Universidad de las Ciencias Informáticas

Facultad 2

Título: Sistema Inteligente de Mitigación de Riesgos para el Centro ISEC

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

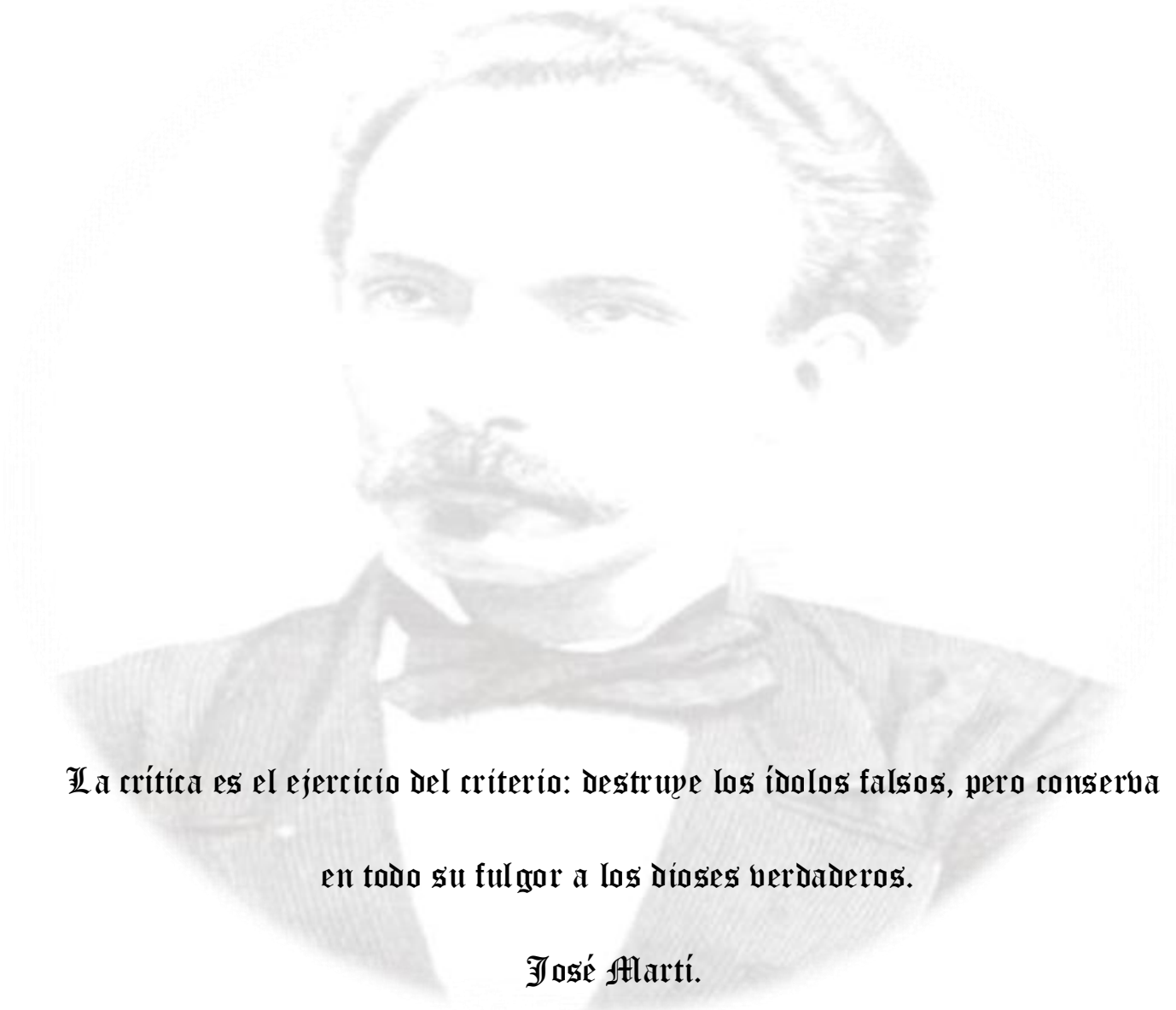
Autor(es): Dasiel Cordero Morales.
Yoanny Torres Rubio.

Tutoras: MSc. Yadira Ruiz Constanten.
Ing. Dariena Ramírez Luján.

Consultante: Dra Natalia Martínez Sánchez.

Junio de 2011.

PENSAMIENTO.



La crítica es el ejercicio del criterio: destruye los ídolos falsos, pero conserva en todo su fulgor a los dioses verdaderos.

José Martí.

DECLARACIÓN DE AUTORÍA.

Declaro que soy el único autor de este trabajo y autorizo al centro de Informatización para la Seguridad Ciudadana de la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio.

Para que así conste firmo la presente a los ____ días del mes de _____ del año _____.

Yoanny Torres Rubio
_____

Dasiel Cordero Morales
_____

RESUMEN.

La acumulación de experiencias en la concepción de proyectos, el surgimiento de organizaciones y estándares dedicados a la planificación y administración de los mismos, han permitido que su desarrollo se haya convertido en una actividad planificada y controlada. La gestión de riesgos durante el ciclo de vida de desarrollo de software es un proceso complejo estrechamente vinculado al dominio que tenga el equipo de desarrollo sobre el tema. De su correcta gestión dependerán gran parte de los resultados, teniendo en cuenta las ventajas que tanto tecnológicas como económicas reportará para el equipo de desarrollo la mitigación de los mismos. Los riesgos han de analizarse para propiciar el aprovechamiento de las diversas oportunidades que puedan ofrecer y de la misma forma, evitar que sean muy severos e irreparables, los daños que puedan provocar.

Los sistemas basados en casos (SBC) ayudan y agilizan la toma de decisiones simulando las cadenas de razonamiento que realiza un experto para resolver un problema de su dominio. Su vinculación con distintos elementos del proceso de desarrollo de software posibilita la obtención de resultados más acertados a partir del conocimiento que se les introduce basado en la experiencia acumulada.

Actualmente el centro de Informatización para la Seguridad Ciudadana ISEC, ha venido confrontando dificultades para una rápida y acertada gestión de riesgos. La utilización de las facilidades que brindan los SBC para la gestión de riesgos en un proyecto de desarrollo de software resultaría ventajoso al brindar las herramientas necesarias para que los líderes de proyecto, basados en experiencias anteriores, realicen una planificación más acertada, teniendo en cuenta los diversos contratiempos que pudieran surgir.

PALABRAS CLAVE: gestión, riesgos, mitigación, proyecto, sistemas basados en casos.

ÍNDICE.

ÍNDICE.

ÍNDICE.

ÍNDICE DE FIGURAS.

ÍNDICE DE TABLAS.

INTRODUCCIÓN.

Desde la segunda mitad del siglo XX, se han venido acumulando experiencias en la concepción de proyectos de manera general, a la par de un creciente desarrollo de la industria y las tecnologías; aparejado al surgimiento de organizaciones y estándares dedicados a planificar, administrar y definir los elementos y métodos necesarios para el desarrollo de un proyecto. El Instituto de Gestión de Proyectos (Project Management Institute PMI®) es una organización internacional sin fines de lucro que asocia a profesionales para la gestión de proyectos (1). Actualmente, es la más grande del mundo en su rubro; dado que se encuentra integrada por más de 260.000 miembros alrededor de 171 países. El mismo, define que: un proyecto es un esfuerzo temporal, único y progresivo, emprendido para crear un producto o un servicio también único (2).

El desarrollo de un proyecto debe ser una actividad planificada y controlada, puesto que requiere participación humana en un tiempo dado para su cumplimiento, además de la utilización de recursos, lo cual depende de una buena gestión. La gestión de proyectos es la disciplina de organizar y administrar recursos de manera tal que se pueda culminar todo el trabajo requerido en el proyecto dentro del alcance, el tiempo, y costo definidos (3). La Guía del Proyecto Órgano de Gestión del Conocimiento (Project Management Body Of Knowledge - PMBOK®), desarrollada por el PMI, contiene una descripción general de los fundamentos de la Gestión de Proyectos. El PMBOK es una colección de procesos y áreas de conocimiento generalmente aceptadas como las mejores prácticas dentro de la gestión de proyectos que provee los fundamentos que son aplicables a un amplio rango de proyectos, incluyendo construcción, software, ingeniería, entre otros. PMBOK reconoce 5 grupos de procesos básicos y 9 áreas de conocimiento comunes a casi todos los proyectos.

Los 5 grupos básicos de procesos son:
Inicio, Plan, Ejecución, Control y Monitoreo y Cierre.

Las nueve áreas del conocimiento mencionadas en el PMBOK son:

- Gestión de la Integración.
- Gestión del Alcance.
- Gestión del Tiempo.

- Gestión de la Calidad.
- Gestión de Costos.
- Gestión del Riesgo.
- Gestión de Recursos Humanos.
- Gestión de la Comunicación.
- Gestión de las Compras y Adquisiciones.

Los proyectos de desarrollo de software se diferencian de los otros proyectos de ingeniería tradicional en la naturaleza lógica del producto. El software se desarrolla, no se fabrica en un sentido clásico. La gestión del proyecto de software es el primer nivel del proceso de ingeniería de software, porque cubre todo el proceso de desarrollo. Para lograr un proyecto de software fructífero se debe comprender el ámbito del trabajo a realizar, los riesgos en los que se puede incurrir, los recursos requeridos, las tareas a llevar a cabo, el esfuerzo a consumir y el plan a seguir.

Como cualquier actividad humana, el desarrollo de un proyecto incluye la ocurrencia de riesgos, los cuales son eventos o condiciones inciertas, que si se producen, afectan de manera positiva o negativa al menos un objetivo del proyecto (4). Un riesgo puede tener una o más causas y en caso de ocurrir, uno o más impactos. Por lo que la mejor manera de mitigar un riesgo, o un conjunto de estos, durante el desarrollo de un proyecto, es una acertada Gestión de Riesgos. Esta última no es más que el arte y la ciencia de identificar, analizar y responder a los riesgos a lo largo de la vida de un proyecto, con el propósito de aumentar la probabilidad y el impacto de los eventos positivos y disminuir la probabilidad y el impacto de los eventos adversos para el proyecto (5). La Gestión de Riesgos en el desarrollo de un proyecto de software, es un proceso generalmente complejo, dependiendo de las características de cada proyecto, que requiere seguimiento constante, y depende de las experiencias acumuladas, del conocimiento adquirido y de los estándares definidos.

Con el desarrollo de las Ciencias y las Tecnologías, han surgido disciplinas capaces de predecir e identificar un resultado determinado o un evento que puede ocurrir simulando el razonamiento humano. A la rama de las Ciencias de la Computación dedicada al desarrollo o uso de los ordenadores, con los que se intenta reproducir los procesos de la inteligencia humana se le denomina Inteligencia Artificial (IA) (6).

Dentro de esta ciencia, existen diferentes técnicas basadas en el conocimiento, que permiten la solución de problemas de forma variada. Una de las más utilizadas son los Sistemas Basados en el Conocimiento (SBC), los cuales se definen como: Un sistema computarizado que usa conocimiento sobre un dominio para arribar a una solución de un problema de ese dominio. Esta solución es esencialmente la misma que la obtenida por una persona experimentada en el dominio del problema cuando se enfrenta al mismo problema (6).

Los SBC están caracterizados por más rasgos que simplemente el hecho de duplicar el conocimiento y experticia de un experto humano para un dominio específico. Otro rasgo que lo distingue de la programación convencional es que pueden dar diversas soluciones. Es posible y resultaría ventajoso, utilizar las facilidades que brindan estos SBC para una acertada gestión de riesgos en un proyecto de desarrollo de software, pues conociendo cómo mitigar algunos riesgos, se puede utilizar este conocimiento para darle solución a problemas de mitigación de los mismos en otros escenarios.

El Centro de Informatización para la Seguridad Ciudadana ISEC, en la Universidad de las Ciencias Informáticas, tiene como labor fundamental el desarrollo de proyectos pertenecientes a ese perfil, ha venido incorporando y desarrollando proyectos de ese tipo de manera vertiginosa. Luego de entrevistas realizadas a diferentes líderes de proyectos, se logró identificar que existen muchas actividades en las que se emplean, según las características de cada proyecto, tiempo y esfuerzo en realizarlas y resolverlas. Tal es el caso de los procesos de identificación y tratamiento de los riesgos, que no se realizan de manera adecuada durante todo el ciclo de desarrollo de software. .Se pudo identificar que existe dualidad de funciones y responsabilidades entre los integrantes del equipo de desarrollo. No se le dedica el tiempo adecuado a la producción y muchas veces no se cuenta con los recursos necesarios. Existiendo en otros casos una mala planificación de los mismos. De manera general, el personal que integra el equipo de desarrollo no cuenta con la preparación ni la experiencia adecuada. No se dispone de un tiempo de capacitación y preparación previo, y si existe no es el que se requiere. Lo antes expuesto unido a la inadecuada gestión del conocimiento provoca inestabilidad en la fuerza de trabajo. Aparejado a esto, surgen problemas de cambios de los requisitos por parte del cliente, además de demora del mismo en la revisión de la documentación del proyecto.

Esta ineficiente gestión de riesgos, es una de las consecuencias fundamentales en el atraso de los cronogramas de los proyectos del centro, afectando todas las fases del desarrollo del software, lo cual incide de forma negativa en el cumplimiento de los objetivos propuestos según el alcance definido en cada

compromiso del Centro y la Universidad, sobre todo por las necesidades de cumplirlos en el tiempo establecido. Por todo lo expuesto anteriormente, surge como problema a resolver: **¿Cómo identificar y mitigar los riesgos en el centro (ISEC) para contribuir a la disminución de los atrasos en los proyectos?** De esta forma el objeto de estudio sobre el cual se enmarca este trabajo de diploma sería: **Herramientas inteligentes para la toma de decisiones**. Exponiéndose así, como Objetivo General**, desarrollar una herramienta inteligente para la identificación y mitigación de Riesgos en los proyectos del Centro ISEC.** El campo de acción lo constituye: **Herramientas Inteligentes basadas en Sistemas Expertos para la gestión de riesgos**. Definiéndose así los siguientes **Objetivos Específicos** y **Tareas de Investigación:**

Objetivos específicos:

- Definir procesos fundamentales relacionados con la gestión de riesgos tras el análisis detallado de las metodologías que abordan el tema.
- Definir requerimientos mínimos indispensables a ser resueltos con el sistema a desarrollar.
- Realizar análisis y diseño del sistema.
- Formalizar un modelo computacional donde se aplique técnicas de inteligencia artificial en la consideración de Mitigación de Riesgos como entidad principal.
- Realizar la implementación computacional del modelo propuesto.

Tareas de Investigación:

- Elaboración del marco teórico de la Investigación.
- Caracterización de la situación existente en el mundo, en la región y en el país en particular sobre la gestión de riegos y definir la posición de los investigadores.
- Análisis detallado sobre los Lineamientos Mínimos de Calidad orientados por la Dirección de Calidad de Software y sus acápites relacionados con los riesgos.
- Análisis de los procesos de la gestión de Riesgos según los siguientes modelos:

  - ✓ Guía de los Fundamentos de la Dirección de Proyectos (PMBOK).
  - ✓ APM Body of Knowledge.
  - ✓ MOGERI.

- Identificación de los riesgos más comunes (positivos y negativos) relacionados con los proyectos del Centro ISEC.
- Clasificación de los riesgos más comunes (positivos y negativos) relacionados con los proyectos del Centro ISEC.

Métodos Teóricos:

- **Analítico–Sintético:** Mientras que el análisis permitió el estudio de cada uno de los factores que influyen en la realización de la Gestión de Riesgos (GR) en su relativa independencia uno de otro, la síntesis permitió descubrir las relaciones existentes entre dichos factores, así como la interacción dialéctica que se establece entre ellos y el condicionamiento mutuo que ejercen sobre la GR.

- **Inductivo-Deductivo:** Se utilizó para el planteamiento del objetivo y la extracción de las ideas fundamentales para la elaboración y fundamentación del trabajo de diploma.

- **Histórico-Lógico:** El método histórico permitió estudiar la trayectoria real de la GR, además de herramientas inteligentes que permitan su tratamiento y acontecimientos fundamentales en el decursar de la historia de ambos. El método lógico permitió investigar las leyes generales del tema y las peculiaridades de los marcos estudiados. La utilización de este método posibilitó que el estudio no se limitara a una simple descripción de los hechos sino que facilitó el descubrimiento de la lógica objetiva del desarrollo histórico de la GR y su integración con herramientas inteligentes para su tratamiento.

Métodos Empíricos:

- **Entrevista**: Se realizó con el objetivo de identificar el grado de conocimiento de los involucrados acerca de la GR, su utilización en la UCI, y su aplicación en los proyectos de desarrollo de software del Centro de Informatización de la Seguridad Ciudadana (ISEC), así como para distinguir aspectos a incluir en el modelo para facilitar su aplicación y efectividad.

El documento está estructurado en 4 capítulos:

Capítulo1. “Fundamentación Teórica”: se especifican conceptos que serán tratados a lo largo del documento y que son de vital importancia para la comprensión del mismo. Se realiza un estudio del arte de sistemas inteligentes para la mitigación de riesgos.

Capítulo 2. “Características del Sistema”: Está orientado a la identificación de las necesidades de los clientes. Descripción general del funcionamiento del sistema.se realiza una detallada descripción de los requisitos funcionales y no funcionales del sistema. Se modelan y describen los Casos de Uso del Sistema. Se realiza el diseño de un prototipo de interfaz inicial en correspondencia a los requisitos funcionales.

Capítulo 3: “Diseño del sistema”: Se documenta la disciplina de diseño, incorporando los elementos definidos por la metodología de desarrollo de software FDD.

Capítulo 4. “Implementación y Prueba”: Se especifican los detalles de los algoritmos utilizados, los diagramas de componentes y la estrategia de prueba empleada.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA.

Introducción del Capítulo.

