Universidad de las Ciencias Informáticas

FACULTAD 6

"PredictNC: Sistema Experto para la predicción de no conformidades en la Actividad de Desarrollo-Producción UCI"

Trabajo de Diploma para optar por el título de

Ingeniero en Ciencias Informáticas

Autores: Rainer González-Quevedo Ávila

Randy Lahera Pestana

Tutor: Ing. Alberto Mendoza Garnache

La Habana, Julio de 2016

"Año 58 de la Revolución"

"Entre los grandes placeres que nos brinda la vida,
está el superar las dificultades paso a paso,
escalando uno a uno los peldaños del éxito, concibiendo
nuevos deseos y viendo los realizados"

Samuel Johnson

Declaración de autoría

Declaramos ser autores de la presente tesis y reconocemos a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

Para que así conste firmo la presente a los ____ días del mes de julio del año 2016.

Rainer González-Quevedo Ávila

_____

Firma del autor

Randy Lahera Pestana

_____

Firma del autor

Ing. Alberto Mendoza Garnache

_____

Firma del tutor

Autores:

Rainer González-Quevedo Ávila

Universidad de las Ciencias Informáticas, La Habana, Cuba

Email: ravila@estudiantes.uci.cu

Randy Lahera Pestana

Universidad de las Ciencias Informáticas, La Habana, Cuba

Email: rlahera@estudiantes.uci.cu.

Tutor:

Ing. Alberto Mendoza Garnache

Universidad de las Ciencias Informáticas, La Habana, Cuba

Email: agarnache@uci.cu.

De Randy:

Agradezco mis padres por haberme traído a la vida, sin ellos nada de esto hubiese sido posible, a mis segundos padres mis abuelos Belkis y Melchor que me criaron desde que tenía uso de razón y me han dado el amor que necesité a lo largo de todos estos años. A mi abuela Elvira por siempre estar presente para mí con sus consejos, a mi única y más querida tía Janet por siempre confiar en mí, gracias por nunca perder la fe. A mis hermanos, espero que les sirva como fuente de inspiración al igual que mis dos primitos, a toda mi familia en general, incluyendo a mi madrastra Raysa, los quiero mucho. A todos los amigos que hice en la universidad, cada uno, aunque no lo crean los llevo por siempre en mi corazón, a mis tutores Garnache y Asiel, por su dedicación y paciencia, a todos mis profesores, mi tía de la residencia, mis compañeros de aula, a todos ustedes muchas gracias por hacer mi estancia en esta escuela mucho mejor.

De Rainer:

A mi mama linda y hermosa, por haber sido madre, ternura, pasión y haber depositado en mí toda la confianza que necesitaba en estos 5 años de carrera.

A mi padre por ser ejemplo, guía y compañero antes todas las cosas y por el apoyo infinito que siempre me brinda en los momentos necesarios.

A mi tía Maricela y Enrique por ser segundos padres en todo momento de forma incondicional.

Agradecimientos

A mi prima Arletty por ser hermana y amiga.

A mi abuela Esther por ser madre y amiga

A mi tía Alina y mi abuela Lidia por ser más que familia en todo momento.

A mis hermanos por siempre estar atentos a cualquier ayuda y apoyo que puedan brindar.

A mis primos por la ayuda que siempre me brindan en cualquier materia y por estar siempre disponibles para mí.

A mi compañero de tesis que fue siempre el motor impulsor.

A Garnache, Asiel y Reicel que fueron el apoyo constante para el desarrollo de esta tesis.

A todos los profesores y estudiantes que de una forma u otra ayudaron al desarrollo satisfactorio tanto de la tesis como en otros momentos de estos 5 años de sacrificios.

A la gente del aula por ser amigos y familia, en especial a Yanitza, Elian y Yanara

A la gente del apartamento que con el paso del tiempo convirtieron los pocos momentos de ocio en en cosas especiales y que nunca vamos a olvidar, Barroso, Henry, Juanpi, Miguel Angel y Juan Manuel.

En fin, a toda esa gente que me ayudaron a completar esta tarea de 5 años y a los que no por hacerme más fuerte y decido en la vida.

De Randy:

A mis abuelos, a ustedes les debo todo lo que soy hoy, y este resultado va en nombre de ustedes, los amo.

De Rainer:

Quiero dedicar esta tesis mis padres que son y serán las personas más importantes de mi vida y espero q nunca me falten y en especial a aquellas personas que pensaron que esta tesis no iba a poder salir adelante gracias por ese apoyo que siempre nos dieron.

