o método están habilitados.
groups	La lista de grupos a los que esta clase o método pertenece.

inheritGroups	Si es verdadero, este método pertenecerá a los grupos especificados en la anotación @Test en la clase nivel.
@DataProvider	Marcar un método como suministro de datos para un método de pruebas. El método debe devolver un objeto [] [], donde a cada objeto [] se le puede asignar la lista de parámetros del método de pruebas. El Método @Test que quiere recibir los datos de este proveedor de datos necesita usar un nombre de proveedor de datos igual al nombre de esta anotación.
name	El nombre del proveedor de datos.
@Factory	Marca un método como una fábrica que retorna objetos que serán usados por TestNG como clase de pruebas. El método debe retornar un objeto.
@Parameters	Describe como se le pasan los parámetros a un método @Test.
value	La lista de variables usadas para llenar los parámetros de este método.
@Test	Marca una clase o un método como parte de una prueba.
alwaysRun	Si se le asigna verdadero, este método de prueba siempre se ejecutará incluso si depende de un método que falla.
dataProvider	El nombre del proveedor de datos para este método de pruebas.
dataProviderClass	La clase donde buscar el proveedor de datos. Si no se especifica, el proveedor de datos se buscará en la clase del actual método de pruebas o en una de sus clases base. Si este atributo se especifica, el proveedor de datos debe ser estático en la clase especificada.

dependsOnGroups	La lista de grupos de la cual el método depende.
dependsOnMethods	La lista de métodos de la cual el método depende.
description	La descripción de este método.
enabled	Si los métodos de esta clase o método están inhabilitados.
expectedExceptions	La lista de excepciones que se espera que un método de prueba arroje. Si no es una excepción o es diferente a las de esta lista, la prueba fracasó.
groups	La lista de grupos a los que esta clase o método pertenece.
invocationCount	El número de veces que el método debe ser invocado.
invocationTimeOut	El número máximo de milisegundos que debe tomar esta prueba por el tiempo acumulado de todas las invocaciones. Este atributo será ignorado si no se especifica invocationCount.
successPercentage	El porcentaje de éxito esperado de este método.
sequential	Si es asignado verdadero, todos los métodos de la clase de pruebas están garantizados para funcionar en secuencia, incluso si las pruebas se encuentran actualmente en ejecución en paralelo = "verdadero". Este atributo sólo se puede utilizar en el nivel de clase y se ignora si se utiliza en el nivel de método.
timeOut	El numero máximo de milisegundos que esta prueba debe tomar.
threadPoolSize	El tamaño de la piscina de hilos de este método. El método será invocado desde múltiples hilos, tal como se especifica por invocationCount. Nota: este atributo se ignora si no se especifica invocationCount

Conclusiones

En este capítulo se ha descrito la propuesta de un procedimiento para realizar Pruebas de Caja Blanca por el grupo de calidad del Área Temática APS. Este muestra cómo se realizarán las diferentes actividades que comprende el flujo de trabajo de Pruebas del Área Temática definido, enfocadas a las Pruebas de Caja Blanca. Se especifica la actividad que realizará cada trabajador y los artefactos de entrada y salida. La herramienta de automatización estudiada para los desarrollos en PHP se no se ajusta al desarrollo del software en el Área Temática. Debido a esto se propone una herramienta para el lenguaje Java, pues los nuevos desarrollos serán implementados en este lenguaje.

El formato usado para el diseño de este procedimiento cumple con los estándares definidos en la Norma ISO 9000 del 2005. *(Ver Anexo F).*

Capítulo 3: Evaluación de Procedimiento Propuesto

En el presente capítulo se realiza la evaluación del procedimiento propuesto. Para ello se le aplica todo el proceso de pruebas de Caja Blanca al Registro de Actividades Diarias (RAD) de la Atención Primaria de Salud, módulo desarrollado en el Área Temática.

3.1. Registro de Actividades Diarias (RAD)

El Registro de Actividades Diarias es un sistema para la gestión de la información sanitaria asociada a las actividades asistenciales (ambulatorias, de urgencia y con las especialidades básicas) y no asistenciales realizadas por el personal de salud de la Atención Primaria de Salud y su procesamiento para la toma de decisiones en la reunión mensual del área de salud y la entrega de guardia. Permite el seguimiento al paciente, y brinda información para el Análisis de la Situación de Salud del área, apoyando la vigilancia de salud y el seguimiento y control de desempeño del personal médico.

Contempla:

- ✓ Registro de la información sanitaria en las consultas y terrenos realizadas por los médicos de la familia e interconsultas con las especialidades básicas.
- ✓ Registros de las actividades no asistenciales de apoyo a la salud realizadas por los médicos y enfermeras de la familia.
- ✓ Consultar las actividades realizadas por el personal de la salud de la APS.
- ✓ Consultar la información sanitaria de actividades asistenciales desarrolladas para el seguimiento del paciente.
- ✓ Procesamiento de la información para la toma de decisiones en la reunión mensual del personal que atiende un área de salud.
- ✓ Procesamiento de la información para la entrega de guardia en instituciones que brindan servicio de urgencia.

3.2. Procedimiento de Pruebas de Caja Blanca al RAD

3.2.1. Planificar Pruebas

Se realizarán Pruebas de Caja Blanca al RAD con el objetivo de buscar errores. Las pruebas estarán enfocadas a verificar la funcionalidad de los métodos implementados por el grupo de trabajo. Para ello se realiza la primera iteración del Plan de Pruebas para varias funcionalidades que en él se describen.

- Plan de Pruebas RAD

1. Introducción

En el presente documento se planifican las Pruebas de Caja Blanca para el Registro de Actividades Diarias (RAD), a partir de las funcionalidades del mismo.

1.1. Alcance

Este Plan se elabora con el fin de planificar las Pruebas de Caja Blanca al RAD de la Atención Primaria de Salud.

1.2. Definiciones, Acrónimos y Abreviaturas

RAD- Registro de Actividades Diarias.

APS -Atención Primaria de Salud.

1.3. Referencias

Código	*Título*
[1]	*RAD_Plan de Desarrollo de Software*

2. **Organización del Equipo de Pruebas**

- ✓ **Administrador de Pruebas:** Responsable del éxito de la prueba. Dirige las actividades en el proceso de pruebas.

  Est Dayanis Isaac Morales.

- ✓ **Analista de Pruebas:** Responsable de identificar los tipos y técnicas de pruebas que se van a realizar.

Est Dayanis Isaac Morales.

- ✓ **Diseñador de prueba:** Planifica, diseña y evalúa los resultados de la prueba.

  Est Dayanis Isaac Morales.

- ✓ **Probadores:** Realizan las pruebas unitarias, de integración y sistema.

  Est Dayanis Isaac Morales.

3. Especificaciones del Software y Hardware

Hardware:

- ✓ Un Servidor de Base de datos
- ✓ Un Servidor de Aplicación
- ✓ Un Servidor Web.
- ✓ Una PC Cliente
- ✓ Servicio de red.

Software

- ✓ Sistema operativo Windows o Linux.
- ✓ Navegador Mozilla 1.5 y Internet Explorer 5.0 o Superior
- ✓ Instalación de PDF y Excel.

4. Descripción del Plan de Pruebas

Las pruebas serán planificadas a partir de las funcionalidades del sistema.

En el nivel Nacional se verificarán los codificadores con sus opciones de Insertar, Modificar, Eliminar y Listar. Ellos son:

- ✓ **Codificador Tipo de Actividad:** Muestra los tipos de actividades que realiza el personal de salud como son: consulta, terreno, etc.

- ✓ **Codificador Actividades del EBS:** Muestra las actividades no asistenciales que realiza el personal de salud.

En el nivel Unidad de Salud se verificarán las siguientes funcionalidades con sus opciones de Insertar y Listar:

- ✓ **Otras Actividades EBS**: Gestiona la información de las actividades no asistenciales que realiza el personal de salud con el objetivo de evaluar su desempeño.

5. Casos de Prueba

A continuación se muestra los diferentes Casos de Uso a los que se hace alusión en la ejecución de las Pruebas de Caja Blanca, especificando los escenarios de cada uno.

Caso	*Datos de la Prueba*				*Resultado Esperado*
1	*Caso de Uso*	*Escenario-Condición*	*Caso de Prueba*		
	Gestionar Actividades EBS	*Insertar Actividades EBS*	*DCPCB_InsActvEBS*		
		Eliminar Actividades EBS.	*DCPCB_ElimActvEBS*		
		Modificar Actividades EBS	*DCPCB_ModActvEBS*		
		Listar Actividades EBS	*DCPCB_ListActvEBS*		
2	*Caso de Uso*	*Escenario-Condición*	*Caso de Prueba*		
	Gestionar Otras Actividades EBS	*Insertar OtrasActividades EBS*	*DCPCB_InsOtrasActvEBS*		

		Listar Otras Actividades EBS	*DCPCB_LisOtrastActvEBS*		
3	*Caso de Uso*	*Escenario-Condición*	*Caso de Prueba*		
	Gestionar Tipo Actividad	*Insertar Tipo Actividad*	*DCPCB_InsTipoActv*		
		Eliminar Tipo Actividad	*DCPCB_ElimTipoActv*		
		Modificar Tipo Actividad	*DCPCB_ModTipoActv*		
		Listar Tipo Actividad	*DCPCB_ListTipoActv*		

6. Estrategia de Prueba

Este plan contempla pruebas unitarias, donde específicamente se verificó el cumplimiento estricto con los estándares de codificación y la funcionalidad de los métodos implementados hasta el momento, usando el método de prueba de caja blanca, siendo diseñados los casos de pruebas con el fin de verificar los procesos del sistema funcionen correctamente.

6.1. Objetivo

Pruebas Unitarias: En esta iteración se comprueban las funcionalidades de los métodos. Se verifica que las operaciones cumplan con los estándares establecidos en el Proyecto y como funcionan los métodos de la aplicación en desarrollo.

6.2. Técnica

Para realizar las Pruebas de Caja Blanca se tendrán en cuenta dos técnicas de prueba: las pruebas de condición y la prueba de Camino Básico. La *Técnica de Pruebas de Condición* que facilitará el trabajo para después aplicar la *Técnica del Camino Básico*. Esta técnica garantizará que durante la prueba se ejecute por lo menos una vez cada sentencia del programa.

6.3. Proceso

El proceso de pruebas en el proyecto funciona de la siguiente forma:

A medida que se validan las descripciones de los casos de uso del sistema es decir, que los requerimientos funcionales son aprobados por el cliente, el Planificador de Pruebas procede a planificar las pruebas apoyándose del Plan de Desarrollo del Software. Elaborado el Plan de Pruebas los Diseñadores de Prueba comienzan a diseñar los casos de pruebas. Por último los Probadores ejecutan estos casos de pruebas elaborando el Resumen de No Conformidades como resultado final, donde se redactan los resultados obtenidos. Estos son enviados al Líder de proyecto y al equipo de desarrollo.

