Serie Científica de la Universidad de las Ciencias Informáticas
Vol. 11, No. 6, Mes Junio, 2018
ISSN: 2306-2495 | RNPS: 2343
Pág. 43-55
http://publicaciones.uci.cu

Tipo de artículo: Artículo original
Temática: Soluciones informáticas
Recibido: 20/04/18 | Aceptado: 05/06/18 | Publicado: 29/06/18

Módulo Visor de Reportes para el Sistema Integrado de Gestión Estadística v4.0

Report Viewer Module for Integrated Statistical Management System v4.0

Ariam Lázaro Álvarez Rodríguez[1], Rene Manuel Puig Pérez[2], Rolando EnriqueFuentes Orozco[3], Glennis Tamayo Morales[4], Juan Miguel Pérez Almaguer[5]

1Universidad de las Ciencias Informáticas, alalvarez@estudiantes.uci.cu
2Universidad de las Ciencias Informáticas, rmpuig@estudiantes.uci.cu
3Universidad de las Ciencias Informáticas, refuentes@uci.cu
4Universidad de las Ciencias Informáticas, gtamayo@uci.cu
5Universidad de las Ciencias Informáticas, jmalmaguer@uci.cu

[*] Autor para correspondencia: alalvarez@estudiantes.uci.cu

Resumen

El sistema Integrado de Gestión Estadística (SIGE); tiene como principal objetivo la informatización de los procesos de captura de datos de una institución. Sus procesos fundamentales son el diseño de plantillas de tipo formulario, la captura de datos sobre estos diseños de plantillas, la validación de los datos capturados y la recuperación de información a través de reportes ofreciendo apoyo en la toma de decisiones en las empresas. Actualmente se está desarrollando la versión 4.0 del producto, utilizando para la lógica del negocio y el acceso a los datos el framework Symfony v2.8 y para el diseño de interfaz gráfica de usuario ExtJS v6.0, lo que ha traído como consecuencia que en estos momentos no es posible la visualización de reportes pues no se integra con el Módulo Visor de Reportes que se utiliza en SIGE v3.0 el cual fue desarrollado con el framework Symfony v1.2.7 y ExtJS v2.2 para el diseño de la interfaz gráfica de la aplicación. En el presente trabajo se trazó como objetivo desarrollar el Módulo Visor de Reportes para SIGE v4.0 que permita mostrar la información almacenada. Realizándose además pruebas de unidad, integración y de sistema al módulo comprobando su correcto funcionamiento.

Palabras clave: *estadística*; *reportes*; *visor*.

Abstract

Serie Científica de la Universidad de las Ciencias Informáticas
Vol. 11, No. 6, Mes Junio, 2018
ISSN: 2306-2495 | RNPS: 2343
Pág. 43-55
http://publicaciones.uci.cu

The Integrated Statistical Management System (SIGE); has as main objective the computerization of the data capture processes of an institution. Its fundamental processes are the design of form-type templates, the capture of data on these designs of templates, the validation of captured data and the retrieval of information through reports offering support in decision-making in companies. Currently, version 4.0 of the product is being developed, using the Symfony v2.8 framework and the graphical user interface ExtJS v6.0 for business logic and data access, which has resulted in These moments it is not possible to display reports because it is not integrated with the Report Viewer Module that is used in SIGE v3.0 which was developed with the framework Symfony v1.2.7 and ExtJS v2.2 for the design of the graphical interface of the application. In the present work, the objective was to develop the Report Viewer Module for SIGE v4.0 that allows displaying the stored information. Also performing tests of unit, integration and system to the module checking its correct operation.

Keywords: *reports; statistics; viewer.*

Introducción

En el mundo contemporáneo la tecnología se ha considerado una herramienta potente para el desarrollo de cualquier organización pues brinda gran ayuda en la toma de decisiones para lograr objetivos y metas propuestas por cada organismo.

Con el creciente avance de las Tecnologías de la Información y las Comunicaciones (TIC), ha aumentado el volumen de información a manejar en las empresas, convirtiéndose esta en el activo fundamental de las mismas. Los datos necesitan ser gestionados y almacenados en bases de datos para luego analizarlos mediante una aplicación que facilite el trabajo; sin la misma sería engorroso su análisis para una posterior toma de decisiones, pues existiría un flujo de información demasiado grande, ocasionando pérdida de tiempo y gasto innecesario de recursos.

