Universidad de las Ciencias Informáticas
Facultad 5

Subsistema de Gestión de Riesgos del Sistema Automatizado de Monitoreo y control de Servicios para el Centro de Soporte UCI

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

Autora: Heydis Gutierrez Solano

Tutores: Dra. Marely del Rosario Cruz Felipe

Msc. Yulio Seriocha García Gallardo

La Habana, 2013.

"Año 55 de la Revolución"

DECLARACIÓN DE AUTORÍA

Declaro por este medio que soy la única autora del presente trabajo de diploma y autorizo a la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio, así como los derechos patrimoniales con carácter exclusivo.

Para que así conste, firmamos la presente declaración jurada de autoría a los _____ días del mes de _____ del año _____.

_____

Firma del autor

Heydis Gutierrez Solano

_____

Firma del tutor

Dra. Marely del Rosario Cruz Felipe

_____

Firma del tutor

Msc. Yulio Seriocha García Gallardo

DATOS DE CONTACTO

Nombre y apellidos del tutor: Dra. Marely del Rosario Cruz Felipe.
Institución: Universidad de las Ciencias Informáticas.
Título: Doctora en Ciencias Técnicas.
Correo electrónico: marely@uci.cu

Jefa de departamento de Programación y Sistemas Digitales de la facultad 5, Profesora Titular, Máster en Telemática y Doctora en Ciencias Técnicas.

Nombre y apellidos del tutor: Yulio Seriocha García Gallardo.
Institución: Universidad de las Ciencias Informáticas.
Título: Máster en Informática Aplicada.
Correo electrónico: ygarciag@uci.cu

Graduado de Ingeniero en Ciencias Informáticas de la UCI del año 2007, se desempeña como especialista del Centro Soporte UCI. Es Máster en Informática Aplicada. Categoría docente: Asistente.

Dedicatoria

A mi madre, por su amor incondicional.
A Iván, mi novio, por siempre estar a mi lado y ser lo más lindo que me ha pasado en la universidad.

Agradecimientos

A mis padres, en especial a mi madre, por estar siempre cuando más la necesito, por depositar toda su confianza en el empeño de hacer de mí la persona que soy, por su infinito amor y comprensión… gracias de todo corazón… estoy muy orgullosa de ser su hija.

A mi hermana, por sus consejos y por su preocupación.

A mi cuñado Over y a mi padrastro Jorge.

A mi novio, por ser la persona que siempre estuvo ahí, apoyándome, exigiéndome y dándome fuerzas para seguir adelante.

A mis tutores por brindarme su apoyo y confiar en mí para el desarrollo del presente trabajo de diploma.

A mis amistades de la universidad por soportarme en estos días llenos de estrés.

A la Revolución y a Fidel por darme la oportunidad de superarme.

A todas las personas que de una forma u otra han contribuido en el desarrollo del presente trabajo de diploma.

Muchas Gracias

RESUMEN

Los riesgos implican un factor muy importante para la gestión de servicios de Tecnologías de Información (TI), garantizando a las empresas poder mitigar problemas que afecten los servicios brindados por ellas. En el Centro de Soporte de la Universidad de las Ciencias Informáticas no se cuenta con una estrategia para la gestión de riesgo de los servicios de TI.

De ahí que en el presente trabajo de diploma, se propone el desarrollo de un Subsistema de Gestión de Riesgos de servicios de Tecnología de Información para el sistema SAMOS. Para ello se realizó un estudio del estado del arte de diferentes estándares para la mejora de servicios de TI y se seleccionó cuál era el indicado para la Gestión de Riesgos de servicios de TI, y se analizaron los procesos de Gestión de Riesgos que realizan diferentes sistemas informáticos.

Fueron descritas las herramientas y tecnologías utilizadas en la solución desarrollada que permitieron una implementación en corto tiempo. Finalmente, se describe de forma detallada todo el proceso de pruebas, mostrando un resultado general de todas las validaciones realizadas al sistema desarrollado.

Palabras claves: Gestión de Riesgos, servicios de Tecnología de Información.

ABSTRACT

The risks imply a very important factor for service management of Information Technology (IT) by guaranteeing to companies to mitigate problems affecting the services provided by them. In Support Centre at the University of Computer Sciences there is not a strategy for risk management of IT services.

Hence in this diploma thesis is proposed the development of a Risk Management Subsystem of Information Technology services for system SAMOS. For it a study was made of the state of art of different standards for the improvement of IT services and selected what was right for Risk Management of IT services, and was analysed the Risk Management processes that perform different informatics systems.

It was described the tools and technologies used in the solution developed that allowed an implementation in a short time. Finally, it describes in detail the entire process of testing, showing a general result of all the developed system validations performed.

Keywords: Risk Management, Information Technology Services.

ÍNDICE DE CONTENIDO

Índice de Contenido

INTRODUCCIÓN

En los últimos años las TI han alcanzado niveles de uso y aplicaciones tan variadas que se ha convertido en un área de gran amplitud e impacto en todos los aspectos de la vida cotidiana, incluyendo la dirección de cualquier empresa, en la cual es casi imprescindible. Es necesario establecer que las TI se le denomina como la utilización de tecnologías para el manejo y procesamiento de información, específicamente la captura, transformación, almacenamiento, protección, y recuperación de datos e información (1).

Cuando los servicios de las TI son críticos, cada una de las actividades que se realizan deben ser ejecutadas con un orden definitivo, de ahí parte la necesidad de gestionar los mismos, lo que representa un elemento crucial para toda empresa que persiga obtener una ventaja competitiva y mejorar la calidad de sus productos.

La Gestión de Servicios de TI (GSTI) es una disciplina basada en procesos horizontales diseñados para facilitar una metodología orientada al cliente, mejorando considerablemente la alineación entre la organización de TI (Proveedora de Servicios de TI) y los clientes (usuarios responsables del uso de estos servicios para el cumplimiento de los objetivos del negocio) poniendo énfasis en los beneficios que puede percibir el cliente final (2).

Dentro de los objetivos de la GSTI se encuentra proporcionar una adecuada administración de la calidad y principalmente reducir los riesgos asociados a los servicios de TI, entre otros. Con la misión de mejorar la GSTI actualmente se han desarrollado un conjunto de estándares y buenas prácticas tales como: ISO/IEC (Organización Internacional de Normalización / Comisión Electrotécnica Internacional, significado de sus siglas en español) 20000, ISO 9001, CMMI-SVC (Modelo de Madurez de la Capacidad Integrado para los Servicios, significado de sus siglas en español), ITIL (Biblioteca de Infraestructura de Tecnologías de Información, significado de sus siglas en español), COBIT (Objetivos de Control para Información y Tecnologías Relacionadas, significado de sus siglas en español).

COBIT aporta un factor diferencial enormemente importante, el cual es la orientación hacia el negocio. Permitiendo a las empresas aumentar el valor de las TI y reducir los riesgos asociados a proyectos tecnológicos, esto a partir de parámetros generalmente aplicables y aceptados, para mejorar las prácticas de planeación, control y seguridad de las TI.

Introducción

Cuba a pesar de ser un país subdesarrollado no está exento del avance de la ciencia y la tecnología; ejemplos que evidencia esto, son las empresas Datys, Desoft, Softel y la creación de la Universidad de las Ciencias Informáticas (UCI) en el año 2002 por el Comandante en Jefe Fidel Castro Ruz.

La UCI es un centro de estudio basado en el concepto de universidad productiva, tiene como objetivos lograr recursos humanos formados e ingreso de divisas por exportación. En este sentido se produce en la universidad una modificación de su plan de estudio y del modelo de producción simultáneamente, potenciando la sinergia entre las áreas de producción, formación, investigación y postgrado (3).

La Universidad cuenta con 22 centros de desarrollo, en los cuales se gestionan proyectos y servicios informáticos. Además cuenta con un Centro de Soporte, el cual brinda servicios de matenimiento y soporte técnico a todas las aplicaciones informáticas y productos desarrollados por la Red de Centros de Desarrollo y resuelve a los clientes las incidencias tecnológicas que tengan solución inmediata al alcance del equipo técnico.

Actualmente en el Centro de Soporte existen diferentes factores que afectan la calidad de los servicios que se prestan, entre los que podemos mencionar:

- La gestión de las incidencias ocurridas a las aplicaciones informáticas; se realiza actualmente sin tener en cuenta los Acuerdos de Niveles de Servicios (ANS) establecidos con el cliente.
- Incumplimiento en los tiempos de respuesta y resolución establecidos con el cliente.
- Los especialistas se demoran mucho en determinar la causa raíz y los síntomas ante un problema detectado.
- Se desconocen los cambios realizados a las aplicaciones informáticas, lo que provoca demoras en el servicio a los clientes.
- Existe un desconocimiento total sobre el nivel de satisfacción de los clientes.
- No se tiene conocimiento de las expectativas del cliente con respecto al servicio de soporte técnico.

- Desconocimiento de las actividades realizadas por los técnicos y especialistas de soporte durante la prestación del servicio.

Estos factores no se tienen en cuenta en ninguna documentación, debido a que no hay ningún método o estrategia que permita la automatización o gestión de los procesos que realiza el mismo, trayendo consigo que exista un mal funcionamiento en la prestación de los servicios, provocando insatifacción en los clientes.

Todos estos problemas conllevaron a la decisión de comenzar el desarrollo del Sistema Automatizado de Monitoreo y Control de Servicios (SAMOS), el cual permitirá llevar una detallada administración de los servicios que se brindan en el Centro de Soporte, aumentando la calidad de los mismos. Hasta el momento se planifica incorporar los módulos de Gestión de Auditoría, Gestión de Mapeo y Evaluación de Competencias, además de integrarse con una Base de Datos de Configuración.

El desarrollo de este sistema, trae consigo el surgimiento de la necesidad de gestionar riesgos para el Centro de Soporte, pues no existe actualmente ninguna gestión de riesgos, ni de las afectaciones en la prestación de los servicios que estos traen consigo, ni de sus soluciones, lo cual provoca un bajo nivel de respuesta a las incidencias tecnológicas.

A partir de la situación antes descrita, el **problema científico** queda planteado de la siguiente forma: ¿Cómo gestionar los riesgos de servicios TI en el Centro de Soporte de la UCI?

Para darle solución al problema antes expuesto se define como **objeto de estudio** la Gestión de Riesgos de servicios TI.

Por lo cual el **objetivo general** que se persigue es: desarrollar un subsistema de Gestión de Riesgos para el SAMOS en el Centro de Soporte UCI utilizando el estándar COBIT.

El **campo de acción** estaría enmarcado en el proceso de la Gestión de Riesgos de servicios de TI del estándar COBIT.

Para dar cumplimiento al objetivo general de investigación y a la situación problémica planteada, se proponen las siguientes **tareas investigativas**:

- Elaboración del diseño teórico de la investigación.

- Análisis de los conceptos y herramientas asociados a la gestión de riesgos de servicios TI para el desarrollo de la fundamentación teórica de la investigación.
- Análisis de las herramientas, metodologías y tecnologías a utilizar para el proceso de desarrollo del sistema.
- Confección de los artefactos correspondientes a las fases de la metodología de desarrollo a utilizar.
- Desarrollo e implementación de la solución propuesta.
- Elaboración del diseño de los casos de prueba de las funcionalidades implementadas.
- Ejecución de las pruebas de funcionalidad para la detección de errores.

Como **idea a defender** se plantea**:** El desarrollo de un sistema de gestión de riesgos para el SAMOS basado en COBIT garantizará la administración de riesgos de TI para el Centro de Soporte UCI.

Los **métodos científicos utilizados** en la presente investigación fueron:

- Métodos Teóricos:
  - Histórico – Lógico**:** Mediante este método se analizó la evolución de los diferentes problemas que traen consigo los riesgos de servicios TI, con el objetivo de realizar una gestión óptima de los mismos.
  - Analítico – Sintético**:** Con el uso de este método de investigación se analizaron las informaciones obtenidas, mediante su desglose, para después realizar una síntesis de las mismas y arribar a las principales ideas.
  - Modelación**:** Este método se utilizó para crear un prototipo funcional del sistema.
- Métodos Empíricos:
  - Consulta de las fuentes de información**:** Para seleccionar la información necesaria para construir el marco teórico.
  - Observación**:** Para reunir información visual sobre lo que el objeto de estudio hace y cómo se comporta.
  - Pruebas**:** Para valorar el desempeño del sistema elaborado.

Para mostrar el desarrollo de la investigación y los resultados de la misma, el trabajo se ha estructurado en tres capítulos, además de las Conclusiones, Recomendaciones, Glosario de Términos, Referencias Bibliográficas, Bibliografía y Anexos.

Capítulo I: Fundamentación teórica

En este capítulo se realiza el estudio del estado del arte sobre la Gestión de Riesgos. Se establece la fundamentación teórica de la investigación a partir de hacer un análisis crítico de las disposiciones establecidas por COBIT y de los principales sistemas de Gestión de Riesgos, además de la caracterización de las metodologías y herramientas de desarrollo de *software* que se utilizarán.

Capítulo II: Propuesta de solución

En el capítulo se definen los requisitos del sistema. Además se elabora una descripción de las funcionalidades mediante historias de usuarios, se muestra una planificación de cómo se trabajaron las iteraciones y también el diseño de la aplicación. Se realizan las descripciones de la arquitectura de *software* y los patrones de diseño utilizados. Además se muestra un modelo de datos, para una mejor comprensión de la información que será manipulada y se lleva a cabo la descripción de las tareas de ingeniería generadas por cada historia de usuario.

Capítulo III: Validación de la solución

El capítulo contiene todo lo relacionado con la validación de la solución propuesta. Se realizan las validaciones de las funcionalidades del sistema mediante pruebas de aceptación para comprobar que este se ajusta a las necesidades del cliente. Y además se realizan pruebas unitarias al código.

CAPÍTULO I: FUNDAMENTACIÓN TEÓRICA

1.1 Introducción

En el presente capítulo se exponen conceptos fundamentales sobre la gestión de riesgos (GR) de servicios de TI. Se hace un estudio de elementos y características fundamentales de COBIT, se realiza una comparación de este conjunto de mejores prácticas con otros estándares para determinar cuál es más factible aplicar al desarrollo de la investigación. Se efectúa un análisis de diferentes sistemas para la GR de servicios existentes en el mundo. Además de la realización de una breve descripción de las diferentes tecnologías a utilizar para lograr el cumplimiento del objetivo general.

1.2 Gestión de riesgos de servicios de TI

La disciplina de la GR es un proceso imprescindible para el logro de la misión de un servicio. La dependencia de los métodos de negocio es tal que los riesgos a los que está sometido este servicio pueden poner en peligro el logro de los objetivos del negocio.

Los objetivos de la GR de servicios de TI son mantener seguros los sistemas tecnológicos que almacenan, procesan o transmiten información sensible en organizaciones. El beneficio que se obtiene de este proceso, es que se puede contar con un tratamiento adecuado para cada riesgo relacionado con la tecnología, de tal forma que el riesgo que prevalece después de su tratamiento sea menor.

Debido a que la definición de GR se verá reflejada a lo largo de toda la investigación por su gran importancia, a continuación se referencian varios conceptos de este término de acuerdo a las posiciones de quienes en el mundo, la estudian y desarrollan.

1.2.1 Conceptos asociados a la GR de servicios de TI

Riesgo

- Según el PMBOK[1] un riesgo es un evento o condición incierta que, si sucede, tiene un efecto en por lo menos uno de los objetivos del proyecto (4).

[1] Guía de los Fundamentos para la Dirección de Proyectos.

- Es la posibilidad que un evento adverso, desgracia o contratiempo pueda manifestarse produciendo una pérdida (5).

- Proviene del italiano *risico* o *rischio* que, a su vez, tiene origen en el árabe clásico *rizq* (**"**lo que depara la providencia**"**). El término hace referencia a la proximidad o contingencia de un posible daño (6).

- La posibilidad de que una amenaza dada impacte las vulnerabilidades de un activo o grupo de activos y cause así, daños a la organización (7).

Tomando en cuenta las definiciones expuesta anteriormente, el riesgo es un suceso que puede ocurrir o no pero en caso de que ocurra provoca un efecto positivo o negativo a un individuo o institución.

Servicio

- Según el estándar ITIL un servicio es un medio para entregar valor a los clientes facilitándoles un resultado deseado sin la necesidad de que estos asuman los costes y riesgos específicos asociados (8).

- Con origen en el término latino *servitĭum*, la palabra servicio define a la actividad y consecuencia de servir (un verbo que se emplea para dar nombre a la condición de alguien que está a disposición de otro para hacer lo que éste exige u ordena) (9).

- Según Lamb, Hair y McDaniel, es el resultado de la aplicación de esfuerzos humanos o mecánicos a personas u objetos. Los servicios se refieren a un hecho, un desempeño o un esfuerzo que no es posible poseer físicamente (10).

- Es el conjunto de actividades que lleva a cabo internamente una empresa para poder responder y satisfacer las necesidades de un cliente (11).

- Son actividades que pueden identificarse aisladamente, esencialmente intangibles que proporcionan satisfacción (12).

Teniendo en cuenta los conceptos anteriores, los servicios son actividades intangibles realizadas por una o varias personas o por un equipo automático de una institución para satisfacer las necesidades de los clientes.

1.2.2 Diferentes conceptos de GR de servicios de TI

- La práctica compuesta de procesos, métodos y herramientas que provee de un entorno disciplinado para la toma de decisiones proactiva en base a determinar constantemente que puede ir mal (riesgos), se emplea para identificar cuáles son los riesgos más importantes en los cuales enfocarse e implementar estrategias para gestionarlos (13).

- El proceso de identificación, medida y administración de los riesgos que amenazan la existencia, los activos, las ganancias o al personal de una organización, o los servicios que ésta provee (14).

- Incluye los procesos relacionados con la planificación de la GR, la identificación y el análisis de riesgos, las respuestas a los riesgos, y el seguimiento y control de riesgos de un proyecto. Los objetivos son aumentar la probabilidad y el impacto de los eventos positivos, así como disminuir la probabilidad y el impacto de los eventos adversos para el proyecto (15).