Resumen

Las no conformidades son irregularidades detectadas en una evaluación del producto o del proceso para desarrollar dicho producto, evaluaciones que forman parte del Sistema de Gestión de la Calidad de la Actividad de Desarrollo-Producción UCI. El área de Aseguramiento de la Calidad de los Procesos y Productos (PPQA, del inglés *Process and Product Quality Assurance*) tiene como objetivos planificar y ejecutar dichas evaluaciones, monitorear las no conformidades y el análisis de tendencias, dichos objetivos tributan a tener una visión del comportamiento de la calidad de los procesos y productos. En la Universidad de las Ciencias Informáticas actualmente se requiere más que una visión del comportamiento, se desea conocer las posibles no conformidades de un proyecto con antelación ahorrando de esta manera tiempo y recursos, contar con dicha información resulta un trabajo engorroso para los especialistas y/o interesados. De manera tal, que se creó PredictNC, un sistema experto basado en casos, que brinda la posibilidad de predecir las posibles no conformidades de un proyecto, así como un análisis de las cantidades de no conformidades por tipo y por área de proceso, permitiendo poder filtrar esas cantidades por los parámetros asociados a las características de los proyectos. Además, cuenta con un gestor de proyectos y no conformidades.

Palabras claves: no conformidades, predicción, sistema experto.

Abstract

Nonconformities are irregularities on an evaluation of the product or process to develop the product, such evaluations are part of the quality management system Activity Development-Production UCI. The area of Quality Assurance Processes and Products (PPQA, English Process and Product Quality Assurance) aims to plan and carry out such assessments, monitor non-conformities and trend analysis, these objectives are taxed to have a vision of behavior quality processes and products. At the University of Information Science, it is more than a vision of behavior currently requires, you want to know the possible nonconformity of a project in advance thus saving time and resources now have this information is a cumbersome job for specialists and / or stakeholders. So that PredictNC, an expert system based on cases, which provides the ability to predict the possible nonconformity of a project and an analysis of the amounts of non-conformities by type and process area was created, allowing power filter those amounts by the parameters associated with the characteristics of the projects. It also has a management which allows to insert, modify, delete, filter and list of projects and their associated non-conformities.

Key words: nonconformities, prediction, expert system.

Índice

Índice de figuras

Índice de tablas

Introducción

A lo largo de la historia el hombre en su búsqueda incansable por alcanzar la perfección ha sido constante. Día a día su afán por conseguir un trabajo bien hecho y la necesidad de comprometerse derivó en el concepto de calidad. La definición de calidad más aceptada en la actualidad es la que compara las expectativas de los clientes con su percepción del servicio. De acuerdo con (García, 2001) *"(…) el desarrollo de la industria de los servicios ha supuesto un desarrollo de una nueva óptica del concepto de calidad que se focaliza más hacia la visión del cliente (…)"*.

El conocimiento generado y adquirido sobre calidad de manera general, fue puesto en práctica a partir de los años 40 en la Informática, específicamente en el desarrollo de software. Según Pressman (Pressman, 1998) *"(…) la calidad de software es la concordancia con los requisitos funcionales y de rendimiento explícitamente establecidos, con los estándares de desarrollo explícitamente documentados, y con las características implícitas que se espera de todo software desarrollado profesionalmente (…)*".

El término calidad no es un concepto aislado en una empresa o proyecto, sino que forma parte de un Sistema de Gestión de la Calidad (SGC). Estos sistemas se encuentran normados bajo la Organización Internacional para la Estandarización (ISO, del inglés *International Organization for Standardization*). Un SGC es un conjunto de normas y estándares ya definidos para hacer cumplir de manera exhaustiva que lo acordado con el cliente tenga la calidad definida mediante un proceso de mejora continua.

Existen en la actualidad varias definiciones de mejora continua, para (Harrington ,1993) un proceso significa cambiarlo para hacerlo más efectivo, eficiente y adaptable. El Modelo Integrado de Madurez de Capacidades (CMMI, del inglés *Capability Maturity Model Integration*) es un enfoque de mejora continua que mide la madurez de un software en una escala del 1 al 5, fue desarrollado por el Instituto de Ingenieros de Software (SEI, del inglés *Software Enginnering Institute*), el mismo tiene como objetivos: producir servicios y productos de alta calidad, crear valor para los accionistas, mejorar la satisfacción del cliente, incrementar la participación en el mercado y ganar reconocimiento en la industria.

En la Universidad de las Ciencias Informáticas (UCI) se aplica el proceso productivo "Actividad de Desarrollo-Producción UCI" sobre la base del modelo de referencia CMMI v1.3 nivel 2. Dicho nivel cuenta con siete áreas, pero por las características con que se desarrolla el software en la UCI solo se aplican seis: Planeación de Proyecto, Monitoreo y Control de Proyecto, Administración de Requisitos, Administración de Configuración, Medición y Análisis y el Aseguramiento de la Calidad de los Procesos y Productos (PPQA, del inglés *Product and Process Quality Assurance*). El proceso de PPQA tiene como

políticas evaluar objetivamente la adherencia a procesos y productos de trabajo, revisar el estado de los procesos, analizar los resultados de las evaluaciones y realizar el análisis de tendencias.

Como parte del proceso, en aras de cumplir con sus políticas, metas, prácticas específicas entre otras, están definidos subprocesos que permiten planificar, organizar y realizar evaluaciones, monitorear las no conformidades detectadas en dichas revisiones como parte del seguimiento y escalamiento de las mismas, de manera que se garantice su resolución y se solucionen las causas que las originaron a través de las acciones correctivas; subprocesos soportados por la herramienta de Gestión de Proyecto GESPRO. Además, se realiza el Análisis de Tendencias, subproceso encargado de recolectar toda la información de las actividades de calidad ejecutadas en el período, efectuándose un análisis de las mismas, con la elaboración del Informe de Tendencias para comunicarlo a la Alta Gerencia y colaborar con la toma de decisiones.