La mayoría de las organizaciones utilizan reportes para analizar los datos almacenados. De esta manera los directivos pueden dar seguimiento al negocio a partir de los reportes de información, propiciando un correcto funcionamiento de la empresa. Al ser este un proceso que ocurre de forma manual se ha necesitado informatizar su funcionamiento, pues el aumento de datos trae consigo la ocurrencia de errores humanos que pueden alterar la información. A raíz de este problema surgen los generadores de reportes, los cuales han demostrado ser una alternativa viable para el control de los datos, como consecuencia, los mismos son considerados una necesidad principal en la Inteligencia de negocio pues organizan la información para que esta sea comprendida por el usuario, mostrándola con un diseño atractivo. Son herramientas complementarias a los sistemas de información, utilizan un lenguaje transparente para el cliente por medio del cual este realizan consultas a la base de datos y se obtiene información de ella en forma de reporte. [1]

Cuba no está ajena al uso de estas tecnologías, las cuales se actualizan constantemente para estar a la par del resto de mundo. Múltiples son las empresas que en el país se encargan del desarrollo de soluciones informáticas destinadas a mejorar la calidad de trabajo en las empresas, siendo la Universidad de las Ciencias Informáticas (UCI) uno de los mayores centros de producción de software. Está estructurada por diversos centros productivos, entre los que se encuentra el Centro de Tecnología de Gestión de Datos (DATEC), ubicado en la Facultad de Ciencias y Tecnologías Computacionales. DATEC se encarga de la realización del proyecto Sistema Integrado de Gestión Estadística (SIGE); es una aplicación web enriquecida desarrollada sobre tecnologías libres que tiene como principal objetivo la informatización de los procesos de captura de datos de una institución. Sus procesos fundamentales son el diseño de plantillas de tipo formulario, la captura de datos sobre estos diseños de plantillas, la validación de los datos capturados y la recuperación de información a través de reportes. El mismo almacena toda la estadística en una sola Base de Datos y ofrece un apoyo en la toma de decisiones en las empresas mediante los reportes. Actualmente SIGE presenta la versión 3, ofreciendo una personalización para cada institución en la que es desplegado. El sistema utiliza un Módulo Visor de Reportes del Generador Dinámico de Reportes (GDR v1.8), para la visualización de los reportes del sistema. Actualmente se está desarrollando la versión 4.0 del producto, utilizando para la lógica del negocio y el acceso a los datos el framework Symfony v2.8 y para el diseño de interfaz gráfica de usuario ExtJS v6.0, lo que ha traído como consecuencia que en estos momentos no es posible la visualización de reportes pues no se integra con el Módulo Visor de Reportes que se utiliza en SIGE v3.0 el cual fue desarrollado con el framework Symfony v1.2.7 y ExtJS v2.2 para el diseño de la interfaz gráfica de la aplicación.

Debido a la necesidad de dar solución a la situación planteada se identifica como problema de la investigación: El Sistema Integrado de Gestión Estadística (SIGE v4.0) no permite visualizar los reportes lo que trae consigo la imposibilidad de mostrar la información almacenada.

Para dar solución al problema de investigación planteado se define como objetivo general desarrollar el Módulo Visor de Reportes para SIGE v4.0 que permita mostrar la información almacenada.

Materiales y métodos o Metodología computacional

Para el cumplimiento del objetivo general planteado se utilizarán las siguientes herramientas y tecnologías:

Una metodología de desarrollo de software es un marco de trabajo que se usa para estructurar, planificar y controlar el proceso de desarrollo de sistemas de información. Una gran variedad de estos marcos de trabajo ha evolucionado durante los años, cada uno con sus propias fortalezas y debilidades. [2], entre las metodologías de desarrollo se encuentra AUP.

AUP es la metodología de desarrollo de software que se utiliza en los proyectos productivos de la UCI. La misma se basa en una variación del "Proceso Unificado Ágil" o Agile Unified Process en inglés. Esta metodología describe de una manera simple y fácil de entender la forma de desarrollar aplicaciones de software de negocio usando técnicas ágiles y conceptos que aún se mantienen cálidos en RUP.