Teniendo en cuenta las definiciones expuesta anteriormente se puede tomar para la presente investigación que la GR es un conjunto de procesos que permite reducir los efectos de los riesgos.

1.3 Principales estándares para la mejora de servicios TI

Actualmente existen diferentes estándares que esquematizan cada uno de los pasos que se deben tener presentes a la hora de llevar a cabo la mejora de servicios de TI para realizar un proceso más organizado. A continuación se presentan una breve descripción de varios de ellos.

1.3.1 COBIT

COBIT fue creado por ISACA (Asociación para la Auditoría y Control de Sistemas de Información, significado de sus siglas en español), e ITGI (Instituto de Administración de las Tecnologías de la Información, significado de sus siglas en español) a inicios de la década de los 90. Es un estándar que les permite a los administradores, usuarios y auditores reducir los espacios existentes entre los riesgos de negocio, las necesidades de control y los aspectos tecnológicos inherentes con el fin de lograr una adecuada gobernabilidad de los recursos de tecnologías de la información.

COBIT constituye una importante herramienta en la estructuración y control de los procesos de TI. De forma que atiende la demanda de las diversas áreas de la empresa, de los accionistas, de los órganos reguladores y de las entidades externas, por alineación, transparencia y ecualización de los riesgos de TI (16).

Actualmente se encuentra en la versión 5 la cual incorpora la última evolución de pensar en la gobernabilidad empresarial y técnicas de gestión, y establece los principios globalmente aceptados, prácticas, herramientas y modelos analíticos para ayudar a aumentar la confianza, y el valor de los sistemas de información. Además construye y amplía COBIT 4.1 mediante la integración de otros marcos, normas y recursos importantes, incluyendo Val IT [2]y Risk IT [3]de ISACA, ITIL y normas conexas de la ISO (Organización Internacional de Normalización, significado de sus siglas en español).

La adopción de COBIT conlleva a los siguientes beneficios (17):

- Proveer la creación de un lenguaje común a todos los involucrados en el control de los procesos.
- Optimizar el costo de los servicios de TI y tecnología.
- Ofrecer información más oportuna y de mayor calidad.
- Los requisitos de seguridad y privacidad serán más claros y la implementación será monitoreada con mayor facilidad.
- Mantener riesgos relacionados con TI a un nivel aceptable.
- Las auditorías serán más eficientes y exitosas.
- El cumplimiento de TI con los requisitos regulatorios será una práctica normal de gestión.

1.3.2 ISO/IEC 20 000

La ISO es la federación de alcance mundial integrada por cuerpos de estandarización nacionales de 162 países. Fue creada con el objetivo de facilitar el intercambio de servicios y bienes, y además para promover la cooperación en la esfera de lo intelectual, científico, tecnológico y económico. Por lo que todos los trabajos realizados por la ISO resultan en acuerdos internacionales los cuales son publicados como estándares internacionales (18).

[2] Marco para la gobernanza de las inversiones empresariales de TI (32).
[3] Marco basado en un conjunto de principios rectores para la gestión de los riesgos de TI (33).

Dentro de sus estándares se encuentra ISO/ IEC 20 000 el cuál es el primer estándar específico para la ITSM (Gestión de Servicios de TI, significado de sus siglas en español) y tiene como objetivo aportar los requisitos necesarios, dentro del marco de un sistema completo e integrado, que permita que una organización proporcione servicios TI gestionados, de calidad y que satisfagan los requisitos de negocio de sus clientes.

Esta norma se concentra en la gestión de problemas de tecnología de la información mediante el uso de un planteamiento de servicio de asistencia, considera también la capacidad del sistema, los niveles de gestión necesarios cuando cambia el sistema, la asignación de presupuestos financieros y el control y distribución del *software*. Es aplicable a cualquier organización, grande o pequeña, de cualquier sector.

Algunas de las principales ventajas de esta norma son:

- Es completamente compatible con ITIL.
- Las auditorías de certificación permiten la evaluación periódica de los procesos de gestión de servicios, lo que ayuda a mantener y mejorar la eficacia.
- Los proveedores de servicios de TI responden mejor a los servicios regidos por aspectos comerciales que a los impulsados por la tecnología.
- Los proveedores de servicios externos pueden utilizar la certificación como elemento diferencial y ampliar el negocio, ya que ésta se está convirtiendo cada vez más en un requisito contractual.
- El proceso de certificación puede reducir la cantidad de auditorías a proveedores y disminuir así los costos.

Dentro de las desventajas que presenta la norma se encuentran:

- La mejora en la provisión de los servicios y la reducción de costos puede ser insuficientemente visibles.
- Si la estructura de procesos se transforman en un objetivo en sí mismo, la calidad del servicio puede verse afectada.
- Puede no haber mejoras, debido a la falta de entendimiento sobre el alcance de los procesos, o los indicadores de performance, o como los procesos deberían ser controlados.

1.3.3 ITIL

El estándar británico propuesto por la OGC (Oficina Gubernamental de Comercio, significado de sus siglas en español) se desarrolló hace más de 15 años para

documentar las mejores prácticas para la gestión de servicios de TI, a través del aporte de expertos, consultores y profesionales de la industria. Está basado en la definición de procesos de mejores prácticas para la gestión y el soporte de servicios de TI, antes que en la definición de un marco de control de amplio alcance.

ITIL aboga que los servicios de TI deben estar alineados con las necesidades del negocio y apoyar los procesos de negocio. Proporciona orientación a las organizaciones sobre cómo utilizar las TI como una herramienta para facilitar el cambio de negocios, la transformación y el crecimiento (19).

Algunas de las principales ventajas que presenta son:

- Implantación de cambios más rápido.
- Reducción del número de cambios que necesiten ser revocados.
- Efectiva gestión de la capacidad.
- Mejor control de activos.
- Mayor alineamiento de TI con el negocio y enfoque al cliente.

Dentro de las desventajas que presenta ITIL, es que puede aportar a las empresas ayuda proporcionando mejores prácticas para la estandarización de los procesos pero, no cubre por igual todos los procesos TI, no hace referencia a los aspectos organizacionales y es débil a la hora de cubrir los aspectos de indicadores y de gobierno.

1.3.4 Valoración del autor

Después de un profundo análisis de las características expuestas de los estándares anteriormente descritos, se identificaron factores que potenciaron la elección de COBIT como guía para el proceso de implementación de la GR de servicios de TI, los cuales son:

- La ISO/IEC 20 000 se enfoca en la administración del servicio de TI, e ITIL se enfoca en la eficacia y eficiencia de los procesos y la gestión de los mismos para mejorar la percepción del cliente.

- COBIT se enfoca más hacia la auditoría del cumplimiento de los procesos con los estándares de autoridades, el control sobre las TI, la medición y la GR. Además en su última versión incluye procesos de los estándares ITIL, ISO/IEC 27000, ISO/IEC 31000, CMMI, entre otros.

1.4 Sistemas informáticos para la Gestión de Riesgos

Debido a que la GR es un proceso imprescindible para el buen funcionamiento de cualquier empresa, actualmente existen varios sistemas que entre sus funcionalidades incluye este proceso. A continuación se describen algunos de ellos.

1.4.1 ORCA GRC SUITE

Es un sistema para la administración de riesgos, cumplimiento y gobierno corporativo, que ayuda a centralizar, analizar y comprender los riesgos asociados con los procesos de negocio críticos, optimizando la seguridad y los niveles de cumplimiento, brindando a los tomadores de decisiones una experiencia única a través de los tableros de control con inteligencia en información GRC (20). En la figura 1 se muestra una imagen del sistema.

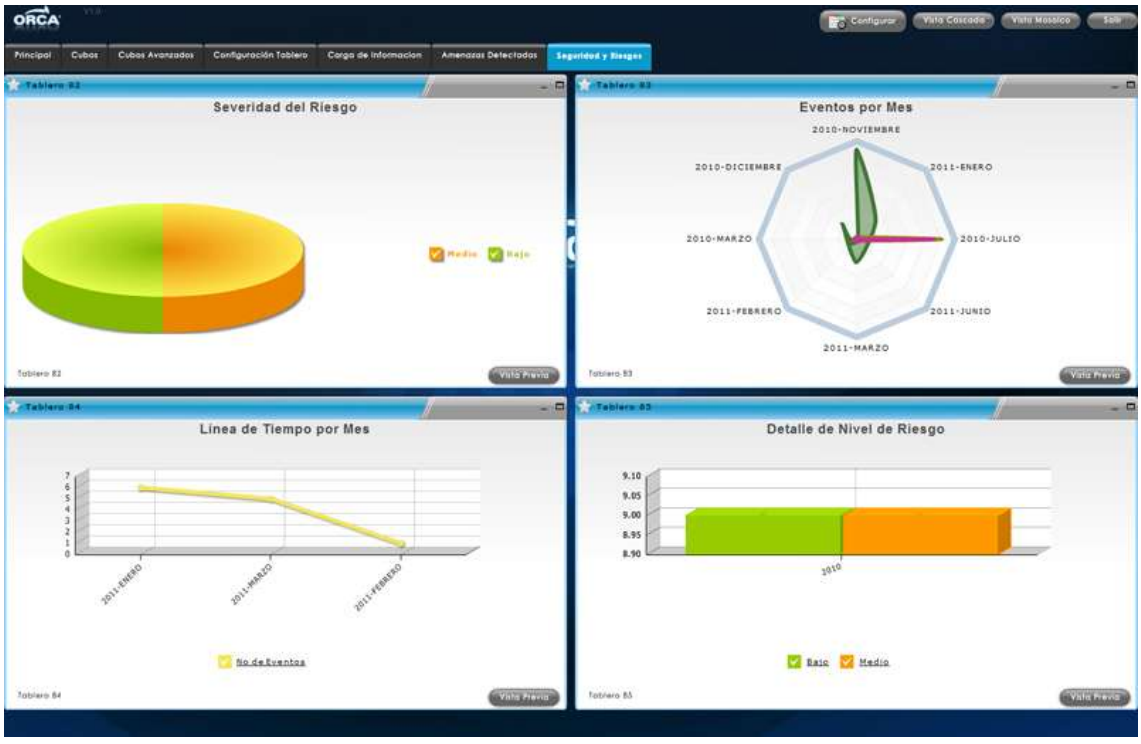


Figura 1. Sistema ORCA GRC SUITE

Según se muestra en la figura 1 este sistema permite mostrar reportes gráficos de de la gestión de los riesgos.

1.4.2 SOFTEXPERT

Es un sistema que sirve como base para la gestión de riesgos corporativos de una empresa a través de la habilidad de unificar y soportar diferentes categorías de riesgos (financiero, seguridad, conformidad, planificación) estando en conformidad con otras

soluciones relacionadas a la gestión de riesgos como Control de Gestión Financiera, la Gestión del Riesgo Operacional, Gestión de Riesgos de TI y Gestión de Cumplimiento General (22). Las funcionalidades proveídas por SoftExpert cumplen con los requisitos establecidos por estándares y reglamentaciones internacionales, tales como ISO 31000, ISO 27001, Sarbanes-Oxley (SOX), AS/NZS: 4360, SEC, NIST, PCAOB, Basiléia II, COSO, COBIT, entre otros. En la figura 2 se muestra una imagen del sistema.

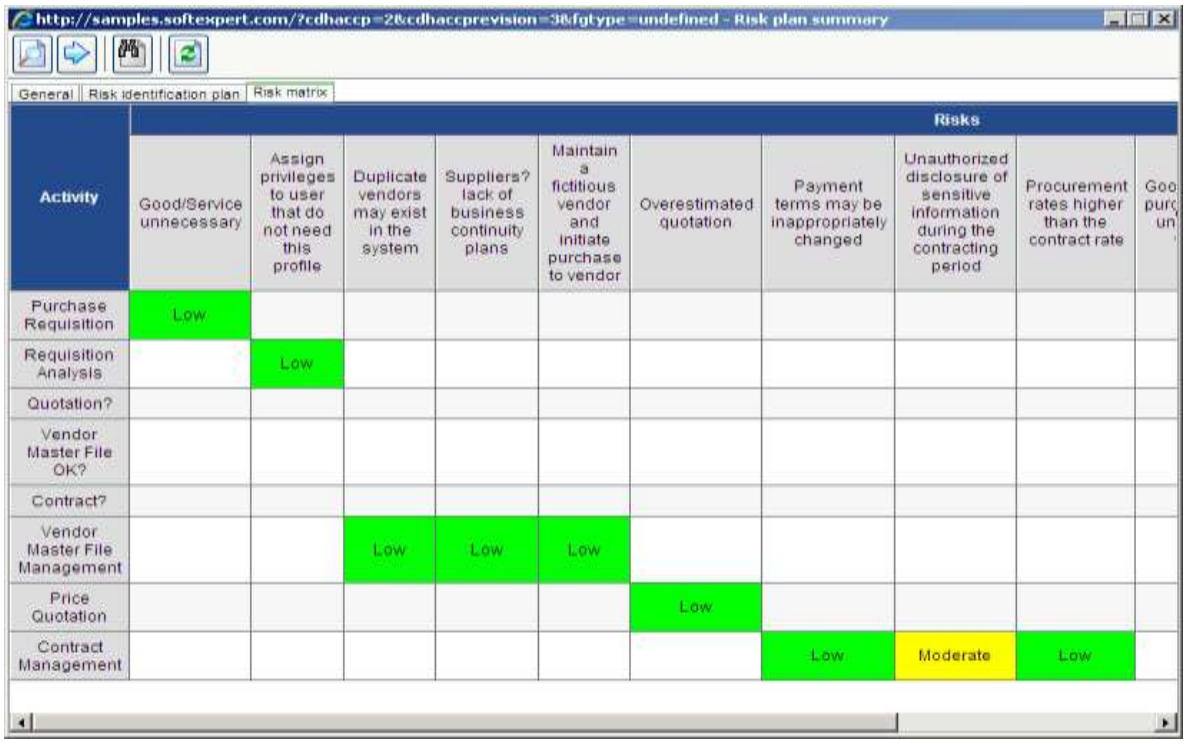


Figura 2. Sistema SOFTEXPERT

Según se muestra en la figura 2 este sistema permite mostrar una matriz de riesgo, donde se relacionan los riesgos y las actividades que estos afectan.

1.4.3 STREAM

Es un sistema libre desarrollado por *Acuity Risk Management LLP*. Dentro de sus características principales presenta que informa en tiempo real el estado de los riesgos, visualiza reportes a través de gráficos y los datos generados pueden ser exportados a MS Excel para su posterior análisis y presentación de informes. Es compatible solo con el Sistema Operativo *Windows*. En la figura 3 se muestra una imagen del sistema.

Figura 3. Sistema STREAM

Según se muestra en la figura 3 este sistema muestra los reportes a través a través de gráficos.

1.4.4 GESPRO 12.05

Es un paquete que posibilita la informatización de las organizaciones, además de la mejora integral de los procesos de planificación, control y seguimiento de proyectos. Desarrollado por la UCI y comercializable desde las empresas asociadas a la Universidad. GESPRO es una plataforma extensible, desarrollada con el *framework* [4] *Ruby on Rails* en su versión 2.3.5, está formada a partir de la integración de más de 18 herramientas libres, comercializadas bajo licencia GPL.

Se ha aplicado con buenos resultados en la red de centros de la Universidad. Sus funcionalidades están basadas en la guía del PMBOK y CMMI. Actualmente es utilizado por más de 7000 usuarios que gestionan actividades de más de 200 proyectos entre los que se encuentran proyectos para la informatización nacional como para la exportación. Este sistema ha sido desarrollado utilizando los modelos de líneas de productos de *software* como modelo industrial de desarrollo (36). En la figura 4 se muestra una imagen del sistema.

[4] Conjunto de procesos, técnicas y archivos previamente confeccionados, que facilitan y aceleran la producción de aplicaciones informáticas (35).

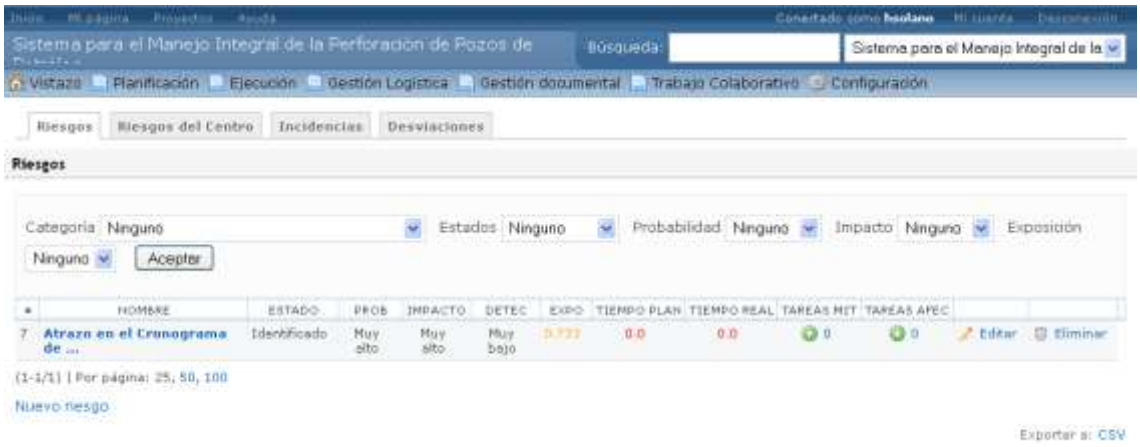


Figura 4. Sistema GESPRO

En la figura 4 se muestra el modulo de la gestión de riesgo, el cual exporta reportes solo en formato CSV.

1.4.5 Valoración del autor

Los dos primeros sistemas informáticos descritos anteriormente realizan una efectiva y rápida GR, presentan los resultados con gráficos que facilitan la compresión y generan detallados reportes, los cuales le permiten al cliente saber qué está yendo mal en su organización. Pero no son una vía para gestionar los riesgos de servicios de TI en el Centro de Soporte UCI debido a que no se comercializan libremente y además se encuentran optimizados para las características específicas de los entornos donde fueron desarrollados.

Lo mismo sucede con el sistema *STREAM*, que aunque se distribuye libremente, su autor, la empresa *Acuity Risk Management LLP* no permite la descarga de su código fuente, lo que imposibilita optimizarlo para las características específicas del Centro de Soporte. Además, otra de las desventajas que presenta es que solo funciona en el sistema operativo *Windows*.