6.4. Casos de Prueba

Todos los casos de pruebas se encuentran en la siguiente dirección:

http://10.36.5.35/repositorio/trunk/RAD/Expediente_del_proyecto/Ingenieria/Implementacion_y_pruebas

6.5. Herramientas

En la presente iteración no se utilizaran herramientas de automatización. Solo un componente de Pruebas que permitirá confirmar la validez de las funcionalidades.

7. Recursos Requeridos

Roles	Cantidad	Responsabilidad
Administrador de Pruebas	1	Identifica, prioriza e implementa los casos de prueba. Genera el Plan de prueba para la iteración. Negocia o acuerda las pruebas. Genera resumen de resultado de prueba. Verifica el progreso y efectividad de las pruebas Genera informe de no conformidades.

Analista de Pruebas	1	Genera procedimientos de prueba. Identifica los casos prueba. Confecciona la lista de ideas para las pruebas. Genera documento con datos de pruebas.
Diseñador de Pruebas	1	Confecciona casos de pruebas. Genera las descripciones de los casos de pruebas. Genera modelo de prueba. Componente de prueba. Especificación de interfaz de prueba. Configuración del ambiente de prueba. Define la suite de pruebas. Identifica y describe apropiadas técnicas de prueba. Identifica herramientas apropiadas de soporte. Define y mantiene la arquitectura de automatización de pruebas. Especifica y verifica las condiciones de las configuraciones de prueba.
Probador	1	Ejecuta las pruebas Registra los resultados de las pruebas. Documenta los pedidos de cambio. Implementa pruebas individuales. Defectos.

Verifica la forma en que se ejecutan las pruebas.

8. Calendario y Plazos.

Para saber cuando la aplicación a probar estará disponible para realizar las pruebas ver la plantilla Cronograma de Pruebas.

9. Definición de los Entregables.

- ✓ Plan de Pruebas
- ✓ Registro de No conformidades
- ✓ Técnica Camino Básico
- ✓ Descripción de Casos de Prueba
- ✓ Cronograma de Pruebas.

10. Seguimiento y Reporte de Defectos.

Los defectos encontrados se registrarán en la plantilla Registro de No Conformidades. Estas deben ser resueltas en un plazo de cinco (5) días. Luego se envía al equipo de Pruebas nuevamente para una nueva revisión. Este proceso de repetirá solo 3 veces.

11. Aprobación del Plan.

Se realiza una reunión donde están involucrados los miembros del Equipo de Calidad y ha quedado aprobado el Plan.

12. Documentación de Resultados

Los resultados estarán ubicados en el repositorio del Área Temática. Cada miembro del equipo de calidad se encargará de jugar su rol en el llenado de las plantillas correspondientes.

Una vez concluido el plan se pasa a elaborar el Cronograma de Pruebas, el cual se muestra a continuación:

- Cronograma de Pruebas RAD

1. Introducción

El presente Cronograma de Pruebas muestra la fecha en que se diseñarán, implementarán y ejecutarán las pruebas al módulo Registro de Actividades Diarias.

1.1. Alcance

El presente cronograma se aplicara al modulo RAD del Área Temática APS.

1.2. Objetivos

El presente cronograma se elabora con el objetivo de conocer la fecha en la que se van a realizar el proceso de pruebas a los módulos desarrollados en el Área Temática APS.

1.3. Referencias

Plan de Pruebas RAD ubicado en:

2. Cronograma de Pruebas

Módulo	**Funcionalidad**	**Fecha lista**	**Fecha de prueba**		
			Diseño	**Implementación**	**Ejecución**
RAD	RAD_InsertarActvEBS	10/04/09	24/04/09	8/05/09	22/05/09
	RAD_EliminArctvEBS	13/04/09	27/04/09	11/05/09	27/05/09
	RAD_ActualizaActvrEBS	15/04/09	29/04/09	13/05/09	27/05/09
	RAD_ListarActvEBS	17/04/09	1/05/09	15/05/09	29/05/09
	RAD_InsertarOtrasActvEBS	10/04/09	24/04/09	8/05/09	22/05/09
	RAD_EliminOtrasArctvEBS	13/04/09	27/04/09	11/05/09	27/05/09
	RAD_ActualizaOtrasActvrEBS	15/04/09	29/04/09	13/05/09	27/05/09
	RAD_ListarOtrasActvEBS	17/04/09	1/05/09	15/05/09	29/05/09

	RAD_InsertarTipoActv	10/04/09	24/04/09	8/05/09	22/05/09
	RAD_EliminTipoArctv	13/04/09	27/04/09	11/05/09	27/05/09
	RAD_ActualizaTipoActvr	15/04/09	29/04/09	13/05/09	27/05/09
	RAD_ListarTipoActv	17/04/09	1/05/09	15/05/09	29/05/09

3.2.2. Analizar y Diseñar Pruebas

De acuerdo al Plan de Pruebas y contando con el código fuente se realizarán las técnicas de pruebas para definir los casos de prueba.

Las **Pruebas de Condición** se realizan de forma manual (Ver Anexo J) y quedarán de la siguiente forma:

Condición (C_1): if ($lugar == "" ||$actividad == "" ||$fecha == "" ||$tiempo == "")

Restricciones de C_1: del tipo (D_1, D_2, D_3, D_4).

Conjunto de restricciones D: {(lugar, actividad, fecha, “”), (lugar, actividad, “”, tiempo), (lugar, “”, fecha, tiempo), (“”, actividad, fecha, tiempo)}.

Estas restricciones se verificarán tras la ejecución de la prueba.

Luego de realizar las pruebas de condición se pasa a realizar la prueba del camino básico plasmando en la plantilla Técnica Camino Básico el flujo de pasos asociados a ésta. Obtenidos los casos de prueba se especifican los datos de entrada que serán insertados en los componentes de pruebas que se implementarán usando como entrada en la plantilla Descripción de Casos de Prueba.

- Funcionalidad Insertar Actividades del Equipo Básico de Salud.

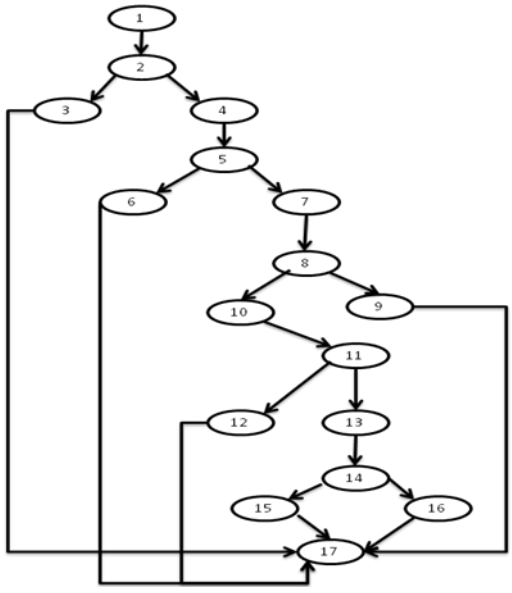
a) Segmento de código: RAD_ InsertarActividadesEBS

```
1<?php
2if(!function_exists('plaser_handler')) 3die('Violación de acceso.');
4require 'PLASER/dbz_class.php';
```

```
4require_once '../general/validar.php';
4global $espacio;
4$methodname = "InsertarActividadesEBS";
4$db = new DBz('m');
4$actividadesEBS = utf8_decode($args['actividad_ebs']);
4$actividadesEBS1 = utf8_decode($args['actividad_ebs']);
5if(!isset($args['actividad_ebs']))
 6return $plaser->fault('ALGUN PARAMETRO NO FUE CREADO....',101);
7$Arraydatos = array($actividadesEBS1);
8if (!ValCadenas($Arraydatos))
{
 9 return $plaser->fault('PARAMETRO INCORRECTO, DEBE SER UNA CADENA...', 102);
}
10$consulta="SELECT *
10FROM
10 `tb_actividades_ebs`
10WHERE
10 (`tb_actividades_ebs`.actividad)='$actividadesEBS'";
10$Arrayresultado = $db->query($consulta);
11if($Arrayresultado->rows > 0)
{
12 return $plaser->fault('LA ACTIVIDAD YA EXISTE.',103);
```

```
}
13 $consulta = "INSERT INTO `tb_actividades_ebs`(actividad) VALUES('$actividadesEBS')";
13 $Arrayrespuesta = $db->query($consulta);
14 if($Arrayrespuesta != 1)
{
 15 $modulo='APS-RAD-Actualizar_Actividades_EBS';
 15 $message=DevuelveMensajeError($Arrayrespuesta,$modulo);
 15 $plaser->fault(($message), 113);
}
else
 16return true;
?>17
```

b) Grafo de flujo



c) Cálculo de la Complejidad Ciclomática

V (G) = *Número de regiones*. **6**

V (G) = Aristas - Nodos + 2. ***21-17+2 = 6***

V (G) = Nodos predicado + 1. ***5*** **+ 1 = 6**

d) Conjunto básico de caminos independientes

Camino 1: 1-2-4-5-7-8-10-11-13-14-15-17.

Camino 2: 1-2-4-5-6-17.

Camino 3: 1-2-4-5-7-8-10-11-13-14-16-17.

Camino 4: 1-2-4-5-7-8-10-11-12-17.

Camino 5: 1-2-4-5-7-8-9-17.

Camino 6: 1-2-3-17.

e) Casos de prueba

Caso de Prueba 1: La actividad no se insertó.

Caso de Prueba 2: No se creó un parámetro.

Caso de Prueba 3: Se inserta la actividad correctamente.

Caso de Prueba 4: La actividad que se inserta está repetida.

Caso de Prueba 5: El parámetro de entrada no es una cadena.

Caso de Prueba 6: La función “plaser_handler” no existe.

f) Para cada uno de los casos de prueba anteriores se muestra la Descripción de Casos de Prueba.

- ✓ ***CP1_RAD_ InsertarActividadesEBS: La actividad no se insertó.***

Descripción

Para la ejecución de esta prueba se ejecuta el componente Agregar Actividades del EBS como resultado de la actividad Implementar Prueba.

Flujo central

- ✓ El componente de prueba muestra la interfaz con las diferentes opciones.
- ✓ El usuario selecciona la opción Agregar para insertar los datos.
- ✓ El componente muestra el campo Actividad que es donde se insertarán los datos.
- ✓ El usuario inserta los datos. Clic en Aceptar.
- ✓ Componente muestra resultado de la acción.

Condiciones de ejecución

Los campos de carácter obligatorio deben estar llenos.

Entrada

En el campo actividad = “Actividad1”.

Resultados esperados

Mensaje: “Error 113 que responde a la no inserción de la actividad”.

Evaluación de la Prueba

Prueba satisfactoria.

- ✓ ***CP2_RAD_ InsertarOtrasActividadesEBS: No se creó un parámetro.***

Descripción

Para la ejecución de esta prueba se ejecuta el componente Agregar Actividades del EBS como resultado de la actividad Implementar Prueba.