Además se utilizó como UML 2.1 como lenguaje de modelado y Visual Paradigm 8.0 como herramienta CASE para establecer la estructura interna del software.

El lenguaje de programación utilizado fue PHP7: Acrónimo de "PHP: HypertextPreprocessor", es un lenguaje de 'scripting' de propósito general y de código abierto que está especialmente pensado para el desarrollo web y que puede ser embebido en páginas HTML. Su sintaxis recurre a C, Java y Perl, siendo así sencillo de aprender. El objetivo principal de este lenguaje es permitir a los desarrolladores web escribir dinámica y rápidamente páginas web generadas; aunque se puede hacer mucho más con PHP. [3]

Symfony 2.8 se utilizócomo famework de desarrollo y la librería de JavaScript ExtJS 6 además de NetBeans 8.0.2 como Entorno de Desarrollo Integrado. Para el trabajo con base de datos PostgreSQL 9.4 y PgAdmin III, como servidor de aplicaciones Apache 2.4.

Resultados y discusión

Para el desarrollo del Módulo Visor de Reportes de SIGE v4.0se siguieron las buenas prácticas aplicadas desde las primeras etapas de desarrollo del proyecto, tanto en la ingeniería del software como en la codificación; empezando con la definición de la arquitectura del software, donde gracias a la flexibilidad,

escalabilidad, mantenibilidad y reusabilidad .

Un ejemplo de estas buenas prácticas es el uso de un estándar de codificación el cual comprende todos los aspectos de la generación de código. Si bien los programadores deben implementar un estándar de forma prudente, éste debe tender siempre a lo práctico. Un código fuente completo debe reflejar un estilo armonioso, como si un único programador hubiera escrito todo el código de una sola vez. Al comenzar un proyecto de software, establezca un estándar de codificación para asegurarse de que todos los programadores del proyecto trabajen de forma coordinada. Cuando el proyecto de software incorpore código fuente previo, o bien cuando realice el mantenimiento de un sistema de software creado anteriormente, el estándar de codificación debería establecer cómo operar con la base de código existente. [4]

Estándares de codificación utilizados

- Los bloques de código siempre deben estar encerrados por llaves, si consta de una línea no es necesario utilizar llaves.
- Los nombres de clases pueden contener sólo caracteres alfanuméricos.
- Los números están permitidos en los nombres de clase.
- Los nombres de funciones y variables deben empezar siempre con una letra minúscula utilizando la forma "camelCase".
- Se recomienda en sentido general la elocuencia, los nombres de las funciones deben describir su propósito y comportamiento.
- Los ficheros tienen que concluir con la extensión .js, y no deben incluir signos de puntuación excepto – (guión medio) o _ (guión bajo).
- Los métodos de los objetos deben ser nombrados utilizando la forma 'camelCase '.
- Usar paréntesis en expresiones que implican distintos operadores para evitar problemas con el orden de precedencia de los operadores. Incluso sí parece claro el orden de precedencia de los operadores, podría no ser así para otros, no se debe asumir que otros programadores conozcan el orden de precedencia.
- Las cadenas tienen que ser definidas utilizando comillas simples siempre que sea posible, para obtener un mejor rendimiento.

A continuación, se muestra un ejemplo del estándar de codificación utilizado en el desarrollo de la aplicación:

```
public function getReportAllAction(Request $request) {
    try {
        $sc = new SdrComunicationApi\AuthenticityServices();
        $response = json_decode($sc->login());

        if ($response)
            $token = $response->items[0]->token;

        $item = array();
        $item['value'] = 'pptx';

        $data = array();
        $data['format'] = $item;
        $params_report_export = array(
            'token' => $token,
            'items' => $data
        );
        $r_export = new SdrComunicationApi\ExportServices();

        $datos = $r_export->findName($params_report_export);
        $valor = json_decode($datos);
        $reportes = $valor->items;


        return new Response(json_encode(array("success" => true, "data" => $reportes,
            "message" => $this->get('translator')->trans('Reportes Obtenidos'), HttpCode::HTTP_OK)));
    } catch (\Exception $e) {
        return new Response(json_encode(array("success" => false,
            "message" => $e->getMessage(), HttpCode::HTTP_SERVER_ERROR)));
    }
}
```