Por su parte GESPRO, a pesar de ser un sistema elaborado en la misma universidad tampoco es una vía para gestionar los riesgos de servicios de TI en el Centro de Soporte debido a que está enfocado solamente para la GR como una de las áreas de procesos de la Gestión de Proyecto, en los proyectos productivos de la universidad.

Lo que se plantea anteriormente demuestra que existen sistemas informáticos que administran riesgos de servicios TI, pero no son viables para el Centro de Soporte, por

tanto se puede comprobar la necesidad de desarrollar un subsistema para la GR dentro del sistema SAMOS.

1.5 Herramientas, tecnologías y metodología para el desarrollo del *software*

Para llevar a cabo el desarrollo del sistema es necesario apoyarse en varias herramientas y tecnologías que posibiliten su creación. A continuación se presenta una breve descripción de las mismas.

1.5.1 Metodologías de desarrollo de *software*

Se entiende por metodología de desarrollo de *software* a un conjunto de pasos y procedimientos que deben seguirse para desarrollar un *software*. Las mismas deben ser seleccionadas y adaptadas de acuerdo al contexto del proyecto, o sea, del tamaño, la experiencia del personal, el tiempo estimado, entre otros factores. Se han desarrollado varias, por la importancia que tienen como guías en el desarrollo del *software*. A continuación se muestra una breve descripción de algunos ejemplos de metodologías utilizadas en la UCI.

Proceso Unificado de Desarrollo de *Software* (RUP)

Se caracteriza por ser iterativo e incremental donde cada fase se desarrolla en iteraciones. También está dirigido por los casos de uso que no son más que el instrumento para validar la arquitectura del *software* y extraer los casos de prueba. Esta metodología se centra en la arquitectura, aquí los modelos son proyecciones del análisis y el diseño constituye la arquitectura del producto a desarrollar. Se utiliza principalmente para proyectos largos y en los que son necesarios establecer una detallada documentación.

Metodología XP

Es una metodología ágil centrada en potenciar las relaciones interpersonales como clave para el éxito en el desarrollo de *software*. Se basa en una retroalimentación continuada entre el cliente y el equipo de desarrollo con una comunicación fluida entre todos los participantes, también busca simplificar las soluciones implementadas y coraje para múltiples cambios. XP define como fases fundamentales: exploración, planificación, diseño, implementación y pruebas.

Dentro de las principales características que presenta se encuentra que está concebida para proyectos pequeños y de desarrollo rápido, organiza el trabajo mediante fases, es flexible a los cambios que puedan ocurrir en las funcionalidades del sistema, además presenta ciclos cortos de trabajo y una generación de artefactos reducida.

Selección de la metodología de desarrollo a utilizar

Para la selección de la metodología de desarrollo de *software* se utilizó el método de estrella de Boehm, el cual consiste en evaluar cada uno de los parámetros representados en las situaciones y características particulares de un proyecto.

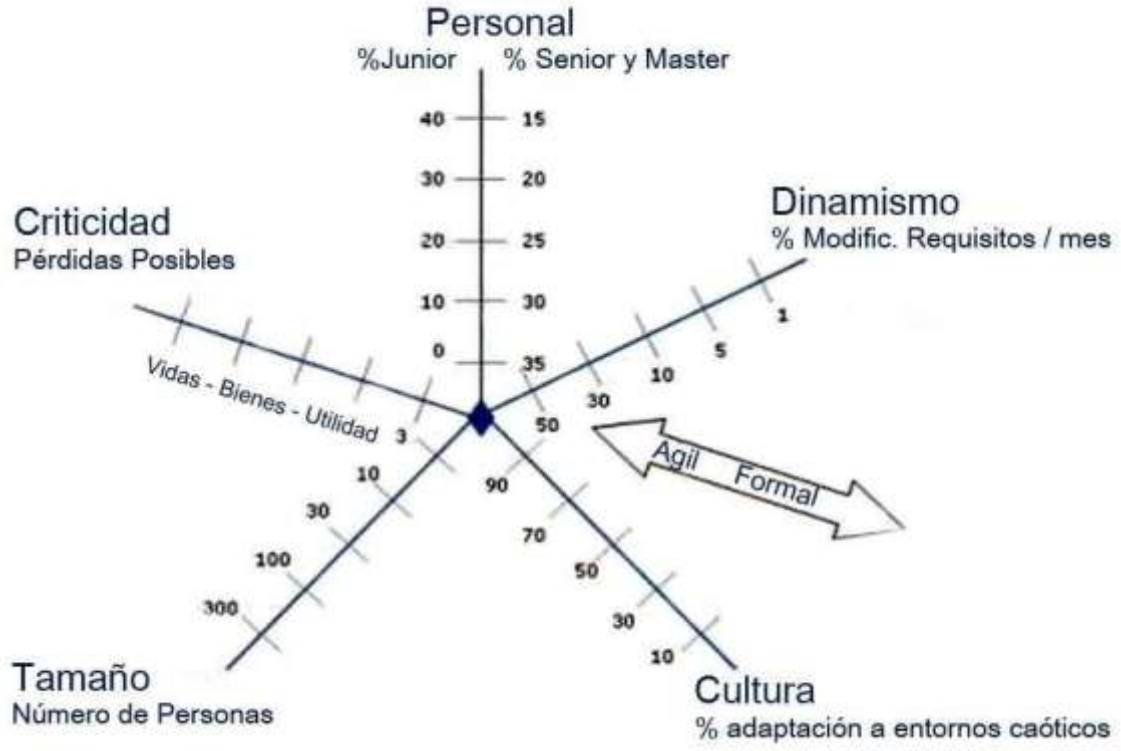


Figura 5. Criterio para la selección apropiada de una metodología de desarrollo, tomado de (34)

Una vez culminado este proceso se obtuvo como resultado que todos los parámetros se comportaron convergiendo hacia el centro, lo cual propició la adopción de una metodología ágil. Se decidió que para la realización del Subsistema de Gestión de Riesgos para el sistema SAMOS se utilizará XP debido a los siguientes elementos:

- XP es una metodología flexible ante los continuos cambios de los requisitos. Ésta condición está presente en el proceso de desarrollo del subsistema.

- XP se utiliza en proyectos con corto tiempo de entrega. Esta condición también se presenta en el proceso de desarrollo del subsistema.
- Además es la metodología que se utiliza para la implementación del sistema SAMOS por el equipo de desarrollo del mismo.

1.5.2 Herramientas para el modelado

Rational Rose Enterprise Edition

Rational Rose Enterprise Edition es una herramienta CASE (Ingeniería de *Software* Asistida por Computadora, significado de sus siglas en español) propietaria, desarrollada por *Rational Corporation*, Permite crear los diagramas que se van generando durante el proceso de desarrollo del *software* y posee un gran número de estereotipos predefinidos que facilitan el proceso de modelación. Otras de las características adicionales que posee son que soporta patrones *Analysis*, ANSI C++, Rose J y *Visual* C++, *Enterprise JavaBeans* 2.0, permite la ingeniería directa e inversa para construcciones comunes en *Java* 1.5. Incluye un complemento de modelado web, que proporciona el modelado para el desarrollo de aplicaciones web. Esta herramienta resulta de gran utilidad para los desarrolladores de proyectos por lograr cubrir todo el ciclo de vida del proceso desde su fase inicial hasta su fase final.

Visual Paradigm

Es una herramienta multiplataforma que utiliza UML (Lenguaje Unificado de Modelado, significado de sus siglas en español) como lenguaje de modelado. Soporta el ciclo de vida completo del desarrollo de una aplicación informática, desde la fase de análisis hasta el despliegue de la misma. Permite realizar todos los tipos de diagramas de clases, ingeniería inversa, generar código desde diagramas, entre muchas otras funcionalidades. Presenta una excelente documentación la herramienta UML CASE basada en abundantes tutoriales de UML, demostraciones interactivas y proyectos UML. *Visual Paradigm* ofrece un diseño enfocado al negocio, permitiendo así generar un *software* de mayor calidad. Presenta una licencia *Open Source* a diferencia de Rational Rose Enterprise Edition.

Selección de la herramienta para el modelado a utilizar

Después de un análisis de las características de las herramientas anteriores se identificaron factores que potenciaron la elección de *Visual Paradigm* como herramienta para el modelado, los elementos determinantes fueron:

- Cumple con las políticas de migración a *software* libre en nuestro país.
- Contiene una interfaz gráfica muy intuitiva y de fácil aprendizaje.
- Soporta múltiples plataformas, tales como *Windows*, Linux, Mac OS y *Java Desktop*.

1.5.3 Marco de desarrollo

Grails

Es un *framework* de desarrollo *Open Source* para aplicaciones web, dentro de la plataforma Java, que utiliza como lenguaje de programación Groovy. El objetivo de Grails es brindar al usuario un entorno de alta productividad, extensible y fácil de utilizar, ofreciendo el balance adecuado entre consistencia y funcionalidades potentes. Nace como la respuesta de Java al cambio de paradigma impuesto por *frameworks* como *Ruby on Rails* y Django (23).

Para el desarrollo del Subsistema de Gestión de Riesgos para el Sistema SAMOS se decidió utilizar Grails debido, entre otras razones, a que posibilita al programador un gasto mínimo de tiempo necesario en tareas repetitivas, donde un cambio de requisitos no obliga a reescribir toda la aplicación, porque la mayoría del código de infraestructura se genera de forma dinámica mientras la aplicación se está ejecutando, logrando así un proceso de desarrollo ágil y productivo. Además, por política del Centro Soporte, es el *framework* de desarrollo que se utiliza para la implementación del sistema SAMOS.

1.5.4 Lenguaje de programación

Groovy

Groovy es un lenguaje de programación con una sintaxis análoga a Java que compila para *Java Bytecode*[5] y corre en la JVM (Máquina Virtual de Java, significado de sus siglas en español). Ha sido influenciado por lenguajes como *Ruby*, Python, Perl, y *Smalltalk*, así como también Java. A diferencia de otros lenguajes que se adaptaron

[5] Archivo binario que contiene un programa ejecutable similar a un módulo objeto.

para la JVM, fue diseñado para la misma, así que no existe incompatibilidad. Aporta una sintaxis que aumenta enormemente la productividad y un entorno de ejecución que nos permite manejar los objetos de forma que en Java serían extremadamente complicados (24).

La primera versión salió en el año 2007 y actualmente está en la versión 2.1. En esta nueva versión, Groovy ofrece soporte completo para el JDK[6] 7 "invocación dinámica" instrucciones de código de bytes y API[7], va más allá de la comprobación estática de tipo convencional con una capacidad de anotación especial para ayudar con la documentación y la seguridad de tipos de DSL[8]. Agrega extensiones estáticas de tipo ortográfico y ofrece opciones adicionales de personalización de compilación (24). Es el lenguaje de programación a utilizar por las características antes expuestas, además de ser el que utiliza el *framework* de desarrollo escogido, Grails.

1.5.5 Entorno de Desarrollo Integrado (IDE)

Intellij IDEA

Es un entorno inteligente para desarrollar aplicaciones *Java*, cliente y servidor, también permite desarrollar aplicaciones para móviles (J2ME). Posee un avanzado editor de código, compatible con multitud de tecnologías (AJAX, JSP, EJB) y, dentro de un mismo entorno, ofrece análisis del código, compilación/ejecución/*debugging*[9], control de versiones, detección de duplicaciones, análisis de dependencias y soporte para *plugins*, además que soporta los siguientes formatos**:** *Java*, JavaScript/ Flex, HTML[10]/XHTML/CSS[11], XML/XSL, Ruby/JRuby, Groovy (24).

Se decidió utilizar este IDE debido a que es el más optimizado para programar en Grails, además que por políticas del Centro de Soporte es el entorno de desarrollo que se utiliza para la implementación del sistema SAMOS.

---

[6] *Software* que provee herramientas de desarrollo para la creación de programas en Java.
[7] Interfaz de programación de aplicaciones o API (del inglés application programming interface) es el conjunto de funciones y procedimientos (o métodos, en la programación orientada a objetos) que ofrece cierta biblioteca para ser utilizado por otro *software* como una capa de abstracción. Usados generalmente en las bibliotecas.

[8] Tipo de lenguaje de programación o lenguaje de especificación en el desarrollo de *software* y la ingeniería de dominio dedicado a un dominio particular problema, una técnica de representación problema particular, y / o una técnica de solución particular.
[9] Depuración. Corrección de errores en programación.
[10] Lenguaje de marcado hipertextual para la elaboración de páginas web.
[11] Hojas de estilo en Cascada

1.5.6 Sistema Gestor de Base de Datos(SGBD)

PostgreSQL

Es un sistema de gestión de bases de datos objeto-relacional, distribuido bajo licencia BSD y con su código fuente disponible libremente. Utiliza un modelo cliente/servidor y usa multiprocesos en vez de multihilos para garantizar la estabilidad del sistema. Un fallo en uno de los procesos no afectará el resto y el sistema continuará funcionando. Está disponible para múltiples sistemas operativos tales como Windows, Linux y Mac OS (25).

Actualmente está ampliamente considerado como el SGBD de código abierto más avanzado, esto está enmarcado por poseer muchas características que tradicionalmente sólo se podían ver en productos comerciales de alto calibre. Por las características antes mencionadas, PostgreSQL es el SGBD a utilizar, además que por política del Centro Soporte, es el SGBD que se utiliza para el desarrollo del sistema SAMOS.

1.5.7 Servidor web

Apache Tomcat

Es un servidor web que se desarrolla en un entorno abierto y participativo publicado bajo la licencia Apache versión 2. Está integrado en la implementación de referencia *Java 2 Enterprise Edition* (J2EE)[12] de *Sun Microsystems* y se encuentra en su versión 7.0.39. Actualmente es el servidor Web más utilizado a la hora de trabajar con Java en entornos web; es una implementación completamente funcional de los estándares de JSP y *Servlets*. También puede especificarse como el manejador de las peticiones de JSP y *Servlets* recibidas por servidores web populares, como el servidor Apache HTTP de la Fundación de *software* de Apache o el servidor *Microsoft Internet Information Server* (IIS). Por las características antes mencionadas se decide utilizar Apache Tomcat como servidor web.

1.5.8 Componentes de diseño

Bootstrap

[12] Plataforma de programación—parte de la Plataforma Java—para desarrollar y ejecutar *software* de aplicaciones en el lenguaje de programación Java.

Es un conjunto de herramientas de interfaz que nos permiten desarrollar rápidamente aplicaciones en una web. A través del uso de técnicas compatibles con los nuevos navegadores, pone a nuestra disposición, por ejemplo, formularios, botones, tablas, cuadrículas, menús y muchas otras herramientas para acelerar nuestro trabajo.

Bootstrap brinda una base pre-codificada de HTML y CSS para armar el diseño de una página o una aplicación web, y al ofrecerse como un recurso de código abierto es fácil de personalizar y adaptar a múltiples propósitos. Fue desarrollado por Mark Otto y Jacbod Thornton de Twitter, como un *framework* para fomentar la consistencia a través de herramientas internas (26).

Kendo UI

Es un *framework* para realizar potentes interfaces compatibles con los mejores navegadores web, sean de escritorio o móviles. Es compatible con sistemas operativos móviles y sus versiones posteriores tales como: Android 2.0, iOS 3.0, BlackBerry OS 6.0, webOS 2.2, además de los siguientes navegadores *Internet Explorer* 7, Firefox 3, Safari 4, Chrome y Opera 10. Está bajo la licencia GPL (27).

Highcharts

Es una biblioteca de gráficos escritos en JavaScript puro, que ofrece una forma fácil de añadir gráficos interactivos a un sitio o aplicación web. Actualmente apoya la línea, spline, área, areaspline, columnas, barras, circulares, de dispersión, patrones angulares, arearange, areasplinerange, columnrange, burbuja, diagrama de caja, barras de error, embudo, cascada y tipos de gráficos polares.

Funciona en todos los navegadores modernos, como el iPhone / iPad, Internet Explorer desde la versión 6 y Mozilla Firefox 3. Una de las características clave de Highcharts es que en cualquiera de las licencias libres o no, se le permite descargar el código fuente y hacer tus propias ediciones, permitiendo así modificaciones personales y una gran flexibilidad (28).

1.6 Conclusiones parciales

En el presente capítulo se efectuó un análisis de los conceptos fundamentales asociados a la TI y a la GR. A partir de este, el autor llegó a la conclusión de que la metodología para la mejora de los servicios de TI que más se adecua para

fundamentar la GR que realizará el subsistema que se propone a implementar será COBIT, en su proceso 9 “Evaluar y Administrar los Riesgos de TI”, debido a que las actividades que el mismo propone cumplen con las necesidades proporcionadas por el cliente.

Además a partir del análisis de diferentes sistemas para la GR de servicios de TI existentes en el mundo se determinó que no se puede utilizar en el Centro de Soporte debido a que la mayoría son privativas y en su totalidad no se ajustan a las necesidades del Centro de Soporte UCI.

Finalmente las tecnologías, herramientas y metodologías para el desarrollo de la aplicación fueron: XP como metodología de desarrollo de *software*, *Visual Paradigm* como herramienta para el modelado, Groovy como lenguaje de programación, Grails como *framework* de desarrollo, Intellij IDEA como entorno de desarrollo, PostgreSQL como SGBD y Apache Tomcat como servidor web.

CAPÍTULO II: PROPUESTA DE SOLUCIÓN

2.1 Introducción

En el presente capítulo se establece una descripción detallada de la solución propuesta utilizando XP como metodología de desarrollo. Se describen las historias de usuario (HU), las cuales contienen los requisitos funcionales del sistema definidos por el cliente y es el primer artefacto generado por la metodología a utilizar. Luego de describirse las HU se realiza la estimación del esfuerzo para cada una, el plan de duración de cada iteración y se define el plan de entregas con las propuestas de liberación de las versiones de la aplicación.

Para el diseño del sistema se describe la arquitectura del módulo, los patrones de diseño que se utilizaron y la descripción en tarjetas Clase-Responsabilidad-Colaboración (CRC).