Flujo central

- ✓ El componente de prueba muestra la interfaz con las diferentes opciones.
- ✓ El usuario selecciona la opción Agregar para insertar los datos.
- ✓ El componente muestra el campo Actividad que es donde se insertarán los datos.
- ✓ El usuario inserta los datos. Clic en Aceptar.
- ✓ Componente muestra resultado de la acción.

Condiciones de ejecución

Los campos de carácter obligatorio deben estar llenos.

Entrada

En el campo actividad = “”.

Resultados esperados

Mensaje: “Error 101 que responde a que algún parámetro no fue creado”.

Evaluación de la Prueba

Prueba satisfactoria.

- ✓ ***CP3_RAD_ InsertarOtrasActividadesEBS: Se inserta la actividad correctamente.***

Descripción

Para la ejecución de esta prueba se ejecuta el componente Agregar Actividades del EBS como resultado de la actividad Implementar Prueba.

Flujo central

- ✓ El componente de prueba muestra la interfaz con las diferentes opciones.
- ✓ El usuario selecciona la opción Agregar para insertar los datos.
- ✓ El componente muestra el campo Actividad que es donde se insertarán los datos.
- ✓ El usuario inserta los datos. Clic en Aceptar.
- ✓ Componente muestra resultado de la acción.

Condiciones de ejecución

Los campos de carácter obligatorio deben estar llenos.

Entrada

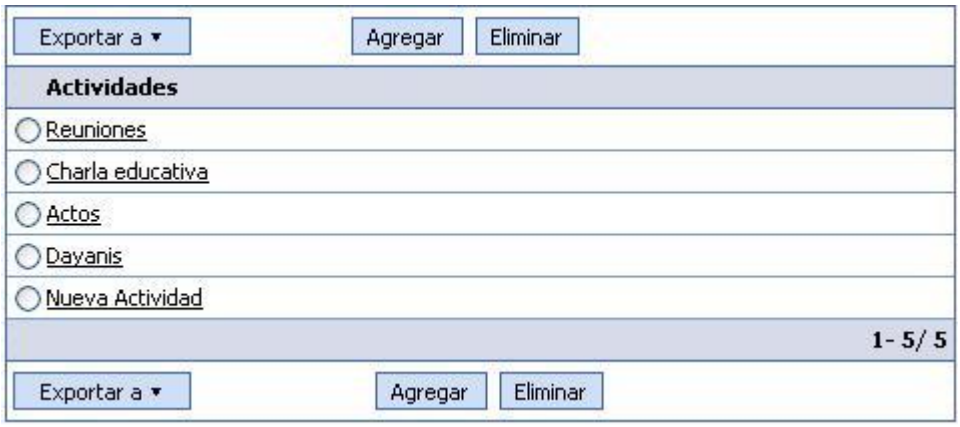
En el campo actividad = “Nueva Actividad”.

Resultados esperados

La actividad se inserta correctamente

Evaluación de la Prueba

Prueba satisfactoria.



- ✓ ***CP4_RAD_ InsertarOtrasActividadesEBS: La actividad que se inserta está repetida.***

Descripción

Para la ejecución de esta prueba se ejecuta el componente Agregar Actividades del EBS como resultado de la actividad Implementar Prueba.

Flujo central

- ✓ El componente de prueba muestra la interfaz con las diferentes opciones.
- ✓ El usuario selecciona la opción Agregar para insertar los datos.
- ✓ El componente muestra el campo Actividad que es donde se insertarán los datos.
- ✓ El usuario inserta los datos. Clic en Aceptar.
- ✓ Componente muestra resultado de la acción.

Condiciones de ejecución

Los campos de carácter obligatorio deben estar llenos.

Entrada

En el campo actividad = “Nueva Actividad”.

Resultados esperados

Mensaje: “Error 103 que responde a que la actividad ya existe”.

Evaluación de la Prueba

Prueba satisfactoria.

- ✓ ***CP5_RAD_ InsertarOtrasActividadesEBS: El parámetro de entrada no es una cadena.***

Descripción

Para la ejecución de esta prueba se ejecuta el componente Agregar Actividades del EBS como resultado de la actividad Implementar Prueba.

Flujo central

- ✓ El componente de prueba muestra la interfaz con las diferentes opciones.
- ✓ El usuario selecciona la opción Agregar para insertar los datos.
- ✓ El componente muestra el campo Actividad que es donde se insertarán los datos.
- ✓ El usuario inserta los datos. Clic en Aceptar.
- ✓ Componente muestra resultado de la acción.

Condiciones de ejecución

Los campos de carácter obligatorio deben estar llenos.

Entrada

En el campo actividad = “123456789”.

Resultados esperados

Mensaje: “Error 102 que responde a que algún parámetro no es correcto, debe ser una cadena”.

Evaluación de la Prueba

Prueba satisfactoria.

- ✓ ***CP6_RAD_ InsertarOtrasActividadesEBS: La función "plaser_handler" no existe.***

Descripción

Para la ejecución de esta prueba se ejecuta el componente Agregar Actividades del EBS como resultado de la actividad Implementar Prueba.

Flujo central

- ✓ El componente de prueba muestra la interfaz con las diferentes opciones.
- ✓ El usuario selecciona la opción Agregar para insertar los datos.
- ✓ El componente muestra el campo Actividad que es donde se insertarán los datos.
- ✓ El usuario inserta los datos. Clic en Aceptar.
- ✓ Componente muestra resultado de la acción.

Condiciones de ejecución

Los campos de carácter obligatorio deben estar llenos. Comentariar la función Plaser Handler.

Entrada

En el campo actividad = "Actividad1".

Resultados esperados

Mensaje: "Violación de acceso.".

Evaluación de la Prueba

Prueba satisfactoria.

Luego de identificar los casos de prueba que verificarán la funcionalidad de los métodos se pasa a especificar los datos de entrada para el componente en la Plantilla **Descripción de Casos de Prueba** (Anexo C) que dan entrada a la próxima actividad.

3.2.3. Implementar Pruebas

Para implementar la prueba se parte de las descripciones de los Casos de Prueba. En ellos se establecen los datos que se deben entrar a los campos de los componentes que se implementarán. A continuación se muestran algunos de ellos.

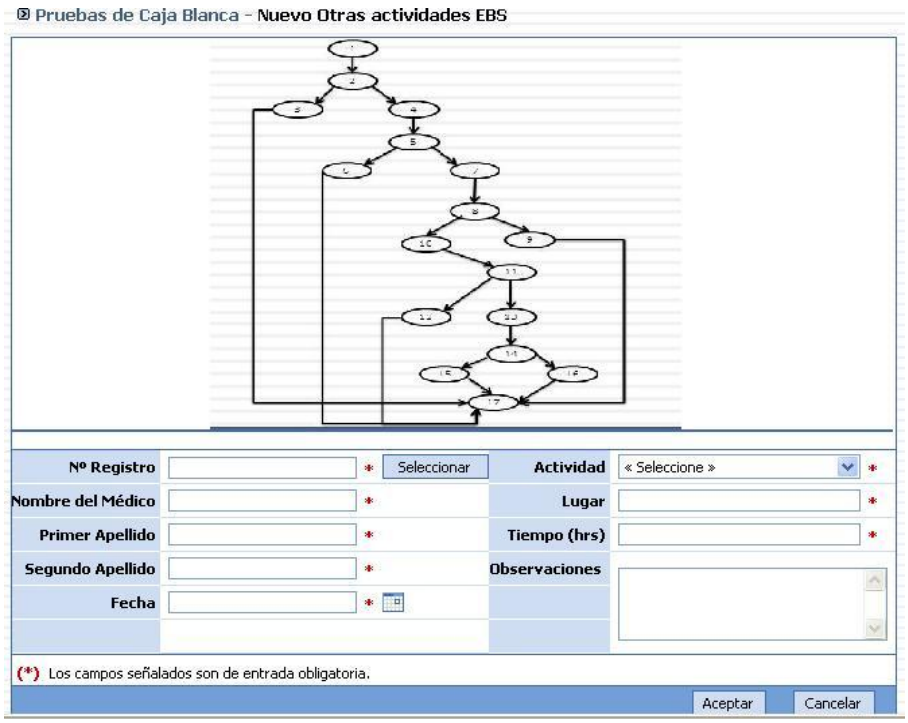


Figura 3.1: Componente Nuevo Otras Actividades EBS.

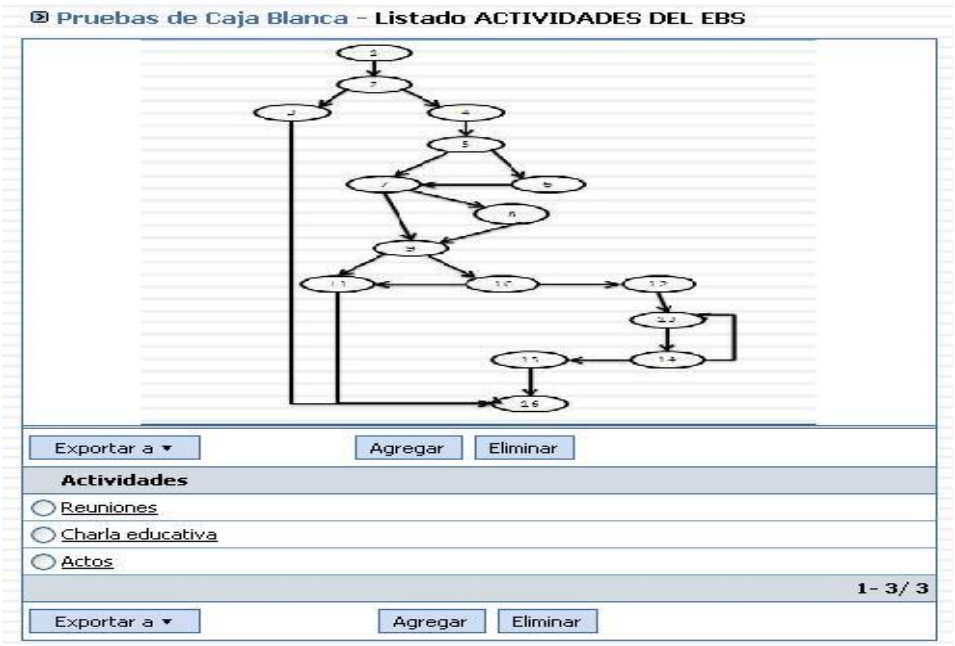


Figura 3.2: Componente Listado Actividades del EBS.