Fig1: Código fuente

Serie Científica de la Universidad de las Ciencias Informáticas
Vol. 11, No. 6, Mes Junio, 2018
ISSN: 2306-2495 | RNPS: 2343
Pág. 43-55
http://publicaciones.uci.cu

El Módulo Visor de Reportes para SIGE v4.0 permite visualizar y actualizar la lista de reportes existentes, buscar un reporte registrado en el sistema, exportar los reportes diseñados a diversos formatos como html, pdf, excel, word, csv, txt,ods, odt, rtf, ppt y xml. Además permite generar un informe asociado a diversos reportes que seleccione el usuario, gestionar parámetros de los reportes y aplicar filtros a un reporte seleccionado.

A continuación se muestran las imágenes de las interfaces del módulo.

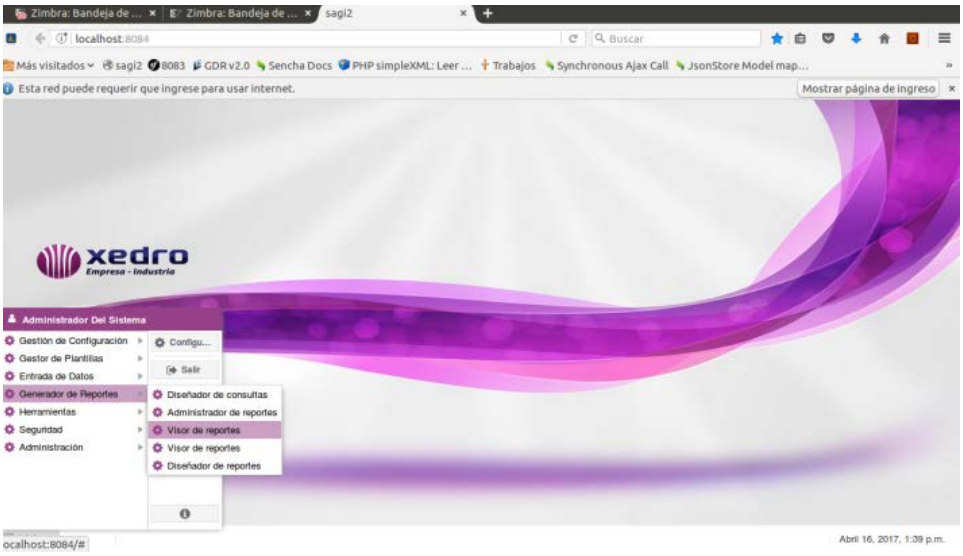

Fig2: Interfaz para acceder al Módulo Visor de Reportes para SIGE v4.0

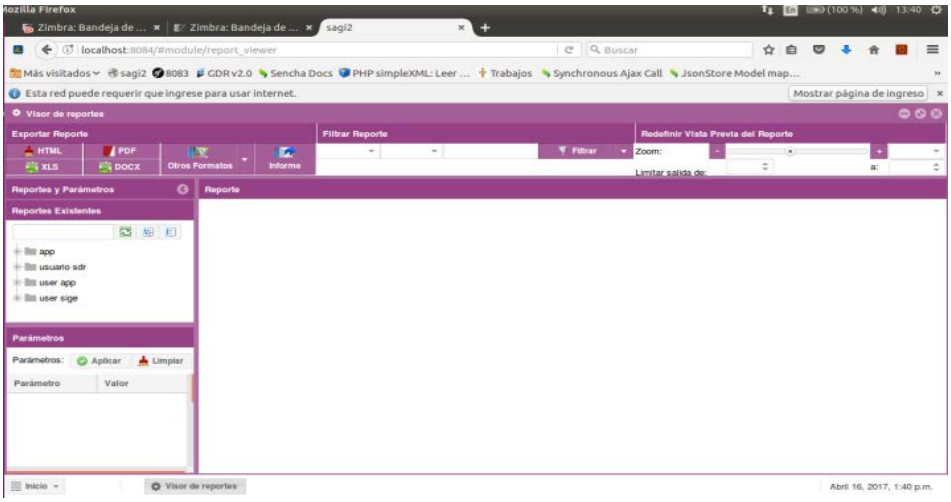

Fig3: Interfaz principal del Módulo Visor de Reportes para SIGE v4.0

Serie Científica de la Universidad de las Ciencias Informáticas
Vol. 11, No. 6, Mes Junio, 2018
ISSN: 2306-2495 | RNPS: 2343
Pág. 43-55
http://publicaciones.uci.cu