2.2 Actores de la solución propuesta

Un actor es alguien o algo, externo al sistema que de cierta forma interactúa con este. Los actores no son solamente roles que juegan personas, sino también organizaciones, *software* y máquinas. A continuación se verán reflejados los actores que interactúan con el sistema en desarrollo:

- Administrador del sistema: Es la persona que posee todos los permisos administrativos del sistema. Se encarga de arreglar posibles fallos y gestionar que funcionen bien todas las conexiones con las bases de datos. Pueden acceder a la aplicación y verificar la autenticidad de varios documentos.

- Gestor de riesgos: Posee los permisos para gestionar riesgos y actividades de respuesta y para generar y descargar reportes y enviar notificaciones a los usuarios.

- Usuario estándar: Es la persona que interactúa con el sistema pero con privilegios básicos de navegación.

2.3 Requisitos de la solución propuesta

Los requisitos para un sistema de *software* determinan lo que hará el sistema y definen las restricciones de su operación e implementación. Los mismos se clasifican

en funcionales y no funcionales. Para la obtención de los mismos se realizaron varios encuentros con el cliente para aclarar dudas que de manera continua pudieran aparecer, en total hubo 5 modificaciones de los mismos.

2.3.1 Requisitos Funcionales

Los requisitos funcionales son capacidades o condiciones que el sistema debe cumplir. A continuación se reflejan los requerimientos del sistema propuesto. En el epígrafe 2.4 se describen detalladamente, mediante Historias de Usuario como lo establece la metodología a utilizar.

RF1. El sistema debe permitir insertar Categorías de Riesgos.

RF2. El sistema debe permitir modificar Categorías de Riesgos.

RF3.El sistema debe permitir eliminar Categorías de Riesgos.

RF4. El sistema debe permitir listar Categorías de Riesgos.

RF5.El sistema debe permitir mostrar una Categoría de Riesgo.

RF6. El sistema debe permitir insertar Sub-Categorías de Riesgos a una Categoría.

RF7. El sistema debe permitir modificar Sub-Categorías de Riesgos de una Categoría.

RF8. El sistema debe permitir eliminar Sub-Categorías de Riesgos de una Categoría.

RF9. El sistema debe permitir listar Sub-Categorías de Riesgos de una Categoría.

RF10. El sistema debe permitir mostrar una Sub-Categorías de Riesgo de una Categoría.

RF11. El sistema debe permitir insertar Riesgos.

RF12. El sistema debe permitir modificar Riesgos.

RF13. El sistema debe permitir eliminar Riesgos.

RF14. El sistema debe permitir listar Riesgos.

RF15. El sistema debe permitir mostrar un Riesgo.

RF16. El sistema debe permitir insertar Actividades de Respuesta a un Riesgo.

RF17. El sistema debe permitir modificar Actividades de Respuesta de un Riesgo.

RF18. El sistema debe permitir eliminar Actividades de Respuesta de un Riesgo.

RF19. El sistema debe permitir listar Actividades de Respuesta de un Riesgos.

RF20. El sistema debe permitir mostrar una Actividad de Respuesta de un Riesgo.

RF21. El sistema debe permitir asociar Actividades de Respuesta a un Riesgo pertenecientes a otros a Riesgos.

RF22. El sistema debe permitir desasociar Actividades de Respuesta de un Riesgo.

RF23. El sistema debe permitir listar Procesos Afectados por un Riesgo.

RF24.El sistema debe permitir mostrar un Proceso Afectado por un Riesgo.

RF25.El sistema debe permitir asociar Procesos a un Riesgo (Proceso Afectado).

RF26. El sistema debe permitir desasociar Procesos Afectados de un Riesgo.

RF27.El sistema debe permitir filtrar Categorías de Riesgos.

RF28. El sistema debe permitir filtrar Sub-Categorías de Riesgos.

RF29. El sistema debe permitir filtrar Riesgos.

RF30. El sistema debe permitir filtrar Actividades de Respuestas.

RF31. El sistema debe permitir filtrar Procesos Afectados.

RF32. El sistema debe permitir exportar listado de Categorías de Riesgos en diversos formatos.

RF33. El sistema debe permitir exportar listado de Sub-Categorías de Riesgos en diversos formatos.

RF34. El sistema debe permitir exportar listado de Riesgos en diversos formatos.

RF35. El sistema debe permitir exportar listado de Actividades de Respuesta en diversos formatos.

RF36. El sistema debe permitir exportar listado de Procesos Afectados en diversos formatos.

RF37. El sistema debe permitir enviar aviso de notificación por correo electrónico.

RF38. El sistema debe permitir visualizar gráficos estadísticos de distribución de Riesgos por Categoría y nivel de criticidad.

RF39. El sistema debe permitir visualizar Matriz de Riesgos.

2.3.2 Requisitos No Funcionales

Los requisitos no funcionales (RNF) son propiedades o cualidades que el producto debe tener. Los mismos no se capturan en la metodología XP mediante historias de usuario, puesto que los clientes generalmente no conocen la terminología técnica que se utiliza para describirlos. A continuación se muestran los RNF definidos para el desarrollo del sistema:

Restricciones de Interfaz o apariencia externa:

RNF1: El sistema deberá presentar un menú lateral que permita el acceso rápido y cómodo a la información, por parte de los usuarios.

Restricciones en el diseño y la implementación:

RNF1: Se utilizará como sistema gestor de base datos a PostgresSQL en su versión 9.1.

RNF2: Se utilizará como lenguaje de programación a Groovy.

RNF3: Se utilizará Grails como *framework* de desarrollo en su versión 2.1.

RNF4: Se utilizará Intellij IDEA en su versión 11.1.4 como entorno de desarrollo.

RNF5: Se utilizará como herramienta de modelado *Visual Paradigm* en su versión 8.0.

Funcionamiento:

- ***Software*:**

RNF1: Para el funcionamiento en la PC cliente será necesario un navegador que cumpla con los estándares definidos por organizaciones internacionales, tales como: Mozilla Firefox, Google Chrome, Opera, Safari.

Seguridad:

- **Confidencialidad:**

  RNF1: El acceso al sistema así como la información que contiene, se encontrarán protegidos contra accesos no autorizados utilizando mecanismos de autenticación propios del sistema.

  RNF2: La autenticación del sistema será la primera acción del usuario, el cual deberá proporcionar un usuario único y una contraseña, los cuales serán de uso exclusivo del propio usuario.

  RNF3: El administrador del sistema como política de seguridad podrá restringir el acceso a las diferentes áreas del sistema a los usuarios.

- **Integridad:**

  RNF1: La información manejada por el sistema solo podrá ser modificada por el personal autorizado.

Fiabilidad:

RNF1: El *software* debe ser capaz de registrar los errores que ocurran durante su ejecución.

RNF2: Ninguna información ingresada en el sistema será eliminada físicamente de la base de datos, independientemente de su inexistencia para el sistema. Permitirá la recuperación de la información de la base de datos, a partir de respaldos, o salvas realizadas.

RNF3: Se realizarán salvas periódicamente de la base de datos para prevenir la pérdida de información.

2.4 Exploración

El ciclo de vida de un proyecto según la metodología XP se inicia con la fase de exploración. En esta fase es donde los clientes plantean a grandes rasgos las historias

de usuario (HU) que son de interés para la entrega del producto. Al mismo tiempo, el equipo de desarrollo se familiariza con las herramientas, tecnologías y prácticas que se utilizarán en el proyecto. Se prueba la tecnología y se exploran las posibilidades de la arquitectura del sistema construyendo un prototipo.

2.4.1 Historias de Usuario

Uno de los artefactos más significativos que genera la metodología a utilizar son las HU, las cuales son escritas por el propio cliente tal y como ve él las necesidades del sistema, aunque el programador puede brindar su ayuda en la identificación de las mismas. Las descripciones son sencillas y cortas para que el programador las implemente en poco tiempo. El tratamiento de las mismas es muy flexible y dinámico debido a que en cualquier momento pueden ser modificadas o reemplazadas por otras más específicas, en total hubo 5 modificaciones de las mismas.

A continuación como resultado de la fase de exploración aparecen descritas 5 HU, como ejemplo, las cuales representan el flujo de actividades que permite la Gestión de Riesgos; en total se definieron 39 (24 con prioridad Alta, 9 Media y 6 Baja). Las restantes pueden ser consultadas en el documento “Historias de Usuario” en la carpeta de anexos:

Tabla 1. HU: Insertar Riesgos

Historia de Usuario	
Número: 11	**Nombre:** Insertar Riesgos
Rol: Gestor de riesgos y administrador del sistema.	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Medio
Descripción: El sistema debe permitir al usuario insertar riesgos, siempre especificando los datos requeridos para la aplicación: ▪ *Nombre ▪ Descripción ▪ *Categoría ▪ Subcategoría ▪ *Impacto ▪ *Probabilidad ▪ *Estado	

▪ Fecha de detección ▪ Fecha de resolución ▪ Causas ▪ Consecuencias *Campos obligatorios.
Información adicional: Si los datos entrados son incorrectos, están en blanco o no coinciden con lo esperado, la aplicación debe mostrar un mensaje correspondiente a cada error.

Tabla 2. HU: Modificar Riesgo

Historia de Usuario	
Número: 12	**Nombre:** Modificar Riesgo
Rol: Gestor de riesgos y administrador del sistema.	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Medio
Descripción: El sistema debe permitir al usuario dado un listado de riesgos existentes, modificar los datos de un riesgo especificado, teniendo en cuenta los campos obligatorios.	
Información adicional: Si los datos entrados son incorrectos, están en blanco o no coinciden con lo esperado, la aplicación debe mostrar un mensaje correspondiente a cada error.	

Tabla 3. HU: Eliminar Riesgo

Historia de Usuario	
Número: 13	**Nombre:** Eliminar Riesgo
Rol: Gestor de riesgos y administrador del sistema.	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Medio
Descripción: El sistema debe permitir al usuario dado un listado de riesgos existentes, eliminar un riesgo especificado.	

Información adicional:

Tabla 4. HU: Listar Riesgos

Historia de Usuario	
Número: 14	**Nombre:** Listar Riesgos
Rol: Gestor de riesgos, administrador del sistema y usuario estándar.	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Medio
Descripción: El sistema debe permitir al usuario ver todos los riesgos insertados en el sistema, mostrando solamente de cada uno de ellos: nombre, nivel de criticidad, procesos afectados y actividades de respuesta.	
Información adicional:	

Tabla 5. Mostrar un Riesgo

Historia de Usuario	
Número: 15	**Nombre:** Mostrar un Riesgo
Rol: Gestor de riesgos, administrador del sistema y usuario estándar.	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Medio
Descripción: El sistema debe permitir al usuario dado un listado de riesgos existentes ver todos los datos de un riesgo seleccionado. Los cuales serían: ▪ Nombre ▪ Descripción ▪ Categoría ▪ Sub-categoría ▪ Impacto ▪ Probabilidad ▪ Nivel de Criticidad ▪ Estado ▪ Fecha de detección ▪ Fecha de resolución ▪ Fecha de creación	

- Fecha de modificación
- Fecha de cierre
- Causas
- Consecuencias
- Añadido por
- Procesos Afectados
- Actividades de Respuesta

Información adicional: Las fechas de creación, modificación y de cierre mostradas no fueron insertadas por el usuario debido a que automáticamente se toma como valor la fecha en la que se realiza la acción, tanto de creación como de modificación o de cierre. Caso parecido ocurre con el autor ("Añadido por"), que es el usuario autenticado en el sistema, así como el nivel de criticidad, que es el promedio entre la probabilidad y el impacto.

2.5 Planificación

Luego de concluida la fase de Exploración, se inicia la fase de Planificación del proyecto, donde el programador realiza una estimación del esfuerzo necesario para cada una de las HU definidas por el cliente. Se toman acuerdos sobre el contenido de la primera entrega y se determina un cronograma en conjunto con el cliente.

2.5.1 Estimación de esfuerzo por HU

A continuación se muestra en la tabla 6 la estimación de las HU por parte del desarrollador, para satisfacer las necesidades del cliente. La estimación incluye todo el esfuerzo asociado a la implementación de cada una de las HU.

Tabla 6. Estimación de esfuerzo por historias de usuario

Número	HU	Estimación(días)
1	Insertar Categoría de Riesgos	2
2	Modificar Categoría de Riesgos	2
3	Eliminar Categoría de Riesgos	2
4	Listar Categorías de Riesgos	2

5	Mostrar una Categoría de Riesgo	2
6	Insertar Sub-Categorías de Riesgo a una Categoría	2.5
7	Modificar Sub-Categorías de Riesgos de una Categoría	2.5
8	Eliminar Sub-Categorías de Riesgos de una Categoría	2.5
9	Listar Sub-Categorías de Riesgos de una Categoría	2.5
10	Mostrar una Sub-Categoría de Riesgos de una Categoría	2.5
11	Insertar Riesgos	2
12	Modificar Riesgo	2
13	Eliminar Riesgo	2
14	Listar Riesgos	2
15	Mostrar un Riesgo	2
16	Insertar Actividades de Respuesta a un Riesgo	2.5
17	Modificar Actividades de Respuesta de un Riesgo	2.5
18	Eliminar Actividades de Respuesta de un Riesgo	2.5
19	Listar las Actividades de Respuesta de un Riesgo	2.5
20	Mostrar datos de una Actividad de Respuesta	2.5
21	Asociar Actividades de Respuesta a un Riesgo	2.5
22	Desasociar Actividades de Respuesta a un Riesgo	2.5
23	Asociar Procesos a un Riesgo(Proceso Afectado)	2.5
24	Desasociar Actividades de Respuesta a un Riesgo	2.5
25	Listar Procesos Afectados	2.5
26	Mostrar un Proceso Afectado por un Riesgo	2.5

27	Filtrar Categorías de Riesgos	3
28	Filtrar Sub-Categorías de Riesgos	3
29	Filtrar Riesgos	3
30	Filtrar Actividades de Respuesta	3
31	Filtrar Procesos Afectados	3
32	Exportar listado de Categorías de Riesgos en diferentes formatos	1
33	Exportar listado de Sub-Categorías de Riesgos en diferentes formatos	1
34	Exportar listado de Riesgos en diferentes formatos	1
35	Exportar listado de Actividades de Respuesta en diferentes formatos	1
36	Exportar listado de Procesos Afectados en diferentes formatos	1
37	Enviar notificaciones por correo electrónico	3
38	Visualizar gráficos estadísticos de distribución de riesgos por Categoría y nivel de criticidad	3.5
39	Visualizar matriz de Riesgos	4.5

Luego de la culminación de este proceso para el presente trabajo de diploma, se determinó un total aproximado de 91 días.

2.5.2 Plan de duración de iteraciones

Después de haber sido estimado el esfuerzo dedicado a la realización de cada una de las HU identificadas, se procede a la creación del plan de iteraciones el cual se expone en la tabla 7 y tiene como objetivo mostrar la duración y el orden en que serán implementadas las mismas.

Tabla 7. Plan de duración de Iteraciones

Iteración	Descripción de la iteración	HU	Estimación(días)
1	En la presente iteración serán implementadas las HU(1-26)	Insertar Categoría de Riesgos Modificar Categoría de Riesgos Eliminar Categoría de Riesgos Listar Categorías de Riesgos Mostrar una Categoría de Riesgo Insertar Sub-Categorías de Riesgo a una Categoría Modificar Sub-Categorías de Riesgos de una Categoría Eliminar Sub-Categorías de Riesgos de una Categoría Listar Sub-Categorías de Riesgos de una Categoría Mostrar una Sub-Categoría de Riesgos de una Categoría Insertar Riesgos Modificar Riesgo Eliminar Riesgo Listar Riesgos Mostrar un Riesgo Insertar Actividades de Respuesta a un Riesgo Modificar Actividades de Respuesta de un Riesgo	60

		Eliminar Actividades de Respuesta de un Riesgo Listar las Actividades de Respuesta de un Riesgo Mostrar datos de una Actividad de Respuesta Asociar Actividades de Respuesta a un Riesgo Desasociar Actividades de Respuesta a un Riesgo Asociar Procesos a un Riesgo(Proceso Afectado) Desasociar Actividades de Respuesta a un Riesgo Listar Procesos Afectados Mostrar un Proceso Afectado por un Riesgo	
2	En la presente iteración serán implementadas las HU restantes.	Filtrar Categorías de Riesgos Filtrar Sub-Categorías de Riesgos Filtrar Riesgos Filtrar Actividades de Respuesta Filtrar Procesos Afectados Exportar listado de Categorías de Riesgos en diferentes formatos Exportar listado de Sub-Categorías de Riesgos en diferentes formatos Exportar listado de Riesgos en	31

		diferentes formatos Exportar listado de Actividades de Respuesta en diferentes formatos Exportar listado de Procesos Afectados en diferentes formatos Enviar notificaciones por correo electrónico Visualizar gráficos estadísticos de distribución de riesgos por Categoría y nivel de criticidad Visualizar matriz de Riesgos	

2.5.3 Plan de entregas

El plan de entrega es el compromiso del equipo de desarrollo con el cliente, pues en él se define cuando será entregado el producto (29). A continuación se muestra en la tabla 8 el plan de entrega para cada iteración:

Tabla 8. Plan de entrega

Producto	**Final de la 1ra iteración (**Semana del 18 al 22 de marzo**)**	**Final de la 2da iteración(**Semana del 6 al 10 de mayo**)**
Subsistema para la gestión de riesgos	Versión 0.5	Versión 1.0

Como se muestra en la tabla 8, se realizarán 2 entregas del sistema, cada una de ellas implementará las funcionalidades especificadas por la iteración a la que fue asignada, comenzando la implementación el 26 de diciembre del 2012 y se hace la primera entrega en la semana del 18 al 22 de marzo del presente año. La segunda entrega perteneciente a la última iteración finalizará en la semana del 6 al 10 de mayo.

2.6 Diseño de la solución propuesta

Una vez planificada la realización del sistema se procede al diseño del mismo, la metodología seleccionada no obliga a la realización de diagramas UML para

representar las clases que contendrán la lógica del negocio, en su lugar propone el uso de tarjetas CRC como técnica de modelado para ayudar a los desarrolladores de *software* a crear diseños de clases orientados a responsabilidades.