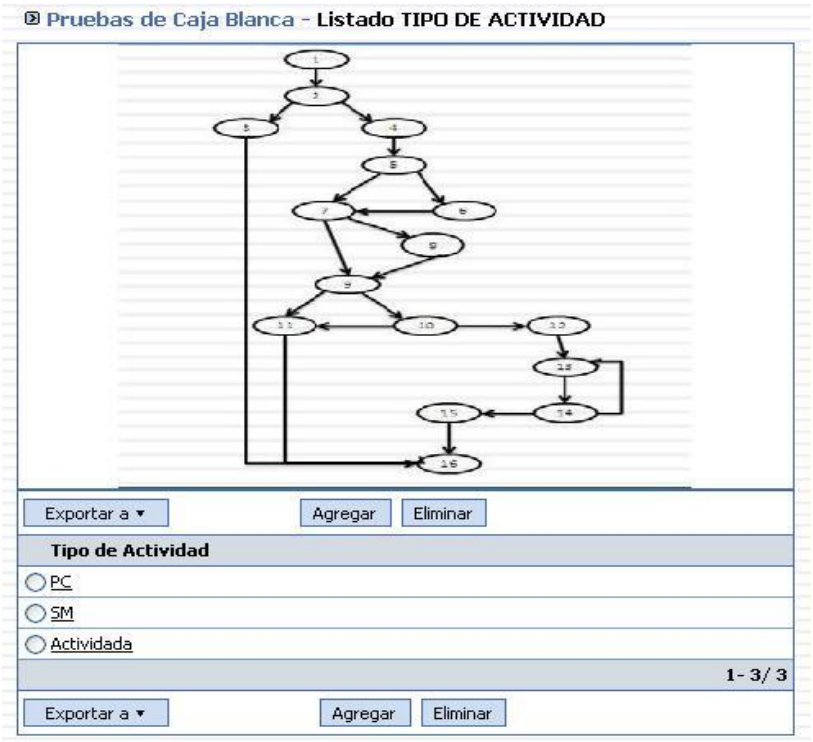


Figura 3.3: Componente Listado Tipo de Actividad.

3.2.4. Ejecutar Prueba

Cumpliendo con el primer objetivo se verificó el cumplimiento de los estándares de codificación especificados por el Área Temática, y se detectó lo siguiente (Ver Anexo I):

Elemento	No	No conformidad	Aspecto correspondiente	Etapa de detección	Importancia	Recomendación
Varias funcionalidades	**1**	**No cumplen con la separación de 4 espacios del margen**	**Identación**	**Prueba**	**Alta**	**Revisar y corregir la identación del código.**

3.3. Resultados de la evaluación del procedimiento.

El procedimiento se evalúa de forma satisfactoria. Para realizar las pruebas se llenaron cada una de las plantillas requeridas posibilitando que cada caso de prueba creado se realizara con el mayor rigor y la eficiencia necesaria. Las pruebas arrojaron varios errores principalmente de estandarización de código, pero fueron ínfimos. Las funciones lanzaron las excepciones correspondientes a cada caso. Mostrando su buen funcionamiento.

Conclusiones

En el presente capítulo se realizó la evaluación del procedimiento planteado en el capítulo anterior. Se tomó como objeto de pruebas el módulo Registro de Actividades Diarias, al cual se le aplica el flujo de actividades propuestas. Se seleccionaron las principales funcionalidades y se aplicaron las Técnicas de prueba correspondientes. Se llenaron las plantillas definidas. Una vez realizadas las pruebas, se procede a verificar eficiencia, robustez y calidad del procedimiento. En base a los resultados obtenidos se califica de satisfactorio.

Conclusiones

Luego de la investigación realizada sobre las pruebas de Caja Blanca y con el fin de fortalecer la calidad del proceso de prueba en el Área Temática APS, así como el cumplimiento de las tareas trazadas, se concluye que:

- ✓ El procedimiento desarrollado cuenta con un grupo de actividades que permiten realizar las pruebas de Caja Blanca de una forma planificada.
- ✓ La versión Symfony Lime v.1.2 para la automatización de pruebas no puede ser utilizada en el Área Temática APS porque no se ajusta a su desarrollo del software.
- ✓ Se propone TestNG como herramienta para realizar pruebas al código para futuros desarrollos en el lenguaje Java.

De esta forma se cumplen los objetivos trazados, que a mediano plazo garantiza la calidad en la implementación de los sistemas en APS. Además se logra un mayor aseguramiento de la calidad por parte del grupo de calidad del Área Temática.

Recomendaciones

Una vez cumplido con el objetivo general propuesto, se recomienda:

- ✓ Desarrollar una herramienta que a partir del código java permita generar los casos de prueba utilizando la técnica de Camino Básico.
- ✓ Desarrollar un procedimiento que integre los dos métodos de prueba: Caja Negra y Caja Blanca para garantizar la eficiencia del proceso de pruebas en el Área Temática.
- ✓ Continuar el estudio de la herramienta TestNG, considerando que los nuevos desarrollos se realizarán en el lenguaje Java.

Referencias Bibliográficas

1. **Colectivo de Autores.** Tecnologías de la Comunicación. [En línea] http://tics.org.ar/.

2. **Red Telemática de Salud en Cuba.** Biblioteca Virtual en Salud. [En línea] http://bvs.sld.cu/revistas/spu/vol32_3_06/spu15306.htm.

3. —. Infomed. [En línea] 1999-2000. http://www.sld.cu/sistema_de_salud/aspectos.html.

4. **Delgado Ramos, Ariel y Vidal Ledo, María.** Informática en la salud pública cubana. [En línea] http://bvs.sld.cu/revistas/spu/vol32_3_06/spu15306.htm.

5. **Noriega Quintana, Darcy Javier, Hernández Pérez, Camilo y San Gabino Merino, Niurka.** Monografias.com. [En línea] 2007. http://www.monografias.com/trabajos59/calidad-software/calidad-software.shtml#xresumen.

6. **Pressman, Roger S.** *Ingeniería del Software Un enfoque práctico.* La Habana: Félix Varela, 2005.

7. *Ídem a la Referencia 6.*

8. **IIEE.** Instituto de Ingenieros Eléctricos y Electrónico. 610-1990.

9. **Lao Medina, Lic. Oslinda y Lodos Vigil, MSc. Jorge.** Sistema de Automatización del Proceso de Pruebas en Segurmática. [En línea] http://crv.matcom.uh.cu/uploads/YadiraFont.pdf..

10. **Lycos Inc.** Estrategia de Pruebas del Software. [En línea] 2001. http://www.angelfire.com/my/jimena/ingsoft/guia9.htm..

11. Tecnología y Synergix. [En línea] http://synergix.wordpress.com/2008/03/15/definimos-pruebas-de-unidad-como/.

12. **V Roca, José María. .** *Pruebas de Integración de Productos: Un enfoque práctico.* 2005.

13. *Ídem a la Referencia 6.*

14. *Ídem a la Referencia 6.*

15. **González Espinosa, Susana y Durán Cutiño, Dania.** *Estrategia para la aplicación de Pruebas de caja Blanca y Pruebas de caja Negra al proyecto Registro y Notarias.* La Habana: s.n., 2008.

16. **Márquez Alpízar, Yaimí y Valdés Hechavarría, Yenni.** *Procedimiento General de Pruebas de caja Blanca aplicando la Técnica del Camino Básico.* La Habana: s.n.

17. **Fuentes Guerra, Yurién Ricardo y Jordán Borjas, Ernesto.** *Implementación de una herramienta para viabilizar el proceso de Pruebas de caja Blanca.* La Habana: s.n., 2008.

18. **González Alonso, Yosbel Ernesto y Cabrera Arribas, Yoan Manuel.** *Proceso de Pruebas del Registro de Áreas de Salud de la Atención Primaria del Sistema de Información para la Salud .* La Habana: s.n., 2007.

19. librosweb.es. [En línea] http://www.librosweb.es/symfony_1_0/.

20. PHPUnit. [En línea] http://www.phpunit.de/wiki.

21. **WORDPRES.COM.** Software Quality. [En línea] http://amunizmartin.wordpress.com/tag/testng/.

22. Software Libre.net. [En línea] http://www.softwarelibre.net/jboss_anuncia_jsfunit_una_herramienta_para_realizar_test_para_aplicaciones_jsf.

23. **Magnolia, Clearspace, JBoss EAP, and RHEL.** JBOSS Community. [En línea] http://www.jboss.org/jrunit/docs/index.html.

24. *Ídem a la Referencia 18.*

25. *Ídem a la Referencia 16.*

Bibliografía

Colectivo de Autores Tecnologías de la Comunicación [En línea]. - http://tics.org.ar/.

Delgado Ramos Ariel y Vidal Ledo María Informática en la salud pública cubana [En línea]. - http://bvs.sld.cu/revistas/spu/vol32_3_06/spu15306.htm.

Fuentes Guerra Yurién Ricardo y Jordán Borjas Ernesto Implementación de una herramienta para viabilizar el proceso de Pruebas de caja Blanca [Informe]. - La Habana: [s.n.], 2008.

González Alonso Yosbel Ernesto y Cabrera Arribas Yoan Manuel Proceso de Pruebas del Registro de Áreas de Salud de la Atención Primaria del Sistema de Información para la Salud [Informe]. - La Habana: [s.n.], 2007.

González Espinosa Susana y Durán Cutiño Dania Estrategia para la aplicación de Pruebas de caja Blanca y Pruebas de caja Negra al proyecto Registro y Notarias [Informe]. - La Habana: [s.n.], 2008.

IIEE Instituto de Ingenieros Eléctricos y Electrónico. - 610-1990.

Lao Medina Lic. Oslinda y Lodos Vigil MSc. Jorge Sistema de Automatización del Proceso de Pruebas en Segurmática [En línea]. - http://crv.matcom.uh.cu/uploads/YadiraFont.pdf..

librosweb.es [En línea]. - http://www.librosweb.es/symfony_1_0/.

Lycos Inc Estrategia de Pruebas del Software [En línea]. - 2001. - http://www.angelfire.com/my/jimena/ingsoft/guia9.htm.

Magnolia, Clearspace, JBoss EAP, and RHEL JBOSS Community [En línea]. - http://www.jboss.org/jrunit/docs/index.html.

Márquez Alpízar Yaimí y Valdés Hechavarría Yenni Procedimiento General de Pruebas de caja Blanca aplicando la Técnica del Camino Básico [Informe]. - La Habana: [s.n.].

Noriega Quintana Darcy Javier, Hernández Pérez Camilo y San Gabino Merino Niurka Monografias.com [En línea]. - 2007. - http://www.monografias.com/trabajos59/calidad-software/calidad-software.shtml#xresumen.

PHPUnit [En línea]. - http://www.phpunit.de/wiki.

Pressman Roger S Ingeniería del Software Un enfoque práctico [Libro]. - La Habana: Félix Varela, 2005.

Red Telemática de Salud en Cuba Biblioteca Virtual en Salud [En línea]. - http://bvs.sld.cu/revistas/spu/vol32_3_06/spu15306.htm.

Red Telemática de Salud en Cuba Infomed [En línea]. - 1999-2000. - http://www.sld.cu/sistema_de_salud/aspectos.html.

Software Libre.net [En línea]. - http://www.softwarelibre.net/jboss_anuncia_jsfunit_una_herramienta_para_realizar_test_para_aplicaciones_jsf.