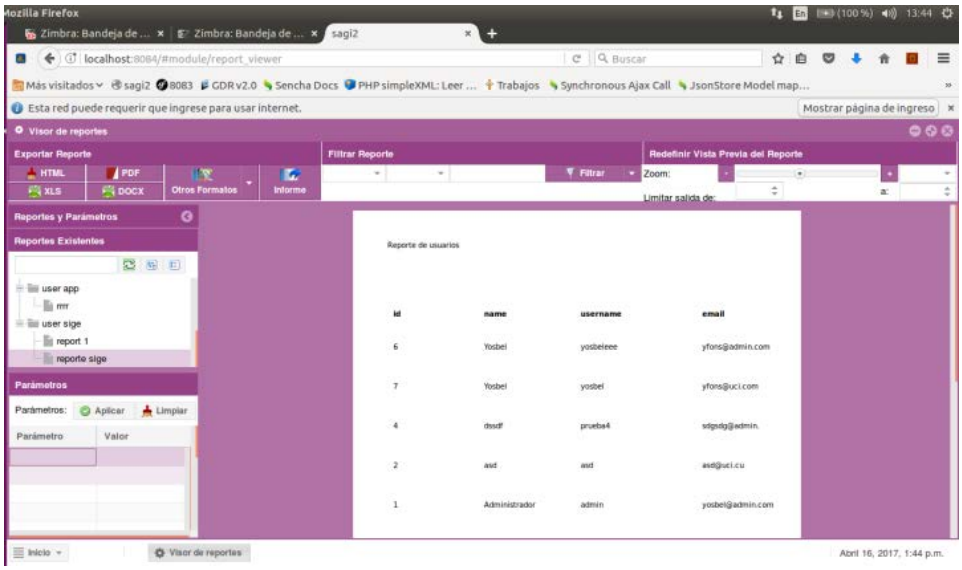

Fig4: Ejemplo de cómo el modulo visualiza los reportes existentes una vez seleccionado

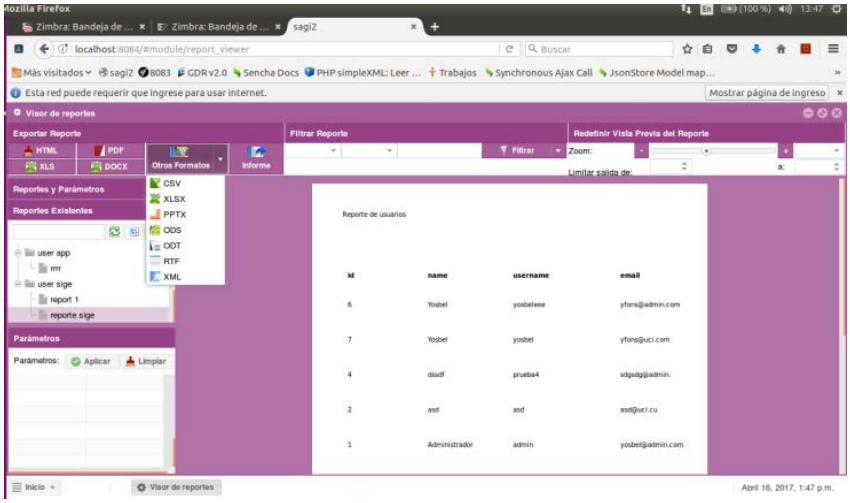

Fig5: Ejemplo de cómo el módulo permite exportar a diferentes formatos el reporte visualizado.

Pruebas de software

Son una actividad en la cual un sistema o componente es ejecutado bajo unas condiciones o requisitos especificados, los resultados son observados y registrados, y una evaluación es hecha de algún aspecto del sistema o componente. [5].