2.6.1 Descripción de la arquitectura de la solución propuesta

Patrón de arquitectura:

Para la arquitectura de la aplicación se utilizó el patrón Modelo-Vista-Controlador (MVC), debido a que el *framework* Grails está basado en el mismo. El MVC separa las interfaces de usuario, los datos y la lógica de control en 3 componentes fundamentales, a continuación se describen los mismos.

Modelo: Contiene todos los elementos que representan los datos manejados por la aplicación. En la figura 7 se muestra la clase “CategoriaRiesgo” como ejemplo de modelos existentes en el sistema.

CategoriaRiesgo.groovy | CategoriaRiesgo | CategoriaRiesgoController | CategoriaRiesgo Views | Tests

```
1   package riskmanagement
2
3   import cmdb.Usuario
4
5   /**...*/
11  class CategoriaRiesgo {
12
13      static hasMany = [riesgos: Riesgo, subCategorias: SubCategoriaRiesgo]
14      static mappedBy = [riesgos: "categoria"]
15
16      static mapping = {
17          tablePerHierarchy false
18      }
19
20      String nombre
21      String descripcion
22
23      Date fechaCreacion
24      Date fechaModificacion
25      Usuario autor
```

Figura 6. Ejemplo de un componente modelo

Vista: Agrupa todos los componentes responsables de mostrar al usuario el estado actual del modelo de datos, y presentarles las distintas funcionalidades disponibles. En la figura 8 se muestra como ejemplo el paquete que conforma la vista correspondiente a la clase “CategoriaRiesgo”.

Figura 7. Ejemplo de un componente vista

Controlador: Contiene todos los elementos que reciben las peticiones de los usuarios, gestionan la aplicación de la lógica del negocio sobre el modelo de datos, y determinan que vista debe mostrarse. La figura 9 muestra la estructura de la clase "CategoriaRiesgoController" como ejemplo de los controladores existentes en el sistema.

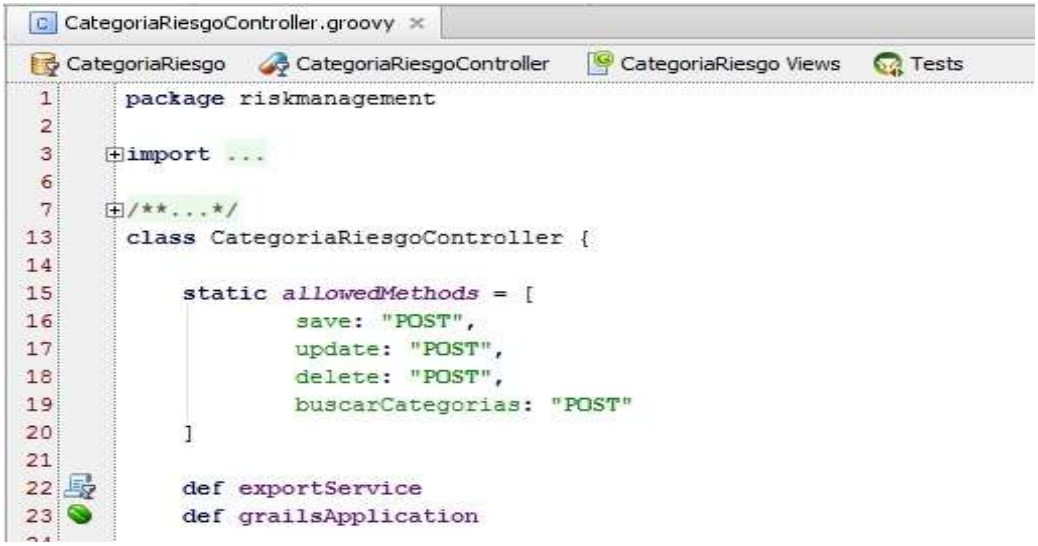


Figura 8. Ejemplo de un componente controlador

En la figura 10 se muestra como se aplica lo anteriormente planteado en la propuesta de solución:

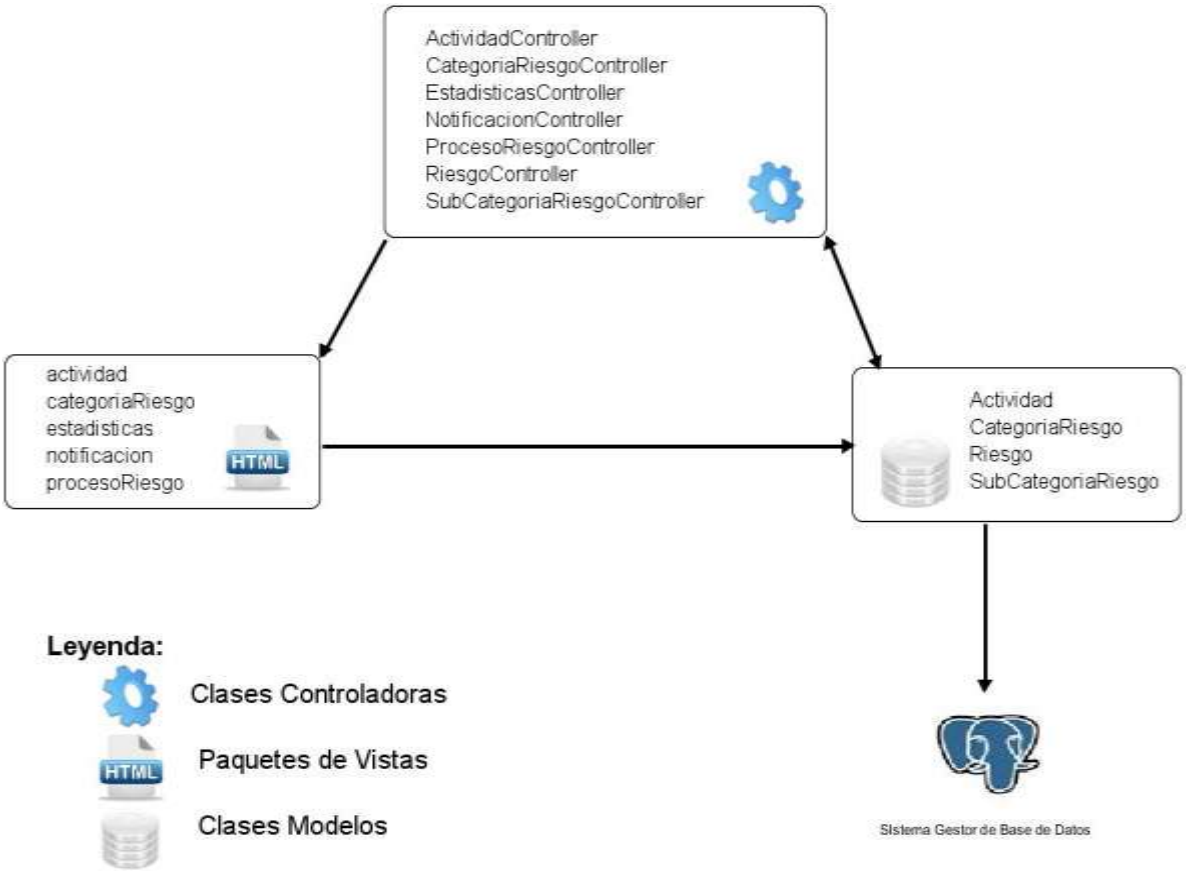


Figura 9. Descripción de la arquitectura

Estructura de la solución propuesta:

El *framework* Grails establece en su estructura clásica del patrón arquitectónico MVC un sistema de paquetes con la siguiente distribución, lo cual es mostrado en la figura 8.

- **grails-app:** Directorio que contiene los archivos fuentes del proyecto de Grails.
  - **conf:** Paquete que contiene los archivos de configuración de una aplicación Grails.
  - **controllers:** Paquete que contiene los controladores de la aplicación Grails, este sería el equivalente a los *servlets*[13]. En los controladores es donde se definen acciones, se re-direcciona o se personaliza el *scaffolding*[14] de la aplicación.

[13] Objetos que corren dentro y fuera del contexto de un contenedor de *servlets* (ejemplo: Tomcat) y extienden su funcionalidad.
[14] Técnica que contiene algunos *frameworks* basados en el patrón arquitectónico MVC, genera controladores y vistas.

- **domain:** Paquete que contiene las clases del dominio, que no son más que las clases persistentes de la aplicación que se convierten en tablas de la base de datos.

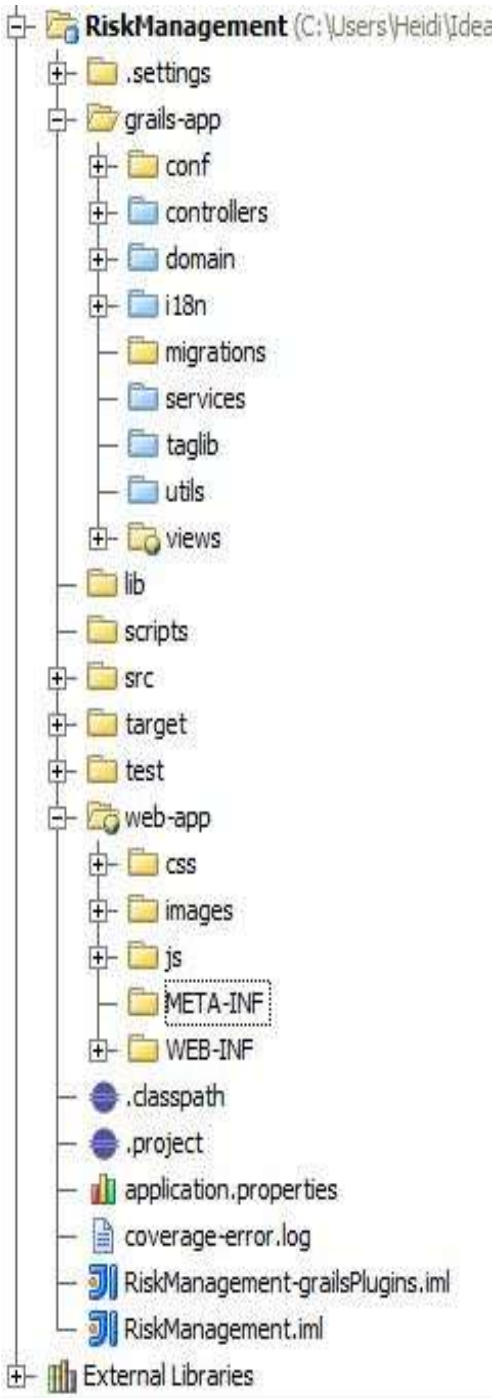


Figura 10. Estructura de la solución propuesta

- **18n:** Paquete que contiene los archivos de internacionalización de la aplicación. Todos los textos son cargados a través de mensajes que están contenidos en estos archivos, en dependencia del idioma establecido por el usuario, será el archivo que cargue la aplicación.

- **services:** Paquete que contiene los servicios facilitan el mantenimiento de la aplicación y forman parte de la lógica del negocio, están pensados para simplificar los controladores.

- **taglib:** Paquete que contiene las librerías tag, las cuales son como grupos de html tags[15]personalizadas que pueden ejecutar código. Grails trae varias por defecto, pero se pueden personalizar y crear nuevas.

- **utils:** Paquete donde se guardan las clases de utilidad para la aplicación, que pueden usarse en otros paquetes antes mencionados.

- **views:** Paquete que contiene todas las vistas correspondientes con las acciones definidas en los diferentes controladores organizadas en subdirectorios con el mismo nombre.

[15] Etiquetas HTML

- **lib:** En este paquete están ubicadas las librerías externas que son usadas por el proyecto.
- **scripts:** Paquete que contiene a los script para correr las pruebas o los creados por el usuario.
- **target:** En este paquete se generan los archivos .class de las clases groovy y java usadas en la aplicación Grails.
- **test:** En este paquete se realizan las pruebas de integración para comprobar la dependencia entre clases. Además contiene las pruebas de unidad de cada clase generada en el proyecto.
- **web-app:** Directorio donde se aloja el sitio web.
    - **css:** Paquete que contiene los archivos css que se utilizan en el proyecto.
    - **images:** Paquete que contiene las imágenes que se utilizan en el proyecto.
    - **js:** Paquete que contiene los archivos js que se utilizan en el proyecto.

Descrita la arquitectura y la estructura de la solución propuesta, se representan en tarjetas CRC las clases del subsistema.

2.6.2 Descripción de las tarjetas CRC

Las tarjetas CRC son utilizadas para representar las responsabilidades de las clases y sus interacciones. A continuación se presentan 4 como ejemplo, en total se crearon 11 tarjetas, las restantes pueden ser consultadas en el documento “Tarjetas CRC” dentro de la carpeta de anexos.

Tabla 9. Tarjeta CRC: CategoriaRiesgo

	Tarjetas CRC
Nombre de la clase: CategoriaRiesgo	
Responsabilidades	**Colaboraciones**

Es la representación de la tabla CategoriaRiesgo de la base de datos. ▪ beforeInsert () ▪ beforeUpdate () ▪ mostrarNombreCorto () ▪ toString ()	▪ Riesgo ▪ SubCategoriaRiesgo ▪ Usuario

Tabla 10. Tarjeta CRC: SubCategoriaRiesgo

Tarjetas CRC	
Nombre de la clase: SubCategoriaRiesgo	
Responsabilidades	**Colaboraciones**
	▪ CategoriaRiesgo

Tabla 11. Tarjeta CRC: Riesgo

Tarjetas CRC	
Nombre de la clase: Riesgo	
Responsabilidades	**Colaboraciones**
Es la representación de la tabla Riesgo de la base de datos. ▪ beforeInsert () ▪ beforeUpdate () ▪ stringToFloat () ▪ floatToString () ▪ mostrarNombreCorto () ▪ toString ()	▪ CategoriaRiesgo ▪ SubCategoriaRiesgo ▪ Actividad ▪ Proceso ▪ Usuario

Tabla 12. Tarjeta CRC: Actividad

Tarjetas CRC	
Nombre de la clase: Actividad	
Responsabilidades	**Colaboraciones**

Es la representación de la tabla Actividad de la base de datos. ▪ beforeInsert () ▪ beforeUpdate () ▪ mostrarNombreCorto (cadena) ▪ toString()	▪ Riesgo ▪ Usuario

2.6.3 Patrones de diseño

Los patrones de diseño son un conjunto de prácticas de óptimo diseño que se utilizan para abordar problemas recurrentes en la programación orientada a objetos. Dentro de las ventajas que presenta se encuentra que soportan la reutilización de arquitecturas *software* y favorecen la reutilización de código.

Patrones GRASP:

Los patrones GRASP son patrones de diseño que describen los principios fundamentales de la asignación de responsabilidades a objetos. GRASP es un acrónimo que significa General Responsibility Asignment *Software* Patterns (patrones generales de *software* para asignar responsabilidades). El nombre se eligió para indicar la importancia de captar estos principios, si se persigue diseñar eficazmente el *software* orientado a objetos (30). A continuación se muestran los patrones GRASP que se reflejan en la solución.

Experto: Es el patrón más utilizado para la asignación de responsabilidades. Se encarga de asignar una responsabilidad al experto en información (la clase que cuenta con la información necesaria para cumplir la responsabilidad). En el sistema se ve evidenciado este patrón a través de las clases encargadas de generar gráficos estadísticos de los Riesgos y enviar notificaciones.

Creador: guía la asignación de responsabilidades relacionadas con la creación de objetos, tarea muy frecuente en los sistemas orientados a objetos. El propósito fundamental de este patrón es encontrar un creador que se deba conectar con el objeto producido en cualquier evento. De manera que todas las clases que representan los datos persistentes en la Base de Datos, conocidas como modelos, resultan ejemplos de este patrón.

Controlador: Consiste en asignar la responsabilidad de gestionar un mensaje de un evento del sistema a una clase que represente una de estas dos opciones:

- Representa el sistema global, dispositivo o subsistema (controlador de fachada).

- Representa un escenario de caso de uso en el que tiene lugar el evento del sistema (controlador de caso de uso o de sesión).

De manera que todas las clases controladoras del sistema resultan ejemplos de este patrón.

2.7 Diseño de la base de datos

Luego de realizarse el diseño del sistema se pasa a la modelación de la base de datos. La figura 9 muestra una representación del diagrama entidad-relación del Subsistema de Gestión de Riesgos del sistema SAMOS, donde se observan las entidades y sus relaciones persistentes en el modelo de datos.