Tecnología y Synergix [En línea]. - http://synergix.wordpress.com/2008/03/15/definimos-pruebas-de-unidad-como/.

V Roca José María . . Pruebas de Integración de Productos: Un enfoque práctico [Libro]. - 2005.

WORDPRES.COM. Software Quality [En línea]. - http://amunizmartin.wordpress.com/tag/testng/.

Anexos

ANEXO A: Plan de Pruebas

UCI Universidad de las Ciencias Informáticas	**Plan de Pruebas del Sistema**

Plan de Pruebas del Sistema

Interno

Atención Primaria de Salud (APS)

<Módulo>

<Versión >

Plan de Pruebas

1. *Introducción*
   - *1.1. Alcance*
   - *1.2. Definiciones, Acrónimos y Abreviaturas*
   - *1.3. Referencias*
2. *Organización del Equipo de Pruebas.*
3. *Especificaciones del Software y Hardware.*
4. *Descripción del Plan de Pruebas.*
5. *Casos de Prueba.*
6. *Estrategia de Prueba.*
   - *6.1. Objetivo*

6.2. Técnica.

6.3. Entorno de Prueba.

6.4. Proceso.

6.5. Casos de Prueba.

6.6. Herramientas.

7. *Recursos Requeridos.*

Roles	***Cantidad***	***Responsabilidad***

8. *Calendario y Plazos.*
9. *Definición de los Entregables.*
10. *Seguimiento y Reporte de Defectos.*
11. *Aprobación del Plan.*
12. *Documentación de Resultados.*

ANEXO B: Cronograma de Pruebas

UCI Universidad de las Ciencias Informáticas	**Cronograma de Pruebas**

Cronograma de Pruebas

Atención Primaria de Salud (APS)

<Modulo>

<Versión>

1. *Introducción*

   1.1. ***Alcance***

   1.2. ***Objetivos***

   1.3. ***Referencias***

2. *Cronograma de Pruebas*

Módulo	Funcionalidad	Fecha lista	Fecha de prueba		
			Diseño	Implementación	Ejecución

ANEXO C: Técnica Camino Básico

UCi Universidad de las Ciencias Informáticas	**TECNICA CAMINO BASICO**

Técnica Camino Básico

Atención Primaria de Salud (APS)

<Módulo>

<Funcionalidad>

<Versión>

1. ***Segmento de código:<Nombre Módulo>_<Nombre del método>***
   *Poner segmento de código que se probará**.***
2. ***Grafo de flujo***
   Poner una foto del grafo de flujo creado.
3. ***Cálculo de la Complejidad Ciclomática***
   Número de regiones. :
   Aristas - Nodos + 2. :
   Nodos predicado + 1. :
4. ***Conjunto básico de caminos independientes***
   Camino 1:
   Camino 2:
5. ***Casos de prueba***
   Caso de Prueba 1:
   Caso de Prueba 2:

Autor:

ANEXO D: Descripción de Casos de Prueba

<Lugar donde se realizan las pruebas>

<Casos de Pruebas>

<Nombre del Proyecto], [Nombre del módulo>

<Nombre del Caso de uso que cubre el caso de prueba>

1. **CP<Nro.>:** <Módulo>_<Funcionalidad>

   1.1. ***Descripción:*** *Breve descripción del proceso.*

   1.2. ***Flujo central****: Describe el flujo de la interacción del probador con el componente.*

   1.5. ***Condiciones de ejecución****: Condiciones que deben haberse creado antes de ejecutar la prueba.*

   1.6. ***Entrada****: Datos de entrada que recibirá el componente de pruebas.*

   1.7. ***Resultados esperados****: Resultado que espera el Probador de la prueba realizada.*

   1.8. ***Evaluación de la Prueba****: Resultados* que mostró la prueba realizada.

ANEXO E: Registro de No Conformidades

No Conformidades Detectadas

Interno

Atención Primaria de Salud (APS)

<Módulo>

<Versión>

Control de versiones

Fecha	Versión	Descripción	Autor

Aspectos generales: *Descripción de Aspectos Generales.*

Elementos Probados: *Descripción general o lista de los Elementos Probados.*

Elementos no Probados y causas: *[Descripción general o lista de los Elementos no Probados]*

Tabla de No Conformidades Detectadas

Elemento	No	No conformidad	Aspecto correspondiente	Etapa de detección	Importancia	Recomendación

Anexos: Documentos anexos utilizados para realizar las pruebas.

ANEXO F: Formato para el Procedimiento

UCI Universidad de las Ciencias Informáticas	**PROCEDIMIENTO PARA LA REALIZACIÓN DE LAS PRUEBAS DE CAJA BLANCA USANDO LA TÉCNICA DEL CAMINO BÁSICO**

1. *Introducción*

   1.1. Objetivo

   1.2. Alcance

   1.3. Definiciones, Acrónimos y Abreviaturas

1. *Desarrollo*

   1.1. Roles y Responsabilidades.

   1.2. Técnicas de prueba.

   1.3. Cronograma de Actividades.

2. *Documentos de referencia.*

3. *Modelos.*

4. *Distribución y Archivo.*

5. *Anexos.*

Elaborado por: *Firma:* *Fecha:*	*Revisado:* *Fecha:*	*Aprobado:* *Fecha:*	*Pág. de*

ANEXO G: Técnica Camino Básico de RAD_TipoActividad

Técnica Camino Básico

Atención Primaria de Salud (APS)

Registro de Actividades Diarias (RAD)

Tipo de Actividad

1.0

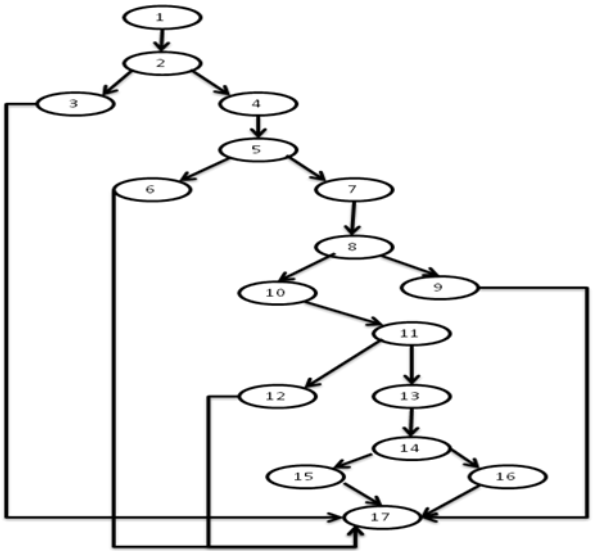
Funcionalidad Insertar Tipo de Actividad

Segmento de código: RAD_ InsertarTipoActividad

```
1<?php
2if(!function_exists('plaser_handler')) 3die('Violación de acceso.');
4require 'PLASER/dbz_class.php';
4require_once '../general/validar.php';
4global $espacio;
4$metodonombre = "InsertarTipoActividad";
4$db = new DBz('m');
4$tipo = utf8_decode($args['tipo']);
5if(!isset($args['tipo']))
 6return $plaser->fault('ALGUN PARAMETRO NO FUE CREADO....',101);
7$Arraydatos = array($tipo);
8if (!ValCadenas($Arraydatos))
{
 9return $plaser->fault('PARAMETRO INCORRECTO, DEBE SER UNA CADENA...', 102);
```

```
}
10$consulta="SELECT * FROM `tb_codificador_tipo_actividad` WHERE (`tb_codificador_tipo_actividad`.tipo_actividad)='$tipo'";
$resultado = $db->query($consulta);
11if($resultado->rows > 0)
{
 12 return $plaser->fault('EL TIPO DE ACTIVIDAD YA EXISTE.',103);
}
13$consulta = "INSERT INTO `tb_codificador_tipo_actividad`(tipo_actividad) VALUES('$tipo')";
13$resultado = $db->query($consulta);
14if($resultado != 1)
{
 15 $modulo='APS-RAD-InsertarTipoActividad';
 15 $message=DevuelveMensajeError($res,$modulo);
 15 $plaser->fault(($message), 113);
 15 $db->close()
}
else
16return true;
17?>
```

Grafo de flujo



Cálculo de la Complejidad Ciclomática

V (G) = *Número de regiones*. **6**

V (G) = Aristas - Nodos + 2. ***21-17+2 = 6***

V (G) = Nodos predicado + 1. ***5*** **+ 1 = 6**

Conjunto básico de caminos independientes

Camino 1: 1-2-4-5-7-8-10-11-13-14-15-17.

Camino 2: 1-2-4-5-6-17.

Camino 3: 1-2-4-5-7-8-10-11-13-14-16-17.

Camino 4: 1-2-4-5-7-8-10-11-12-17.

Camino 5: 1-2-4-5-7-8-9-17.

Camino 6: 1-2-3-17.

Casos de prueba

Caso de Prueba 1: No se inserta el tipo de actividad.

Caso de Prueba 2: No se creo un parámetro.

Caso de Prueba 3: Se inserta el tipo de actividad correctamente.

Caso de Prueba 4: La actividad repetida.

Caso de Prueba 5: El parámetro de entrada no es una cadena.

Caso de Prueba 6: función 'plaser_handler' no existe.

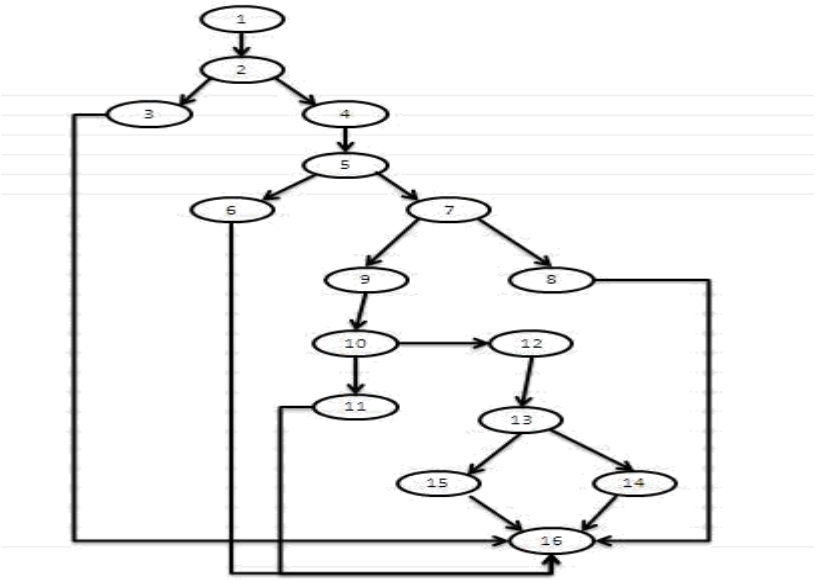
Funcionalidad Eliminar Tipo de Actividad