A continuación se muestra las pruebas realizadas al Módulo Visor de Reportes para SIGE v4.0 para comprobar su correcto funcionamiento.

Prueba de Unidad

Es la prueba enfocada a los elementos testeables más pequeño del software. Es aplicable a componentes representados en el modelo de implementación para verificar que los flujos de control y de datos están cubiertos, y que ellos funcionen como se espera. La prueba de unidad siempre está orientada a caja blanca. Antes de iniciar cualquier otra prueba es preciso probar el flujo de datos de la interfaz del componente. Si los datos no entran correctamente, todas las demás pruebas no tienen sentido. El diseño de casos de prueba de una unidad comienza una vez que se ha desarrollado, revisado y verificado en su sintaxis el código a nivel fuente. [5]

Método de Caja Blanca

Las pruebas de Caja Blanca se centran en los detalles procedimentales del software, por lo que su diseño está fuertemente ligado al código fuente. Se escogen distintos valores de entrada para examinar cada uno de los posibles flujos de ejecución del programa y cerciorarse de que se devuelven valores de salida adecuados.[5]

Mediante la prueba de Caja Blanca se pueden obtener casos de prueba que:

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

Por lo anteriormente mencionado se considera a la prueba de Caja Blanca como uno de los tipos de pruebas más importante que se le aplica al software, logrando como resultado que disminuya el número de errores existente en los sistemas, aumentando así la calidad y confiabilidad. [5].

Técnica del Camino Básico

La técnica del camino básico permite obtener una medida de la complejidad lógica de un diseño y utilizar esta medida como una guía para la definición de un conjunto básico. En la técnica se derivan los casos de prueba a partir de un conjunto dado de caminos independientes por los cuales se puede circular el flujo de control. Para obtener el conjunto de caminos se realiza el Grafo de Flujo asociado y se calcula su complejidad ciclomática. [5].

A continuación, se muestran los pasos para aplicar la técnica:

- A partir del diseño o del código fuente, se dibuja el grafo de flujo asociado.
- Se calcula la complejidad ciclomática del grafo.

Serie Científica de la Universidad de las Ciencias Informáticas
Vol. 11, No. 6, Mes Junio, 2018
ISSN: 2306-2495 | RNPS: 2343
Pág. 43-55
http://publicaciones.uci.cu

- Se determina un conjunto básico de caminos independientes.
- Se preparan los casos de prueba que obliguen a la ejecución de cada camino del conjunto básico.

Los casos de prueba derivados del conjunto básico garantizan que durante la prueba se ejecuta por lo menos una vez cada sentencia del programa.

Notación del Grafo de Flujo

Para aplicar la técnica del camino básico se debe introducir una sencilla notación para la representación del flujo de control, el cual puede representarse por un Grafo de Flujo. Cada nodo del grafo corresponde a una o más sentencias de código fuente. Todo segmento de código de cualquier programa se puede traducir a un Grafo de Flujo. [5].

```
public function exportReportAction(Request $request) {
    try {
        $sc = new SdrComunicationApi\AuthenticityServices();
        $response = json_decode($sc->login());

        if ($response)
            $token = $response->items[0]->token;
        $params = array(
        );
        $item = array();
        $item['reportId'] = $request->get('idReporte');
        $item['dbConnectionId'] = $request->get('connectionId');
        $item['params'] = json_encode($params);
        $export[] = $item;
        $data = array();
        $data['export'] = $export;
        $data['format'] = $request->get('format');
        $params_report_export = array(
            'token' => $token,
            'items' => $data
        );
        $s_report = new SdrComunicationApi\ReportServices();
        $dato = $s_report->export($params_report_export);
        $valor = json_decode($dato);
        $url = $valor->url;
        return new Response(json_encode(array("success" => true, "data" => $url,
            "message" => $this->get('translator')->trans('Reporte Exportado Satisfactoriamente'), HttpCode::HTTP_OK)));
    } catch (\Exception $e) {
        return new Response(json_encode(array("success" => false,
            "message" => $e->getMessage(), HttpCode::HTTP_SERVER_ERROR)));
    }
```