En el presente capítulo se describen los antecedentes y el estado existente en el mundo, en la región y en el país en particular sobre la gestión de riegos, además de la inteligencia artificial como técnica que simplifica y facilita la gestión de riesgos en proyectos de desarrollo de software. Se incluyen también conceptos teóricos de las tecnologías que se utilizan para implementar la solución que se propone en esta investigación, así como las metodologías de desarrollo de software utilizadas.

1.1 Gestión de Riesgos (GR):

La GR es el arte y la ciencia de identificar, analizar, y responder a los riesgos a lo largo de la vida de un proyecto (7), con el propósito de aumentar la probabilidad y el impacto de los eventos positivos y disminuir la probabilidad y el impacto de los eventos adversos para el proyecto.

La “GR del proyecto”, incluye los procesos siguientes:

Planificación de la gestión de riesgos.
Consiste en definir y planificar las actividades del proyecto que gestionen los riesgos, con el fin de proporcionar recursos y tiempo suficientes para las actividades de esta. Una planificación de la GR cuidadosa y explícita mejora la posibilidad de éxito de los otros cinco procesos. Este proceso debe completarse en las fases tempranas de la planificación del proyecto.

La principal salida de este proceso es el Plan de GR, el cual documenta la metodología, roles, responsabilidades, presupuesto y recursos para implementar dicho plan de riesgos, es decir, en él se describe como se estructurará y realizará la gestión de riesgos en el proyecto. El Plan de GR incluye:

- Actividades organizativas (Metodología, Roles y Responsabilidades, Tolerancias de los interesados).
- Categorización de riesgos (Categorías de riesgo, Definiciones de probabilidad e impacto de los riesgos, Formato de informes).
- Actividades de seguimiento (Preparación del presupuesto, periodicidad, seguimiento).

Identificación de los riesgos:

Es el proceso por el cual se determina qué riesgos pueden afectar al proyecto y se documentan sus características. Constituye un proceso interactivo, ya que se descubrirán nuevos riesgos a medida que se avance con el ciclo de vida del proyecto.

Análisis cualitativo de los riesgos:

Es el proceso que permite evaluar los riesgos. Además identifica las causas probables de su ocurrencia y determina los riesgos con mayores probabilidades de ocurrencia e impacto en el proyecto. Esto involucra una estimación de incertidumbre del riesgo y su impacto así como los análisis de causas y relaciones entre riesgos.

Análisis cuantitativo de los riesgos:

Consiste en la evaluación cuantitativa de los riesgos identificados. Generalmente sigue al proceso Análisis Cualitativo de Riesgos. En algunos casos, es posible que no sea necesario el Análisis Cuantitativo de Riesgos para desarrollar respuestas efectivas a los mismos. La disponibilidad de tiempo y presupuesto, y la necesidad de enunciados cualitativos o cuantitativos acerca de los riesgos y sus impactos, determinarán qué métodos usar en cualquier proyecto en particular. Los elementos de evaluación cuantitativa más comunes son: la probabilidad de ocurrencia y su impacto.

Planificación de la respuesta los riesgos:

Es el proceso de desarrollar opciones y de determinar acciones para reducir las amenazas de los objetivos del proyecto y aprovechar las oportunidades. Incluye la identificación y asignación de individuos para tomar la responsabilidad de responder a cada riesgo. Este proceso aborda los riesgos en función de su prioridad y asegura que los riesgos identificados sean tratados correctamente. Permite además introducir recursos y actividades en el presupuesto, cronograma y plan de gestión del proyecto. Por tanto la eficacia de la planificación determinará directamente si el riesgo aumenta o disminuye para el proyecto.

Existen tres estrategias que se ocupan de las amenazas o los riesgos que pueden tener impactos negativos sobre los objetivos del proyecto en caso de ocurrir. Las estrategias son: evitar, transferir o mitigar. A continuación se detallan las especificaciones de cada una de las estrategias:

Estrategias para Riesgos Negativos o Amenazas.

- Evitar.
- Transferir.
- Mitigar.

Estrategias para Riesgos Positivos u Oportunidades.

- Explotar.
- Compartir.
- Mejorar.

Seguimiento y control de riesgos.

Es el proceso de identificar, analizar y planificar nuevos riesgos, realizar el seguimiento de los riesgos identificados y los que se encuentran en la lista de supervisión, volver a analizar los riesgos existentes, realizar el seguimiento de las condiciones que disparan los planes para contingencias, realizar el seguimiento de los riesgos residuales y revisar la ejecución de las respuestas a los riesgos mientras se evalúa su efectividad. El proceso Seguimiento y Control de Riesgos aplica técnicas, como el análisis de variación y de tendencias, que requieren el uso de datos de rendimiento generados durante la ejecución del proyecto. El proceso Seguimiento y Control de Riesgos, así como los demás procesos de gestión de riesgos, es un proceso continuo que se realiza durante la vida del proyecto.

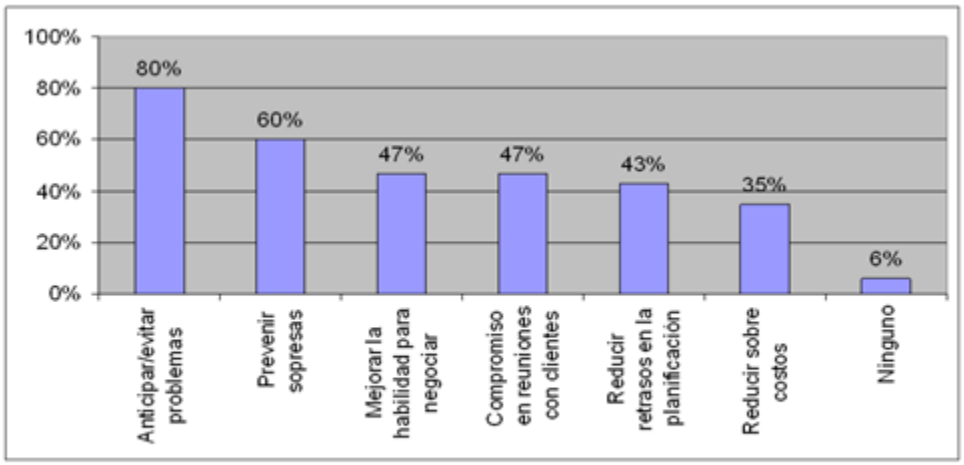
1.1.1 Antecedentes y situación actual de la gestión de riesgos.

Con el surgimiento del Instituto de Gestión de Proyectos, a principios de la década de los 70, que unió a profesionales de la gestión de proyectos, y con la creación por parte de esta misma institución en la década de los 90 del Proyecto Órgano de Gestión del Conocimiento (PMBOK Project Management Body Of Knowledge), se definieron las principales normas y estándares para la gestión de proyectos, incluyendo la gestión de riesgos.

Durante el año 2001 (Kuala Lumpur Composite Index / KLCI Research Group August 2001) se realizó y publicó un estudio con más de 260 organizaciones de todo el mundo y el resultado arrojó, como puede apreciarse en la figura 1.1, que el 3% de los entrevistados no utilizaba ningún marco de gestión del riesgo, el 18% utilizaba algún marco caótico para identificar sus riesgos, el 37% de los participantes habían utilizado algún marco informal, el 28% utilizaban procedimientos repetitivos y sólo un 14% utilizaba un enfoque formal para identificar riesgos (8). Según este estudio, las razones más comunes para utilizar un marco informal son: la falta de procedimientos, las necesidades del proyecto mal definidas, la organización

inexperta o inmadura y el compromiso del equipo. Aunque los diversos enfoques de GR aparecieron hace varias décadas, sigue siendo evidente la poca utilización de sus técnicas en los proyectos de desarrollo de software actuales (9).

Fig 1.1 Estudios Realizados por KLCI.



*Kulik, Peter and Catherine Weber, "Software Risk Management Practices – 2001," KLCI Research Group (August 2001).

Fig 1.2 Beneficios de la Gestión de Riesgos.

ISO/IEC 12207 es una norma técnica que establece un marco de referencia común para los procesos del ciclo de vida del software con una terminología bien definida. Contiene procesos, actividades y tareas para aplicar durante la adquisición de un sistema que contiene software, un producto software puro o un servicio software y durante el suministro, desarrollo, operación y mantenimiento de productos software. Los procesos que se emplean son: los procesos principales, los procesos de apoyo y los procesos organizativos del ciclo de vida. Dentro de los procesos organizativos del ciclo de vida se incluye la GR que tiene en este caso el propósito de identificar, analizar, tratar y monitorear los riesgos continuamente (10).

La Escuela de Negocios de la Universidad de Québec en Montreal, publicó una encuesta (11) que forma parte de la segunda fase de una investigación conjunta realizada con el Departamento de Búsquedas de

PMI / PMI Research Department y los resultados arrojados enfatizan en el estudio del uso e importancia en la gestión de proyectos, de herramientas y técnicas específicas para la gestión de riesgos, como son el análisis de requerimientos basados en la complejidad que adquieren hoy los ambientes operacionales, el nivel de uso de las herramientas en el proyecto, así como la planificación y el soporte además de las nuevas fuentes de riesgos.

1.1.2 Modelo de GR.

Un modelo de GR es una guía que define las tareas necesarias para la identificación de los riesgos y el análisis de estos. Además de proporcionar una vía para identificar factores potenciales de riesgos así como medidas para reducirlos o lograr que impacten según su propia característica. Además proveen métodos y definen tareas para el aseguramiento de los proyectos.

El marco de esta investigación, tendrá sus bases sobre MOGERI (Modelo de GR) definido para la UCI y las peculiaridades de su proceso productivo, en aras de la mejora del mismo. Recoge las prácticas adecuadas según el entorno donde debe aplicarse. Define procesos que permiten planificar las actividades sobre los riesgos, identificarlos, analizarlos, planificar las respuestas ante ellos, seguir y controlar los riesgos en el contexto del proyecto, así como comunicar la información generada al respecto (12)**.**

1.1.3 Gestión de Riesgos (GR) en Cuba.

El tratamiento de los riesgos se manifiesta en los Planes de Seguridad y los Planes de Contingencia de manera general, siendo débil la GR como elemento de la propia Gestión del Proyecto y como una actividad más en el desarrollo de software. Estos planes son vitales, pero frecuentemente solo tienen en cuenta los riesgos de tipo tecnológico.

Por otra parte, puede citarse el estudio de la Ing. Leidy Fernández y la Dra. Lourdes García (de la Empresa de Telecomunicaciones de Cuba SA y la Universidad Central de las Villas respectivamente) quienes en el artículo Gestión del riesgo en la fase de ingeniería de requisitos de un proyecto software proponen la GR como una actividad conformada por los siguientes pasos:

- **Paso 1**: Identificación de riesgos.
- **Paso 2**: Evaluación de los riesgos.
- **Paso 3**: Planificación de riesgos.

- **Paso 4**: Supervisión de los riesgos.

Las autoras concluyen que el tratamiento proactivo de los riesgos asociados a los requisitos del software permite al gestor del proyecto adoptar, desarrollar e implementar adecuadamente las actividades de gestión de estos, en función de obtener productos de calidad que satisfagan las necesidades del cliente, manteniendo el equilibrio de plazo y costo del proyecto en virtud de lograr un mejor desempeño del proceso de Ingeniería de Requisitos en la pequeña y mediana empresa de software. El manejo de los riesgos asociados a los requisitos, organizados y gestionados a través de las diferentes taxonomías propuestas en este trabajo, puede constituirse en una útil herramienta para los gestores y equipos de desarrollo.

1.1.4 Gestión de Riesgos en la Universidad de Las Ciencias Informáticas (UCI).

La meta de transformar la informática en un campo próspero internacionalmente y en una de las ramas más productivas para Cuba, requiere inevitablemente lograr el respaldo de un sólido sistema de educación superior. Corresponde a las universidades cubanas, garantizar la formación de profesionales en la informática y la computación, listos para enfrentar la producción de software con responsabilidad y creatividad. La Universidad de las Ciencias Informáticas (UCI) debe alcanzar este propósito con la ejecución de un ambicioso programa curricular vinculado al desarrollo de productos y servicios con la aplicación de las más modernas tecnologías en la docencia.

En entrevistas realizadas durante la investigación al personal involucrado en los proyectos de desarrollo de software en la UCI, específicamente al centro ISEC, se reconoce la carencia de conocimientos relacionados con la GR y por tanto de su aplicación. De un total de 45 entrevistados, el 80% considera que se conocen algunos riesgos que pueden afectar el desarrollo del proyecto, pero el 100% reconoce que no son debidamente identificados utilizando alguna guía formal. En general los entrevistados, conceden gran importancia a la GR para el cumplimiento de los objetivos del proyecto y considera necesario en consecuencia, la creación y aplicación de un modelo con este propósito en la UCI.

En la UCI, se ejecutan acciones integradas al propio proceso docente de pregrado y además a la formación posgraduada. En el primer caso con cursos optativos, tanto en el plano docente como productivo de GR, para el aumento de los conocimientos y las buenas prácticas. Dentro de las asignaturas impartidas en la carrera se destaca en este tema la asignatura Gestión de Software la cual maneja el tema de gestión de riesgos aunque no con la profundidad requerida, pues solo se le dedican cuatro horas

clases. Se tiene conocimiento además de la aplicación del Modelo de Gestión de Riesgos MOGERI al proyecto Sistema de Información Geográfica de la Universidad. Para esto se utilizaron diversas técnicas de GR como análisis, planificación, priorización y control de los riesgos.

1.1.5 Herramientas de Software de gestión de riesgos.

Existen una gran cantidad de herramientas software de gestión de riesgos disponibles en el mercado y que siguen determinadas metodologías. Estas herramientas se enfocan sólo en una categoría de riesgos (TRIMS – Sistema de Mitigación e Identificación de Riesgos Técnicos / Technical Risk Identification and Mitigation System), o están orientadas a compañías maduras que poseen una amplia base de datos organizacional que les permite generar información de categorías propias de riesgos (Risk Trak y Welcome Risk), o bien emplean un mecanismo que no se orienta al uso de clasificaciones de riesgos (ARM – Active Risk Manager).

Se destaca, la herramienta Chinchón –Análisis del riesgo, desarrollada en Java, libre, basada en el modelo MAGERIT, para la gestión de riesgos, pero sólo se enfoca en la fase de análisis cuantitativo del riesgo. La siguiente tabla muestra un análisis de algunas herramientas para la GR y se establece una comparación entre ellas para destacar sus elementos más representativos.

Producto	Proveedor	Descripción	Plataforma
Active Risk Manager (ARM)	Strategic Thought	Herramienta integrada de gestión de riesgos que brinda una solución para la identificación de riesgos mediante la utilización de la información contenida en el WBS de proyecto.	Web Based
Technical Risk Identification and Mitigation System (TRIMS)	Best Manufacturing Practices	Herramienta integrada de gestión de riesgos que emplea ingeniería de conocimientos y que se enfoca en la identificación y medición de riesgos técnicos de proyectos.	Win32
RiskTrak	Risk Services &Technology	Herramienta integrada de gestión de riesgos que brinda una solución para la identificación de riesgos mediante el empleo de bases de datos.	Win32
WelcomRisk	Welcom	Herramienta integrada de gestión de riesgos que brinda una solución para la identificación	Win32

		sistemática de riesgos mediante la utilización de bibliotecas configurables de categorías de riesgos.	
Chinchón – Análisis del riesgo	Free	Chinchón es una herramienta para analizar cuantitativamente el riesgo de un sistema (de información). La herramienta sigue el modelo Magerit 1.0	Java

Tabla 1.1 Herramientas para la Gestión de riesgos.

En la actualidad, las herramientas existentes no sólo se basan en la madurez de las empresas o de sus bases de datos organizacionales, sino que también incluyen patrones de reconocimiento de comportamientos o experiencias anteriores, es decir herramientas que aprovechando las novedosas técnicas de inteligencia artificial, pretenden, facilitan o ayudan a dar solución a la problemática de la gestión de riesgos.

1.2 Antecedentes y situación actual de las herramientas Inteligentes para la gestión de riesgos.

Si bien las ideas fundamentales de la Inteligencia Artificial, se remontan a la lógica y algoritmos de los griegos, y a las matemáticas de los árabes, varios siglos antes de Cristo, el concepto de obtener razonamiento artificial aparece en el siglo XIV. A finales del siglo XIX se obtienen lógicas formales suficientemente poderosas y a mediados del siglo XX, se obtienen máquinas capaces de hacer uso de tales lógicas y algoritmos de solución. Esta ciencia surge en una reunión realizada en el Dartmouth College (Hanover, EEUU) en 1956.

1.2.1 Inteligencia Artificial (IA).

A pesar de que la mayoría de los intentos para definir términos complejos y a la vez ampliamente usados suelen ser inútiles, es positivo al menos esbozar los límites aproximados, en los que enmarcar el concepto de IA. Esta ciencia estudia cómo lograr que las máquinas realicen tareas que, por el momento son realizadas mejor por los humanos.

Según la bibliografía consultada la IA se divide de manera general en dos escuelas de pensamiento:

La inteligencia artificial convencional: Se conoce también como IA simbólico-deductiva. Está basada en el análisis formal y estadístico del comportamiento humano ante diferentes problemas:

- ✓ Razonamiento basado en casos: Ayuda a tomar decisiones mientras se resuelven ciertos problemas además de ser muy importantes requieren de un buen funcionamiento.
- ✓ Sistemas expertos: Infieren una solución a través del conocimiento previo del contexto en que se aplica y se ocupa de ciertas reglas o relaciones.
- ✓ Redes bayesianas: Propone soluciones mediante inferencia probabilística.
- ✓ Inteligencia artificial basada en comportamientos: que tienen autonomía y pueden auto-regularse y controlarse para mejorar.
- ✓ Smart process management: facilita la toma de decisiones complejas, proponiendo una solución a un determinado problema al igual que lo haría un especialista en la actividad.