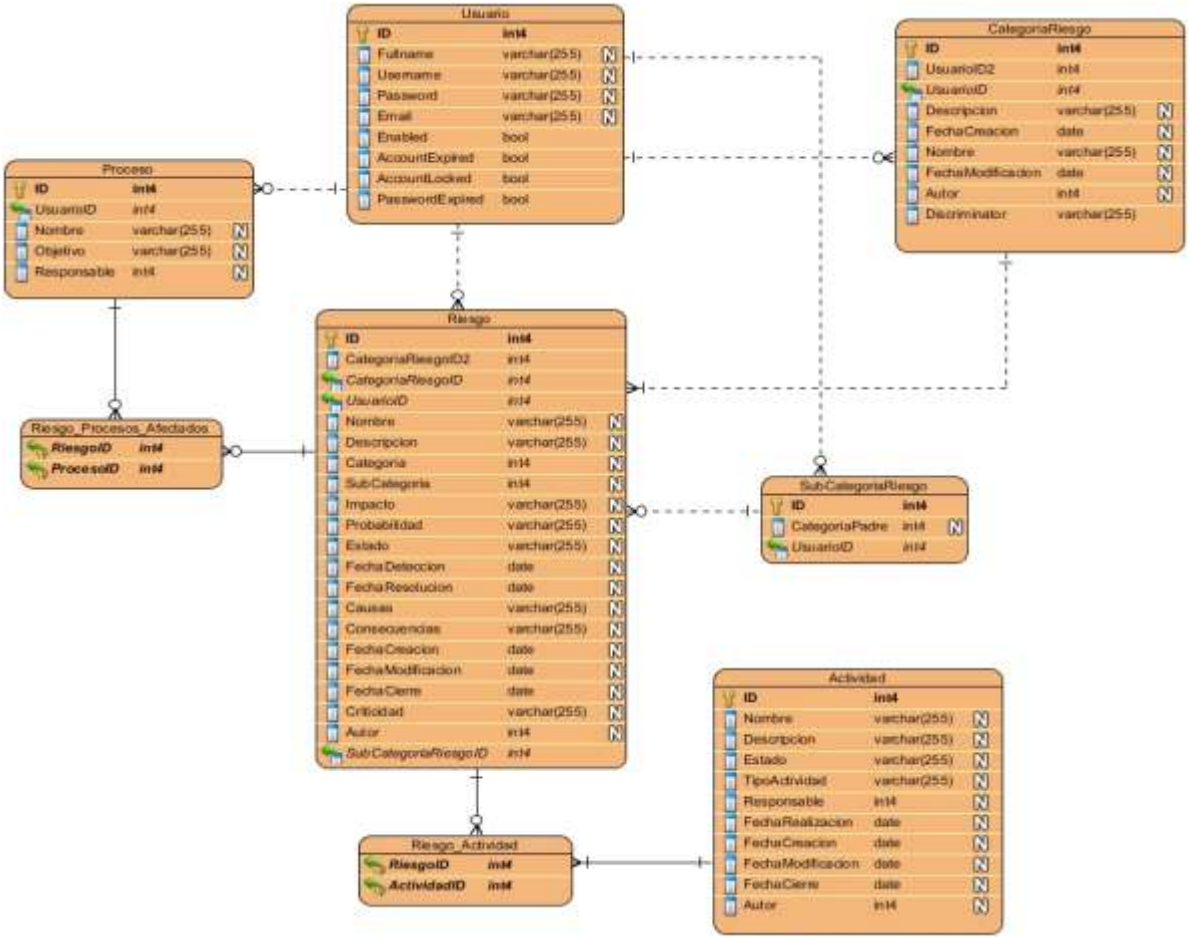


Figura 11. Diagrama Entidad-Relación

Como se muestra en la figura 9, la base de datos cuenta con 6 entidades persistentes, siendo la entidad “Riesgo” la de mayor importancia, debido a que prácticamente la mayoría de las restantes tienen una relación directa con ella. Estas relaciones vienen dadas a que el subsistema a desarrollar es para la GR de servicios TI. Un riesgo presenta una categoría y una sub-categoría, además puede afectar uno o varios procesos que se realizan en el Centro de Soporte, como por ejemplo “Gestión de entregas y despliegues” y un proceso puede verse afectado por varios riesgos, de ahí parte entonces la relación de muchos-a-muchos entre ambas entidades.

Para darle solución a los riesgos se realizan actividades de respuesta las cuales mitigan, aceptan, evitan o comparten a uno o varios riesgos de acuerdo a sus características, estando presente también la relación antes mencionada. El usuario podrá realizar acciones referentes a los riesgos, las categorías, las sub-categorías, las actividades de respuesta y los procesos afectados siempre y cuando tenga los permisos pertinentes, de acuerdo al rol que desempeñe en el sistema.

2.8 Implementación de la solución propuesta

Después de finalizada la fase de diseño, se procede a la implementación de las historias de usuario en su correspondiente iteración, obteniéndose en cada una de ellas una versión funcional del producto. Desde el inicio se realizan revisiones del plan de iteraciones, el cual se modifica y es mejorado en caso de ser necesario, esto se realiza de manera continua a lo largo del proyecto.

2.8.1 Tareas de Ingeniería

Cada una de las historias de usuario se dividirá en tareas a desarrollar, las cuales serán creadas para ayudar a organizar la implementación de las mismas. Estas tareas serán para el uso estricto de los programadores y pueden ser escritas en lenguaje técnico y no necesariamente entendible por el cliente. A continuación se presentan las tareas generales para el flujo de acciones que permiten la GR como ejemplo, en total se realizaron 126, las restantes pueden ser consultadas en el documento “Tareas de Ingeniería” dentro de la carpeta de anexos:

Tareas de Ingeniería generales para el flujo de acciones que permiten “Gestionar Riesgos”

Tabla 13. Tarea de Ingeniería: Diseño e implementación de la clase del dominio “Riesgo”

	Tarea de Ingeniería
Número Tarea: 37	**Número Historia de Usuario:** -
Nombre Tarea: Diseño e implementación de la clase del dominio “Riesgo”	
Tipo de Tarea : Desarrollo	**Programador Responsable:** Heydis Gutierrez Solano
Descripción: Definición y desarrollo de la clase del dominio “Riesgo”, así como de sus atributos, métodos y relaciones con el resto de las clases. Esta clase se utiliza como comunicación entre el sistema y los datos persistentes que existen en la tabla “riesgo” en la base de datos.	

Tabla 14. Tarea de Ingeniería: Diseño e implementación de la clase controladora “RiesgoController”

	Tarea de Ingeniería
Número Tarea: 38	**Número Historia de Usuario:** -
Nombre Tarea: Diseño e implementación de la clase controladora “RiesgoController”	
Tipo de Tarea : Desarrollo	**Programador Responsable:** Heydis Gutierrez Solano
Descripción: Definición y desarrollo de la clase controladora “RiesgoController”, así como de sus diferentes acciones, utilizando la técnica de *scaffolding.* Esta clase controla el flujo de acciones que permiten gestionar Riesgos.	

Tabla 15. Tarea de Ingeniería: Creación del paquete “riesgo”

	Tarea de Ingeniería
Número Tarea: 39	**Número Historia de Usuario:** -
Nombre Tarea: Creación del paquete “riesgo”	
Tipo de Tarea : Desarrollo	**Programador Responsable:** Heydis Gutierrez Solano
Descripción: Se crea el paquete “riesgo”, utilizando la técnica de *scaffolding*, que contendrá las vistas relacionadas con el flujo de acciones de los riesgos.	

2.9 Conclusiones parciales

Capítulo II: Propuesta de solución

En el presente capítulo se detallaron las características fundamentales de la propuesta de solución para comprender mejor las acciones a desarrollar por el subsistema de GR y lograr un mejor entendimiento del negocio en cuestión. Se obtuvieron 39 requisitos funcionales para el sistema modelado y se generaron los artefactos definidos por la metodología de desarrollo, además del diagrama Entidad-Relación, con un total de 6 entidades.

Estos artefactos, unidos al buen uso de los patrones de diseño: experto, creador y controlador permitieron sentar las bases para la implementación del *software*. Para una mejor organización de la implementación se hicieron uso de las tareas de ingeniería. Partiendo de todo lo planteado y analizado en este capítulo se afirma que se ha logrado una implementación del diseño. Por tanto se concluye que en este capítulo se implementó un subsistema de GR para el sistema SAMOS, en estado funcional.

CAPÍTULO III: VALIDACIÓN DE LA SOLUCIÓN PROPUESTA

3.1 Introducción

En el presente capítulo se abordarán las pruebas a la solución propuesta. Estas pruebas son realizadas con el objetivo de comprobar que todos los componentes cumplan con los requisitos funcionales propuestos.

3.2 Pruebas

Uno de los pasos más importantes de la metodología XP es el proceso de pruebas, el cuál anima al desarrollador a probar constantemente la aplicación tantas veces como sea necesario. Mediante esta filosofía se reduce el número de errores no detectados, contribuyendo todo esto a la elevación de la calidad del producto desarrollado.

La metodología XP divide las pruebas en dos grupos: pruebas unitarias, desarrolladas por los programadores, encargadas de verificar el código de forma automática y las pruebas de aceptación, destinadas a evaluar si al final de una iteración se obtuvo la funcionalidad requerida, además de comprobar que dicha funcionalidad sea la esperada por el cliente realizando las pruebas sobre la interfaz de la aplicación.

3.2.1 Pruebas Unitarias

Las pruebas unitarias son aquellas que permiten comprobar el funcionamiento de cada componente del sistema por separado, lo que aplicado al paradigma de la orientación a objetos es equivalente a hablar de pruebas que permiten comprobar el funcionamiento correcto de una clase de modo que, para cada uno de sus métodos, o al menos para cada método no trivial, se pueden ejecutar casos de prueba independientes (31).Para la realización de las mismas se utilizó el método Camino Básico y además la biblioteca JUnit, la cual está integrada en el framework de desarrollo.

Camino Básico

El método Camino Básico consiste en derivar casos de prueba a partir de un conjunto dado de caminos independientes por los cuales puede circular el flujo de control. Los casos de prueba derivados del conjunto básico garantizan que durante la prueba se ejecuta por lo menos una vez cada sentencia del programa.

Para obtener dicho conjunto de caminos independientes se construye el grafo de flujo asociado y se calcula su complejidad ciclomática.

El grafo de flujo está formado por 3 componentes: nodos, aristas y regiones. Cada nodo representa una o más secuencias procedimentales, las aristas representan el flujo de control y las regiones son las áreas delimitadas por las aristas y los nodos, incluyendo también el área exterior del grafo.

La complejidad ciclomática 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. Se calcula de 3 formas:

V (G) = R: donde R es el número de regiones del grafo de flujo.

V (G) = A - N + 2: donde A es la cantidad de aristas del grafo y N la cantidad de nodos.

V (G) = P + 1 donde P: es la cantidad de nodos predicados en el grafo.

A continuación se detallan como ejemplo, las pruebas unitarias realizadas a las funcionalidades “save”, “show” y “update” correspondientes a la tarjeta CRC “RiesgoController”, las restantes pueden ser consultadas en el documento “Pruebas Unitarias(Camino Básico)” dentro de la carpeta de anexos.

```
    @Secured("hasRole('Gestor de Riesgos')")
    def save() {
/* 1 */ def riesgoInstance = new Riesgo()

/* 2 */ if (sec?.username()) {
/* 3 */     riesgoInstance?.autor = Usuario.findByUsername(sec.username()?.toString())
        }
        else {
/* 4 */     flash.exception = message(code: 'default.not.created.no.user.logged.message', args: [message(cod
/* 4 */     redirect action: "list"
            return
        }

/* 5 */ if (params?.fechaDeteccion && params.fechaDeteccion != 'null') {
/* 6 */     params.fechaDeteccion = new Date(params.fechaDeteccion?.toString())
        }

/* 7 */ if (params?.fechaResolucion && params.fechaResolucion != 'null') {
/* 8 */     params.fechaResolucion = new Date(params.fechaResolucion?.toString())
        }

/* 9 */ riesgoInstance?.properties = params

/* 10 */if (!riesgoInstance.save(flush: true)) {
/* 11 */    render view: "create", model: [riesgoInstance: riesgoInstance]
            return
        }

/* 12 */flash.message = message(code: 'default.created.message', args: [message(code: 'riesgo.label', defaul
/* 12 */redirect action: "show", id: riesgoInstance?.id
    }
```

Figura 12. Código del método “save”

Usando el código de la figura anterior como base, se dibuja el correspondiente grafo de flujo:

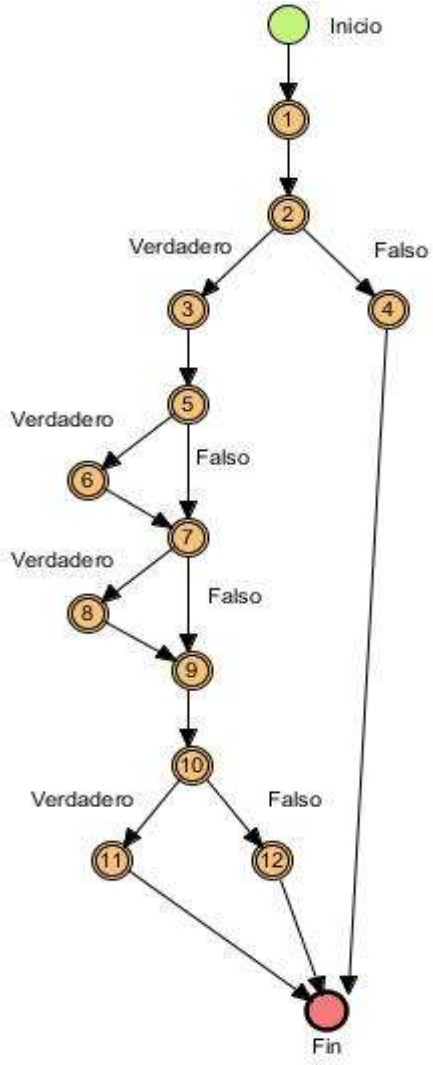


Figura 13. Grafo de flujo correspondiente al método “save”

Luego de realizado el grafo de flujo se determina la complejidad ciclomática [V (G)] del grafo de flujo resultante:

V (G) = # regiones = 5

V (G) = A – N + 2 = (17- 14) +2 = 5

V (G) = P + 1 = 4+1 = 5

Luego de determinada la complejidad ciclomática se tiene que 5 es el número máximo de caminos a tener en cuenta para realizar las pruebas.

Caminos independientes determinados:

- Camino 1: Inicio-1-2-3-5-7-8-9-10-12-fin
- Camino 2: Inicio-1-2-3-5-6-7-9-10-12-fin
- Camino 3: Inicio-1-2-3-5-6-7-8-9-10-12-fin
- Camino 4: Inicio-1-2-3-5-6-7-8-9-10-11-fin

- Camino 5: Inicio-1-2-4-fin

En la figura 12 se muestra el código del método “show”.

```
    def show(Long id) {
/* 1 */ def riesgoInstance

/* 2 */ if (id != null) {
/* 3 */     riesgoInstance = Riesgo.get(id)

/* 4 */     if (!riesgoInstance) {
/* 5 */         flash.exception = message(code: 'default.not.found.message', args: [message(code: 'riesgo.label', defaul
/* 5 */         redirect action: "list"
                return
            }
        }
        else {
/* 6 */     flash.exception = message(code: 'default.not.received.message', args: [message(code: 'riesgo.label', default
/* 6 */     redirect action: "list"
            return
        }

/* 7 */ [riesgoInstance: riesgoInstance]
    }
```

Figura 14. Código del método “show”

Usando el código de la figura 12 como base, se dibuja el correspondiente grafo de flujo:

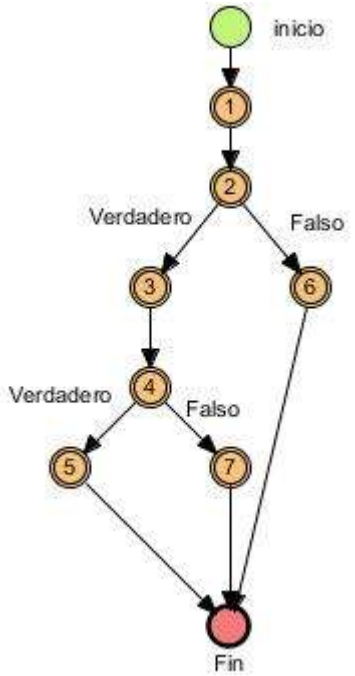


Figura 15. Grafo de flujo correspondiente al método "show"

Luego de realizado el grafo de flujo se determina la complejidad ciclomática [V (G)] del grafo de flujo resultante:

V (G) = # regiones = 3

V (G) = A – N + 2 = (10 - 9) + 2 = 3

V (G) = P + 1 = 2 + 1 = 3

Luego de determinada la complejidad ciclomática se tiene que 3 es el número máximo de caminos a tener en cuenta para realizar las pruebas.

Caminos independientes determinados:

- Camino 1: Inicio-1-2-3-4-5-Fin
- Camino 2: Inicio-1-2-3-4-7-Fin
- Camino 3: Inicio-1-2-6-Fin

En la figura 12 se muestra el código del método “update”

```
    @Secured("hasRole('Gestor de Riesgos')")
    def update(Long id, Long version) {
/* 1 */ def riesgoInstance

/* 2 */ if (id != null) {
/* 3 */     riesgoInstance = Riesgo.get(id)

/* 4 */     if (!riesgoInstance) {
/* 5 */         flash.exception = message(code: 'default.not.found.message', args: [message(code: 'riesgo.
/* 5 */         redirect action: "list"
                return
            }

/* 6 */     if (version != null && riesgoInstance?.version > version) {
/* 7 */         riesgoInstance.errors.rejectValue("version", "default.optimistic.locking.failure", [messag
/* 7 */         render view: "edit", model: [riesgoInstance: riesgoInstance]
                return
            }

/* 8 */     if (params?.fechaDeteccion && params.fechaDeteccion != 'null') {
/* 9 */         params.fechaDeteccion = new Date(params.fechaDeteccion?.toString())
            }

/* 10 */    if (params?.fechaResolucion && params.fechaResolucion != 'null') {
/* 11 */        params.fechaResolucion = new Date(params.fechaResolucion?.toString())
            }

/* 12 */    riesgoInstance?.properties = params

/* 13 */    if (!riesgoInstance.save(flush: true)) {
/* 14 */        render view: "edit", model: [riesgoInstance: riesgoInstance]
                return
            }

/* 15 */    flash.message = message(code: 'default.updated.message', args: [message(code: 'riesgo.label',
/* 15 */    redirect action: "show", id: riesgoInstance?.id

        }
        else {
/* 16 */    flash.exception = message(code: 'default.not.received.message', args: [message(code: 'riesgo.l
/* 16 */    redirect action: "list"
        }
    }
```

Figura 16. Código del método “update”

Usando el código de la figura 14 como base, se dibuja el correspondiente grafo de flujo:

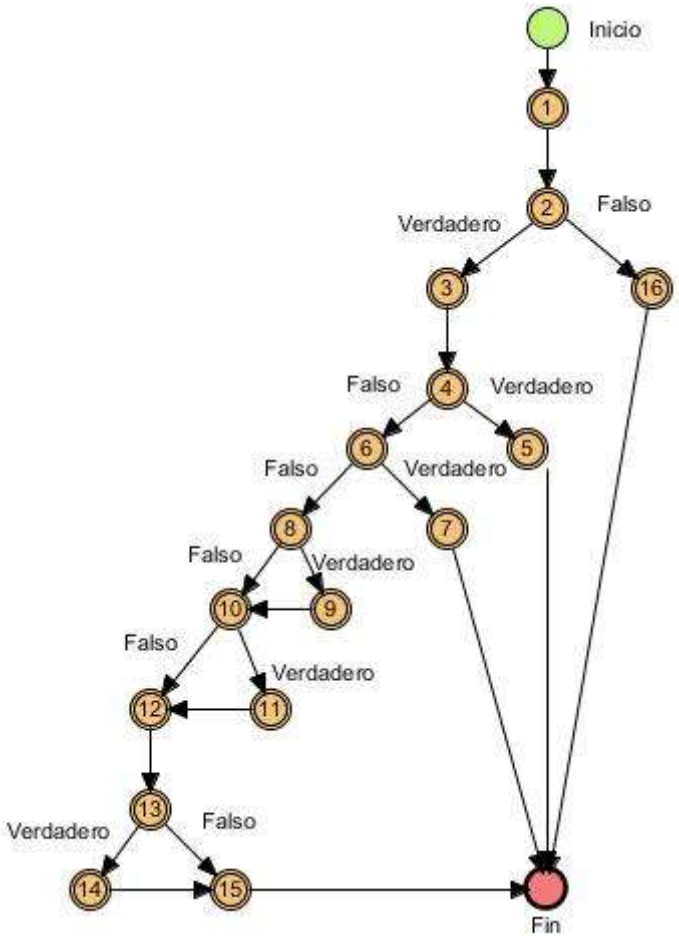


Figura 17. Grafo de flujo correspondiente al método “update“

Luego de realizado el grafo de flujo se determina la complejidad ciclomática [V (G)] del grafo de flujo resultante:

V (G) = # regiones = 7

V (G) = A – N + 2 = (23 - 18) + 2 = 7

V (G) = P + 1 = 6 + 1 = 7

Luego de determinada la complejidad ciclomática se tiene que 7 es el número máximo de caminos a tener en cuenta para realizar las pruebas.