El análisis de tendencia perteneciente al proceso de PPQA se realiza actualmente en la UCI de forma manual, donde se registra en un documento el análisis de varios indicadores asociados a las no conformidades detectadas, el estado de las revisiones, el cumplimiento de las planificaciones, lecciones aprendidas, solicitudes de mejoras, entre otros temas que se informan periódicamente a la Alta Gerencia.

Con los especialistas se analizan las no conformidades en talleres, capacitaciones, entrenamientos y cursos de postgrado, por esta vía resulta un poco difícil y trabajoso realizar un estudio sobre cuáles son las no conformidades que predominan en un proyecto determinado o en un área o etapa de desarrollo, cuáles son las no conformidades que se pudieran prevenir y dar solución antes de que ocurran las mismas. La UCI actualmente debido al desarrollo numeroso de proyectos requiere más que una visión del comportamiento de las no conformidades detectadas, desea conocer las posibles no conformidades de un proyecto con antelación ahorrando de esta manera tiempo y recursos, contar con dicha información resulta un trabajo engorroso y difícil para los especialistas y/o interesados, pues si bien la herramienta de GESPRO permite dar seguimiento y control a las no conformidades no es capaz de prevenirlas.

Por todo lo antes expuesto se plantea como **problema de la investigación**: ¿Cómo contribuir a la predicción de no conformidades en la Actividad de Desarrollo-Producción UCI?
A partir del problema de investigación se puede diferir como **objeto de estudio** de la investigación: Proceso de predicción utilizando algoritmos de inteligencia artificial, enmarcado en el **campo de acción**: Proceso de predicción de no conformidades utilizando algoritmos de inteligencia artificial en la Actividad de Desarrollo-Producción UCI.

La presente investigación tiene como **objetivo general**: Desarrollar un sistema experto para la predicción de las no conformidades en la Actividad de Desarrollo-Producción UCI.
Para guiar la investigación se plantean los siguientes **objetivos específicos**:

- ✓ Realizar la fundamentación teórica de las principales definiciones asociadas a la inteligencia artificial y los sistemas expertos, así como de las herramientas, tecnologías y metodologías a utilizar en el desarrollo del sistema experto para la predicción de las no conformidades en la Actividad de Desarrollo-Producción UCI.
- ✓ Determinar la viabilidad del sistema experto para la predicción de las no conformidades en la Actividad de Desarrollo-Producción UCI.
- ✓ Describir el procedimiento para la obtención del conocimiento de la base de conocimientos para el desarrollo del sistema experto para la predicción de las no conformidades en la Actividad de Desarrollo-Producción UCI.
- ✓ Diseñar el sistema experto para la predicción de las no conformidades en la Actividad de Desarrollo-Producción UCI.
- ✓ Implementar del sistema experto para la predicción de las no conformidades en la Actividad de Desarrollo-Producción UCI.
- ✓ Realizar las pruebas de software al sistema experto para la predicción de las no conformidades en la Actividad de Desarrollo-Producción UCI.

Con el propósito de cumplir con los objetivos planteados fueron formuladas las siguientes **preguntas científicas**:

1. ¿Cuáles son los fundamentos teóricos asociados a la inteligencia artificial y los sistemas expertos?
2. ¿Qué herramientas, tecnologías y metodologías utilizar para el desarrollo del sistema experto para la predicción de las no conformidades en la Actividad de Desarrollo-Producción UCI?
3. ¿Qué características debe poseer sistema experto para la predicción de las no conformidades en la Actividad de Desarrollo-Producción UCI para que cumpla con las necesidades del cliente?

4. ¿Cómo construir el sistema experto para la predicción de las no conformidades en la Actividad de Desarrollo-Producción UCI de forma tal que constituya una herramienta de apoyo en la toma de decisiones?

5. ¿Cómo probar el sistema experto para la predicción de las no conformidades en la Actividad de Desarrollo-Producción UCI para garantizar que todas sus funcionalidades se ejecuten correctamente?

Para cumplir con lo antes descrito en la investigación se han trazado las siguientes **tareas de la investigación**:

1. Selección de las herramientas y metodologías a utilizar para guiar el desarrollo del sistema experto para la predicción de las no conformidades en la Actividad de Desarrollo-Producción UCI.