La inteligencia computacional: También conocida como IA su simbólica-inductiva, implica desarrollo o aprendizaje interactivo (por ejemplo, modificaciones interactivas de los parámetros en sistemas conexionistas). El aprendizaje se realiza basándose en datos empíricos.

En el marco de esta investigación, se tendrá en cuenta sólo la inteligencia artificial convencional, pues sobre esta escuela de pensamiento se basa la solución propuesta ya que la solución se centra en un sistema experto, específicamente un sistema basado en casos. Esta selección está determinada por varios factores. El primero de ellos está dado porque se modela de la manera más sencilla posible el conocimiento, se pueden devolver determinadas respuestas y conocer los detalles del por qué del resultado y por último, simula de manera similar el comportamiento del razonamiento humano, comparando los nuevos casos, conocimiento y experiencia acumulada.

1.2.2 Herramientas Inteligentes para la gestión de riesgos. Situación actual.

Existen diferentes herramientas que permiten, con el uso de técnicas de IA, la gestión de riesgos de alguna manera. Estos sistemas, no se adaptan a las necesidades del centro, pues o los riesgos que gestionan son ajenos a los de ciclo de vida del software o se centran en etapas específicas del análisis de los riesgos. Por otra parte constituyen ejemplos de sistemas o herramientas que tratan riesgos de diferentes tipos, según las características para las cuales se concibieron.

El Sistema Inteligente de Administración de Riesgo SIAR®.

El SIAR® es una solución automatizada para las aduanas que utiliza modelos econométricos que identifican los criterios relacionados con la evaluación del riesgo en el ámbito de las transacciones

comerciales. Desarrollado por Cotecna, con sede en Suiza. Aplica una serie de criterios de manera sistemática para determinar el nivel de riesgo de cada transacción y asigna los niveles de intervención de las aduanas según el nivel de riesgo determinado y los recursos disponibles.

Sistema Inteligente de Gestión de Vulnerabilidades Informáticas (SIGVI) - (Ecuador 2008-2009)

Este proyecto tiene como finalidad el propiciar la construcción de la aplicación computacional SIGVI, un software desarrollado dentro del compromiso institucional de la Fundación Universitaria Iberoamericana (FUNIBER) con el software libre. SIGVI facilita la gestión de alertas por vulnerabilidades para que las organizaciones actúen con tiempo suficiente para poder corregir las vulnerabilidades antes de que el riesgo potencial de su ocurrencia se convierta en una amenaza real o en un desastre.

SIGVI resulta útil a diversas organizaciones que deben superar una serie de vulnerabilidades y que no cuentan con un mecanismo certero, seguro, confiable y constante de recepción, procesamiento, discernimiento y ejecución de las alertas que se reciben. Con SIGVI una organización puede procesar las alertas con eficiencia y eficacia y rápidamente plantear, generar y ejecutar una solución o una actuación que evite la vulnerabilidad probable..

1.2.3 Herramientas inteligentes en Cuba.

En Cuba, las principales manifestaciones de esta ciencia, se centran fundamentalmente en el campo de la medicina, se han experimentado avances en este sentido a través de historias clínicas electrónicas con insospechadas posibilidades en el futuro, sistemas para tratamientos estadísticos como el APUS que es capaz de ofrecer información gerencial para la toma de decisiones, agentes inteligentes para el diagnóstico de trastornos ginecológicos. Además, Cuba cuenta con un Centro de Cibernética Aplicada a la Medicina (CECAM), que concentra esfuerzos en disímiles direcciones de las aplicaciones e investigaciones médicas y con intereses marcados en el campo del intelecto artificial.

1.2.4 Sistemas basados en el conocimiento (SBC).

En la década del 70 se reconoció que los métodos de solución de problemas generales eran insuficientes para resolver los problemas orientados a aplicaciones. Se determinó que era necesario conocimiento específico sobre el problema, limitado a los dominios de aplicación de interés, en lugar de conocimiento general aplicable a muchos dominios. Este reconocimiento condujo al desarrollo de Sistemas Basados en

el Conocimiento (SBC).Los sistemas expertos o basados en el conocimiento, típicos del campo de la IA, no son más que programas para computadoras que simulan las cadenas de razonamiento que realiza un experto para resolver un problema de su dominio. Para conseguirlo, se dota al sistema de un conjunto de principios o reglas que infieren nuevas evidencias a partir de la información previamente conocida.

En términos generales, un SBC puede ser definido como un sistema computarizado que usa conocimiento sobre un dominio para arribar a una solución de un problema de ese dominio (13). Esta solución es esencialmente la misma que la obtenida por una persona experimentada en el dominio del problema cuando se enfrenta al mismo problema.

A los SBC lo caracterizan más rasgos que simplemente el hecho de duplicar el conocimiento y la experticia de un humano para un dominio específico. Tres conceptos fundamentales relativos a los SBC los distinguen de los programas basados en búsqueda general:

- ✓ La separación del conocimiento de cómo éste es usado.
- ✓ El uso de conocimiento muy específico del dominio.
- ✓ Naturaleza heurística, en lugar de algorítmica, del conocimiento empleado.

1.2.5 Ventajas y desventajas de los SBC.

Los SBC tienen ventajas y desventajas cuando se comparan con otras soluciones como el software convencional o expertos humanos.

1.2.5.1 Ventajas:

1. Amplia distribución de experticia escasa.
2. Fácil modificación (conocimiento explícito y accesible).
3. Consistencia en las respuestas (los expertos humanos pueden diferir en sus explicaciones, incluso un mismo experto puede responder de forma diferente en momentos diferentes).
4. Gran accesibilidad (los SBC trabajan las 24 horas todos los días).
5. Preservación de la experticidad (constituye una memoria institucional y poseen la capacidad para adquirir nuevo conocimiento y perfeccionar el que poseen).
6. Solución de problemas que incluyen datos incompletos.
7. Explicación de soluciones (justifica sus conclusiones y explica por qué hace una pregunta).
8. Permite evaluar el efecto de nuevas estrategias añadiendo o modificando conocimiento.

9. Constituye un entrenador en el dominio de aplicación.

1.2.5.2 Desventajas:

1. Las respuestas no siempre son correctas.
2. Conocimiento limitado al dominio de experticidad.
3. Ausencia de sentido común.
4. No reconocen el límite de su conocimiento.

1.2.6 Sistemas Basados en el Conocimiento para la GR.

Un ejemplo de este tipo de sistemas lo constituye el Sistema de Información Normativo Aplicado al Control (SINAC) fruto de la colaboración entre la Intervención General de la Administración del Estado y la Dirección General de Informática Presupuestaria de España. Su función primordial consiste en ayudar en la fiscalización y control de la actividad económica del Sector Público. Otro ejemplo significativo podría ser la Metodología de Análisis y Gestión de Riesgos de los sistemas de Información de las administraciones públicas (MAGERIT), elaborada por un equipo interdisciplinario del comité técnico de Seguridad de los Sistemas de Información y Tratamiento Automatizado de Datos Personales, SSITAD, del Consejo Superior de Informática y que consiste en un método formal para investigar los riesgos que soportan los Sistemas de Información, y para recomendar las medidas apropiadas que deberían adoptarse para controlar estos riesgos; por tanto, permite aportar racionalidad en el conocimiento del estado de seguridad de los Sistemas de Información y en la introducción de medidas de seguridad.

Estos sistemas no se adaptan a las necesidades existentes en el Centro ISEC, puesto que el primero se centra en el control de la actividad económica del Sector Público y el segundo, aunque tiene su basamento en una metodología de GR se centra fundamentalmente en la seguridad de las aplicaciones; por lo que ninguno trata o incluye la GR en proyectos de desarrollo de software.

1.3 Sistemas Basados en el Conocimiento. Razonamiento Basado en Casos RBC.

El Razonamiento Basado en Casos (RBC) representa un método para resolver problemas no estructurados, en el cual el razonamiento se realiza a partir de una memoria asociativa que usa un algoritmo para determinar una medida de semejanza entre dos objetos. Debe destacarse que es una técnica, en la cual la memoria se sitúa como fundamento de la Inteligencia Artificial y más concretamente de los sistemas basados en el conocimiento.

1.3.1 Arquitectura de los sistemas basados en casos.

Un sistema basado en casos tiene dos componentes principales: una base de casos y un resolvedor de problemas. La base de casos contiene las descripciones de los problemas resueltos previamente. Cada caso puede describir un episodio particular o una generalización de un conjunto de episodios relacionados. En el estilo de solución de problemas se recupera un caso semejante al nuevo y la solución del problema recuperado se propone como solución potencial del nuevo problema. Esto se deriva de un proceso de adaptación en el cual se adecua la vieja solución a la nueva situación.

1.3.2 Herramientas que utilizan Sistemas Basados en Casos.

Un ejemplo relevante en Cuba, es el Sistema Inteligente de Selección de Información (SISI), elaborado en el Centro de Cibernética Aplicada a la Medicina (CECAM), que implementa el razonamiento basado en casos orientado a tareas de diagnóstico médico. Este sistema o herramienta recoge elementos actualizados que son capaces de simular en una mayor escala el proceso de enseñanza y aprendizaje en un entorno Web donde se integran la programación para el cliente (Javascript) y para el servidor (PHP); bases de datos en MYSQL e información sobre todo lo relacionado con sistemas tutoriales inteligentes, inteligencia artificial distribuida y multimedia.

Además, se conoce de una herramienta de apoyo a las revisiones de proyectos de software utilizando conocimiento basado en casos, elaborada en el Centro de estudios de Ingeniería y Sistemas CEIS, del Instituto Politécnico “José Antonio Echeverría” CUJAE, en Cuba.

En este trabajo se presenta una propuesta de estructura de almacenamiento y de función de semejanza entre proyectos de software para aplicar el Razonamiento Basado en Casos (RBC), en las Revisiones a los proyectos. Estas revisiones se enfocan en la etapa de Requisitos, debido a la importancia de la detección de los defectos en las etapas tempranas del desarrollo de los proyectos. Se enuncian los algoritmos propuestos para considerar la semejanza entre cada par de proyectos. Además se describen las características generales de una herramienta automatizada que implementa el RBC de forma que se pueda acumular la experiencia en la detección de defectos en los proyectos de software y emplearla en la ejecución de nuevas revisiones.

1.4 Metodologías de desarrollo de software.

Una metodología de desarrollo de software es un conjunto de procedimientos, herramientas y un soporte documental que ayuda a los desarrolladores a realizar nuevo software (14). Estas metodologías se basan en una combinación de los modelos de procesos genéricos (cascada, incremental, evolutivo, etc.) y definen con precisión los roles, actividades y artefactos, además de las técnicas y prácticas recomendadas.

Los requisitos de un software a otro son tan variados y cambiantes, que ha dado lugar a que exista una gran variedad de metodologías para la creación del software. Se podrían clasificar en dos grandes grupos:

- Las metodologías orientadas al control de los procesos, estableciendo rigurosamente las actividades a desarrollar, herramientas a utilizar y notaciones que se usarán. Estas metodologías son llamadas Metodologías Pesadas.
- Las metodologías orientadas a la interacción con el cliente y el desarrollo incremental del software, mostrando versiones parcialmente funcionales del software al cliente en intervalos cortos de tiempo, para que pueda evaluar y sugerir cambios en el producto según se va desarrollando. Estas son llamadas Metodologías ligeras/ágiles.

Para la realización de la etapa más compleja de este trabajo de diploma, diseño e implementación, fue seleccionada una metodología ágil de desarrollo de software, basándose en las características propias del resultado a obtener, el tiempo del que se dispone, los recursos, y el equipo de desarrollo, el cual está integrado solo por dos personas, y explotando las principales ideas planteadas por las metodologías de este tipo que valoran al individuo y las iteraciones del equipo más que a las herramientas o los procesos utilizados, se le da más importancia a crear un producto software que funcione que escribir mucha documentación, resaltando la capacidad de respuesta ante un cambio realizado que el seguimiento estricto de un plan. Con este propósito, de las metodologías agiles existentes se seleccionó Desarrollo Basado en Funcionalidades / Feature Driven Development (FDD).

1.4.1 Desarrollo Basado en Funcionalidades / Feature Driven Development (FDD).

FDD es una metodología ágil para el desarrollo de sistemas a corto plazo. Se basa en un proceso de iteraciones cortas de aproximadamente dos semanas que producen un software funcional que el cliente y la dirección de la empresa, centro o proyecto pueden ver y monitorear. Estas iteraciones se deciden en

base a las funcionalidades que son pequeñas partes del software con significado para el cliente. Ayuda a contrarrestar situaciones como el exceso en el presupuesto, fallas en el programa o el hecho de entregar menos de lo deseado y con su utilización se obtienen resultados periódicos y tangibles.

El proceso consiste de cinco pasos secuenciales durante los cuales se diseña y se construye el sistema:

Fig. 1.3 Procesos FDD.

Desarrollo de un modelo general: Cuando comienza el desarrollo, los expertos del dominio están al tanto de la visión, el contexto y los requerimientos del sistema a construir. Se divide el dominio global en áreas que son analizadas detalladamente. Los desarrolladores construyen un diagrama de clases o de objetos por cada área y se construye un modelo global del sistema.

Construcción de una lista de funcionalidades: Una funcionalidad constituye pequeñas partes del software con significado para el cliente. Se elabora una lista de funcionalidades que resuma la funcionalidad general del sistema; la lista es elaborada por los desarrolladores y es evaluada por el cliente y se divide en subconjuntos según la afinidad y la dependencia de las funcionalidades, la lista es finalmente revisada por los usuarios y los responsables para su validación y aprobación.

Planeación por funcionalidad: Se procede a ordenar los conjuntos de funcionalidades conforme a su prioridad y dependencia, y se asigna a los programadores jefes.

Diseño por funcionalidad y Construcción por funcionalidad: Se **selecciona** un conjunto de funcionalidades de la lista, se procede a diseñar y construir la funcionalidad mediante un proceso iterativo. Una iteración puede tomar de unos pocos días a un máximo de dos semanas. El proceso iterativo incluye inspección de diseño, codificación, pruebas unitarias, integración e inspección de código.

La metodología FDD no hace énfasis en la obtención de los requerimientos sino en cómo se realizan las fases de diseño y construcción. Al comenzar el primero de sus procesos ya los expertos del dominio están al tanto de la visión, el contexto y los requerimientos del sistema a construir. En el sitio oficial de esta metodología, en un acápite que detalla su utilización en aplicaciones Web (http://www.featuredrivendevelopment.com/node/550) se plantea, citando a Fred Brooks, que FDD no cubre todos los aspectos del desarrollo. Hay muchos que deben manejarse fuera de dicha metodología para dejar documentado los resultados que como parte del proceso de desarrollo se van obteniendo. Tal es el caso del levantamiento de requisitos, diseño de interfaz, pruebas e implementación. Por lo que sugiere el apoyo en otras metodologías, para calzar estas insuficiencias.

Sin embargo, las facilidades que brinda en el diseño e implementación, así como las ventajas asociadas al tamaño del equipo de desarrollo, la poca carga de trabajo que genera referente a los artefactos en estas fases, en las que es la obtención del código la tarea que requiere mayor esfuerzo, hace que sea la más adecuada. Por otra parte, su estrecho vínculo con Grails, el framework de desarrollo utilizado, posibilita su perfecta integración con el mismo, pues funcionalidades complejas como las de tipo CRUD, se vuelven muy sencillas con el uso de las bondades de Grails. Tal es el caso del"scaffolding" quien provee facilidades para este tipo de funcionalidad. Además de las características del lenguaje Groovy y los frameworks incorporados a Grails agilizan el desarrollo y definen una estrategia de "Convención sobre Configuración", con la premisa de "No te repitas" posibilitando la reutilización del código y la obtención de resultados tangibles en poco tiempo.

1.4.2 Proceso unificado de Rational / Rational Unified Process (RUP).

RUP es una metodología de desarrollo de software que intenta integrar todos los aspectos a tener en cuenta durante todo el ciclo de vida del software, con el objetivo de hacer abarcables tanto pequeños como grandes proyectos de software. Además Rational proporciona herramientas para todos los pasos del desarrollo así como documentación en línea para sus usuarios. Las características principales están dadas en que es un proceso iterativo e incremental, centrado en la arquitectura y dirigido por casos de usos. Además su desarrollo se basa en componentes, tiene un único lenguaje para modelar constituyendo un proceso integrado (15).

La siguiente imagen muestra detalladamente los elementos que forman parte de la metodología RUP:

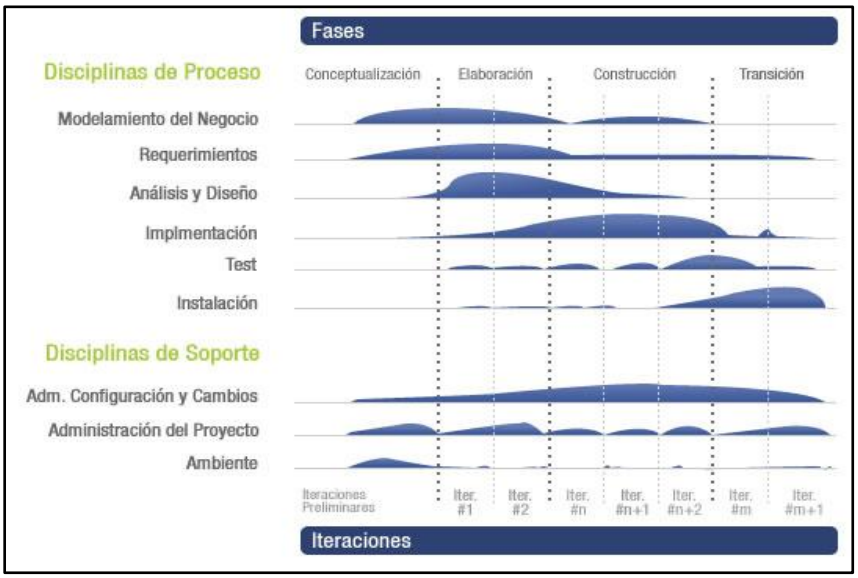


Fig 1.4 Proceso unificado de Rational / Rational Unified Process (RUP).Fases y flujos de trabajo.