Segmento de código: RAD_ EliminarTipoActividad

```
1<?php
2if(!function_exists('plaser_handler')) 3die('Violación de acceso.');
4require 'PLASER/dbz_class.php';
4require_once '../general/validar.php';
4global $espacio;
4$metodonombre = "EliminarTipoActividad";
4$db = new DBz('m');//
4$id_tipo_actividad = $args['id_tipo_actividad'];
5if(!isset($args['id_tipo_actividad']))
        6return $plaser->fault('FALTA UN ELEMENTO DE ENTRADA...',14);
7if($id_tipo_actividad == '')
        8return $plaser->fault('FALTA UN PARAMETRO POR PASAR O HA PASADO CERO EN ALGUN PARAMETRO...',19);
9$Arraydatos = array($id_tipo_actividad);
10if (!ValEnterosPositivos($Arraydatos))
```

```
{
 11 $plaser->fault('PARAMETRO INCORRECTO, DEBE SER UN ENTERO POSITIVO...', 26);
}
12$consulta="DELETE FROM tb_codificador_tipo_actividad WHERE (`tb_codificador_tipo_actividad`.id_tipo_actividad = $id_tipo_actividad)";
12$resultado = $db->query($consulta);
13if($resultado != 1)
{
 14$message = DevuelveMensajeError($resultado);
 14 $plaser->fault('No se puede eliminar dicho tipo de actividad ya que ha sido asignada',$message ,35);
 14 $db->close();
}
15return new SOAP_Value($metodonombre, "{urn:$espacio}$metodonombre", array($message));
16?>
```

Grafo de flujo



Cálculo de la Complejidad Ciclomática

V (G) = *Número de regiones*. **6**

V (G) = Aristas - Nodos + 2. ***20-16+2 = 6***

V (G) = Nodos predicado + 1. ***5*** **+ 1 = 6**

Conjunto básico de caminos independientes

Camino 1:1-2-4-5-7-9-10-12-13-15-16.

Camino 2: 1-2-4-5-6-16.

Camino 3: 1-2-4-5-7-8-16.

Camino 4: 1-2-4-5-7-9-10-11-16.

Camino 5: 1-2-4-5-7-9-10-12-13-14-16.

Camino 6: 1-2-3-16.

Casos de prueba

Caso de Prueba 1: todo ok.

Caso de Prueba 2: Falta un elemento de entrada.

Caso de Prueba 3: No se entro el parámetro o el id es 0.

Caso de Prueba 4: El parámetro que se entra es incorrecto.

Caso de Prueba 5: Muchas respuestas para el pedio en la base de datos. .

Caso de Prueba 6: función 'plaser_handler' no existe.

Funcionalidad Actualizar Tipo de Actividad

a) **Segmento de código:** RAD_ ActualizarTipoActividad

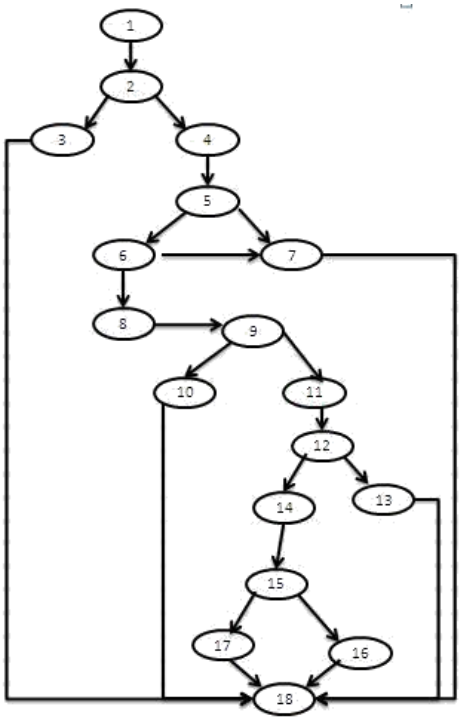
```
1 <?php
2 if(!function_exists('plaser_handler'))3 die('Violacion de acceso.');
4 require 'PLASER/dbz_class.php';
4 require_once '../general/validar.php';
4 global $espacio;
4 $metodonombre = "ActualizarTipoActividad";
4 $db = new DBz('m');
4 $id_tipo_actividad = intval($args['id_tipo_actividad']);
4 $tipo_actividad = utf8_decode($args['tipo_actividad']);
5 if(!isset($args['id_tipo_actividad']) || 6 !isset($args['tipo_actividad']))
7 return $plaser->fault('ALGUN PARAMETRO NO FUE CREADO....',101);
8 $Arraydatos = array($tipo_actividad);
9 if (!ValCadenas($Arraydatos))
```

```
{
10 return $plaser->fault('PARAMETRO INCORRECTO, DEBE SER UNA CADENA...', 102);
}
11 $consulta="SELECT * FROM tb_codificador_tipo_actividad WHERE (tb_codificador_tipo_actividad.tipo_actividad = '$tipo_actividad')";
 $resultado = $db->query($consulta);
12 if ($resultado->rows>0)
{
13 return $plaser->fault('EL TIPO DE ACTIVIDAD ES INCORRECTA, YA EXISTE ....',101);
}
14 $consulta = "UPDATE tb_codificador_tipo_actividad SET tb_codificador_tipo_actividad.id_tipo_actividad = $id_tipo_actividad, tb_codificador_tipo_actividad.tipo_actividad = '$tipo_actividad 'WHERE (tb_codificador_tipo_actividad.id_tipo_actividad = $id_tipo_actividad)";
$resultado = $db->query($consulta);
15 if($resultado != 1)
{
16 $modulo='APS-RAD-ActualizarTipoActividad';
16 $message=DevuelveMensajeError($res,$modulo);
16 $plaser->fault(($message), 113);
16 $db->close();
}
else
```

17 return true;

18 ?>

c) **Grafo de flujo**



d) **Cálculo de la Complejidad Ciclomática**

V (G) = *Número de regiones*. **7**

V (G) = Aristas - Nodos + 2. ***23-18+2 = 7***

V (G) = Nodos predicado + 1. ***6*** **+ 1 = 7**

e) Conjunto básico de caminos independientes

Camino 1: 1-2-4-5-6-8-9-11-12-14-15-16-18.

Camino 2: 1-2-4-5-7-18.

Camino 3: 1-2-4-5-6-7-18.

Camino 4: 1-2-4-5-6-8-9-10-18.

Camino 5: 1-2-4-5-6-8-9-11-12-13-18.

Camino 6: 11-2-4-5-6-8-9-11-12-14-15-17-18.

Camino 7: 1-2-3-18.

f) Casos de prueba

Caso de Prueba 1: todo ok.

Caso de Prueba 2: Falta un elemento de entrada.

Caso de Prueba 3: Falta un elemento de entrada.

Caso de Prueba 4: El parámetro entrado no es cadena.

Caso de Prueba 5: La actividad ya existe.

Caso de Prueba 6: Muchos resultados para una llmada.

Caso de Prueba 7: función 'plaser_handler' no existe.

Funcionalidad Listar Tipo de Actividad

a) Segmento de código: RAD_ ActualizarTipoActividad

```
1 <?php
2 if(!function_exists('plaser_handler')) 3 die('Violacion de acceso.');
4 require_once 'PLASER/dbz_class.php';
4 global $espacio;
4 $metodonombre = "Listar_Tipo_Actividad";
4 $db = new DBz('m');
4 $offset = $args['offset'];
4 $cantidad = $args['cantidad'];
```

4 $consulta = "SELECT `tb_codificador_tipo_actividad`.id_tipo_actividad,`tb_codificador_tipo_actividad`.tipo_actividad FROM `tb_codificador_tipo_actividad` ";

4 $respuesta = $db->query($consulta);

4 $total = $respuesta->rows;

5 if($offset == '')

{

6 $offset = 0;

}

7 if($cantidad == '')

{

8 $cantidad = $total;

}

9 if (($offset <> 0) or**10** ($total > $cantidad))

{

11$consulta = $consulta . " LIMIT ".$offset.",".$cantidad;

11$respuesta = $db->query($consulta,DBX_RESULT_ASSOC);

}

12 $i = 0;

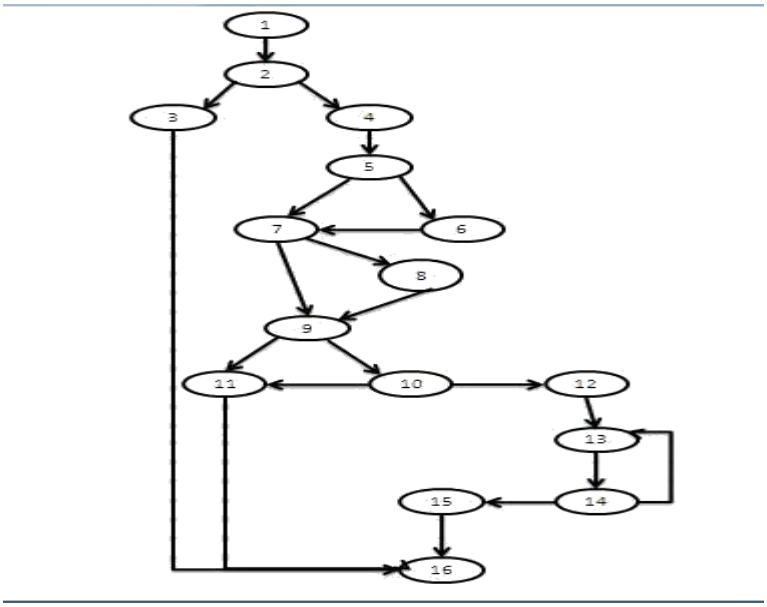
13 foreach ($respuesta->data as $row)

{

14 $Arrayresultado['Listado_Tipo_Actividad'][$i]['id_tipo_actividad'] = $row[0];

```
        $Arrayresultado['Listado_Tipo_Actividad'][$i]['tipo_actividad'] = utf8_encode($row[1]);
$Arrayarreglototal['Listado_Tipo_Actividad'][$i]['descripcion'] = utf8_encode($row[1]);
        $i++;
}
 15 $Arrayresultado['offset'] = $offset;
 15 $Arrayresultado['cantidad']= $cantidad;
 15 $Arrayresultado['total'] = $total;
 15 $db->close();
16 ?>
```

c) **Grafo de flujo**



d) **Cálculo de la Complejidad Ciclomática**

V (G) = *Número de regiones*. **7**

V (G) = Aristas - Nodos + 2. ***21-16+2 = 7***

V (G) = Nodos predicado + 1. ***6*** **+ 1 = 7**

e) Conjunto básico de caminos independientes

Camino 1: 1-2-4-5-7-9-10-12-13-14-15-16.

Camino 2: 1-2-3-16.

Camino 3: 1-2-4-5-6-7-16.

Camino 4: 1-2-4-5-7-8-9-16.

Camino 5: 1-2-4-5-7-9-11-16.

Camino 6: 1-2-4-5-7-9-10-11-16.

Camino 7: 1-2-4-5-7-9-10-12-13-14-13.

f) Casos de prueba

Caso de Prueba 1: La actividad se inserta correctamente.