2. Selección de los algoritmos de clasificación y de selección de atributos para validar la base de conocimientos

3. Evaluación del sistema experto para determinar su viabilidad y la motivación para el esfuerzo.

4. Aplicación de técnicas de obtención de la información para conformar la base de conocimientos.

5. Interpretación y análisis del conocimiento para identificar los requisitos del sistema.

6. Diseño de la arquitectura del sistema experto para definir las capas que componen la solución.

7. Diseño del diagrama de clases para identificar las clases existentes y sus relaciones.

8. Diseño del modelo de datos para identificar las tablas y sus relaciones.

9. Selección de las técnicas de representación del conocimiento y de control para el desarrollo del sistema experto.

10. Selección del estándar de codificación para guiar la implementación del sistema experto.

11. Implementación de la base de conocimientos y las interfaces del sistema experto

12. Aplicación de pruebas de software al sistema experto para comprobar su correcto funcionamiento.

En el desarrollo del presente trabajo se utilizan los siguientes **métodos de investigación:**

Métodos teóricos

- **Analítico-Sintético**, el cual permite realizar un análisis de elementos claves del Modelo de Desarrollo-Producción UCI como son el proceso de aseguramiento de la calidad de procesos y productos, sus respectivos subprocesos y el sistema de gestión de la calidad en sí; la síntesis se produce sobre la base de los resultados del análisis de tendencias registrado en el Informe de Tendencias a los niveles de Entidad Desarrolladora y Alta Gerencia. Además, dicho método permitirá realizar un análisis del impacto de los resultados del sistema desarrollado como parte de la validación y cumplimiento de los objetivos de la presente investigación.

- **Modelación**, método mediante el cual se generan abstracciones tales como los modelos, diagramas, artefactos o productos de trabajos que permiten entender las necesidades, definir metas, objetivos, procesos como parte del desarrollo del sistema de gestión de conocimiento e incluso explicar la solución que se propone en la investigación. Se utilizó principalmente en el modelado del diagrama de casos de uso del sistema, el modelo conceptual, los diagramas de clases del diseño, el diagrama de componentes, entre otros.

Métodos empíricos

- **Entrevistas y Análisis de documentos**, combinación de gran utilidad en la recogida de información durante los procesos de la ingeniería de requisitos que se apliquen en la investigación y en la validación a la hora de analizar el impacto de los resultados de la misma. Las entrevistas fueron aplicadas a los especialistas del área de calidad para obtener la información necesaria para el desarrollo del sistema experto. El análisis de documentos se realizó para consolidar los conocimientos asociados a la inteligencia artificial, sistemas expertos y otros temas presentes en la investigación.

El presente documento, está estructurado en 3 capítulos, a continuación, se muestra una breve descripción de cada uno de estos:

Capítulo 1. Fundamentación teórica para la creación de un sistema experto

En este capítulo se exponen varios conceptos referentes a la inteligencia artificial, se realiza un estudio de los sistemas expertos, sus ventajas y arquitectura, así como de algunos sistemas expertos utilizados en la predicción como apoyo en la toma de decisiones y de los algoritmos de clasificación y de selección de

atributos. También son descritas varias metodologías para el desarrollo de sistemas expertos y es argumentada la seleccionada para el desarrollo de PredictNC. Se exponen las características de las herramientas y tecnologías a utilizar en el desarrollo de la solución, así como de los patrones de diseños estudiados.

Capítulo 2. Análisis y diseño del sistema experto

En este capítulo se realiza una evaluación del sistema experto a desarrollar reflejado en el estudio de viabilidad. Se expone la manera en que se adquiere el conocimiento y las técnicas aplicadas para su análisis e interpretación reflejados en el modelo conceptual y la identificación de los requisitos del sistema. Se diseñan y describen el diagrama de caso de uso, los diagramas de clases del diseño y el modelo de datos. Además, se seleccionan las técnicas de representación del conocimiento y de control aplicadas.

Capítulo 3. Implementación y prueba del sistema experto

En este capítulo se presenta la organización de los componentes del sistema mediante el diagrama de componentes y son descritos cada uno de estos. Se selecciona el estándar de codificación para guiar la implementación. Se describen los elementos que conforman la base de conocimiento y se muestran algunas interfaces del sistema. Se exponen las pruebas realizadas a la base de conocimiento y los resultados arrojados por las mismas, así como las pruebas realizadas al sistema, la cantidad de no conformidades, su tipo y las corregidas.

Capítulo 1. Fundamentación teórica para la creación de un sistema experto

En este capítulo se exponen varios conceptos referentes a la inteligencia artificial, se realiza un estudio de los sistemas expertos, sus ventajas y arquitectura, así como un análisis de algunos sistemas expertos utilizados en la predicción como apoyo en la toma de decisiones y de algoritmos de clasificación y de selección de atributos. También son descritas varias metodologías para el desarrollo de sistemas expertos y es argumentada la seleccionada para el desarrollo de PredictNC. Se exponen las características de las herramientas y tecnologías a utilizar en el desarrollo de la solución, así como de los patrones de diseños estudiados.

1.1 La inteligencia artificial

El término de inteligencia artificial (IA) es muy utilizado actualmente, muchas son las definiciones que autores emiten, pero todas convergen que en esencia es la capacidad para imitar al razonamiento e inteligencia humana, es decir*: "La inteligencia artificial es una de las áreas más fascinantes y con más retos de las ciencias de la computación ya que ha tomado a la inteligencia como la característica universalmente aceptada para diferenciar a los humanos de otras criaturas ya sean vivas o inanimadas, para construir programas o computadoras inteligentes (…)".* (De Ávila, 2008).