Fig6: Fragmento de código a analizar

Serie Científica de la Universidad de las Ciencias Informáticas
Vol. 11, No. 6, Mes Junio, 2018
ISSN: 2306-2495 | RNPS: 2343
Pág. 43-55
http://publicaciones.uci.cu

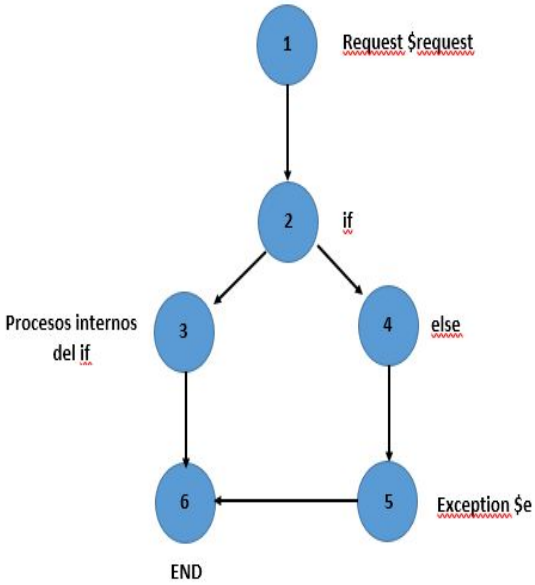

Fig7: Representación del grafo de flujo

Complejidad ciclomática

La complejidad ciclomática es una métrica de software que proporciona una medición cuantitativa de la complejidad lógica de un programa. El valor calculado como complejidad define el número de caminos independientes del conjunto básico de un programa y da un límite superior para el número de pruebas que se deben realizar para asegurar que se ejecute cada sentencia al menos una vez. [5]

Un camino independiente es cualquier camino del programa que introduce por lo menos un nuevo conjunto de sentencias de procesamiento o una nueva condición. El camino independiente se debe mover por lo menos a una arista que no haya sido recorrida anteriormente.

En la Figura 14 un ejemplo de camino independiente sería:

Camino 1: 1 – 2 – 3 – 6.

Camino 2: 1 – 2 – 4 – 5 – 6.

La complejidad ciclomáticaV(G), se define como:

V(G) = A – N + 2

Dónde: A es el número de aristas del grafo y N es el número de nodos.

V(G) = 6 aristas - 6 nodos + 2 = 2.

Concluyéndose así que la complejidad ciclomática del grafo de flujo de l figura 15 es igual a 2.

Serie Científica de la Universidad de las Ciencias Informáticas
Vol. 11, No. 6, Mes Junio, 2018
ISSN: 2306-2495 | RNPS: 2343
Pág. 43-55
http://publicaciones.uci.cu

Una vez concluido el cálculo de la complejidad se evalúa el riesgo de realización del método.

Tabla 1: Evaluación del riesgo según la complejidad ciclomática.

Complejidad Ciclomática	Evaluación del Riesgo
1-10	Programa simple, sin mucho riesgo
11-20	Más complejo, riesgo moderado
21-50	Complejo, programa de alto riesgo
50	Programa no testeable, muy alto riesgo

Dependiendo del resultado de la complejidad ciclomática que supone la realización del método y su impacto en la aplicación se define la funcionalidad como “simple, sin mucho riesgo”

Derivación de casos de prueba

Luego de tener elaborados los Grafos de Flujos y los caminos a recorrer, se preparan los casos de prueba que forzarán la ejecución de cada uno de esos caminos. Se escogen los datos de forma que las condiciones de los nodos predicados estén adecuadamente establecidas, con el fin de comprobar cada camino.Luego de confeccionados los casos de prueba se ejecutaron cada uno de estos y se comparan los resultados con los esperados. Una vez terminados todos los casos de prueba, se está seguro de que todas las sentencias del programa se han ejecutado por lo menos una vez.

1.1.1. Prueba de Integración

La prueba de integración es ejecutada para asegurar que los componentes en el modelo de implementación operen correctamente cuando son combinados para ejecutar un caso de uso. Se prueba un paquete o un conjunto de paquetes del modelo de implementación. Estas pruebas descubren errores en las especificaciones de las interfaces de los paquetes. Esta prueba debe ser responsabilidad de desarrolladores y de independientes, sin solaparse las pruebas. [5].