En el desarrollo de esta tesis de grado se utiliza la metodología RUP como apoyo a la metodología FDD, seleccionada para guiar el diseño y la implementación. Se generarán algunos artefactos correspondientes al modelamiento del negocio, requerimientos, implementación y pruebas para garantizar que quede documentada la mayor cantidad de detalles posibles, para el posterior desarrollo del sistema. Estos artefactos serán:

- **Modelo de dominio:** El Modelo de Dominio es una representación visual estática del entorno real objeto del proyecto. Cuyo objetivo es ayudar a comprender los conceptos que utilizan los usuarios y los conceptos con los que trabajará la aplicación (16). Se determinó su utilización porque no hay una definición clara de los procesos relacionados con el sistema a desarrollar.
- **Lista de Requisitos:** Los requisitos se presentan en una lista ordenada, categorizada según su ámbito y la influencia y prioridad respecto al entorno de aplicación del proyecto. Esta lista es elaborada siguiendo las recomendaciones de los estándares definidos por la organización a la que pertenece (17).

1.5 Lenguaje de modelado.

El Lenguaje Unificado de Modelado (UML) es un lenguaje de modelado visual que se usa para especificar, visualizar, construir y documentar artefactos de un sistema de software. Captura decisiones y conocimiento sobre los sistemas que se deben construir. Se usa para entender, diseñar, configurar,

mantener, y controlar la información sobre tales sistemas (18). Está pensado para usarse con todos los métodos de desarrollo, etapas del ciclo de vida, dominios de aplicación y medios. Ha sido diseñado por la combinación de estándares de tres metodologías procedentes de tres desarrolladores J. Rumbaugh, G. Booch e I. Jacobson.

UML ayuda al usuario a entender la tecnología en realidad y la posibilidad de mostrar una idea antes de invertir en proyectos que no estén seguros en su desarrollo. Entre las propiedades que tiene este importante lenguaje se encuentra que modela estructuras complejas, soporta estructuras orientadas a objetos, como objetos, clases, componentes y nodos, es un lenguaje distribuido y adaptado a las conectividades actuales y futuras y se define el comportamiento del sistema: casos de uso y diagramas de interacción.

1.5.1 Visual Paradigm.

Visual Paradigm es una herramienta que es soportada por cualquier sistema operativo. Es una herramienta visual de ingeniería de software para el modelado, que tiene un entorno de trabajo que muestra colección de menús, barra de herramientas y ventanas que permite realizar diagramas. Puede ser utilizada por una gran variedad de usuarios como analistas de sistemas, analistas de negocios e ingenieros de software. Provee soporte para la generación de código y la ingeniería inversa. Se integra con algunas herramientas como: Eclipse, Netbeans, Jbuilder, Oracle, entre otras.Se utilizará Visual Paradigm Suite versión 3.4.

1.6 Plataforma de desarrollo

Es un entorno común donde se desarrolla la programación de un grupo de aplicaciones.

1.6.1 Plataforma Java.

La plataforma Java es un entorno capaz de ejecutar aplicaciones desarrolladas usando el lenguaje de programación Java u otros lenguajes que compilen a bytecode. Provee una máquina virtual que se encarga de la ejecución de aplicaciones, un conjunto de bibliotecas estándares que ofrecen funcionalidades comunes así como una interfaz para la programación de aplicaciones haciendo uso del lenguaje Java. Está compuesta por una gran cantidad de tecnologías como Java Edición Estándar (JSE), Java Edición Micro (JME) y Java Edición Empresarial (JEE). Esto permite que los usuarios interactúen con la máquina virtual de Java y exploten el conjunto de funcionalidades brindadas por la misma, así como sus

bibliotecas. Además las aplicaciones sobre esta plataforma pueden también ser desarrolladas en la web. Para el desarrollo de aplicaciones, se utiliza un conjunto de herramientas conocidas como JDK (Java Development Kit, o herramientas de desarrollo para Java).

1.7 Lenguaje de Programación.

1.7.1 Groovy.

Groovy 1.0 apareció el 2 de enero de 2007. Después de varias versiones beta y otras tantas candidatas a release, el 7 de diciembre de 2007 apareció la versión Groovy 1.1 que finalmente fue renombrada a Groovy 1.5 con el fin de notar la gran cantidad de cambios que ha sufrido con respecto a la versión 1.0. En diciembre de 2009 se publicó la versión 1.7.

Es un lenguaje orientado a objetos, nacido con la misión de acoplarse de forma efectiva con Java, trabajar en conjunto con la Máquina Virtual de Java JVM y soportar los tipos de datos estándar, pero añadiendo características dinámicas y sintácticas presentes en otros lenguajes como Python, Smalltalk o Ruby (19). El código fuente Groovy se compila a bytecodes, y es posible instanciar objetos Java desde Groovy y viceversa. Lo que Groovy aporta es una sintaxis que aumenta enormemente la productividad y un entorno de ejecución que nos permite manejar los objetos de formas que en Java serían extremadamente complicados. Todo lo anterior, unido a que la mayor parte de código escrito en Java es totalmente válido en Groovy hace que este lenguaje sea de muy fácil adopción.

1.8 Entorno de desarrollo Integrado.

Los Entornos de Desarrollo Integrados / Integrated Developtment Enviroment (IDEs) son un conjunto de herramientas para el programador que suelen incluir en una misma suite, un buen editor de código, administrador de proyectos y archivos, enlace transparente a compiladores y debuggers e integración con sistemas controladores de versiones o repositorios.

1.8.1 NetBeans.

NetBeans es un entorno de desarrollo hecho principalmente para el lenguaje de programación Java, pero puede servir para cualquier otro lenguaje de programación. Es un producto libre y gratuito sin restricciones de uso. Incluye soporte para Groovy y Grails desde la versión 6.5, incluyendo un tipo de proyecto y

posibilidad de ejecutar comandos Grails desde el propio IDE. El soporte para Groovy incluye resaltado de sintaxis y autocompletado de nombres de variables y métodos, y sigue evolucionando a muy buen ritmo.

Se utilizará la versión 6.9.

1.9 Framework.

Constituye una arquitectura informática unificada que abarca, Interfaz de usuario programada API, aplicaciones, programación interactiva, sensor de manejo y gestión de la información. Son diseñados con la intención de facilitar el desarrollo de software, permitiendo a los diseñadores y programadores centrarse en la identificación de requisitos de software y disminuir los tediosos detalles de bajo nivel de proveer un sistema funcional. Entre sus objetivos se encuentra acelerar el desarrollo, reutilizar código existente y promover buenas prácticas de desarrollo como el uso de patrones.

1.9.1 Grails.

Grails es un framework para aplicaciones web libre desarrollado sobre el lenguaje de programación Groovy (el cual a su vez se basa en la Plataforma Java). Grails pretende ser un marco de trabajo altamente productivo siguiendo paradigmas tales como convención sobre configuración, proporcionando un entorno de desarrollo estandarizado y ocultando gran parte de los detalles de configuración al programador.

Grails fue conocido como 'Groovy on Rails' (el nombre cambió en respuesta al pedido de David Heinemeier Hansson, fundador de Ruby on Rails). Se inició en julio de 2005, con la versión 0.1 29 de marzo de 2006. La versión 1.0 es anunciada el 18 de febrero de 2008. En diciembre de 2009 se publicó la versión 1.2, y en mayo de 2010 la versión 1.3.

Se utilizará la versión 1.3.5 de Grails.

1.9.1.1 Características.

Grails se ha desarrollado con una serie de objetivos:

- Ofrecer un framework web de alta productividad para la plataforma Java.
- Reutilizar tecnologías Java ya probadas como Hibernate y Spring bajo una interfaz simple y consistente.

- Ofrecer un framework consistente que reduzca la confusión y que sea fácil de aprender.
- Ofrecer documentación para las partes del framework relevantes para sus usuarios.
- Proporcionar lo que los usuarios necesitan en áreas que a menudo son complejas e inconsistentes:
    - Framework de persistencia potente y consistente.
    - Patrones de visualizaciones potentes y fáciles de usar con GSP (Groovy Server Pages).
    - Bibliotecas de etiquetas dinámicas para crear fácilmente componentes web.
    - Buen soporte de Ajax que sea fácil de extender y personalizar.
- Proporcionar aplicaciones ejemplo que muestren la potencia del framework.
- Proporcionar un entorno de desarrollo orientado a pruebas.
- Proporciona un entorno completo de desarrollo, incluyendo un servidor web y recarga automática de recursos.

Grails tiene tres características que intentan incrementar su productividad comparándolo con los framework Java tradicionales:

- Inexistencia de configuración XML.
- Entorno de desarrollo preparado para funcionar desde el primer momento.
- Funcionalidad disponible mediante métodos dinámicos.

El framework web de Grails se ha diseñado según el paradigma Modelo Vista Controlador.

1.10 PostgreSQL.

PostgreSQL es un sistema de gestión de bases de datos objeto-relacional, distribuido bajo licencia BSD y con su código fuente disponible libremente. Soporta gran parte del SQL estándar y muchas funcionalidades como son: consultas complejas, disparadores, vistas, integridad transaccional, así como el control de versiones concurrentes que es una estrategia de almacenamiento que permite trabajar con grandes volúmenes de datos. Ha sido diseñado y creado para tener un mantenimiento y ajuste mucho menor que otros productos, conservando todas las características de estabilidad y rendimiento. Cuenta con interfaces para distintos lenguajes, como son: C/C++, Java, Delphi, Python, Perl, PHP, entre otros.

Era necesaria para el presente trabajo la utilización de un gestor de base de datos potente para el almacenamiento de un alto volumen de información. Dentro de los que cumplen con las características

necesarias para la solución de la problemática del presente trabajo se encuentran: Oracle, Microsoft SQL Server y PostgreSQL, por lo que fue seleccionado PostgreSQL por ser el único software libre de todos.. Se utilizará PostgreeSQL 8.2.

1.11 CSS.

CSS es un lenguaje de hojas de estilos creado para controlar el aspecto o presentación de los documentos electrónicos definidos con HTML y XHTML. Estos estilos definen la forma de mostrar los elementos en el documento de este tipo. Además es interpretado por las páginas o vistas GSP que brinda el framework. CSS es la mejor forma de separar los contenidos y su presentación y es imprescindible para crear páginas web complejas.

La versión de CSS utilizada en este trabajo será CSS3, interpretadas por navegadores como Safari Chrome entre otros, así como Firefox 1.3.X y 1.4.X.

Conclusiones.

En el presente capítulo se realizó un estudio del estado del arte del proceso de gestión de riesgos y los sistemas que actualmente se enfocan en esa área, identificándose a MOGERI como modelo de gestión de riesgos en el cual basar la investigación. Este estudio arrojó como resultado que los sistemas estudiados no satisfacen las necesidades de ISEC en cuanto a la gestión de riesgos, por lo cual se concluyó que la solución adecuada es el desarrollo de un sistema inteligente de mitigación de riesgos que permita utilizar el conocimiento de expertos en el área. Se seleccionó FDD para guiar el proceso de desarrollo de software y en cuanto a la implementación se determinó utilizar la plataforma Java y el IDE NetBeans 6.9 con el lenguaje de programación Groovy en su versión 1.7. Para facilitar el desarrollo de la propuesta se usa el framework Grails 1.3 el cual está basado en el lenguaje Groovy. Finalmente se seleccionó PostgreSQL 8.2 como sistema gestor de base de datos para el almacenamiento y manejo de los mismos.

CAPÍTULO 2: CARACTERÍSTICAS DEL SISTEMA.

Introducción.

En el presente Capítulo se describe la propuesta de solución al problema. Se describe el objeto de automatización. Además se detalla el modelo de dominio y se enumeran los requisitos funcionales y no funcionales, así como una descripción de los casos de uso del sistema y los diagramas relacionados.

2.1 Objeto de Estudio.

El objeto de estudio sobre el cual se enmarca este trabajo de diploma son las herramientas inteligentes para la toma de decisiones.

2.2 Problema y situación problemática.

Tras la reciente creación del Centro de Informatización de la Seguridad Ciudadana en la Universidad de Ciencias Informáticas, el cual, tiene como labor fundamental el desarrollo de proyectos pertenecientes a ese perfil. Luego de entrevistas realizadas a diferentes líderes de proyectos, se logró identificar que existen muchas actividades en las que se emplean, según las características de cada proyecto, tiempo y esfuerzo en realizarlas y resolverlas. Tal es el caso de los procesos de identificación y tratamiento de los riesgos, que no se realizan de manera adecuada durante todo el ciclo de desarrollo de software. Se pudo identificar que existe dualidad de funciones y responsabilidades entre los integrantes del equipo de desarrollo. No se le dedica el tiempo adecuado a la producción y muchas veces no se cuenta con los recursos necesarios. Existiendo en otros casos una mala planificación de los mismos.

De manera general, el personal que integra el equipo de desarrollo no cuenta con la preparación ni la experiencia adecuada. No se dispone con un tiempo de capacitación y preparación previo, y si existe no es el que se requiere. Lo antes expuesto unido a la inadecuada gestión del conocimiento provoca inestabilidad en la fuerza de trabajo. Aparejado a esto, surgen problemas de cambios de los requisitos por parte del cliente, además de demora del mismo en la revisión de la documentación del proyecto.

Esta ineficiente gestión de riesgos, es una de las consecuencias fundamentales en el atraso de los cronogramas de los proyectos del centro, afectando todas las fases del desarrollo del software, lo cual incide de forma negativa en el cumplimiento de los objetivos propuestos según el alcance definido en cada

compromiso del Centro y la Universidad, sobre todo por las necesidades de cumplirlos en el tiempo establecido.

2.3 Objeto de automatización.

Actualmente el centro ISEC no cuenta con una herramienta que permita, ayude o agilice el proceso de gestión de riesgos en el ciclo de vida de los proyectos de desarrollo de software. Este trabajo tiene como objetivo proveer a los proyectos del centro de una herramienta inteligente para la Mitigación de Riesgos, y de esta forma automatizar este complejo proceso de vital importancia en todo proyecto; con el fin de identificar los riesgos más comunes a ocurrir, según las características del proyecto en cuestión, así como posibles estrategias para su tratamiento.

La identificación y mitigación de riesgos a lo largo del ciclo de vida del software en el centro ISEC y la UCI, es realizado por parte de la dirección de cada proyecto, analizando y calculando según diferentes criterios de estimación y con el uso de metodologías de GR en cada proyecto. Este es un proceso largo y complejo, que se realiza de forma manual y que incluye el seguimiento y control de muchos de los integrantes del equipo de desarrollo. El Sistema Inteligente de Mitigación de Riesgos, permitirá automatizar una parte importante de este proceso. Pues brinda la posibilidad de que dadas características relevantes del proyecto, se logre obtener de forma automática los riesgos potenciales a incidir a lo largo del ciclo de desarrollo así como una posible mitigación de los mismos. Lo anterior, permitirá tener una visión adelantada de lo que puede ocurrir además de posibles ideas para enfrentarlo.

2.4 Información que se maneja.

La información que se maneja en este trabajo está relacionada con los riesgos más comunes que pueden presentarse o se han presentado en diferentes proyectos de desarrollo de software en la Universidad de las Ciencias Informáticas, así como algunas acciones o estrategias realizadas para el tratamiento de estos riesgos. Además, se utilizará información referente a las características propias de los proyectos del centro ISEC.

2.5 Descripción de la solución propuesta.

El Sistema Inteligente de Mitigación de Riesgos se basa en la información que introduce el usuario referente a la descripción de un proyecto. Utiliza técnicas de Inteligencia Artificial, específicamente de un

sistema basado en el conocimiento, y dentro de este, el desarrollo de un sistema experto a partir del razonamiento basado en casos, para distinguir por rasgos la descripción dada. Luego los compara con los almacenados en la base de casos con que cuenta el sistema experto y devuelve un grupo de ítems semejantes. A partir de ellos y con el uso de un razonador (el cual forma parte también de la arquitectura de un sistema basado en casos) devuelve una respuesta que se asemeje a la obtenida según la similitud del caso entrado con el o los que se tiene en la base de casos, y a partir de esta información, obtener las posibles acciones a realizar o estrategias a tomar. Y por último, almacenar el caso si no se encuentra o si es posible obtener información de él así como si puede convertirse en uno nuevo por sus propias características.

2.6 Modelo de dominio.

Se detalla el modelo de dominio realizado para darle solución a las funcionalidades. Se especifica además con un lenguaje más comprensible tanto para los clientes como para los desarrolladores cada uno de los conceptos y sus relaciones.

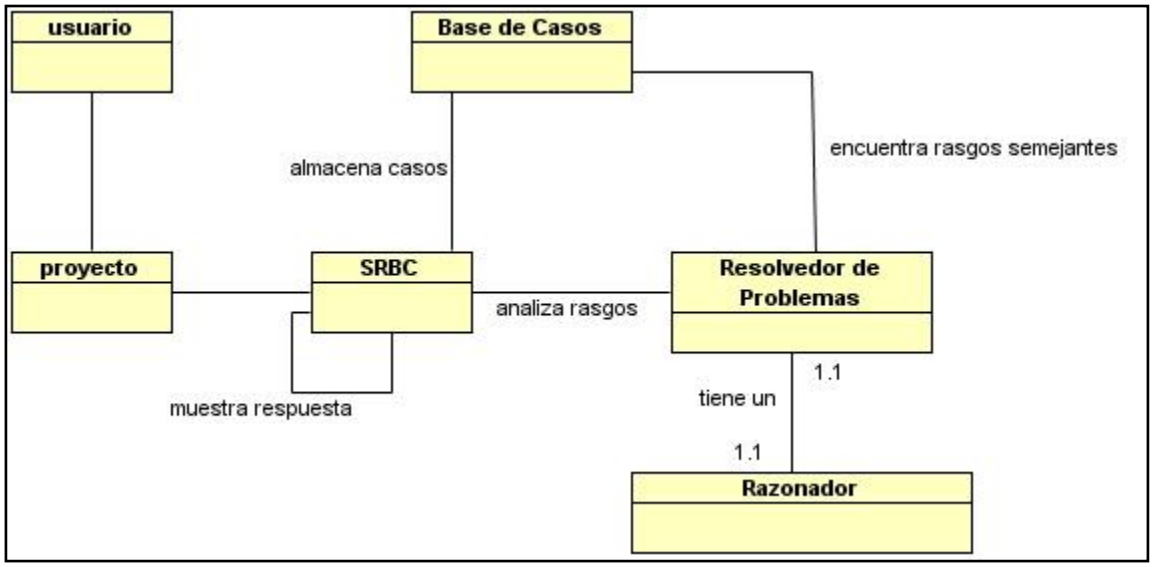


Fig.2.1 Modelo de Dominio. Sistema Inteligente para la Mitigación de Riesgos.