Caso de Prueba 2: función 'plaser_handler' no existe.

Caso de Prueba 3: Variable offset ≠“” y cantidad < total.

Caso de Prueba 4: Variable offset ≠“” y cantidad < total.

Caso de Prueba 5: Variables offset = 0 y cantidad ≠ ”” > total.

Caso de Prueba 6: Variable offset ≠ “” y = “”.

Caso de Prueba 7: Varias páginas.

Autor: Dayanis Isaac Morales

Fecha: Del 24 de Abril al 1ro de Mayo del 2009

ANEXO H: Descripción de Casos de Prueba Insertar Tipo de Actividad

LAB 204

Casos de Prueba: Insertar Tipo Actividad.

Atención Primaria de Salud (APS), Registro de Actividades Diarias (RAD)

CU: Gestionar Tipo de Actividad

- ✓ ***CP1_RAD_ InsertarTipoActividad: No se inserta el tipo de actividad.***

Descripción

Para la ejecución de esta prueba se ejecuta el componente Listado Tipo de Actividad como resultado de la actividad Implementar Prueba.

Flujo central

- ✓ El componente de prueba muestra la interfaz con las diferentes opciones.
- ✓ El usuario selecciona la opción Agregar.
- ✓ El componente muestra los campos que se deben llenar.
- ✓ El usuario llena los campo y selecciona la opción Aceptar.
- ✓ Componente muestra resultado de la acción.

Condiciones de ejecución

Los campos de carácter obligatorio deben estar llenos.

Entrada

En el campo actividad = “Actividad9”.

Resultados esperados

Mensaje: “Mensaje 113 que responde a.”

Evaluación de la Prueba

- ✓ ***CP2_RAD_ InsertarTipoActividad: No se creo un parámetro.***

Descripción

Para la ejecución de esta prueba se ejecuta el componente Listado Tipo de Actividad como resultado de la actividad Implementar Prueba.

Flujo central

- ✓ El componente de prueba muestra la interfaz con las diferentes opciones.
- ✓ El usuario selecciona la opción Agregar.
- ✓ El componente muestra los campos que se deben llenar.
- ✓ El usuario llena los campo y selecciona la opción Aceptar.
- ✓ Componente muestra resultado de la acción.

Condiciones de ejecución

Los campos de carácter obligatorio deben estar llenos.

Entrada

En el campo tipo = “”.

Resultados esperados

Mensaje: “Mensaje 101 que responde a que un parámetro no fue creado”

Evaluación de la Prueba

Prueba satisfactoria.

- ✓ ***CP3_RAD_ InsertarTipoActividad: Se inserta el tipo de actividad correctamente.***

Descripción

Para la ejecución de esta prueba se ejecuta el componente Listado Tipo de Actividad como resultado de la actividad Implementar Prueba.

Flujo central

- ✓ El componente de prueba muestra la interfaz con las diferentes opciones.
- ✓ El usuario selecciona la opción Agregar.
- ✓ El componente muestra los campos que se deben llenar.
- ✓ El usuario llena los campo y selecciona la opción Aceptar.
- ✓ Componente muestra resultado de la acción.

Condiciones de ejecución

Los campos de carácter obligatorio deben estar llenos.

Entrada

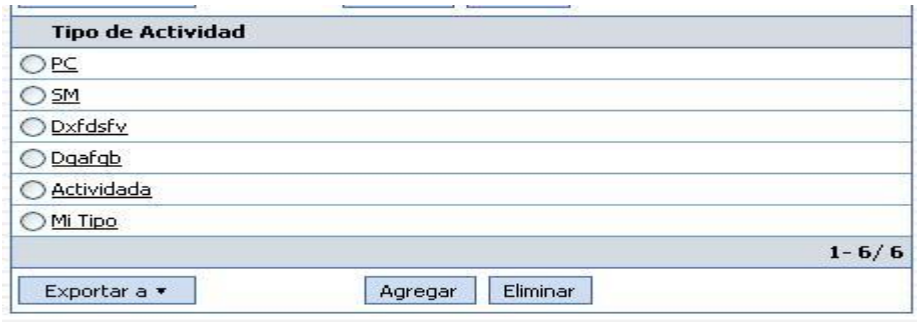
En el campo tipo actividad = “Mi Tipo”.

Resultados esperados

La actividad se inserta correctamente.

Evaluación de la Prueba

Prueba satisfactoria.



- ✓ ***CP4_RAD_ InsertarTipoActividad: La actividad repetida.***

Descripción

Para la ejecución de esta prueba se ejecuta el componente Listado Tipo de Actividad como resultado de la actividad Implementar Prueba.

Flujo central

- ✓ El componente de prueba muestra la interfaz con las diferentes opciones.
- ✓ El usuario selecciona la opción Agregar.
- ✓ El componente muestra los campos que se deben llenar.
- ✓ El usuario llena los campo y selecciona la opción Aceptar.
- ✓ Componente muestra resultado de la acción.

Condiciones de ejecución

Los campos de carácter obligatorio deben estar llenos.

Entrada

En el campo tipo lugar = “Mi Tipo”.

Resultados esperados

Mensaje: “Error 103 que responde a que el tipo de actividad ya existe”.

Evaluación de la Prueba

Prueba satisfactoria.

- ✓ ***CP5_RAD_ InsertarTipoActividad: El parámetro de entrada no es una cadena.***

Descripción

Para la ejecución de esta prueba se ejecuta el componente Listado Tipo de Actividad como resultado de la actividad Implementar Prueba.

Flujo central

- ✓ El componente de prueba muestra la interfaz con las diferentes opciones.
- ✓ El usuario selecciona la opción Agregar.
- ✓ El componente muestra los campos que se deben llenar.

- ✓ El usuario llena los campo y selecciona la opción Aceptar.
- ✓ Componente muestra resultado de la acción.

Condiciones de ejecución

Los campos de carácter obligatorio deben estar llenos.

Entrada

En el campo tipo de actividad = "//**-oko".

Resultados esperados

Mensaje: "Error 102 que responde a que el parámetro debe ser una cadena".

Evaluación de la Prueba

Prueba satisfactoria.

- ✓ ***CP6_RAD_ InsertarTipoActividad: La función "plaser_handler" no existe.***

Descripción

Para la ejecución de esta prueba se ejecuta el componente Listado Tipo de Actividad como resultado de la actividad Implementar Prueba.

Flujo central

- ✓ El componente de prueba muestra la interfaz con las diferentes opciones.
- ✓ El usuario selecciona la opción Agregar.
- ✓ El componente muestra los campos que se deben llenar.
- ✓ El usuario llena los campo y selecciona la opción Aceptar.
- ✓ Componente muestra resultado de la acción.

Condiciones de ejecución

Los campos de carácter obligatorio deben estar llenos. Comentariar la función Plaser Handler.

Entrada

En el campo tipo actividad = “Reunión 6”.

Resultados esperados

Mensaje: “Violación de acceso.”.

Evaluación de la Prueba

Prueba satisfactoria.

Autor: Dayanis Isaac Morales

Fecha : Del 8 al 15 de Mayo del 2009

ANEXO I: Registro No Conformidades RAD

No Conformidades Detectadas

Interno

Atención Primaria de Salud (APS)

Registro de Actividades Diarias

1.0

Control de versiones

Fecha	Versión	Descripción	Autor
Del 22/05/09 Al 29/05/09	1.0	Registra los defectos encontrados en el código.	Est Dayanis Isaac Morales.

Aspectos generales: *Descripción de Aspectos Generales.*

Elementos Probados: *Descripción general o lista de los Elementos Probados.*

Elementos no Probados y causas: *[Descripción general o lista de los Elementos no Probados]*

Tabla de No Conformidades Detectadas

Elemento	No	No conformidad	Aspecto correspondiente	Etapa de detección	Importancia	Recomendación
Insertar ActivEBS	1	No cumplen con la separación de 4 espacios del margen	Identación	Prueba	Media	Revisar y corregir la identación del código.

Eliminar ActivEBS	2	No cumplen con la separación de 4 espacios del margen	Identación	Prueba	Media	Revisar y corregir la identación del código.
Actualizar ActivEBS	3	No cumplen con la separación de 4 espacios del margen	Identación	Prueba	Media	Revisar y corregir la identación del código.
Listar ActivEBS	4	No cumplen con la separación de 4 espacios del margen	Identación	Prueba	Media	Revisar y corregir la identación del código.
Insertar Tipo Activ	5	No cumplen con la separación de 4 espacios del margen	Identación	Prueba	Media	Revisar y corregir la identación del código.
Eliminar Tipo Activ	6	No cumplen con la separación de 4 espacios del margen	Identación	Prueba	Media	Revisar y corregir la identación del código.
Actualizar Tipo Activ	7	No cumplen con la separación de 4 espacios del	Identación	Prueba	Media	Revisar y corregir la identación del código.

		margen				
Listar Tipo Activ	8	No cumplen con la separación de 4 espacios del margen	Identación	Prueba	Media	Revisar y corregir la identación del código.
Insertar OActivEBS	9	No cumplen con la separación de 4 espacios del margen	Identación	Prueba	Media	Revisar y corregir la identación del código.
Listar ActivEBS	**10**	**No cumplen con la separación de 4 espacios del margen**	**Identación**	**Prueba**	**Media**	**Revisar y corregir la identación del código.**

Anexos:

- ✓ Plan de Pruebas RAD.
- ✓ Técnica Camino Básico
  - RAD_TCB_ActvEBS
  - RAD_TCB_TipoActv
  - RAD_TCB_OtrasActvEBS
- ✓ Descripción de Casos de Pruebas RAD.
  - *RAD_DCPCB_InsActvEBS*

- *RAD_DCPCB_ElimActvEBS*
- *RAD_DCPCB_ModActvEBS*
- *RAD_DCPCB_ListActvEBS*
- *RAD_DCPCB_InsTipoActv*
- *RAD_DCPCB_ElimTipoActv*
- *RAD_DCPCB_ModTipoActv*
- *RAD_DCPCB_ListTipoActv*
- *RAD_DCPCB_InsOtrasActvEBS*
- *RAD_DCPCB_ListOtrasActvEBS*

✓ Estándares de Codificación APS.

- APS_Estandares_Caracteres ExtranosUso de los arreglos.
- APS_Estandares_Codificacion
- APS_Estandares_Programar_CP

ANEXO J: Pruebas de Condición

- Insertar Actividades EBS

- ✓ ***Condición 1: if(!function_exists('plaser_handler'))***

Restricciones: (D_1).

Conjunto de restricciones D: {(V: Existe función “plaser_handler”), (F: No existe función “plaser_handler”)}

- ✓ ***Condición 2: if(!isset($args['actividad_ebs']))***

Restricciones: (D_1).

Conjunto de restricciones D: {(V), (F)}.

- ✓ ***Condición 3: if (!ValCadenas($Arraydatos))***

Restricciones: (D_1).

Conjunto de restricciones D: {(V) (F)}.