1.2 Los sistemas expertos

Los Sistemas Expertos (SE) pueden ser considerados como un subconjunto de la IA (Rossini, 2000). Un SE es un sistema que emplea conocimiento humano capturado en una computadora para resolver problemas que normalmente requieran de expertos humanos. Los sistemas bien diseñados imitan el proceso de razonamiento que los expertos utilizan para resolver problemas específicos.

La introducción de los SE en el mercado mundial ha crecido considerablemente con el transcurso de los años, y uno de los motivos se debe a su amplia gama de utilización. Se define a un SE como un "*sistema computacional que adquiere conocimiento especializado en un campo específico para explotarlo mediante métodos de razonamiento que emulan el desempeño del experto humano en la solución de problemas".* (Bello, 2005).

1.2.1 Ventajas y desventajas de un sistema experto

La información que posee un experto en un área específica contiene un alto valor y a veces es insustituible, sin embargo, en la mayoría de las áreas existen más problemas a resolver que expertos, de

ahí la necesidad y la importancia de los SE. Según Yogesh y Yogyata (Yogesh, et al., 2012) la experticia de los SE presenta notables ventajas frente a la de los expertos humanos como:

- ✓ Permanente y consistente, pues no está relacionado con el estado "físico o mental" del experto.
- ✓ Mayor confiabilidad: pues los especialistas al apoyarse en los SE aumentan su nivel de confiabilidad debido a que el sistema proporciona una segunda opinión e indica que el especialista ha tomado la decisión correcta.
- ✓ Respuesta rápida: hay situaciones que exigen respuestas rápidas, que a veces los humanos no pueden proporcionar por diferentes razones, de modo que un SE en tiempo real resulta una buena elección.

Los SE como toda aplicación informática posee limitaciones y problemas, a continuación, se citan algunos de estos (Peña, 2006):

- ✓ Dominio limitado.
- ✓ No poseen sentido común.
- ✓ Posibilidad de error.
- ✓ No pueden refinar su propia base de conocimientos.
- ✓ Algunos tienen un costo elevado de desarrollo.

1.2.2 Arquitectura de un sistema experto

Existen muchas literaturas y autores que definen varias arquitecturas para un SE, pero todas coinciden en los principales elementos. En la Fig. 1 se muestran los principales componentes donde las flechas representan el flujo de información dependiendo del sentido de las mismas.

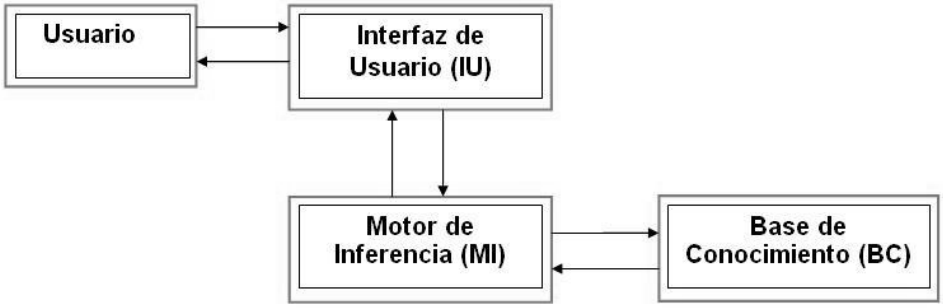

Fig. 1: Arquitectura de un Sistema Experto

- ✓ Interfaz de usuario: proporciona el resultado del motor de inferencia al usuario, así como las explicaciones pertinentes al mismo. Debe proveer un buen mecanismo para que la información que se muestre no sea escasa y sirva de ayuda al usuario.
- ✓ Motor de inferencia: es el núcleo del SE, es un programa que emplea conocimientos de un dominio para solucionar un problema dado.
- ✓ Base de conocimiento: contiene toda la información específica relativa al campo del saber deseado. Está escrita en un lenguaje específico de representación del conocimiento que contiene y en el cual el experto puede definir su propio vocabulario técnico.

1.2.3 Metodologías para el desarrollo de sistemas expertos

En informática, específicamente en el desarrollo de software una metodología es un conjunto de actividades que guían el proceso de desarrollo de software desde su inicio hasta el fin, es decir: *"(…) propuestas metodológicas que inciden en distintas dimensiones del proceso de desarrollo (...)"*, según lo expone el Laboratorio Nacional de Calidad de Software de España. (INTECO, 2009)

Existen muchas metodologías para el desarrollo de sistemas expertos; varios autores han definido sus propias metodologías y las han ajustado a sus necesidades, por lo tanto, no existe una genérica o estándar para la creación de un SE. Tras una revisión bibliográfica se encontraron metodologías tales como: la Metodología de Ingeniería del Conocimiento de John Durkin, Buchanan, Grover, CRIP-DM, entre otras.

En la Tabla 1 se presenta un resumen de algunas de las metodologías estudiadas para el desarrollo de PredictNC, haciendo énfasis en los aspectos que más influyen para la selección de una u otra metodología.