Usuario: Encargado de introducir la descripción o información del proyecto a analizar.

Proyecto: Conjunto de características correspondientes a un proyecto entrado por el usuario.

SRBC: Sistema de Razonamiento Basado en Casos. Encargado de manejar y visualizar la información o conjunto de características entradas por el usuario, así como la respuesta y el almacenamiento de casos.

Base de Casos: Conjunto de casos y correspondientes rasgos, previamente almacenados para el análisis de las semejanzas con los rasgos obtenidos de la descripción insertada.

Resolvedor de Problemas: Se encarga de manejar los rasgos inicialmente insertados. Contiene un razonador de casos.

Razonador: Encargado de a partir de los ítems semejantes que provee el recuperador de casos, evaluar el resultado del mismo.

2.7 Especificación de las funcionalidades.

2.7.1 Requisitos Funcionales.

Los requerimientos funcionales son condiciones o capacidades que el sistema debe cumplir. A continuación se enumeran los que han sido identificados.

RF1. Autenticar usuario.

- Usuario.
- Contraseña.

RF2. Insertar Usuario.

- Nombre.
- Nombre de usuario
- Rol.
- Proyecto.
- Contraseña.

RF3. Modificar usuario.

- Datos a modificar (Remitirse a **RF1**).

RF4. Eliminar usuario.

- Usuario.

RF5. Listar Usuarios.

RF6. Insertar Proyecto.

- Nombre de proyecto.
- Líder del Proyecto.
- Facultad.
- Cantidad de profesores.
- Cantidad de profesores en adiestramiento.
- Cantidad de profesores prestando servicios.
- Cantidad de estudiantes de 2do.
- Cantidad de estudiantes de 3ro.
- Cantidad de estudiantes de 4to.
- Cantidad de estudiantes de 5to.
- Cantidad total de PCs.
- Cantidad disponible de PCs.
- Tiempo por PC.
- Familiaridad con el modelo de proyecto.
- Dominio del negocio.
- Experiencia en la plataforma.
- Nivel de conocimiento del lenguaje.
- Nivel de conocimiento de los desarrolladores.
- Experiencia del equipo de desarrollo.
- Motivación del equipo de desarrollo.
- Nivel de dificultad del lenguaje.
- Cohesión del equipo.
- Experiencia con las herramientas.
- Complejidad del producto.
- Cantidad de tiempo promedio dedicado al proyecto.
- Información que maneja.

- Estabilidad de los requerimientos.
- Cantidad de funcionalidades simples.
- Cantidad de funcionalidades medias.
- Cantidad de funcionalidades complejas.
- Cronograma.
- Tipo cliente.
- Relación y vinculación del cliente.
- Nivel de acceso del cliente.
- Usuario que inserta el proyecto.

RF7. Modificar Proyecto.

- Modificar cualquiera de los datos insertados (Remitirse a **RF6**).

RF8. Eliminar Proyecto.

- ID_proyecto.

RF9. Listar Proyectos.

RF10. Identificar riesgos.

- Proyecto (Remitirse a **RF6**).
- Riesgo (Remitirse a **RF17**).
- Rasgo (Remitirse a **RF13**).
- Clasificación (Remitirse a **RF6**).

RF11. Mostrar resultados.

- Riesgos identificados.

RF12. Aprender caso.

- Rasgos.
- Caso evaluado.

RF13. Insertar Rasgo.

- Cantidad de profesores.

- Cantidad de profesores en adiestramiento.
- Cantidad de profesores prestando servicios.
- Cantidad de estudiantes de 2do.
- Cantidad de estudiantes de 3ro.
- Cantidad de estudiantes de 4to.
- Cantidad de estudiantes de 5to.
- Cantidad total de PCs.
- Cantidad disponible de PCs.
- Tiempo por PC.
- Familiaridad con el modelo de proyecto.
- Dominio del negocio.
- Experiencia en la plataforma.
- Nivel de conocimiento del lenguaje.
- Nivel de conocimiento de los desarrolladores.
- Experiencia del equipo de desarrollo.
- Motivación del equipo de desarrollo.
- Nivel de dificultad del lenguaje.
- Cohesión del equipo.
- Experiencia con las herramientas.
- Complejidad del producto.
- Cantidad de tiempo promedio dedicado al proyecto.
- Información que maneja.
- Estabilidad de los requerimientos.
- Cantidad de funcionalidades simples.
- Cantidad de funcionalidades medias.
- Cantidad de funcionalidades complejas.
- Cronograma.
- Tipo cliente.
- Relación y vinculación del cliente.
- Nivel de acceso del cliente.

- Riesgos asociados.

RF14. Eliminar Rasgo.

- Id_Rasgo.

RF15 Listar Rasgos.

RF16. Modificar Rasgo.

- Id_Rasgo.
- Campos a modificar.

RF17. Insertar Riesgos.

- Nombre de riesgo.
- Clasificación.
- Posible Mitigación.

RF18. Modificar Riesgo.

- Datos a modificar (Remitirse a **RF17**).

RF19. Eliminar Riesgo.

- ID_riesgo.

RF20. Listar Riesgos.

RF21. Insertar Clasificación.

- Nombre de clasificación

RF22. Modificar Clasificación.

- Datos a modificar (Remitirse a **RF21**).

RF23. Eliminar Clasificación.

- ID_ clasificación

RF24. Listar Clasificaciones.

RF25. Insertar Evaluación.

- Evaluación.
- Proyecto.
- Comentario.

RF26. Modificar Evaluación

- Datos a modificar (Remitirse a **RF25**).

RF27. Listar Evaluación.

RF28. Eliminar Evaluación.

- Evaluación.

2.7.2 Requisitos no Funcionales.

Los requisitos no funcionales son fundamentales en el éxito del producto. Normalmente están vinculados a requerimientos funcionales. Se han seleccionado los siguientes requisitos no funcionales.

Requisitos de Apariencia o interfaz externa:

- El sistema brindará una interfaz amigable para sus usuarios, con predominio de color azul en diversas tonalidades y gris. Las letras serán legibles con formato Verdana, Arial, Helvetica, Sans-Serif, de 15 píxeles azules para los links y 13 píxeles negras para los textos.

- Los errores que sean visibles al usuario, indicarán las posibles causas y sus soluciones

Requisitos Software.

- La PC donde se ejecute la aplicación debe tener instalado Firefox 3.X o superior, Chrome 5.X o superior o Safari, debido a que los estándares de CSS3 no son reconocidos por muchas de las versiones de Internet Explorer (6.X, 7.X). Debe tener alojado el SGBD PostgreSQL 8.2 o contar con dicho servidor externo.

- En caso de ser externo el servidor de base de datos, debe tener instalado Tener instalado PostgreSQL 8.2

- La Computadora que sirva como servidor de la aplicación, debe tener instalado el Apache Tomcat como servidor Web, preferentemente una versión superior a 6.X y la Máquina Virtual de Java.

Requisitos de Hardware.

- Es necesario contar con una computadora con un Microprocesador con velocidad superior a los 1.6 GHz, y con una memoria RAM superior a los 256 Mega Bytes, si en ella no se encuentra el servidor de Base de Datos.

- Si el servidor de base de datos es externo, debe estar alojado en una computadora con 1 GB de RAM, 500 HDD y tener la red disponible con una tarjeta de 10/100/1000 Mb/s.

Requisitos de diseño.

- El sistema implementado será una aplicación web.

- El sistema se implementará usando la plataforma JAVA.

- El sistema estará basado en un estilo arquitectónico en capas. Las capas estarán distribuidas de la siguiente manera:

    - Capa web: Encapsula las vistas y la lógica de presentación. En la lógica de presentación se maneja todo el flujo web utilizando la implementación del patrón Modelo-Vista-Controlador que brinda el framework Grails. Las vistas son los recursos que junto al modelo generado por los controladores le permiten al cliente visualizar la información.

    - Capa de dominio: Contiene las entidades que persistirán en la base de datos, además encapsula toda la lógica relacionada con el dominio que abarca el negocio. Esta brinda las funcionalidades mediante fachadas de servicio que son utilizadas por los controladores en la capa web y se implementan procesos de negocio que perduran en el tiempo.

- Capa de datos: Encapsula la lógica de acceso a datos abstrayendo a las capas superiores del mecanismo de acceso a la base de datos. Dicha abstracción se realiza a través del framework de persistencia GORM, que implementa Grails, éste agrega métodos de forma dinámica a las entidades del dominio para que estas sean capaces de interactuar con el gestor de base de datos por sí mismas.

Ayuda y documentación en línea.

- Se dispondrá de una ayuda, que será accedida desde la aplicación a través del navegador web, para facilitar a los usuarios la correcta utilización del sistema las 24 horas del día, los 7 días de la semana.

Requisitos de Seguridad.

- Seguridad y control a nivel de usuarios y contraseñas, garantizando el acceso de los mismos sólo a los niveles establecidos de acuerdo a la función que realicen. Las contraseñas sólo podrán ser cambiadas por el propio usuario o por el administrador informático del sistema.

2.8 Diagrama de Casos de uso del Sistema.

2.8.1 Definición delos actores del sistema.

Actores	Justificación
Usuario	Es un actor del sistema el cual debe registrase, autenticarse, insertar la descripción de un proyecto, identificar los riesgos posibles a incidir así como evaluar el resultado de la identificación de los riesgos.
Consultor	Es un actor del sistema especializado del actor Usuario. Es el encargado insertar, actualizar y eliminar riesgos, rasgos, clasificaciones además de ser el protagonista del aprendizaje del sistema pues también se encarga de aprender los nuevos casos.
Administrador	Es un actor del sistema especializado del actor Consultor. Es el encargado de insertar, modificar y eliminar los usuarios del

	sistema además de realizar todas las acciones posibles de usuario y consultor.

Tabla 1.2 Descripción de los actores del sistema.

2.8.2 Diagrama de casos de uso del sistema a automatizar.

Los diagramas de casos de uso se emplean para modelar la vista de casos de uso de un sistema. La mayoría de las veces, esto implica modelar el contexto del sistema, subsistema o clase, o el modelado de los requisitos de comportamiento de esos elementos.

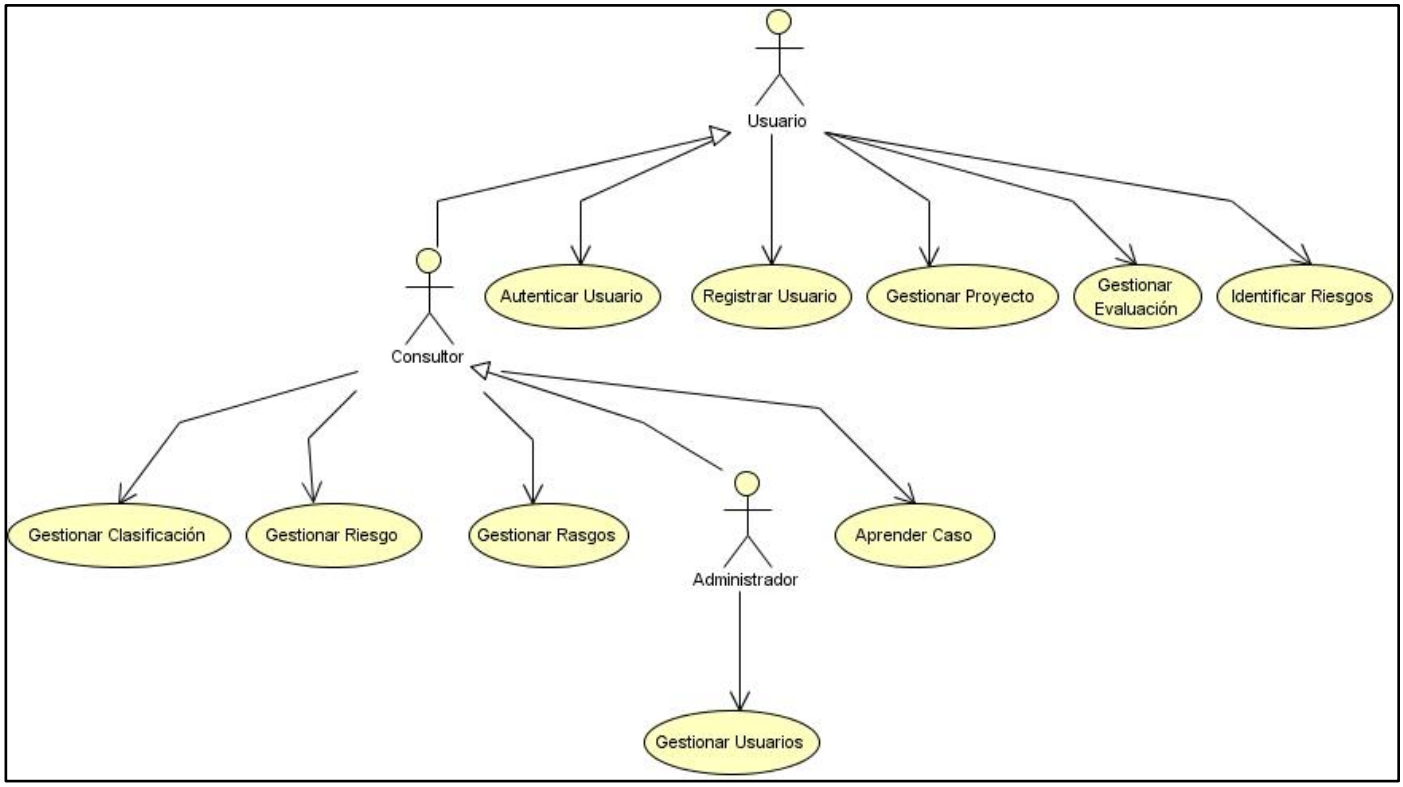


Fig. 2.2 Diagrama de Caso de Uso de Sistema.

2.8.3 Descripción de los Casos de uso del Sistema para el Sistema Inteligente de Mitigación de Riesgos.

En el **Anexo 1** se muestran las descripciones textuales de los casos de usos con las interfaces

correspondientes.

Conclusiones del Capítulo.

En el presente capítulo se describió la solución propuesta de acuerdo con los elementos del negocio. Se presentaron además los artefactos generados como parte de la caracterización y definición del sistema. El modelo de dominio es el primero de los artefactos desarrollados y se usa para relacionar los conceptos manejados. Se definieron además los requisitos funcionales y no funcionales de acuerdo con las características del entorno donde se usará la aplicación. Estas funcionalidades se agruparon a través del diagrama de casos de uso presentado. Se detalló cada caso de uso definido en una especificación que se muestra como anexo al presente documento.

CAPÍTULO 3: DISEÑO DEL SISTEMA.

Introducción.

En este capítulo se hace referencia a los aspectos correspondientes al diseño, en función de la propuesta del sistema y teniendo en cuenta las funcionalidades seleccionadas. Se confeccionan los diagramas de clases del diseño y los diagramas de interacción. Se describen las clases del diseño más importantes en función de establecer la estructura del sistema. Se especifican los patrones de diseño y se realiza la propuesta de interfaz de usuario para el sistema.

3.1 Patrones utilizados.

Los patrones son una estructura de implementación que logra una finalidad determinada a un lenguaje de programación de alto nivel. En otras palabras, brindan una solución ya probada y documentada a problemas de desarrollo de software que están sujetos a contextos similares. Existen diferentes tipos de patrones: patrones arquitectónicos y patrones de diseño. Un patrón arquitectónico especifica un conjunto predefinido de subsistemas con sus responsabilidades y una serie de recomendaciones para organizar los distintos componentes. Un patrón de diseño está relacionado con los aspectos del diseño de los subsistemas. Es una solución estándar para un problema común de programación y una técnica para flexibilizar el código haciéndolo satisfacer ciertos criterios.

3.1.1 Patrones arquitectónicos.

Son los que definen la estructura de un sistema software, los cuales a su vez se componen de subsistemas con sus responsabilidades, también tienen una serie de directivas para organizar los componentes del mismo sistema, con el objetivo de facilitar la tarea del diseño de tal sistema.

Modelo Vista Controlador (MVC): Es un patrón de arquitectura de software que separa los datos de una aplicación, la interfaz de usuario, y la lógica de control en tres componentes distintos. El patrón el cual define un estilo arquitectónico de llamada y retorno, se ve frecuentemente en aplicaciones web, donde la vista es la página HTML y el código que provee de datos dinámicos a la página. El modelo es el Sistema de Gestión de Base de Datos y la Lógica de negocio, y el controlador es el responsable de recibir los eventos de entrada desde la vista.

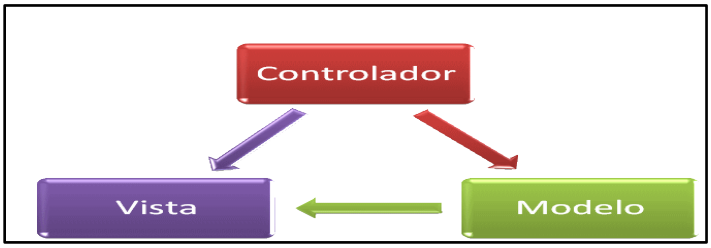


Fig 3.1 Modelo Vista Controlador.