Caminos independientes determinados:

- Camino 1: Inicio-1-2-16-Fin
- Camino 2: Inicio-1-2-3-4-5-Fin
- Camino 3: Inicio-1-2-3-4-6-7-Fin
- Camino 4: Inicio-1-2-3-4-6-8-10-12-13-15-Fin
- Camino 5: Inicio-1-2-3-4-6-8-9-10-12-13-15-Fin
- Camino 6: Inicio-1-2-3-4-6-8-9-10-11-12-13-15-Fin
- Camino 7: Inicio-1-2-3-4-6-8-9-10-11-12-13-14-Fin

JUnit

Es un conjunto de bibliotecas creadas por Erich Gamma y Kent Beck que son utilizadas en programación para hacer pruebas unitarias de aplicaciones Java. Permite realizar la ejecución de clases de manera controlada, para poder evaluar si el funcionamiento de cada uno de los métodos se comporta como se espera. A continuación se presenta como ejemplo, la realización de las mismas a las clases "RiesgoController" y "ActividadController", las restantes pueden ser consultadas dentro de la carpeta de anexos.

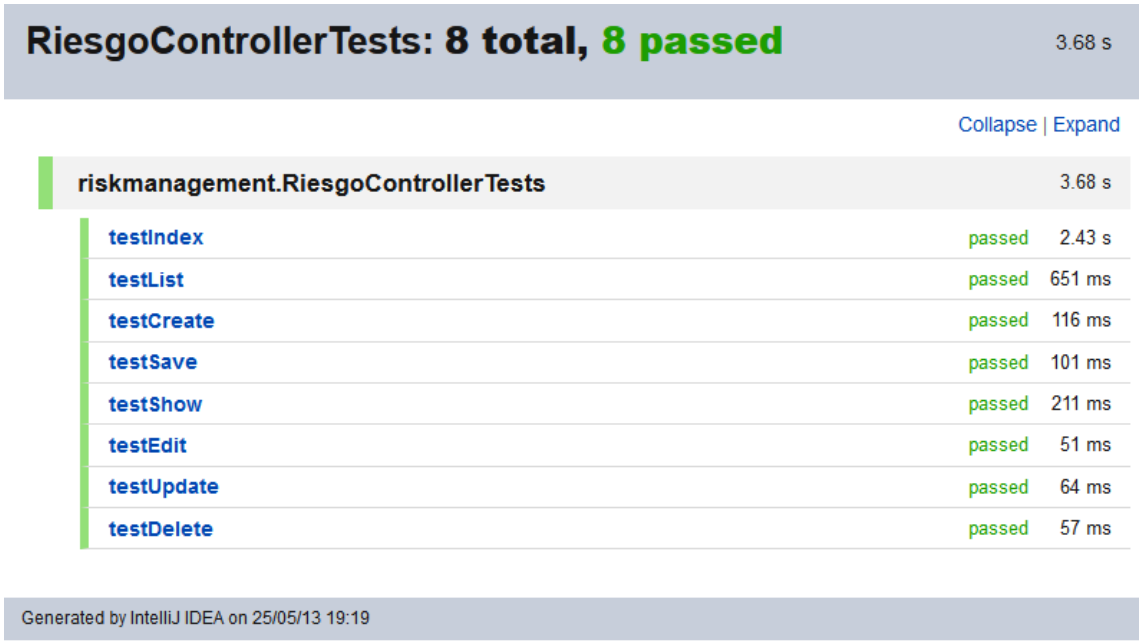


Figura 18. Pruebas Unitarias con JUnit: Clase "RiesgoController"

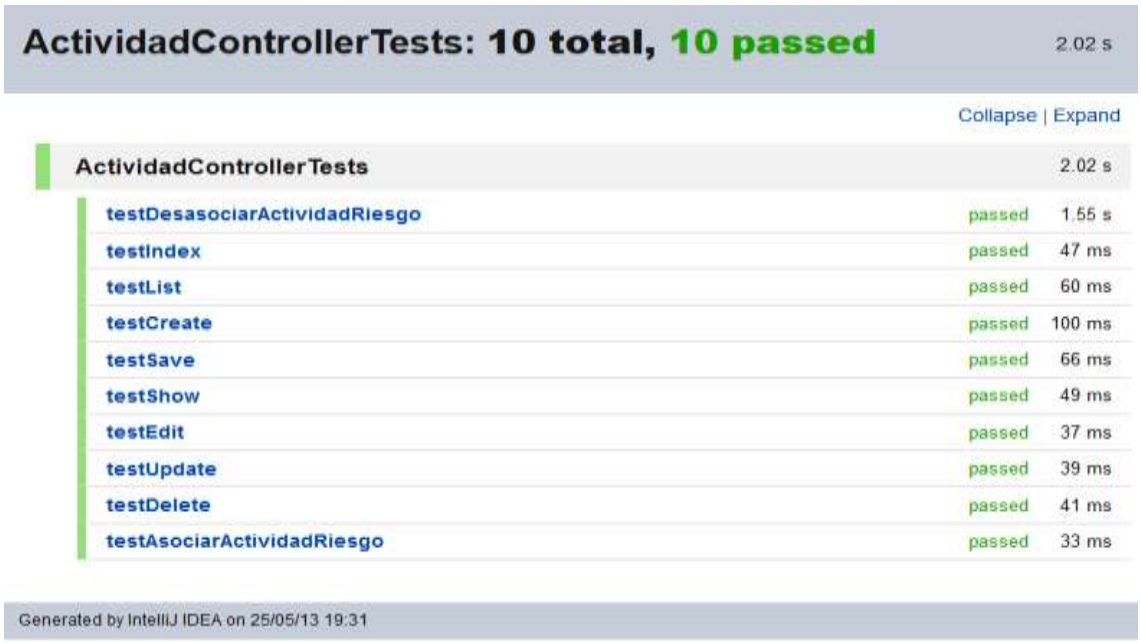


Figura 19. Pruebas unitarias con JUnit: Clase “ActividadController”

Resultados de las Pruebas Unitarias

De ambas formas, las pruebas dieron resultados satisfactorios, demostrando así un correcto funcionamiento del sistema.

3.2.2 Pruebas de Aceptación

Además de las pruebas unitarias anteriormente expuestas, se realizaron las llamadas pruebas de aceptación, las cuales son realizadas en base a las historias de usuarios, en cada ciclo de la iteración del desarrollo. El cliente debe especificar uno o diversos escenarios para comprobar que una historia de usuario ha sido correctamente implementada, y es responsable de verificar que los resultados de estas pruebas sean correctos. Una HU no se puede considerar terminada hasta tanto pase correctamente todas las pruebas de aceptación.

A continuación se muestran como ejemplo las pruebas de aceptación para las HU Insertar Actividades de Respuesta a un Riesgo, las restantes pueden ser consultadas en el documento “Casos de Prueba de Aceptación” dentro de la carpeta de anexos.

Tabla 16. Caso de Prueba: HU16_P1

	Caso de Prueba de Aceptación
Código Caso de Prueba:	**Nombre Historia de Usuario:** Insertar Actividades de

HU16_P1	Respuesta a un Riesgo

Nombre de la persona que realiza la prueba: Heydis Gutierrez Solano

Descripción de la Prueba: Prueba para comprobar que el sistema permite insertar actividades de respuesta a un riesgo especificado, en caso que el usuario inserte todos los datos correctamente.

Condiciones de Ejecución:

- El usuario debe estar autenticado en el sistema.
- El usuario debe contar con el privilegio para ejecutar la operación.

Entrada / Pasos de ejecución:

- Acceder al vínculo “Riesgos” en el menú lateral de la aplicación.
- Acceder al vínculo “Riesgos” en el sub-menú desplegado Esta acción re-direcciona a la página “Listado de Riesgos”.
- Acceder a las actividades de respuesta de un riesgo especificado, ubicada en la columna “Actividades de Respuesta”. Esta acción re-direcciona a la página “Actividades de Respuesta al Riesgo: nombre del riesgo seleccionado”.
- Acceder al botón “Nueva Actividad”. Esta acción re-direcciona a la página “Crear Actividad”.
- Introducir datos correctamente.
- Acceder al botón “Crear”.

Resultado Esperado: El sistema adiciona la actividad de respuesta al listado ya existente y muestra un mensaje indicando el éxito de la operación.

Resultado Obtenido:

Evaluación de la Prueba: Satisfactoria

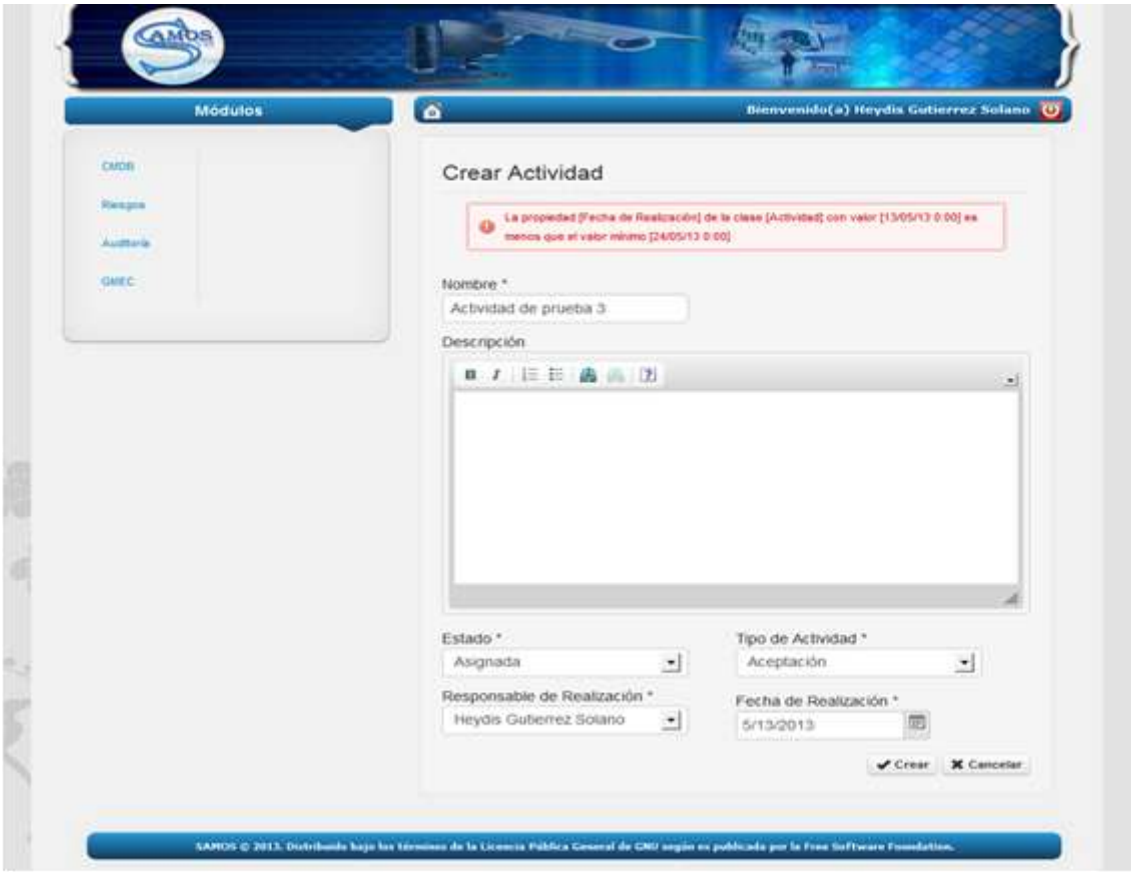
Tabla 17. Caso de Prueba: HU16_P2

Caso de Prueba de Aceptación	
Código Caso de Prueba: HU16_P2	**Nombre Historia de Usuario:** Insertar Actividades de Respuesta a un Riesgo
Nombre de la persona que realiza la prueba: Heydis Gutierrez Solano	
Descripción de la Prueba: Prueba para comprobar que el sistema no permite adicionar actividades de respuesta a un riesgo especificado, en caso que el usuario inserte datos incorrectamente.	
Condiciones de Ejecución: ▪ El usuario debe estar autenticado en el sistema. ▪ El usuario debe contar con el privilegio para ejecutar la operación.	
Entrada / Pasos de ejecución: ▪ Acceder al vínculo “Riesgos” en el menú lateral de la aplicación.	

- Acceder al vínculo “Riesgos” en el sub-menú desplegado Esta acción re-direcciona a la página “Listado de Riesgos”.
- Acceder a las actividades de respuesta de un riesgo especificado, ubicada en la columna “Actividades de Respuesta”. Esta acción re-direcciona a la página “Actividades de Respuesta al Riesgo: nombre del riesgo seleccionado”.
- Acceder al botón “Nueva Actividad”. Esta acción re-direcciona a la página “Crear Actividad”.
- Introducir datos incorrectamente.
- Acceder al botón “Crear”.

Resultado Esperado: El sistema muestra un mensaje indicando el error.

Resultado Obtenido:



Evaluación de la Prueba: Satisfactoria

Resultados de las pruebas de aceptación

Se realizaron 3 iteraciones de pruebas donde se detectaron varias no conformidades (NC) que fueron mitigadas durante el transcurso de la presente etapa. En la primera iteración de pruebas se detectaron un total de 9 NC, donde la mayoría eran por problemas de validación. Posteriormente en la segunda iteración se validó la corrección de los errores encontrados en la primera, aunque se detectaron 3 nuevas NC y por último en la tercera iteración no se encontraron NC, constituyendo esto un 100% de pruebas de aceptación exitosas, dicha iteración se realizó de manera conjunta con el cliente. Para una mayor constancia de la validación del producto, el Centro de Soporte generó un aval evidenciando su satisfacción, dicho documento se muestra anexado en el presente documento.

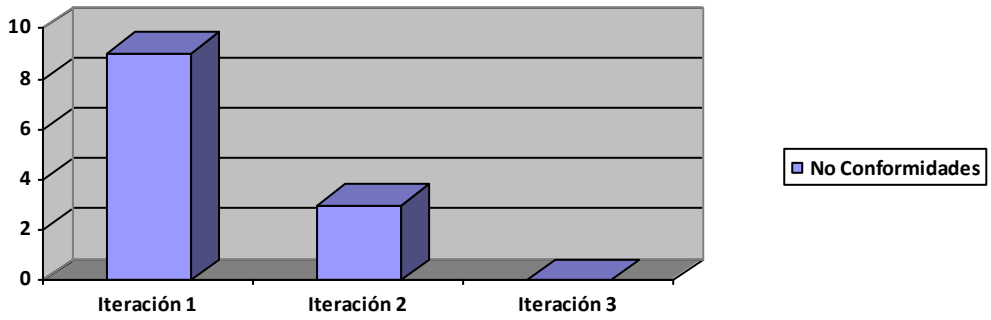


Figura 20. Resultados de las NC

3.3 Conclusiones Parciales

En el presente capítulo se ejecutaron las pruebas que propone la metodología de desarrollo de *software* seleccionada, permitiendo demostrar que el subsistema se encuentra listo y cumple con las exigencias del cliente lo cual fue comprobado mediante las pruebas de aceptación, participando el cliente en la validación realizada. Además se realizaron pruebas unitarias demostrando el correcto funcionamiento del sistema.

CONCLUSIONES GENERALES

Con el desarrollo del presente trabajo y la implementación del subsistema de GR para el sistema SAMOS, se concluye afirmando que:

- Se aportó un marco de trabajo de GR de servicios de TI donde:
  - Se identifican.
  - Se evalúan a través del análisis cualitativo.
  - Se brinda respuesta a los riesgos haciendo uso de estrategias o actividades para evitar, reducir, compartir o aceptar según la naturaleza de cada uno.
  - Se realiza un monitoreo y control de un Plan de Acción de Riesgos utilizando gráficos estadísticos y matriz de riesgo.
- La solución implementada cumple con las normas establecidas por el marco de trabajo de COBIT respecto a la GR.
- El uso de la metodología ágil de desarrollo XP, permitió desarrollar en el período de tiempo determinado, un producto con las funcionalidades especificadas por el cliente.
- Se validó la solución propuesta mediante pruebas de aceptación y unitarias, las cuales fueron vencidas satisfactoriamente, demostrando así, el correcto funcionamiento del sistema y el buen cumplimiento de las expectativas del cliente.

RECOMENDACIONES

Con la intensión de asegurar la posterior mejora del Subsistema de Gestión de Riesgos del SAMOS, se propone a continuación la recomendación siguiente:

- Implementar un modelo para el análisis cuantitativo de los riesgos.