- ✓ ***Condición 4: if($Arrayresultado->rows > 0)***

Restricciones: (D_1, D_2,).

Conjunto de restricciones D: {(0, 0), (1, 0), (2, 0), (3, 0), (4, 0),…, (n, 0)}.

- ✓ ***Condición 5: if($Arrayrespuesta != 1)***

Restricciones: (D_1, D_2,).

Conjunto de restricciones D: {(0, 1), (1, 1), (2, 1), (3, 1), (4, 1),…, (n, 1)}.

- Eliminar Actividades EBS

- ✓ ***Condición 6: if($id_actividadesEBS == '')***

Restricciones: (D_1, D_2,).

Conjunto de restricciones D: {(actividad, actividad) (actividad, “”)}

- ✓ ***Condición 7: if (!ValEnterosPositivos($Arraydatos))***

Restricciones: (D_1)

Conjunto de restricciones D: {(V), (F)}.

- ✓ ***Condición 8: if($resultado != 1)***

Restricciones: (D_1, D_2).

Conjunto de restricciones D: {(0, 1), (1, 1), (2, 1), (3, 1), (4, 1),…, (n, 1)}.

- ✓ ***Condición 9: if($Arrayresultado != 1)***

Restricciones: (D_1, D_2).

Conjunto de restricciones D: {(0, 1), (1, 1), (2, 1), (3, 1), (4, 1),…, (n, 1)}.

- Actualizar Actividades EBS

- ✓ ***Condición 10: if($offset ==”'')***

Restricciones: (D_1, D_2).

Conjunto de restricciones D: {(ofsett, a), (ofsett, b), (ofsett, c), (ofsett, d), (ofsett, “”),…, (ofsett, n)}.

- ✓ ***Condición 11: if($cantidad == '')***

Restricciones: (D_1, D2).

Conjunto de restricciones D: {(cantidad, 1), (cantidad, 2), (cantidad, “”),…,(cantidad, n)}.

- ✓ ***Condición 12: if (($offset <> 0) or ($total > $cantidad))***

Restricciones: (D_1, D_2,).

Conjunto de restricciones D: {(V,V), (V, F), (F, V)}.

- ✓ ***Condición 13: if($tipo=='.pdf')***

Restricciones: (D_1, D_2).

Conjunto de restricciones D: {(tipo, pdf), (tipo, excel), (tipo, word)}.

- ✓ ***Condición 14: foreach($respuesta->data as $Arraytemporal)***

Restricciones: (D_1, D_2).

Conjunto de restricciones D: {()}

- ✓ ***Condición 15: foreach ($respuesta->data as $row)***

Restricciones: (D_1, D_2).

Conjunto de restricciones D: {(V), (F)}

- ➢ Insertar Tipo de Actividad

- ✓ ***Condición 16: if(!isset($args['tipo']))***

Restricciones: ***(D_1).***

Conjunto de restricciones D: {(V), (F)}

- Insertar Otras Actividades del EBS

- ✓ ***Condición 17: if ($lugar == "" ||$actividad == "" ||$fecha == "" ||$tiempo == "")***

Restricciones: (D_1, D_2, D_3, D_4).

Conjunto de restricciones D: {(lugar, actividad, fecha, “”), (lugar, actividad, “”, tiempo), (lugar, “”, fecha, tiempo), (“”, actividad, fecha, tiempo), (“, “”, fecha, tiempo),, (lugar, “”, “, tiempo),, (lugar, “”, fecha, “),(“,actividad, ““, tiempo), (“,actividad, fecha, “”),(lugar, actividad, “,tiempo}.

- ✓ ***Condición 18: if (!ValCadenas($cadena))***

Restricciones: (D_1).

Conjunto de restricciones D: {(V), (F)}

- Buscar Otras Actividades del EBS

- ✓ ***Condición 19: if($fecha_fin=="")***

Restricciones: (D_1, D_2).

Conjunto de restricciones D: {(fecha, “”), (Fecha, fecha)}.

- ✓ ***Condición 20: if($a['mday']<10)***

Restricciones: (D_1, D_2).

Conjunto de restricciones D: {(a, 1), (a, 2), (a, 11), (a, 10), (a, 13)}

- ✓ ***Condición 21: if ($a['mon']<10)***

Restricciones: (D_1, D_2, D_3, D_4).

Conjunto de restricciones D: {(a, 1), (a, 2), (a, 11), (a, 10), (a, 13)}.

- ✓ ***Condición 22: if($fecha_fin<$fecha_inicio)***

Restricciones: (D_1, D_2).

Conjunto de restricciones D: {(fin, ini), (fin-1, ini), (fin + 1, ini), (fin, ini - 1), (fin, ini +1) }.

- ✓ ***Condición 23: if($id_medico !="")***

Restricciones: (D_1, D_2, D_3, D_4).

Conjunto de restricciones D: {()}.

- ✓ ***Condición 24:if($id_actividad!="")***

Restricciones: (D_1, D_2, D_3, D_4).

Conjunto de restricciones D: {(medico, “”), (medico, medico)}.

- ✓ ***Condición 25: if($id_ebs!="")***

Restricciones: (D_1, D_2).

Conjunto de restricciones D: {(id, “”), (id, id)}.

- ✓ ***Condición 26: if($fecha_inicio !="")***

Restricciones: (D_1, D_2).

Conjunto de restricciones D: {(fecha, “”), (Fecha, fecha)}.

- ✓ ***Condición 28: if($ordenar!="")***

Restricciones: (D_1, D_2).

Conjunto de restricciones D: {(ordenar, ordenar), (ordenar, ordenar + 1), (ordenar, ordenar-1)}.

- ✓ ***Condición 29: if($ordenar=="fecha")***

Restricciones: (D_1, D_2).

Conjunto de restricciones D: {(ordenar, ordenar), (ordenar, ordenar + 1), (ordenar, ordenar-1)}.}.

- ✓ ***Condición 30: if($ordenar=="lugar")***

Restricciones: (D_1, D_2).

Conjunto de restricciones D: {(ordenar, ordenar), (ordenar, ordenar + 1), (ordenar, ordenar-1)}.}.

- ✓ ***Condición 30: if($ordenar=="tiempo")***

Restricciones: (D_1, D_2).

Conjunto de restricciones D: {(ordenar, ordenar), (ordenar, ordenar + 1), (ordenar, ordenar-1)}.

- ✓ ***Condición 31: if($offset == "}***

Restricciones: (D_1, D_2,).

Conjunto de restricciones D: {(offset, “”), (offset, a), (offset, b),…,(offset, z)}.

- ✓ ***Condición 32: if($cantidad == ")***

Restricciones: (D_1, D_2).

Conjunto de restricciones D: {(cantidad, “”), (cantidad, 1), (cantidad, 2),…, (cantidad, n)}.

- ✓ ***Condición 33: if (($offset <> 0) or ($total > $cantidad))***

Restricciones: (D_1, D_2).

Conjunto de restricciones D: {(V, V), (V,F), (F,V)}.

ANEXO K: Pruebas Unitarias con Symfony Lime

El listado muestra un conjunto típico de pruebas unitarias para la función strtolower(). En primer lugar, se instancia el objeto lime_test. Cada prueba unitaria consiste en una llamada a un método de la instancia de lime_test. El último parámetro de estos métodos siempre es una cadena de texto opcional que se utiliza como resultado del método.

Listado 1- Archivo de ejemplo de prueba unitaria, en `test/unit/strtolowerTest.php`

```
<?php
 include(dirname(__FILE__).'/../bootstrap/unit.php');
require_once(dirname(__FILE__).'/../../lib/strtolower.php');
 $t = new lime_test(7, new lime_output_color());
 // strtolower()
$t->diag('strtolower()');
$t->isa_ok(strtolower('Foo'), 'string',
 'strtolower() returns a string');
$t->is(strtolower('FOO'), 'foo',
 'strtolower() transforms the input to lowercase');
$t->is(strtolower('foo'), 'foo',
 'strtolower() leaves lowercase characters unchanged');
$t->is(strtolower('12#?@~'), '12#?@~',
 'strtolower() leaves non alphabetical characters unchanged');
$t->is(strtolower('FOO BAR'), 'foo bar',
 'strtolower() leaves blanks alone');
$t->is(strtolower('FoO bAr'), 'foo bar',
 'strtolower() deals with mixed case input');
$t->is(strtolower(''), 'foo',
 'strtolower() transforms empty strings into foo');
```

Para ejecutar el conjunto de pruebas, se utiliza la tarea `test-unit` desde la línea de comandos. El resultado de esta tarea en la línea de comandos es muy explícito, lo que permite localizar fácilmente las pruebas que han fallado y las que se han ejecutado correctamente. El listado2 muestra el resultado del ejemplo anterior.

Listado 2 - Ejecutando una prueba unitaria desde la línea de comandos

```
> symfony test-unit strtolower
 1..7
# strtolower()
ok 1 - strtolower() returns a string
ok 2 - strtolower() transforms the input to lowercase
ok 3 - strtolower() leaves lowercase characters unchanged
ok 4 - strtolower() leaves non alphabetical characters unchanged
ok 5 - strtolower() leaves blanks alone
ok 6 - strtolower() deals with mixed case input
not ok 7 - strtolower() transforms empty strings into foo
# Failed test (.\batch\test.php at line 21)
# got: ''
# expected: 'foo'
# Looks like you failed 1 tests of 7.
```

Glosario de Términos

Aplicación: En informática, una aplicación es un tipo de programa informático diseñado para facilitar al usuario la realización de un determinado tipo de trabajo.

Área Temática: Conjunto de proyectos que poseen características comunes, es decir que todos se realizan con el mismo fin determinado y para el mismo perfil para así organizar mejor la producción de la Facultad Siete.

Herramientas: Se aplica el término *herramienta* a un producto CASE que da soporte a una tarea concreta dentro de las actividades de desarrollo de software. Dicho soporte consistirá en una serie de *servicios*, cada uno de los cuales automatiza una operación individual.

ISO 9000: La familia de normas ISO 9000 son normas de "calidad" y "gestión continua de calidad", establecidas por la Organización Internacional para la Estandarización (ISO) que se pueden aplicar en cualquier tipo de *organización* o actividad sistemática, que esté orientada a la producción de bienes o servicios. Se componen de estándares y guías relacionados con sistemas de gestión y de herramientas específicas como los métodos de auditoría (el proceso de verificar que los sistemas de gestión cumplen con el estándar).

Proceso: Un proceso puede ser definido como un conjunto de actividades enlazadas entre sí que, partiendo de uno o más entradas los transforman, generando un resultado.

Propuesta: Idea u oferta que se realiza con el propósito de ponerlo en práctica y mejorar lo que está definido.

RUP: El Proceso Unificado Racional (*Rational Unified Process* en inglés, habitualmente resumido como RUP) es un proceso de desarrollo de software y junto con el Lenguaje Unificado de Modelado UML, constituye la metodología estándar más utilizada para el análisis, implementación y documentación de sistemas orientados a objetos.