Grails se vale de las bondades del patrón **MVC** brindando las vistas en forma o extensión gsp. Este tipo de páginas conforman la capa de interacción con el cliente. Los controladores, encargados de mantener el flujo de comunicación entre las vistas y el modelo, están definidos por los groovy controllers o controladores groovy, los cuales especifican cómo acceder a los datos y el envío de las respuestas a las vistas. El modelo, definido por clases de dominio de groovy que se mapean en la base de datos utilizada, complementan la tercera capa y permiten el manejo y almacenamiento de los datos. De esta forma, queda definida la utilización del patrón **MVC** en la interacción y desarrollo con el framework. Por su parte, Grails define una cuarta capa, la capa de Servicios, la cual separa la lógica de negocio de los controladores y la incluye en ella.

3.1.2 Patrones de Diseño.

Un patrón de diseño es una solución a un problema de diseño. Para que una solución sea considerada un patrón debe poseer ciertas características. Una de ellas es que debe haber comprobado su efectividad resolviendo problemas similares en ocasiones anteriores. Otra es que debe ser reusable, lo que significa que es aplicable a diferentes problemas de diseño en distintas circunstancias.

Inversión del control (IoC): La inversión de control es un patrón utilizado en Grails, según el cual las dependencias de un componente no deben gestionarse desde el propio componente para que éste sólo contenga la lógica necesaria para hacer su trabajo.

Al utilizar servicios en Grails, solo se escribe una variable con su nombre, sin tener en cuenta la instanciación del servicio y su configuración. Grails configura Spring para que gestione su ciclo de vida y sus dependencias. El objetivo de esta técnica es mantener los componentes tan sencillos como sea posible a la vista del programador, incluyendo únicamente código que tenga relación con la lógica de negocio. Así, la aplicación será más fácil de comprender y mantener.

3.1.2.1 Patrones GRASP.

Son patrones generales de software para asignación de responsabilidades, es el acrónimo de Patrones de Software de Asignación de Responsabilidades Generales (General Responsibility Assignment Software Patterns). Aunque se considera que más que patrones propiamente dichos, son una serie de "buenas prácticas" de aplicación recomendable en el diseño de software (20).

Patrón Controlador: Sirve como intermediario entre una determinada interfaz y el algoritmo que la implementa, de tal forma que es la que recibe los datos del usuario y la que los envía a las distintas clases según el método llamado. Este patrón sugiere que la lógica de negocios debe estar separada de la capa de presentación, esto para aumentar la reutilización de código y a la vez tener un mayor control.

Los controladores en Grails, son los encargados de posibilitar una capa intermedia entre las vistas y el modelo. Además controla el flujo de información referente a la lógica de negocio.

Alta cohesión: Se refiere a que la información que almacena una clase debe de ser coherente y debe estar (en la medida de lo posible) relacionada con la ella misma.

Bajo acoplamiento: Se refiera a tener las clases lo menos ligadas entre sí que se pueda. De tal forma que en caso de producirse una modificación en alguna de ellas, se tenga la mínima repercusión posible en el resto de clases, potenciando la reutilización, y disminuyendo la dependencia entre las clases.

Experto: La responsabilidad de realizar una labor es de la clase que tiene o puede tener los datos involucrados (atributos). Una clase, contiene toda la información necesaria para realizar la labor que tiene encomendada. De esta manera se garantiza una alta cohesión y un bajo acoplamiento.

3.1.2.2 Patrones GOF (Gang of Four).

Singleton: Garantiza que una clase sólo tenga una instancia, y proporciona un punto de acceso global a ella. Se utiliza en la instanciación de los servicios en los controladores. Por defecto en Grails todos los servicios son Singleton.

Decorador / Decorator: Añade responsabilidades adicionales a un objeto dinámicamente, proporcionando una alternativa flexible a la especialización mediante herencia, cuando se trata de añadir

funcionalidades. El uso de este patrón se centra en la utilización de SiteMesh integrado a la arquitectura del framework. El mismo permite a partir de una plantilla inicial de la página incorporar modificaciones a cada vista en particular.

3.2 Diagramas de Clases del Diseño.

Un diagrama de clases es un tipo de diagrama estático que describe la estructura de un sistema mostrando sus clases, atributos y las relaciones entre ellos. Los diagramas de clases son utilizados durante el proceso de análisis y diseño de los sistemas, donde se crea el diseño conceptual de la información que se manejará en el sistema, y los componentes que se encargarán del funcionamiento y la relación entre uno y otro.

Los diagramas de clases del diseño (DCD) del Sistema Inteligente de Mitigación de Riesgos, fueron realizados a partir de lo planteado en la metodología utilizada, es decir diseñando por funcionalidad. Tendiéndose un DCD por funcionalidad del sistema.

La leyenda muestra la definición de las clases en los DCD.

Define las Vistas | Define las Clases (Entidades) | Define los Controladores | Define los Servicios

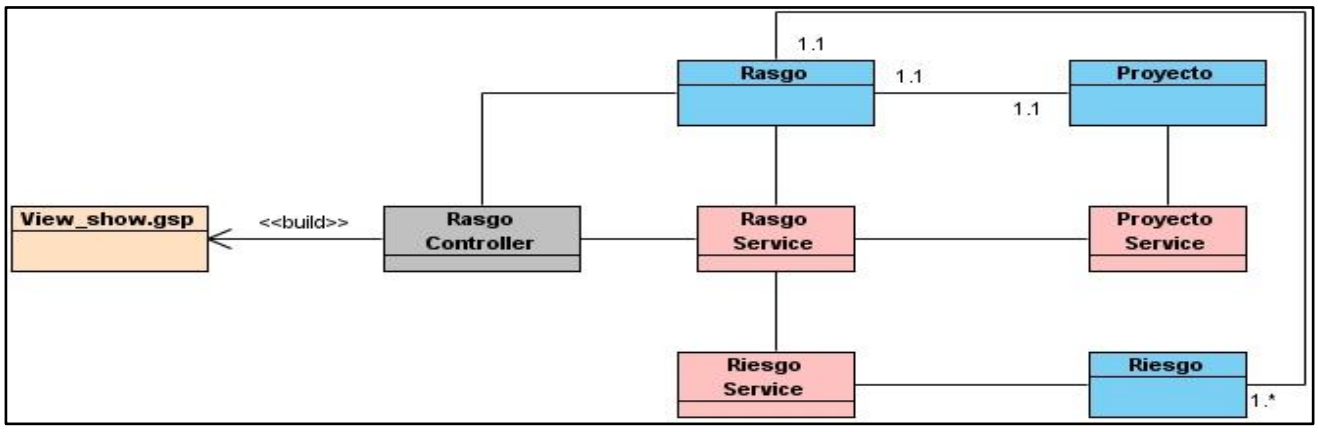


Fig. 3.2 Diagrama de Clases del Diseño Caso de Aprender Caso.

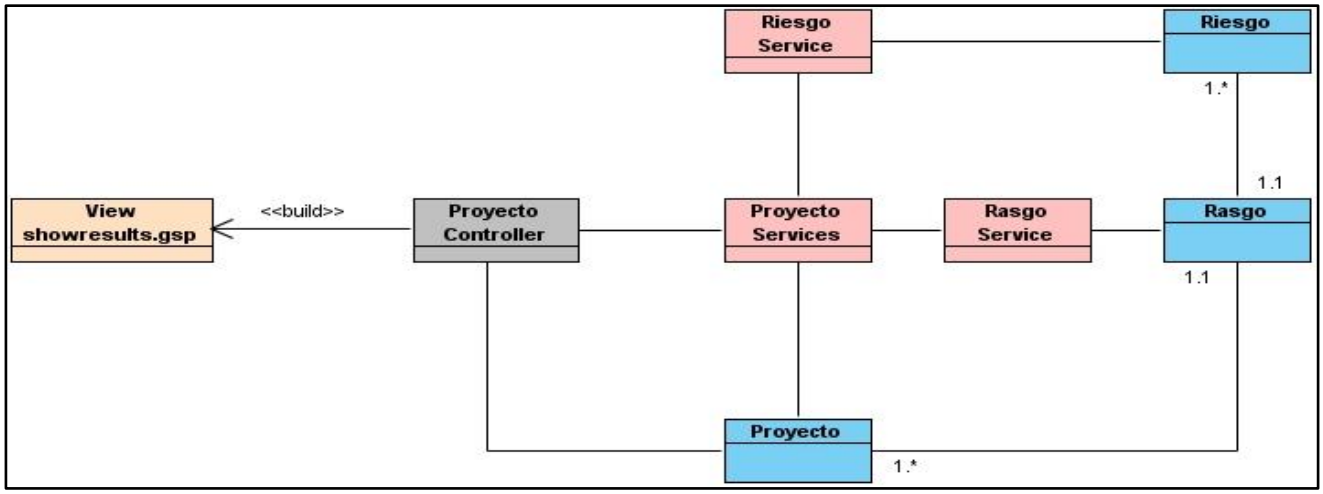


Fig. 3.3 Diagrama de Clases del Diseño. Caso de Uso Identificar Riesgos.

Ver **Anexo 2**.

3.3 Descripción de las Clases Fundamentales (Entidades).

El modelo de datos en una aplicación Grails está compuesto por las clases del Dominio (entidades). Grails utiliza GORM (Grails Object Relational Mapping) / (Mapeo relacional de Objetos Grails), para controlar el ciclo de vida de las entidades y proporcionar una serie de métodos dinámicos (se crean en tiempo de ejecución para cada entidad) que facilitan enormemente las búsquedas. GORM está construido sobre Hibernate, una herramienta de mapeo objeto-relacional, que se encarga de relacionar las clases con tablas de una base de datos, y sus propiedades con campos en las tablas. Cada operación que se realice sobre los objetos del modelo de datos será traducida por Hibernate a las sentencias SQL necesarias para quedar reflejado en la base de datos.

3.3.1 Clase Riesgo.

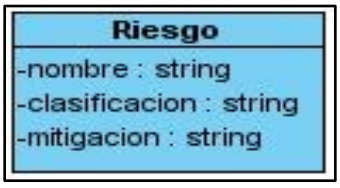


Fig. 3.4 Clase riesgo.

Propósito: Proporcionar un medio para la creación y manejo de los datos de los riesgos que se identificarán por parte del sistema.

3.3.2 Clase Evaluación.

Fig. 3.5. Clase Evaluación.

Propósito: Posibilitar y facilitar el proceso de aprendizaje del sistema, pues a partir de una evaluación dada por el usuario, el administrador del sistema decidirá aprender tales rasgos para así formar un nuevo caso.

3.3.3 Clase Rasgo.

Fig. 3.6 Clase Rasgo.

Propósito: Definir el valor de los rasgos predictivos así como de los rasgos objetivos, a partir de los cuales se identificarán los riesgos asociados a los mismos.

3.3.4 Clase Evaluación.

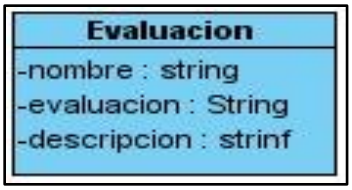


Fig. 3.7 Clase Evaluación.

Propósito: Facilitar el proceso de aprendizaje del sistema, pues a partir de una evaluación dada el consultor del sistema decidiría aprender tales rasgos para así formar un nuevo caso.

3.3.5 Clase Proyecto.

Fig. 3.8 Clase Proyecto.

Propósito: Proporcionar el medio necesario para la creación y manejo de la descripción o los datos correspondientes a un proyecto de desarrollo de software, a partir de los cuales, y con el uso de la clase rasgo, se identificarán los riesgos asociados a tal descripción.

3.4 Servicios Principales.

Los servicios en Grails, no son más que componentes que implementan la lógica de negocio. Pues permiten separar el acceso a datos de los controladores, dejándole a estos sólo el control del flujo de la información.

3.4.1 EvaluacionService.

El propósito de este servicio, es proporcionar una vía para emitir un criterio sobre el resultado de la identificación de los riesgos. Posibilitando un mecanismo para el aprendizaje del sistema.

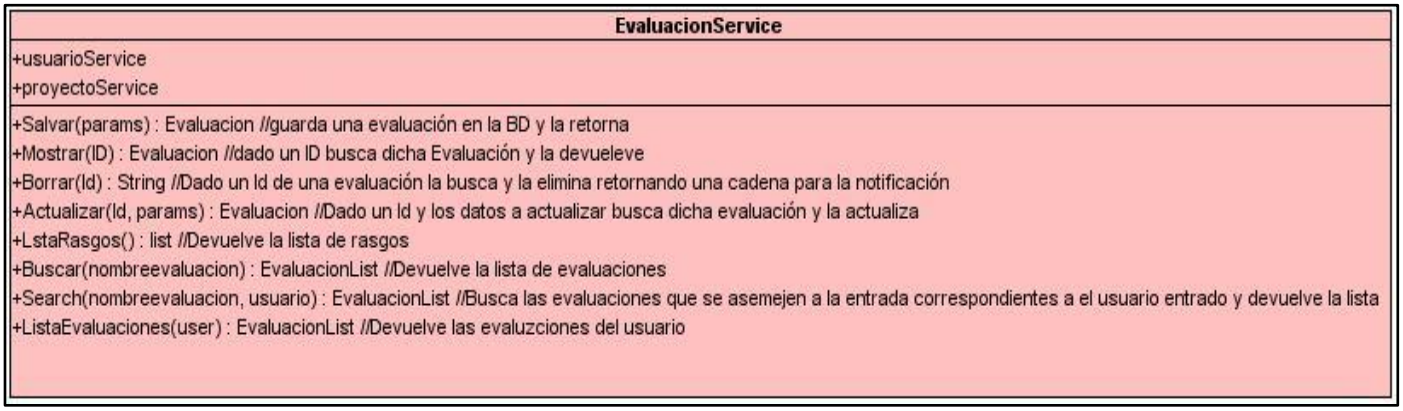


Fig. 3.9 EvaluacionService.

3.4.2 RasgoService.

El propósito de este servicio es manipular los datos referentes a los casos en la base de conocimientos. Además contiene la lógica necesaria para el aprendizaje del sistema, dado una evaluación a un proyecto.

RasgoService
+riesgoService
+Salvar(params) : Rasgo //guarda un rasgo en la BD y lo retorna +Salvar_Caso(caso) : Rasgo //Dado un proyecto evaluado busca que no se asemeje con los guardados en la base de casos, lo construye y lo retorna +Asociar_Riesgo(rasgo, riesgonew) : //Dado un rasgo y un caso asocia los riesgos del rasgo al caso +Mostrar(ID) : Rasgo //dado un ID busca dicho Rasgo y lo devueleve +Borrar(Id) : String //Dado un Id de un rasgo lo busca y lo elimina retornando una cadena para la notificación +Actualizar(Id, params) : Rasgo //Dado un Id y los datos a actualizar busca dicho rasgo y lo actualiza +LstaRasgos() : list //Devuelve la lista de rasgos +Buscar(nombre) : Rasgo //Busca dado un caso a aprender cuan semejante es a los almacenados en la base de casos +Search(rasgo) : Rasgo //Busca un rasgo que se asemeje al entrado con la utilización de comodines

Fig. 3.10 RasgoService.

3.4.3 ProyectoService.

ProyectoService
+riesgoService +rasgoService +usuarioService
+Salvar(params) : Proyecto //guarda un proyecto en la BD y lo retorna +Asociar_Riesgo(rasgo, riesgonew) : //Dado un rasgo y un riesgo asigna dicho riesgo al rasgo +Mostrar(ID) : Proyecto //dado un ID busca dicho Proyecto y lo devueleve +Borrar(Id) : String //Dado un Id de un proyecto lo busca y lo elimina retornando una cadena para la notificación +Actualizar(Id, params) : Proyecto //Dado un Id y los datos a actualizar busca dicho proyecto y lo actualiza +LstaProyectos() : list //Devuelve la lista de proyectos +Buscar(nombreproyecto) : ProyectoList //Busca y devuelve la lsita de proyectos semejantes al nombre entrado +Search(nombreproyecto, usuario) : Proyecto //Busca y retorna el proyecto semejante al entrando (usando comodines) que corresponde a un usuario +Obtain_Risk(ID) : RiesgoList //Dado un id de un proyecto lo obtiene y aplica a cada caso la funcion de semejanza para devolver los posibles riesgos +Semejanza(proyecto, rasgo) : float //Dado un proyecto y un caso con sus rasgos compara cada uno por campos utilizando las funciones de semejanza +FuncSemejanzaNumerica(valor1, valor2) : float //compara los valores según rangos establecidos para los datos numéricos +FuncSemejanzaTexto5(texto1, texto2) : float //compara los valores para el modelo de proyecto,dominio del negocio,experiencia en la plataforma y el nivel de conocimiento del lenguaje y de de los desarrolladores +FuncSemejanzaTexto4(texto1, texto2) : float //compara los valores para la experiencia del equipo, la motivación del equipo,nivel de dificultad del lenguaje, cohesión del equipo, experiencia en las herramientas y la estabilidad de los requerimientos +FuncSemejanzaTexto3(texto1, texto2) : float //compara los valores para la complejidad de la aplicacion, el tiempo promedio dedicado al proyecto, la información que se maneja, el cronograma, la relación con el cliente +FuncSemejanzaTexto2(texto1, texto2) : float //Compara los valores entrados para el nivel de acceso del cliente +FuncSemejanzaCliente(texto1, texto2) : float //Compara los valores entrados para el tipo de cliente

Fig. 3.11 ProyectoService.

El propósito de este servicio, es proporcionar una vía para la manipulación de los datos referente a un proyecto. Dígase inserción, actualización, lectura y eliminación. Además, contiene la lógica necesaria para lograr la inteligencia del sistema conteniendo las operaciones de consulta y cálculo necesarias para el comportamiento en interpretación de la información referente al proyecto y los posibles riesgos a incidir dada su descripción.