GLOSARIO DE TÉRMINOS

Gestión de Servicios de TI: Práctica basada en el proceso destinado a armonizar la entrega de TI con las necesidades de la empresa, que hace hincapié en los beneficios para los clientes. Ayuda a las organizaciones a definir los procesos que aseguran que estas interacciones generan un conjunto de respuestas oportunas consistentes que garanticen un alto nivel de calidad del servicio (45).

Estándar: Conjunto de reglas que deben cumplir los productos, procedimientos o investigaciones que afirmen ser compatibles con el mismo producto. Los estándares ofrecen muchos beneficios, reduciendo las diferencias entre los productos y generando un ambiente de estabilidad, madurez y calidad en beneficio de consumidores e inversores (38).

Proceso: Conjunto de acciones o actividades sistematizadas que se realizan o tienen lugar con un fin (8).

ISACA: Organización líder en Auditoria de Sistemas y Seguridad de los Activos de Información.

Proyecto: Conjunto de las actividades que desarrolla una persona o una entidad para alcanzar un determinado objetivo. Estas actividades se encuentran interrelacionadas y se desarrollan de manera coordinada (15).

Software: Es una palabra que proviene del idioma inglés, pero que gracias a la masificación de uso, ha sido aceptada por la Real Academia Española (RAE). Según la RAE, el software es un conjunto de programas, instrucciones y reglas informáticas que permiten ejecutar distintas tareas en una computadora (15).

JAVA: Es un lenguaje de programación orientado a objetos desarrollado por la Sun Microsystems, una compañía famosa sobre todo por sus estaciones UNIX de faja alta. Inspirado al C++, Java fue proyectado con la finalidad de obtener un producto de pequeñas dimensiones, simple y portátil sobre diferentes plataformas y sistemas operativos, sea a nivel de código fuente que a nivel de código binario; lo que significa que los programas Java pueden ser ejecutados sobre cualquier computadora en la cual sea instalada la máquina virtual (38).

Máquina Virtual Java: Aplicación que interpreta y ejecuta programas escritos en el lenguaje de programación Java (39).

Lenguaje de programación: Es aquel elemento dentro de la informática que nos permite crear programas mediante un conjunto de instrucciones, operadores y reglas de sintaxis; que pone a disposición del programador para que este pueda comunicarse con los dispositivos hardware y software existentes (40).

Entorno de Desarrollo Integrado: Es un programa informático compuesto por un conjunto de herramientas para programar en un lenguaje de programación o varios, donde podemos encontrar como mínimo un editor, compilador, interprete y depurador de uno o varios lenguajes de programación (41).

Sistema Gestor de Base de Datos (SGBD): es un sistema de software que permite la definición de bases de datos; así como la elección de las estructuras de datos necesarios para el almacenamiento y búsqueda de los datos, ya sea de forma interactiva o a través de un lenguaje de programación. Un SGBD relacional es un modelo de datos que facilita a los usuarios describir los datos que serán almacenados en la base de datos junto con un grupo de operaciones para manejar los datos (46).

Servidor web: Es un programa que sirve para atender y responder a las diferentes peticiones de los navegadores, proporcionando los recursos que soliciten usando el protocolo HTTP o el protocolo HTTPS (la versión cifrada y autenticada) (42).

Modelo de datos: es una estructura abstracta que documenta y organiza la información para la comunicación entre el personal del departamento técnico y el resto de los empleados. En la informática, difiere en cuanto a su enfoque, el cual se centra en el planeamiento del desarrollo de aplicaciones y la decisión de cómo se almacenarán los datos y cómo se accederá a ellos (15).

Diagrama Entidad-Relación: Es un modelo de red que describe la distribución de los datos almacenados en un sistema de forma abstracta. Se emplean para modelar bases de datos que pertenecen a un sistema informático (39).

Casos de Prueba: Un conjunto de entradas, condiciones de ejecución y resultados esperados desarrollados para un objetivo particular, un caso de prueba es utilizado por el analista para determinar si el requisito de una aplicación es parcial o completamente satisfactorio (43).

Grafo de Flujo: representación, en forma de grafo dirigido, de todos los caminos que pueden ser atravesados a través de un programa durante su ejecución (44).

REFERENCIAS BIBLIOGRÁFICAS

1. **Meltom Technologies.** deGerencia.com. [En línea] 2013. [Citado el: 22 de enero de 2013.] http://www.degerencia.com/tema/tecnologia_de_informacion.

2. **Piñero, Pedro.** *Conferencia 1: Contratación, Factibilidad de Proyectos y Negociación.* La Habana : Universidad de las Ciencias Informáticas : s.n., 2010.

3. **PMI.** *Guía de los Fundamentos para la Dirección de Proyectos. (Guía del PMBOK®).* Pennsylvania : s.n., 2008.

4.**Pressman, Roger.** *Ingeniería del software: Un enfoque práctico.* Madrid y Carchelejo (España) : s.n., 2001. 5ta edición.

5. **ISO/IEC.** *Information technology. Security techniques. Management of information and communications technology security.* 2004.

6.**ITIL.** ITILV3 Gestión de servicios de TI. [En línea] ITIL, 2012. http://itilv3.osiatis.es/gestion_servicios_ti.php..

7. **Charles, Lamb, Joseph, Hair and McDaniel, Carl.** *Marketing.* s.l. : International Thomson, 2002. Sexta Edición.

8.**definicionABC**. [En línea] 2013. http://www.definicionabc.com.

9. **Grande, Estaban.** *Marketing de los servicios.* Madrid : ESIC., 2005. 4ta edicion.

10. **PMI.** *Guía de los Fundamentos de la Dirección de Proyectos.* Pennsylvania, : Project Management Institute, 2004.

11. **AEC"asociación española para la calidad**. [En línea] 2013. [Citado el: 18 de abril de 2013.] http://www.aec.es/web/guest/centro-conocimiento/cobit.

12. [En línea] 2009. http://www.softwareseleccion.com/softexpert+erm+suite-p-1011.

13. **TwitterBootstrap** | the fuel blog. [En línea] 21 de enero de 2013. www.fuelgrafics.com/blog/tag/twitter-bootstrap/.

14. **Yiser.** Master Executive en administración y Dirección de empresas. [En línea] 20 de mayo de 2013. http://www.eoi.es/blogs/madeon/2013/04/15/combinacion-de-las-herramientas-cobit-y-pmbok-para-potenciar-la-gestion-exitosa-de-los-proyectos/.

15. **WordPress.** Definicion.de. [En línea] 2013. [Citado el: 18 de marzo de 2013.] http://definicion.de/sitio/.

16. **Softonic**. [En línea] INTERSHARE, 2013. http://intellij-idea.softonic.com/.

17. **SIG Implementation.** SIG Implementation.SistemasIntegrados de Gestion. [En línea] 2012. http://www.implementacionsig.com.

18. **SICELCA.** SICELCA IT Systems. [En línea] 2013. [Citado el: 6 de mayo de 2013.] http://www.sicelca.com/.

19. **postgresql-es**. [En línea] 2012. http://www.postgresql.org.es/sobre_postgresql.

20. **ORCA SOFTWARE GRC SUITE**. [En línea] 2012. http://www.gcpglobal.com/orca-descripcion.php.

21. **Murphy, R y y otros.** Continuous Risk Management Guidebook. [En línea] Software Engineering Institute, 2008. [Citado el: 11 de abril de 2013.] http://www.sei.cmu.edu/publications/books/other-books/crm.guidebk.html..

22. **Morales.** AUDITOL, Red de conocimiento de auditoría y control interno. [En línea] 2013. http://www.auditool.org/index.php?option=com_content&view=article&id=700:administracion-de-riesgos-conceptos-fundamentales-parte-1&catid=39:trip-deals&Itemid=56.

23. **Groovy**. [En línea] 16 de abril de 2013. http://groovy.codehaus.org/.

24. **Definicion.de**. [En línea] 2013. [Citado el: 19 de marzo de 2013.] http://definicion.de/servicio/.

25. **Estratega,consultoria**. [En línea] 2010. http://estratega.org/site/draft-created-on-09092010-at-1833/.

26. **1024**. [En línea] 2011. http://www.1024.com.uy/revista/index.php/galileo-galilei/110-grails-el-santo-grial-de-java.2011.

27. **html5facil**. [En línea] 2013. http://html5facil.com/tips/kendo-ui-quiere-que-tus-interfaces-sean-realizadas-mas-rapidas-y-mejores.

28. **highcharts**. [En línea] 2013. www.highcharts.com/.

29. **Cuba Rondon, Enelis Blanca.** *Fusión de la información de los servicios de la comunidad universitaria de la UCI con el metaverso.* 2012.

30. **Larman, Craig.** *UML Y PATRONES. Introducción al analisis y diseño orientado a objetos.* Mexico : s.n., 1999.

31. **Scrum Manager.** Scrum Manager. [En línea] 23 de enero de 2013. [Citado el: 16 de mayo de 2013.] http://www.scrummanager.net/bok/index.php?title=Pruebas_unitarias.

32. **ISACA**. Sitio oficial de ISACA. [En línea] ISACA, 2013. [Citado el: 3 de Junio de 2013.] referencia: http://www.isaca.org/Knowledge-Center/Val-IT-IT-Value-Delivery-/Pages/Val-.

33. —. Sitio oficial de ISACA. [En línea] ISACA, 2013. [Citado el: 3 de Junio de 2013.] http://www.isaca.org/Knowledge-Center/Risk-IT-IT-Risk-Management/Pages/Risk-IT1.aspx?utm_source=multiple&utm_medium=multiple&utm_content=friendly&utm_campaign=riskit.

34. **Argota, Irina**. Na-vegando en la dynamic Matrix del conocimiento... [En línea] 17 de Abril de 2012. [Citado el: 3 de Junio de 2013.]

http://vegamatrica.blogspot.com/2012/04/metodologia-agil-o-formal-en-tu-empresa.html.

35. **Zadunaisky, Mauro**. PlusGlobal. [En línea] 12 de Marzo de 2013. [Citado el: 3 de Junio de 2013.] http://es.plusglobal.com/blog/agiliza-tu-trabajo-con-un-*framework*/.

36. **Martínez Vigil , Iván**. Módulo de Gestión de Riesgos versión 2.0 para el sistema GESPRO 12.05. La Habana : s.n., 2012.

37. **UNAM**. UNAM-Facultad de Ingeniería Biometría Infórmatica. [En línea] [Citado el: 3 de Junio de 2013.] http://redyseguridad.fi-p.unam.mx/proyectos/biometria/estandares/estandar.html.

38. **Babylon.** Babylon. [Online] 2012. [Cited: Junio 4, 2013.] http://diccionario.babylon.com/java/.

39. **ALEGSA.** alegsa. [En línea] 2012. [Citado el: 4 de Junio de 2013.] http://www.alegsa.com.ar/Dic/maquina%20virtual%20java.php.

40. **definición.org.** [En línea] 2013. [Citado el: 4 de Junio de 2013.] http://www.definicion.org/lenguaje-de-programacion.

41. **Regalut.** Desarrollo Movil Multiplataforma. [En línea] 26 de Agosto de 2012. [Citado el: 4 de Junio de 2013.] http://desarrollomovilmultiplataforma.blogspot.com/2012/08/aspectos-teoricos-entorno-de-desarrollo.html.

42. **cibernetia.** Cibernetia. [En línea] 2013. [Citado el: 4 de Junio de 2013.] http://www.cibernetia.com/manuales/instalacion_servidor_web/1_conceptos_basicos.php.

43. **plaza-entretenimiento.** Plaza de Entretenimiento Huetamo . [En línea] 13 de Mayo de 2010. [Citado el: 4 de Junio de 2013.] http://plaza-entretenimiento.blogspot.es/.

44. **AVRORA.** AVRORA. [En línea] [Citado el: 4 de Junio de 2013.] http://compilers.cs.ucla.edu/avrora/cfg.html.

45. **The University of Chicago.** ITServices. [En línea] 2012. [Citado el: 4 de Junio de 2013.] https://itservices.uchicago.edu/page/it-service-management-initiatives.

46. **ECURED.** ECURED. [En línea] 14 de Diciembre de 2010. [Citado el: 4 de Junio de 2013.] http://www.ecured.cu/index.php/Sistema_Gestor_de_Base_de_Datos.

BIBLIOGRAFÍA

1. **ITGI, ISACA.** *Cobit 4.1. Marco de trabajo. Objetivos de control. Directrices gerenciales. Modelos de Madurez.* 2007.

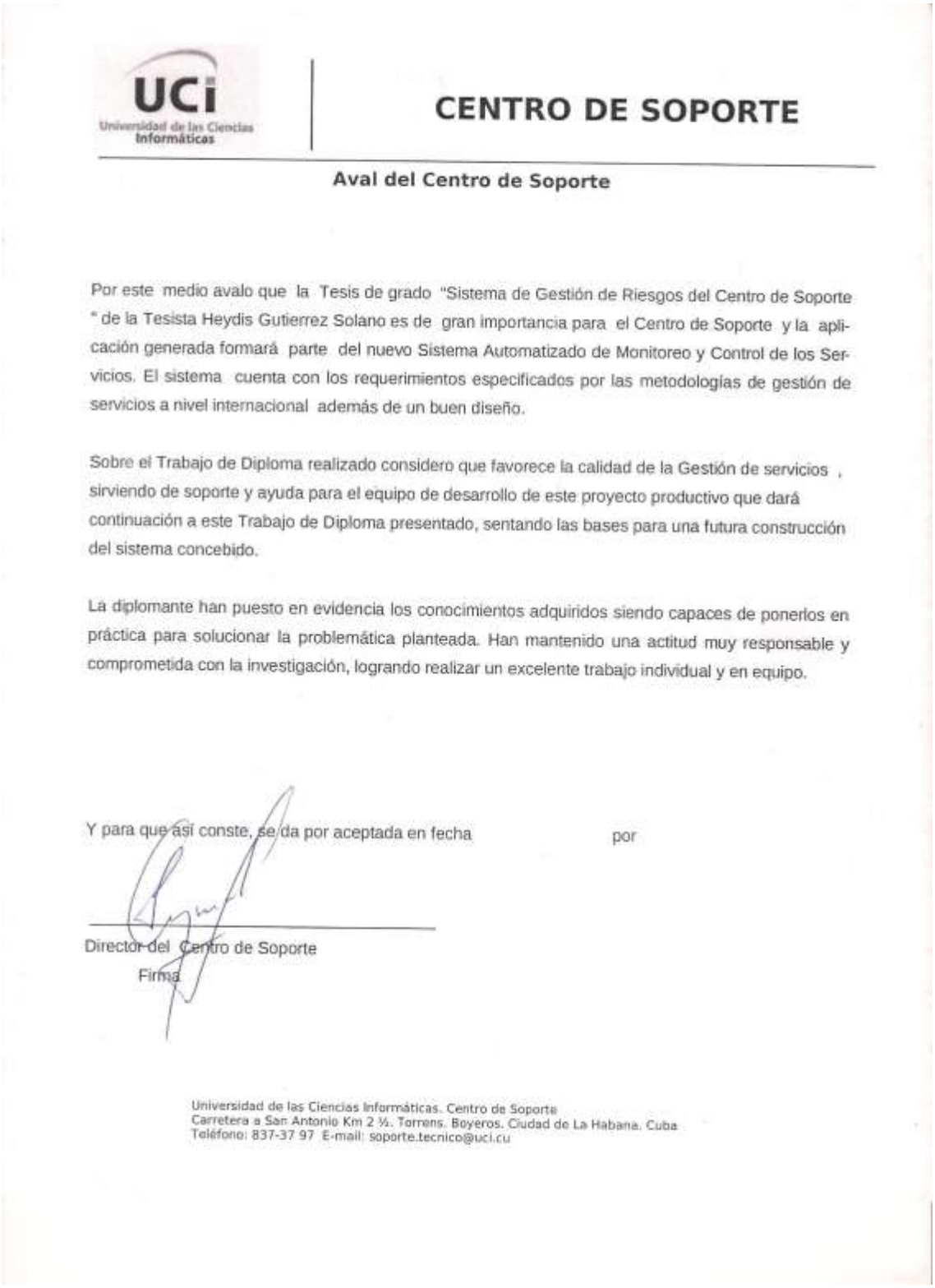
2. **Larman, Craig.** *UML Y PATRONES. Introducción al analisis y diseño orientado a objetos.* Mexico : s.n., 1999.

3. **Brito, Nacho.** *Manual de desarrollo web con Grails.* 2009.

4. **Pivotal**. Grails. [Online] Pivotal, 2013. http://www.grails.org/.

ANEXOS

1. Aval del Centro de Soporte UCI



UCI
Universidad de las Ciencias Informáticas

CENTRO DE SOPORTE

Aval del Centro de Soporte

Por este medio avalo que la Tesis de grado "Sistema de Gestión de Riesgos del Centro de Soporte " de la Tesista Heydis Gutierrez Solano es de gran importancia para el Centro de Soporte y la aplicación generada formará parte del nuevo Sistema Automatizado de Monitoreo y Control de los Servicios. El sistema cuenta con los requerimientos especificados por las metodologías de gestión de servicios a nivel internacional además de un buen diseño.

Sobre el Trabajo de Diploma realizado considero que favorece la calidad de la Gestión de servicios , sirviendo de soporte y ayuda para el equipo de desarrollo de este proyecto productivo que dará continuación a este Trabajo de Diploma presentado, sentando las bases para una futura construcción del sistema concebido.

La diplomante han puesto en evidencia los conocimientos adquiridos siendo capaces de ponerlos en práctica para solucionar la problemática planteada. Han mantenido una actitud muy responsable y comprometida con la investigación, logrando realizar un excelente trabajo individual y en equipo.

Y para que así conste, se da por aceptada en fecha por

Director del Centro de Soporte
Firma

Universidad de las Ciencias Informáticas. Centro de Soporte
Carretera a San Antonio Km 2 ½. Torrens. Boyeros. Ciudad de La Habana. Cuba
Teléfono: 837-37 97 E-mail: soporte.tecnico@uci.cu

Figura 21. Aval del Centro de Soporte UCI