Tabla 1. Comparación de las metodologías estudiadas

Metodología	***Documentación generada***	***Incluye la obtención del conocimiento experto***	***Incluye la fase de Evaluación***	***Tipo de equipo de desarrollo recomendado (en cuanto a cantidad de desarrolladores)***
Buchanan	Mucha	Si	No	Mediano o Grande
Grover	Mucha	Si	No	Grande

CRIP-DM	Poca	Si	No	Mediano o Pequeño
John Durkin	Poca	Si	Si	Pequeño, aunque no necesariamente

Las metodologías de Buchanan y Grover guían el proceso de desarrollo de SE generando una exhaustiva documentación, ya que están pensadas para la construcción de grandes SE, además de que requieren un equipo de desarrollo mediano o grande; por tal motivo la elaboración correcta de toda esa documentación y la aplicación adecuada de cada fase requiere de un personal más numeroso del que se dispone para la construcción de PredictNC. CRIP-DM es una de las metodologías más aplicada en el mundo para la construcción de un SE, sus fases se encuentran bien detalladas y existe una amplia documentación de la misma, pero, no cuenta con una fase capaz de evaluar la viabilidad del proyecto para construir un SE. Estas son las principales razones por la que no se escoge ninguna de las tres para el desarrollo del SE que aborda el presente trabajo. Por otra parte, la metodología propuesta por John Durkin, "*Metodología de Ingeniería del Conocimiento*" genera poca documentación y es aplicable para equipos de desarrollo pequeños, además cuenta con seis fases (siendo la de Evaluación la primera de estas) cada una bien detalladas de manera que no se necesita tener mucha experiencia con la misma y el equipo de desarrollo puede ser pequeño. Por tal motivo se escoge la Metodología de Ingeniería del Conocimiento según John Durkin para la construcción del SE.

En la Fig. 2 se muestra una breve descripción de la Metodología de Ingeniería del Conocimiento según John Durkin (Durkin, 1994). La imagen muestra las seis fases que componen dicha metodología y las relaciones entre cada una de sus fases.

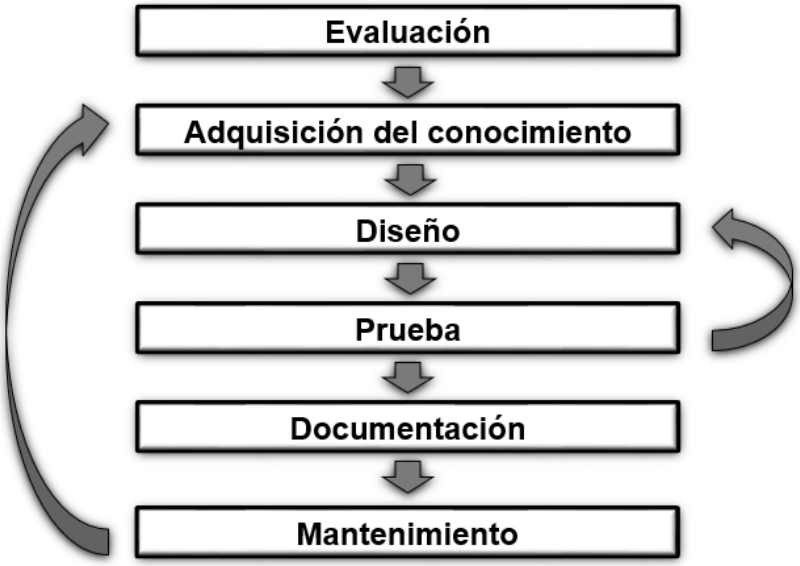

Fig. 2. Metodología de Ingeniería del conocimiento según John Durkin

Si bien todas las metodologías antes mencionadas fueron diseñadas para desarrollar un SE, la calidad del producto que se desea obtener depende de cuan bien se apliquen las fases y tareas que se plantean en cada una de las metodologías.

Es necesario destacar que por las características propias de la investigación no se aplicarán todas las fases ni tareas que propone dicha metodología. A continuación, se describirán las fases y tareas que se ejecutarán para llevar a cabo la construcción del SE.

FASE 1: Evaluación

1.1 Motivación para el esfuerzo: Consiste en determinar ¿Por qué está la organización motivada para seguir Sistemas Expertos? Algunas organizaciones están mirando resolver un problema particular mientras que otras están interesadas en encontrar qué puede hacer la tecnología por ellos. De acuerdo a lo antes mencionado existen dos posiciones que puede asumir una organización al incursionar en la tecnología de Sistemas Expertos: Conducida por el Problema o Conducida por la Solución.

1.2 Estudio de viabilidad: En esta tarea lo primordial es tratar de determinar si el proyecto tendrá éxito. Se consideran dos puntos a evaluar

Primero: Una lista de ítems que debería reunir el proyecto es verificada. Estos ítems incluyen los recursos propios, un recurso de conocimiento y personal del Proyecto.