Es una técnica sistemática para construir la estructura del programa mientras que, al mismo tiempo, se llevan a cabo pruebas para detectar errores asociados con la interacción. En este se verifica que módulos individuales son combinados y probados como un grupo. En una interfaz es posible perder datos, un módulo podría tener un efecto adverso e inadvertido sobre otro [5].

Se decide aplicar como tipo de prueba la prueba de integración ascendente, debido a que la aplicación a implementar es un módulo que se integrará posteriormente a un sistema se tiene en cuenta que comenzará la construcción en el nivel más bajo, llamado también nivel atómico, en el que se encuentran los componentes más bajos de la estructura del programa. Debido a que los componentes se integran de abajo hacia arriba siempre está disponible el

procesamiento requerido para los componentes subordinados a un determinado nivel y se elimina la necesidad de resguardarlos.

Pruebas de Sistema

Son las pruebas que se hacen 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. [5]..

Proporciona un aseguramiento final de que el software cumple con todos los requisitos funcionales, de comportamiento y desempeño. La prueba se concentra en las acciones visibles para el usuario y en la salida que este puede reconocer. La validación se alcanza cuando el software funciona de tal manera que satisface las expectativas razonables (especificación de requisitos de software) del cliente. [5].

Se decide realizar las Pruebas Funcionales como tipo de prueba pues asegura que los requisitos funcionales estén implementados según los requerimientos, esto incluye el correcto funcionamiento de la navegación, entrada de datos, procesamiento, validación y obtención de resultados. Las pruebas funcionales están enfocadas a los requisitos funcionales, aunque pueden estar basadas directamente en los Casos de Uso y las reglas del negocio. La meta es verificar que existe un correcto funcionamiento en la parte visual de la aplicación, verificando la navegabilidad, la validación de datos, así como la entrada y salida de los mismos. Estas pruebas están basadas en la técnica de Caja Negra, la cual verifica la aplicación y sus procesos internos mediante la interacción con la aplicación y analiza la salida y los resultados.

Prueba de Caja Negra: Se centra en los requisitos funcionales del software. Permite a los ingenieros de software obtener conjuntos de condiciones de entrada que ejerciten completamente todos los requisitos funcionales de un programa. [5].

Partición Equivalente: Es una técnica de prueba de caja negra que divide el dominio de entrada de un programa en clases de datos a partir de las cuales pueden derivarse casos de prueba. El diseño de casos de prueba para partición equivalente se basa en una evaluación de las clases de equivalencia para una condición de entrada. [5].

Los resultados de las pruebas fueron satisfactorios, en la primera iteración se detectaron 17 No Conformidades de mediana complejidad que fueron resueltas y en la segunda iteración no se detectó ninguna No Conformidad, permitiendo arribar a la conclusión de que la nueva versión está en óptimas condiciones para su explotación en un ambiente de despliegue real.

Conclusiones

Se desarrolló un Módulo Visor de Reportes para la versión 4.0 del Sistema Integrado de Gestión Estadística el cual permite la visualización y exportación de los reportes del sistema ayudando a la toma de decisiones a los usuarios finales del sistema.

Se le realizaron pruebas de unidad, integración y de sistema al módulo comprobando su correcto funcionamiento.

Referencias

[1] Silva Hernández, Iliana. Generador Automático de Reportes Dinámicos: Departamento de Computación, Centro de Investigación y de Estudios Avanzados del IPN 2003. Ciudad México : s.n., 2010.

[2] Laboratorio Nacional de Calidad de Software. 2009. INGENIERÍA DEL SOFTWARE: METODOLOGÍAS Y CICLOS DE VIDA. Madrid : Instituto Nacional de Tecnologías de la Comunicación INTECO , 2009.

[3] KANEIWA, Ken, MIZOGUCHI, Riichiro y NGUYEN, Philip HP. 2015. A Logical and Ontological Framework for Compositional Concepts of Objects and Properties. s.l. : New Generation Computing, 2015.

[4] BRUEGGE, Bernd, DUTOIT, Allen. Ingeniería de software orientado a objetos. . 2008.

[5]PRESSMAN, Roger. Ingeniería del Software. Un enfoque práctico. s.l. : McGraw-Hill/Interamericana, 2009.