3.5 Diagramas de Interacción.

Los diagramas de iteración que serán utilizados en este trabajo son los de secuencia, estos permiten ver la relación entre las clases y los objetos que participan, así como los mensajes y el tiempo de vida que tiene cada objeto.

Ver **Anexo 3.**

Conclusiones del Capítulo.

En el capítulo 3 se mostraron los diagramas de clases del diseño elaborados para darle solución al sistema, así como la descripción de las clases más importantes y los servicios de mayor relevancia utilizados. Se presentaron los patrones de diseño y los arquitectónicos seleccionados para el diseño y la implementación del sistema. Se elaboraron diagramas de secuencia para modelar la interacción entre las clases, los cuales se muestran como anexos al documento.

CAPÍTULO 4: IMPLEMENTACIÓN Y PRUEBA.

Introducción.

En el presente capítulo se mostrará el diagrama de componentes Sistema Inteligente de Mitigación de Riesgos y la descripción de cada uno de los componentes que lo integran. Además se hará alusión de los algoritmos utilizados y se detallará la solución propuesta. Por último se realizarán pruebas que contribuyen a la calidad de la aplicación en general.

4.1 Diagrama de Componentes.

El diagrama de componentes describe los elementos físicos del sistema y sus relaciones. Para el caso de este trabajo se utilizan los paquetes de clases a nivel lógico descomponiendo de una forma más original el sistema a implementar.

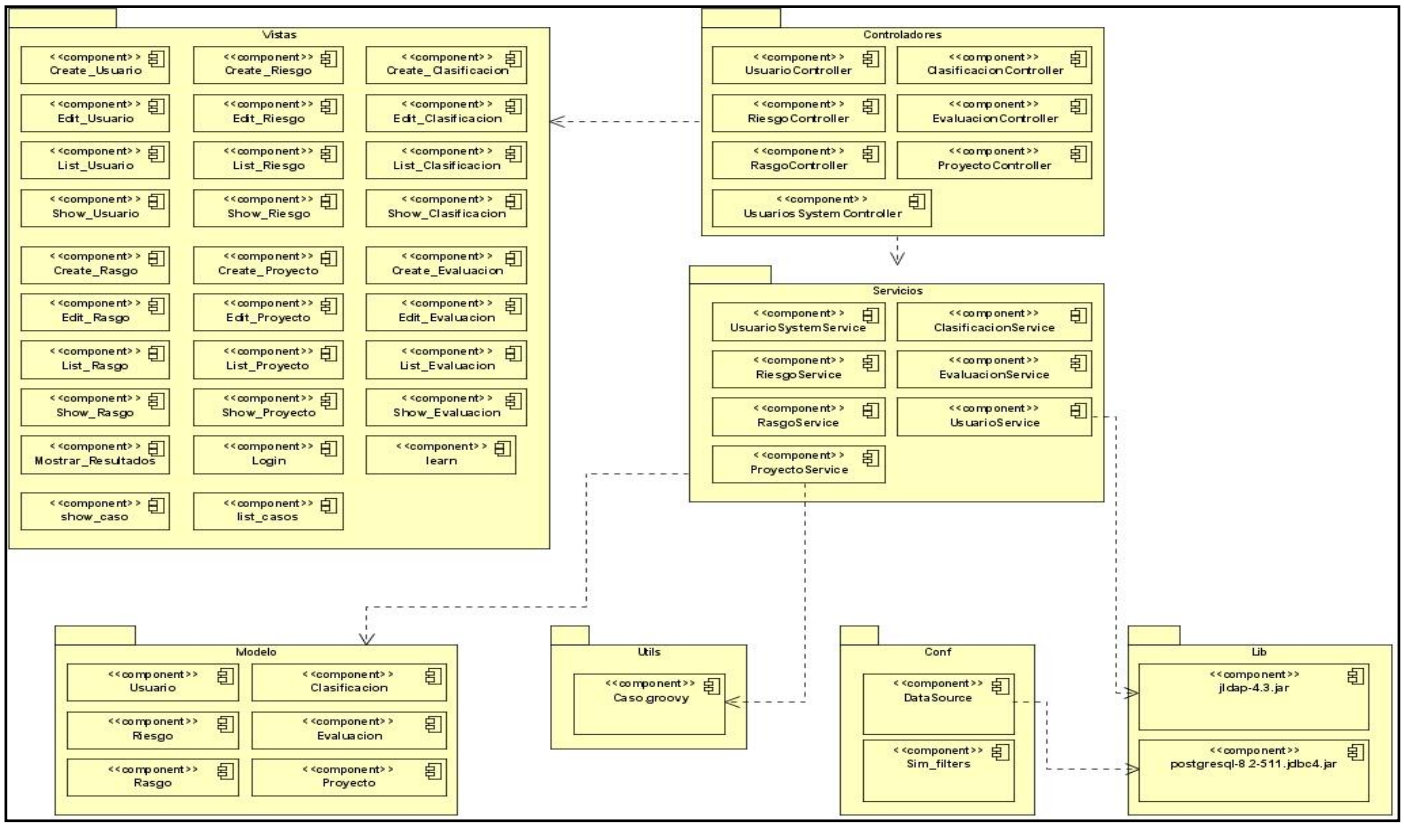


Fig. 4.1 Diagrama de Componentes.

Este es otro de los artefactos de la metodología RUP que se utilizan para garantizar una documentación mas detallada del sistema desarrollado, dado que FDD solo considera el código fuente como principal resultado obtenido en la fase de Construcción por Funcionalidad (BBF).

4.2 Descripción de los Componentes.

4.2.1 Paquete de componentes "Vistas".

Fig. 4.2 Paquete Vista.

El paquete de componentes "Vista" es una representación lógica de las páginas gsp o vistas con las cuales interactúa el usuario y permiten la consulta e inserción de datos al sistema.

4.2.2 Paquete de componentes "Controladores".

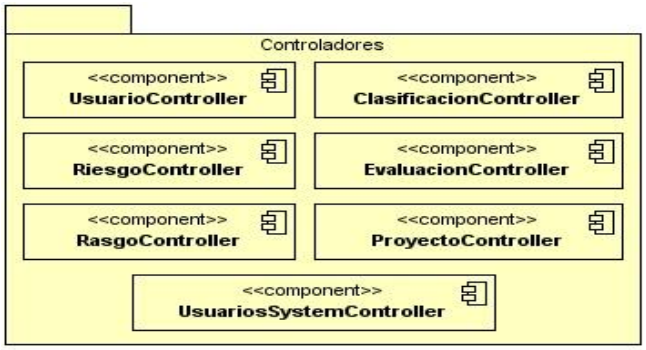


Fig. 4.3 Paquete Controladores.

El paquete de componentes "Controladores" es una representación lógica del conjunto de operaciones que permite mantener el flujo de comunicación entre las vistas y el modelo. Para esto se hace uso de los servicios como intermediarios entre los controladores y las clases del dominio o modelo.

4.2.3 Paquete de componentes "Servicios".

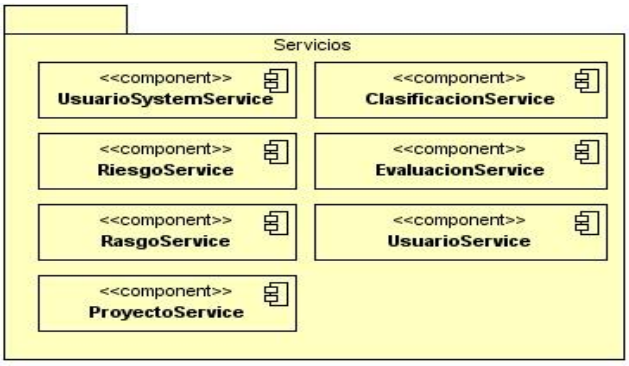


Fig. 4.4 Paquete Servicios.

El paquete de componentes "Servicios" es una representación de las operaciones de lógica de negocio, que permite comunicar los controladores con el modelo y manejar los datos.

4.2.4 Paquete de componentes "Modelo".

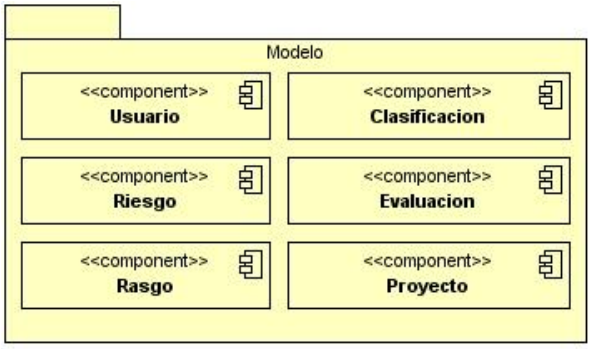


Fig. 4.5 Paquete Modelo.

Son las clases que representa todo el negocio del sistema. Dichas clases del dominio constituyen entidades o tablas en la base de datos mapeadas por el GORM.

4.2.5 Paquete de componentes "Conf".

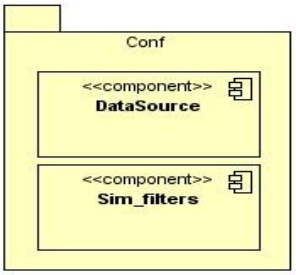


Fig. 4.6 Paquete Conf.

Este paquete es utilizado para la configuración y definición del acceso local o remoto a los datos. Además garantiza con la utilización del componente SimFilters la seguridad y control de acceso a través del manejo y control de las acciones en los controladores para lograr el acceso a las acciones realizables por cada usuario del sistema. Constituye un componente del Framework donde se ubican ficheros groovy de configuración.

4.2.6 Paquete de componentes "Lib".

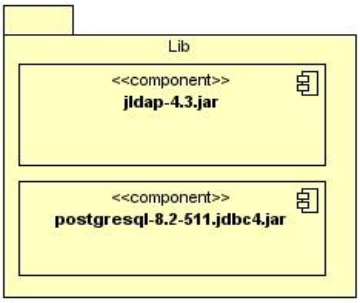


Fig. 4.7 Paquete Conf.

Este paquete es utilizado para la conexión a la Base de datos con el uso del driver correspondiente al gestor de Bases de datos utilizado. Además, se utiliza otra librería que permite la conexión a LDAP (jldap-4.3).

4.2.7 Paquete de componentes "Utils".

Fig. 4.8 Paquete Conf.

Este paquete contiene una clase auxiliar utilizada, para contar con una estructura que permita guardar un caso y su correspondiente valor de semejanza.

4.3 Descripción de los algoritmos utilizados.

4.3.1 Identificar Riesgos.

La identificación de riesgos es la funcionalidad de mayor relevancia en el sistema. A través de los datos del proyecto de un usuario, este último ejecutará la acción Identificar Riesgos. De esta forma, se compararán los datos entrados con los casos almacenados en la base de conocimientos a partir de las funciones de semejanza correspondientes para luego evaluar el caso. Cada dato, según su valor y dominio tendrá una función de semejanza, la cual comparará dicho valor con su rasgo correspondiente en

cada Caso de la Base de Conocimientos. Al terminar de analizar y comparar cada rasgo predictor (columna de la Base de Casos o campo independiente), se procederá a evaluarlo (rasgo objetivo o fila de la Base de casos). La función de evaluación correspondiente a los casos es la siguiente:

$$F(E) = \frac{\sum_{i=0}^{n} Wi\delta i(R1,R2)}{\sum_{i=0}^{n} Wi}$$

Donde W representa el peso o nivel de importancia de cada rasgo predictor y δ representa la función de semejanza definida para la comparación de el caso existente (R2) con el caso nuevo (R1).

Para hallar la semejanza de cada rasgo predictor con los datos insertados, fueron utilizadas diferentes funciones de según los valores de dominio que abarcan estos últimos. Las figuras muestran los cuatro grupos en que fueron agrupadas las funciones de comparación de valores discretos. Dicha comparación está definida por valores según la semejanza entre ellos.

	Muy Alta	Alta	Baja	Ninguna	ninguna
Muy Alta	1	0,7	0,4	0,1	0
Alta	0,7	1	0,6	0,2	0
Media	0,4	0,6	1	0,5	0
Baja	0,1	0,2	0,5	1	0,3
Ninguna	0	0	0	0,3	1

Tabla 4.1 Valores de comparación y dominios.

Los valores de comparación y los dominios anteriores, están definidos para el modelo de proyecto, dominio del negocio, experiencia en la plataforma, nivel de conocimiento del lenguaje, las herramientas y los desarrolladores.

La siguiente figura muestra también los valores y dominios correspondientes a la experiencia del equipo, la motivación, la experiencia del mismo, el nivel de dificultad del lenguaje, la cohesión del equipo, la experiencia en las herramientas y la estabilidad de los requisitos.

Estabilidad de Requerimientos Experiencia/motivación/Dificultad lenguaje/Cohesión del equipo/Experiencia con las herramientas	Muy Estables	Estables	Poco Estables	Inestables
	Mucha	Media	Baja	Ninguna
Mucha/Muy Estables	1	0,6	0,2	0
Media/Estables	0,6	1	0,5	0
Baja/Poco Estables	0,2	0,5	1	0,1
Ninguna/Inestables	0	0	0,1	1

Tabla 4.2 Valores de comparación y dominios.

Se muestran además, los valores de dominio y de comparación para la complejidad de la aplicación, el tiempo promedio dedicado al proyecto, la información que se maneja, la situación del cronograma y la relación con el cliente.

Cronograma	Muy Variable	Variable	Poco Variable
Relación con el cliente	Buena	Regular	Mala
Información que se maneja	Confidencial	Pública	Privada
Complejidad de la aplicación	Alta	Media	Baja
Alta/Confidencial/Buena	1	0,5	0
Media/Pública/Regular	0,5	1	0,2
Baja/Privada/Mala	0	0,2	1

Tabla 4.3 Valores de comparación y dominios.

También se exponen los datos para la comparación y el dominio de la relación con el cliente.

Relación con el cliente	Accesible	Poco Accesible
Accesible	1	0,5
Poco Accesible	0,5	1

Tabla 4.4 Valores de comparación y dominios.

La comparación se realiza cada dato nuevo insertado referente a un proyecto, con cada rasgo predictor de la base de casos retornando su valor de semejanza. Si son iguales se retornará 1 (valor que constituye la semejanza). Sino, se procederá a devolver el valor correspondiente a cada situación expresada en la tabla, comparando el valor de dominio de la columna correspondiente a la variable retornando el valor de semejanza perteneciente a tal comparación Esto será multiplicado por el peso o nivel de importancia

definida para cada rasgo predictor. Evaluando el resultado con la utilización de la fórmula matemática mostrada anteriormente.

Los valores numéricos o continuos, son comparados siguiendo una estrategia de pertenencia a intervalos. Si los valores son iguales se retorna 1. De no ser iguales pero, si el módulo de la resta de ambos es igual a 2 se retorna 0.7 (lo que constituye su valor de semejanza), y si este cálculo retorna 3 o 4 entonces se devolverá 0.3. De lo contrario se asumirá que los valores son muy diferentes y se retornará 0.

$$F(S)\begin{cases} V1 = V2 & \delta i = 1 \\ |V1 - V2| \leq 2 & \delta i = 0.7 \\ 2 < |V1 - V2| \leq 4 & \delta i = 0.3 \\ 0 & \end{cases}$$

De esta forma, cada caso nuevo será comparado con los existentes y posteriormente evaluado. Este proceso se realiza de forma iterativa para todos los rasgos de la Base de casos obteniendo así el caso que exprese mayor semejanza a la descripción entrada. Luego, se obtendrán los riesgos asociados a este caso y se le mostrarán al usuario en una página con algunos datos del proyecto y los datos de los posibles riesgos a incidir.

4.4 Vista de Despliegue.

4.4.1 Diagrama de despliegue.

El Diagrama de Despliegue es un tipo de diagrama del Lenguaje Unificado de Modelado que se utiliza para modelar el hardware utilizado en las implementaciones de sistemas y las relaciones entre sus componentes.

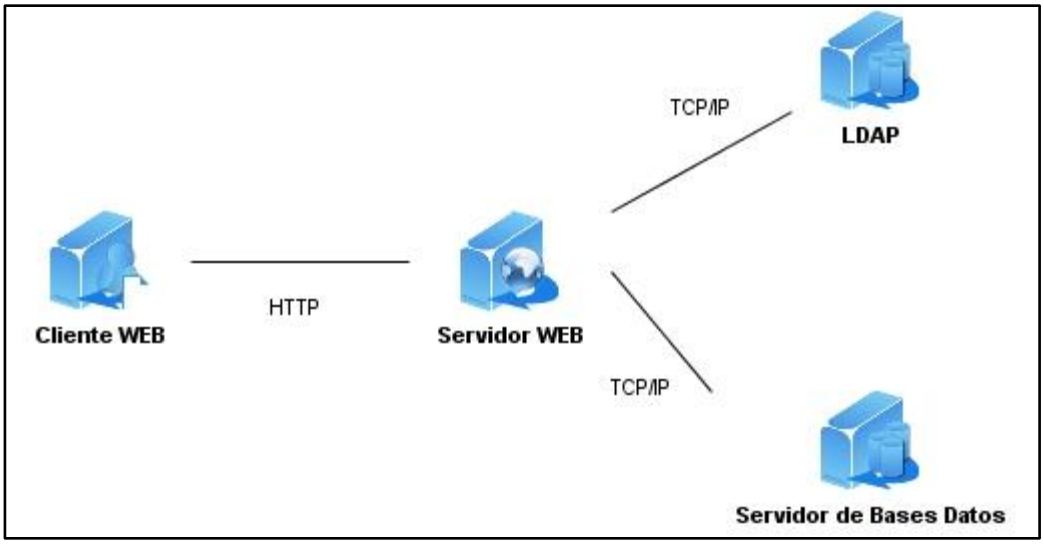


Fig. 4.9 Diagrama de Despliegue.