- ✓ Disponibilidad de conocimiento para la solución del problema.
- ✓ Disponibilidad de un Ingeniero del Conocimiento.
- ✓ Disponibilidad de software de desarrollo de sistema.
- ✓ Disponibilidad de facilidades de computador.

Segundo: Considerar asuntos que son importantes para el éxito del proyecto, pero los cuales son subjetivos de naturaleza y requieren algún juicio para determinar. Estos incluyen características del problema, características de la gente involucrada del proyecto y asuntos de despliegue.

FASE 2: Adquisición del conocimiento

2.1 Recolección del conocimiento: Es la tarea de adquirir conocimiento del experto. Este esfuerzo requiere entrenamiento en técnicas de entrevistas. Además, requiere buenas habilidades de comunicación interpersonal y la habilidad para obtener la cooperación del experto.

2.2 Interpretación: La interpretación de la información recolectada envuelve la identificación de piezas claves del conocimiento, como conceptos, reglas y estrategias.

2.3 Análisis: Envuelve el estudio de las piezas claves del conocimiento durante la tarea de interpretación. Este esfuerzo proporciona la visión de formar las teorías en la organización del conocimiento y estrategias de solución de problemas.

FASE 3: Diseño

3.1 Seleccionar técnica de representación del conocimiento: Consiste en seleccionar la técnica de representación del conocimiento que más se ajuste a las características que posee la información.

3.2 Seleccionar técnica de control: Se seleccionan las técnicas de control, las cuáles pueden ser encadenamiento hacia delante o encadenamiento hacia atrás, la selección de una de estas depende de la existencia o no de una hipótesis a demostrar.

3.3 Desarrollo del prototipo: Se confecciona la base de conocimientos y se aplican los algoritmos de selección de atributos y clasificación.

3.4 Desarrollo de las interfaces: Se implementan las interfaces mediante las cuáles el usuario interactúa con el sistema.

FASE 4: Pruebas

4.1 Validación del Sistema: Un sistema experto modela la decisión de un experto humano. Si se diseñó correctamente, el sistema deriva los mismos resultados que el experto y razona de una manera similar al experto. Por consiguiente, el esfuerzo de aprobación debe dirigirse a lo siguiente:

- ✓ Validar los resultados del sistema.
- ✓ Validar qué proceso razona el sistema (base de conocimiento).

FASE 5: Documentación

5.1 Relación de temas que deben ser documentados: Como un proyecto de sistema experto la cantidad de conocimiento recolectado del experto crece. Después de un tiempo, la cantidad de información es abrumadora. Durante el esfuerzo de desarrollo, se necesitará volver a menudo a la documentación para grabar la nueva información o estudiar previamente la información descubierta. Desde que muchos proyectos requieren un reporte final de proyecto, la información grabada en la documentación sirve como una fuente valiosa. A continuación, se exponen los elementos que deben ser documentados:

- ✓ Conocimiento
- ✓ Gráficos de conocimiento
- ✓ Código fuente
- ✓ Pruebas
- ✓ Reportes.

5.2 Organización de la documentación: Además de contener la información listada en la sección anterior, la documentación debe ser organizada para facilitar el desarrollo del sistema. Para muchos proyectos de sistema expertos necesita escribir un reporte final. Hay variaciones de que será presentado en este reporte que depende de la organización para quien el trabajo fue hecho. El contenido del reporte final del proyecto debe incluir lo siguiente:

- ✓ Página del título
- ✓ Tabla de contenidos.
- ✓ Resumen ejecutivo

- ✓ Visión global del proyecto
- ✓ Descripción del programa
- ✓ Resultados de las pruebas
- ✓ Resumen
- ✓ Referencias
- ✓ Bibliografías
- ✓ Apéndices.

1.2.4 Tipos de sistemas expertos

Determinado por la forma de representación del conocimiento se da lugar a diferentes variantes de SE, entre los más conocidos y empleados en labores de predicción se encuentran: los Sistemas Basados en Reglas (SBR), los Sistemas Basados en Frames (SBF), los Sistemas Basados en Casos (SBC), los Sistemas Basados en Probabilidades (SBP), las Redes Expertas y los Sistemas Basados en Modelos. Para el desarrollo de PredictNC se utilizará un SBC dada las características en que se encuentra la información disponible.

Sistemas Expertos Basados en Casos

El aprendizaje basado en casos consiste precisamente en aprender a partir de experiencias o casos pasados. El aprendizaje basado en casos también se conoce como razonamiento basado en casos, por el hecho de que este tipo de aprendizaje no se concibe sin el proceso de razonamiento que conlleva la obtención de una nueva experiencia. El Razonamiento Basado en Casos (RBC) es el proceso que se usa para solucionar nuevos problemas basándose en las soluciones de problemas anteriores. (Bello, 2002)

Según Bello (Bello, 2002), *"En el RBC, las descripciones de experiencias pasadas de los especialistas humanos, son representadas en casos y almacenadas en una base de conocimiento para su posterior recuperación, los cuales se usan cuando el usuario encuentra un nuevo caso con parámetros similares. El sistema busca en los casos almacenados, casos con características similares al problema que se está tratando de resolver (…)".*

Para comprender mejor este tipo de sistemas se debe conocer que un caso es la definición completa, clara y precisa de las características de un problema particular que lo distinguen de otros problemas y las acciones que se deben tomar para su corrección. (Mestizo, et al., 2010)

Ventajas del uso del razonamiento basado en casos

El razonamiento basado en casos provee numerosas ventajas (Cáceres, 2011):

1. Reduce la tarea de adquisición de conocimiento pues la misma consiste, básicamente, en una colección relevante de experiencias (casos), su representación y almacenamiento.