Un cliente web, es una PC tanto portátil como de escritorio, la cual tiene un sistema operativo instalado y un navegador disponible, preferentemente Firefox en cualquier versión. El servidor web, es una PC de escritorio o un servidor, la cual tiene instalado un servidor web, que no es más que una aplicación que procesa cualquier aplicación del lado del servidor permitiendo conexiones con clientes generando una respuesta en cualquier lenguaje. Generalmente las conexiones con el servidor son realizadas a través del protocolo HTTP. El servidor de Base de Datos es un Computadora la cual tiene instalado un programa que permite organizar datos en una o más tablas relacionadas. LDAP son las siglas de Lightweight Directory Access Protocol (en español Protocolo Ligero de Acceso a Directorios) que hacen referencia a un protocolo a nivel de aplicación el cual permite el acceso a un servicio de directorio ordenado y distribuido para buscar diversa información en un entorno de red (21). LDAP también es considerado una base de datos (aunque su sistema de almacenamiento puede ser diferente) a la que pueden realizarse consultas.

4.5 Pruebas al Sistema.

Las pruebas son una actividad en la cual un sistema o componente es ejecutado bajo unas condiciones específicas. Los resultados son observados y registrados, y una evaluación es hecha de algún aspecto del sistema o componente. La prueba de software es un elemento crítico para la garantía de la calidad del software y representa una revisión final de las especificaciones del diseño y de la codificación. La prueba está enfocada principalmente en la evaluación y determinación de la calidad del producto. Estas están definidas en estrategias, niveles, tipos, métodos y técnicas de prueba.

4.5.1 Estrategia de Prueba.

La estrategia de prueba describe el enfoque y los objetivos generales de las actividades de prueba. Incluye los niveles de prueba (unidad, integración, etc.) a ser diseccionados y el tipo de prueba a ser ejecutadas (funcional, stress, etc.).

La estrategia de prueba define:

- Técnicas de pruebas (manual o automática) y herramientas a ser usadas.
- Qué criterios de éxitos y culminación de la prueba serán usados.
- Consideraciones especiales afectadas por requerimientos de recursos o que tengan implicaciones en la planificación.

4.5.2 Niveles de Pruebas.

Se distinguen los siguientes niveles de pruebas:

- **Prueba de desarrollador**
- **Prueba independiente**
- **Prueba de Unidad**
- **Prueba de Integración**
- **Prueba de sistema**
- **Prueba de aceptación.**

Prueba de unidad: Enfocada a los elementos más pequeños del software que pueden ser probados. Aplicable a componentes representados en el modelo de implementación para verificar que los flujos de control y de datos están cubiertos, y que funcionen como se espera. La prueba de unidad siempre está orientada a caja blanca.

Prueba de Sistema: Se hacen cuando el software está funcionando como un todo. Es la actividad de prueba dirigida a verificar el programa final, después que todos los componentes de software y hardware han sido integrados. En un ciclo iterativo estas pruebas ocurren más temprano, tan pronto como subconjuntos bien formados de comportamiento de caso de uso son implementados.

4.5.3 Tipo de Prueba.

Funcionalidad	Usabilidad	Fiabilidad	Rendimiento	Soportabilidad
Función: Pruebas fijando su atención en la validación de las funciones, métodos, servicios, caso de uso.	**Usabilidad:** Prueba enfocada a factores humanos, estéticos, consistencia en la interfaz de usuario, ayuda sensitiva al contexto y en línea, asistente documentación de usuarios y materiales de entrenamiento.	**Integridad:** Enfocada a la valoración de la robustez (resistencia a fallos).	**Benchmark:** es un tipo de prueba que compara el rendimiento de un elemento nuevo o desconocido a uno de carga de trabajo de referencia conocido.	**Configuración:** Enfocada a asegurar que funciona en diferentes configuraciones de hardware y software. Esta prueba es implementada también como prueba de rendimiento del sistema
Seguridad: Asegurar que los datos o el sistema solamente es accedido por los actores deseados.		**Estructura:** Enfocada a la valoración a la adherencia a su diseño y formación. Este tipo de prueba es hecho a las aplicaciones Web asegurando que todos los enlaces están conectados, el contenido deseado es	**Contención:** Enfocada a la validación de las habilidades del elemento a probar para manejar aceptablemente la demanda de múltiples actores sobre un mismo recurso (registro de recursos, memoria, etc).	**Instalación:** Enfocada a asegurar la instalación en diferentes configuraciones de hardware y software bajo diferentes condiciones (insuficiente espacio en disco, etc)

		mostrado.		
Volumen: Enfocada en verificar las habilidades de los programas para manejar grandes cantidades de datos.		**Stress:** Enfocada a evaluar cómo el sistema responde bajo condiciones anormales. (extrema sobrecarga, insuficiente memoria, servicios y hardware no disponible, recursos compartidos no disponible).	**Carga:** Usada para validar y valorar la aceptabilidad de lo límites operacionales de un sistema bajo carga de trabajo variable, mientras el sistema bajo prueba permanece constante. La variación en carga es simular la carga de trabajo promedio y con picos que ocurre dentro de tolerancias operacionales normales.	
			Performance profile: Enfocadas a monitorear el tiempo en flujo de ejecución, acceso a datos, para identificar y direccionar los cuellos de botellas y los procesos ineficientes.	

Tabla 4.5 Tipos de Pruebas.

4.5.4 Métodos de Pruebas.

Los métodos de pruebas son el de caja negra y caja blanca.

La prueba de caja negra se refiere a las pruebas que se llevan a cabo sobre la interfaz del software. O sea, los casos de prueba pretenden demostrar que las funciones del software son operativas, que la entrada se acepta de forma adecuada y que se produce un resultado correcto, así como que la integridad de la información externa se mantiene.

La prueba de la caja blanca del software comprueba los caminos lógicos del software proponiendo casos de prueba que se ejerciten con conjuntos específicos de condiciones y/o bucles. Se puede examinar el estado del programa en varios puntos para determinar si el estado real coincide con el esperado o mencionado.

4.5.5 Estrategia de prueba seguida.

La estrategia seguida para la realización de las pruebas al Sistema Inteligente de Mitigación de Riesgos contempla dos niveles: el nivel de pruebas de Unidad y el nivel de prueba de Sistema. A su vez en el primer nivel, se realizaron pruebas automáticas haciendo uso del sistema de Testing o Prueba que brinda el propio Framework utilizado. El segundo nivel de pruebas incluye la técnica manual incluyendo los tipos de prueba de funcionalidad y usabilidad con el uso de diseño de casos de prueba utilizando el método de caja negra.

Los test unitarios son aquellos en los que se verifica que un método se comporta como debería, sin tener en cuenta su entorno. Esto significa que cuando se ejecutan las pruebas unitarias de un método Grails no inyectará ninguno de los métodos dinámicos con los se cuenta la aplicación cuando se está ejecutando. Así que, cuando se trabaja con entidades o servicios son los desarrolladores los responsables de crear y gestionar todos los objetos.

Durante el proceso de desarrollo, podemos probar el estado de la aplicación mediante el comando:

```
grails test-app
```

Ver Anexos **4** y **5**.

A partir de esto, se ejecutarán todas las pruebas unitarias y de integración del proyecto, y generará un informe al que se puede recurrir en caso de fallos. El informe se genera en texto y HTML. Para crear un test unitario de un Servicio se ejecuta el comando:

grails create-unit-test<nombre de la prueba>

Creándose una nueva batería de pruebas en la carpeta test/unit del proyecto. La clase generada hereda de GrailsUnitTestCase, lo que hace que se tenga a disposición una serie de métodos para que facilite la tarea de probar los servicios definidos. Este test (prueba) se ejecutará sin levantar el contexto Grails, de forma que las entidades no disponen de los métodos dinámicos inyectados por GORM, ni se realiza la inyección de dependencias. Para esto, GRAILS aprovecha el soporte nativo en Groovy para distintas formas de **Mocks** y **Stubs**.

La clase GrailsUnitTestCase pone a nuestra disposición métodos para que nuestros objetos se comporten en pruebas como lo harían en ejecución. Gracias a estos métodos, se pueden probar los artefactos y usar los métodos dinámicos de Grails sin necesidad de levantar el contexto de ejecución.

La prueba de Caja Negra se centra principalmente en los requisitos funcionales del software. Estas pruebas permiten obtener un conjunto de condiciones de entrada que ejerciten completamente todos los requisitos funcionales de un programa. En ellas se ignora la estructura de control, concentrándose en los requisitos funcionales del sistema y ejercitándolos. Ver **anexo 6**.

Conclusiones.

En el presente capítulo se mostró y detalló el algoritmo seguido para la implementación de la identificación de los riesgos. Se definió la distribución física del sistema a través de nodos en el diagrama de despliegue y se visualizó en los diagramas de componentes la organización y dependencia existente entre el código implementado. El desarrollo estuvo enfocado en darle cumplimiento a los requisitos funcionales y no funcionales definidos, lo cual se comprobó con la realización pruebas al sistema definiendo para ello una estrategia que integra niveles, tipos y métodos de prueba.

CONCLUSIONES.

En el presente trabajo se describe la solución propuesta para desarrollar una herramienta capaz de ayudar a la identificación y mitigación de los riesgos a lo largo del ciclo de vida de desarrollo de software con el nombre “Sistema Inteligente de Mitigación de Riesgos para el centro ISEC”:

- Se definieron los procesos fundamentales relacionados con la gestión de riesgos y se seleccionó MOGERI como una metodología adecuada para ello de acuerdo con las características de ISEC, tras el análisis detallado de los métodos que abordan el tema.
- Se utilizaron herramientas dentro de las que se encuentran, como IDE de desarrollo NetBeans 6.9, Grails como framework, Groovy como lenguaje de programación y para el modelado se utilizó Visual Paradigm 3.4.
- Se usa FDD como metodología de desarrollo, pues se integra perfectamente con el ritmo de trabajo ágil definido por el framework, apoyándose en algunos artefactos generados por RUP para cubrir las etapas de desarrollo en las que FDD no centra su atención.
- Se definieron requisitos indispensables a ser resueltos con el sistema a desarrollar.
- Se realizó el diseño del sistema propuesto.
- Se formalizó un modelo computacional donde se aplican técnicas de inteligencia artificial en la consideración de Mitigación de Riesgos como entidad principal.
- Se realizó la implementación computacional del modelo propuesto.
- Se realizaron pruebas al sistema desarrollado de acuerdo con una estrategia definida para ello.
- Aplicar el modelo propuesto a la solución de proyectos reales del centro.

RECOMENDACIONES.

Este trabajo da solución al problema planteado y satisface los objetivos trazados al inicio de la investigación, pero existen elementos que se le pueden incorporar en un futuro para una mayor eficiencia. Debido a esto, al concluir este trabajo y en aras de darle continuidad al mismo, con el objetivo de obtener una solución más completa se presentan un conjunto recomendaciones que se deberían tener en cuenta para una posterior versión:

- Incluir en el sistema, específicamente en la programación inteligente el trabajo con valores de incertidumbre.
- Permitir cuantificar las pérdidas económicas que pudieran existir tras la ocurrencia de un riesgo.
- Investigar y tener en cuenta otras estrategias de aprendizaje.
- Investigar y tener en cuenta otras funciones de comparación además de otra función de evaluación para cada rasgo predictor y objetivo respectivamente.
- Incorporar al sistema un foro, para la discusión de los resultados obtenidos y permitir el intercambio de opiniones entre los usuarios.

REFERENCIAS BIBLIOGRÁFICAS.

1. *Institute, P. M. Guía de los Fundamentos de la Dirección de Proyectos (Guía del PMBOK).* Four Campus Boulevard. Newton Square, Pennsylvania 19073-3299 : Project Management Institute,Inc,, 2004.

2. **Proyecto, Lider de.** Lider de Proyecto. [Online] [Cited: 12 10, 2010.] http://liderdeproyecto.com/manual/que_es_el_pmbok.html.

3. **Expert, Soft.** Soft Expert. [Online] [Cited: 12 10, 2010.] http://www.softexpert.es/norma-pmbok.php..

4. **Pressman, R. S.** *Ingeniería del Software. Un enfoque Práctico.* México, McGraw-Hill : Version 1.1. CMMI. C. M. S. E. Institute. Pittsburgh, Carnegie Mellon Software Engineering Institute., 2002.

5. Real Academia de La Lengua Española. [Online] [Cited: 11 22, 2010.] http://buscon.rae.es/draeI/SrvltConsulta?TIPO_BUS=3&LEMA=inteligencia%20artificial.

6. **Daniel, Gálvez.** *Curso de Sistemas Basados en el Conocimiento.* 1998.

7. **Alarcos.** Ingeniería de Software 1.Tema04.pdf. [Online] [Cited: 11 17, 2010.] http://alarcos.inf-cr.uclm.es/doc/ISOFTWAREI/Tema04.pdf.

8. **Kulik, Peter and Catherine Webber.** *Software Risk Management Practices.* 2001.

9. **Estéves, J and Pastor, A.** *Implementación y Mejora del Método de Gestión Riesgos del SEI en un proyecto universitario de desarrollo de software.* 2005.

10. *Norma Técnica peruana NTP-ISO/IEC 12207.* **INDECOPI.** 2006.

11. **UQAM.** Welcome to the study on The Reality of Project Management Practice 2007. [Online] Diciembre 20, 2010. http://www.surveymonkey.com/DisplaySummary.asp?SID=1993451&U=199345126557.

12. *Revista Española de Innovación, Calidad e Ingeniería de Software*. **Yeleny Zulueta, Eder Despaine.** No. 3, Barcelona : s.n., 2009, Vol. Volume 5.

13. **Lio, Gálvez Daniel.** *Curso de Sistemas Basados en el Conocimiento.* 1998.

14. **Coad P., Lefebvre E. ,De Lucas.** *Metodologías de Desarrollo(FDD). Java Modeling In Color With UML.* Prentice Hall : Enterprise Components and Process, 1999.

15. *Guía a Rational Unified Process.* **Martínez, Alejandro and Martínez, Raúl.** Universidad de Castilla la Mancha : Escuela Politécnica Superior de Albacete.

16. [Online] [Cited: Junio 7, 2011.] http://migueljaque.com/index.php/tecnicas/tecnicasmodnegocio/37-modelado_negocio/46-modelo-de-dominio?tmpl=component&print=1&page=.

17. [Online] [Cited: Junio 7, 2011.] http://bloqnum.com/pfc/proyecto/node11.html.

18. **James Rumbaugh, Ivar Jacabson, Grady Booch.** *El Lenguaje Unificado de Modelado. Manual de Referencia.*

19. **Dierk Konig, Andrew Glover.** *Groovy in Action.* s.l. : Manning Publications Co, 2007.

20. Adictos al Trabajo. [Online] [Cited: 03 13, 2011.] http://www.adictosaltrabajo.com/tutoriales/tutoriales.php?pagina=grasp.

21. **Glen Smith, Peter Ledbrook.** *Grails in Action.* s.l. : Manning Publications Co, 2009.

BIBLIOGRAFÍA.

1. *Institute, P. M. Guía de los Fundamentos de la Dirección de Proyectos (Guía del PMBOK).* Four Campus Boulevard. Newton Square, Pennsylvania 19073-3299 : Project Management Institute,Inc,, 2004.

2. **Proyecto, Lider de.** Lider de Proyecto. [Online] [Cited: 12 10, 2010.] http://liderdeproyecto.com/manual/que_es_el_pmbok.html.

3. **Expert, Soft.** Soft Expert. [Online] [Cited: 12 10, 2010.] http://www.softexpert.es/norma-pmbok.php..

4. **Pressman, R. S.** *Ingeniería del Software. Un enfoque Práctico.* México, McGraw-Hill : Version 1.1. CMMI. C. M. S. E. Institute. Pittsburgh, Carnegie Mellon Software Engineering Institute., 2002.

5. Real Academia de La Lengua Española. [Online] [Cited: 11 22, 2010.] http://buscon.rae.es/draeI/SrvltConsulta?TIPO_BUS=3&LEMA=inteligencia%20artificial.

6. **Daniel, Gálvez.** *Curso de Sistemas Basados en el Conocimiento.* 1998.

7. **Alarcos.** Ingeniería de Software 1.Tema04.pdf. [Online] [Cited: 11 17, 2010.] http://alarcos.inf-cr.uclm.es/doc/ISOFTWAREI/Tema04.pdf.

8. **Kulik, Peter and Catherine Webber.** *Software Risk Management Practices.* 2001.

9. **Estéves, J and Pastor, A.** *Implementación y Mejora del Método de Gestión Riesgos del SEI en un proyecto universitario de desarrollo de software.* 2005.

10. *Norma Técnica peruana NTP-ISO/IEC 12207.* **INDECOPI.** 2006.

11. **UQAM.** Welcome to the study on The Reality of Project Management Practice 2007. [Online] Diciembre 20, 2010. http://www.surveymonkey.com/DisplaySummary.asp?SID=1993451&U=199345126557.

12. *Revista Española de Innovación, Calidad e Ingeniería de Software.* **Yeleny Zulueta, Eder Despaine.** No. 3, Barcelona : s.n., 2009, Vol. Volume 5.

13. **Lio, Gálvez Daniel.** *Curso de Sistemas Basados en el Conocimiento.* 1998.

14. **Coad P., Lefebvre E. ,De Lucas.** *Metodologías de Desarrollo(FDD). Java Modeling In Color With UML.* Prentice Hall : Enterprise Components and Process, 1999.

15. *Guía a Rational Unified Process.* **Martínez, Alejandro and Martínez, Raúl.** Universidad de Castilla la Mancha : Escuela Politécnica Superior de Albacete.

16. [Online] [Cited: Junio 7, 2011.] http://migueljaque.com/index.php/tecnicas/tecnicasmodnegocio/37-modelado_negocio/46-modelo-de-dominio?tmpl=component&print=1&page=.

17. [Online] [Cited: Junio 7, 2011.] http://bloqnum.com/pfc/proyecto/node11.html..