2. Evita la repetición de errores del pasado, pues los sistemas que guardan tanto los fallos como los éxitos, así como las causas de los fallos, utilizan esta información para predecir fallos futuros.

3. Provee gran flexibilidad en el modelado del conocimiento, pues usan las experiencias pasadas como dominio y dan una solución razonable.

4. Permite al razonador proponer soluciones en dominios que no son del todo entendidos por él. Hay dominios que son imposibles de entender completamente, a menudo porque dependen del comportamiento humano impredecible, por ejemplo, la economía. Sin embargo, el RBC permite tomar ciertas premisas y predicciones basándose en lo que funcionó en el pasado.

5. Permite hacer predicciones del posible éxito de una solución propuesta. Cuando la información se almacena teniendo en cuenta el nivel de éxito de las soluciones previas, el razonador basado en casos puede ser capaz de predecir el éxito de una solución propuesta para el problema actual.

6. Aprende con el tiempo. A medida que los razonadores basados en casos son usados, encuentran más situaciones de problemas y crean soluciones por lo que el sistema tendrá más variedad de situaciones, grado de refinamiento y éxito.

7. Se proponen soluciones a problemas rápidamente en dominios que requieren un gran procesamiento para crear una solución.

8. Proveen un medio de justificación. El razonamiento basado en casos puede dar un caso previo y su solución (con éxito) para convencer al usuario, o justificar, una solución propuesta al problema actual. Si el usuario deseara una medida de calidad de la solución, el sistema podría cuantificar cuanto éxito tuvo el caso pasado y qué grado de similitud hay con el caso actual.

9. Es un reflejo del razonamiento humano, ya que los seres humanos usan una forma de razonamiento basado en casos. Esto es una gran ventaja a la hora de poder entender el funcionamiento del sistema, así como la justificación de una solución propuesta por un sistema basado en casos.

Inconvenientes del uso de RBC

1. Puede haber una tendencia a usar los casos previos ciegamente, confiando en la experiencia previa sin validarla en la nueva situación.

2. Los casos previos pueden predisponer demasiado al razonador a la hora de resolver un nuevo problema.

Confiar en experiencias previas sin validarlas puede generar soluciones y evaluaciones ineficientes o incorrectas. La recuperación de casos inapropiados puede costar un tiempo considerable o llevar a errores muy costosos. El uso de sistemas basados en casos debe servir de apoyo o ayuda a la toma de decisiones, nunca para suplantar al experto humano, pues este es quien debe tomar la decisión final basado en el resultado que le brinda el sistema experto, considerando siempre que los mismos pueden presentar una probabilidad de error.

1.2.5 Algoritmos de selección de atributos

Un caso está compuesto por sus atributos, pero no siempre todos los atributos que lo componen tienen la misma relevancia, por lo que es necesario determinar cuáles son los atributos de mayor relevancia. Los algoritmos de selección de atributos estudiados fueron los siguientes:

- ✓ CfsSubsetEval: Evalúa un subconjunto de atributos considerando la habilidad predictiva individual de cada variable, así como el grado de redundancia entre ellas. Se prefieren los subconjuntos de atributos que estén altamente correlacionados con la clase y tengan baja intercorrelación. (Hall, 1998)
- ✓ CorrelationAttributeEval: Evalúa el valor de un atributo mediante la medición de la correlación (de Pearson) entre ella y la clase. Atributos nominales se consideran en un valor de base, mediante el tratamiento de cada valor como un indicador. Una correlación general para un atributo nominal se llega a través de un promedio ponderado. (Hall, 1998)
- ✓ ReliefFAttributeEval: Evalúa el valor de un atributo mediante el muestreo en varias ocasiones a una instancia y teniendo en cuenta el valor del atributo dado para la instancia más cercana de la misma y a diferentes clases. Puede funcionar tanto en los datos de clases discretas y continuas. (Kenji, et al., 2002)
- ✓ WrapperSubsetEval: Evalúa el valor de un atributo mediante el uso de un esquema de aprendizaje. La validación cruzada se utiliza para estimar la exactitud del sistema de aprendizaje para un conjunto de atributos. (Konavi, et al., 2007)

1.2.6 Algoritmos de clasificación

Una vez definido que el tipo de sistema experto a utilizar es basado en casos y seleccionados los atributos de mayor relevancia, se hace necesario la selección de un método que permita dado un nuevo caso identificar a cuál o cuáles de los casos almacenados en la base de conocimiento es más semejante. A continuación, se describen cuáles fueron estudiados y posteriormente empleados para validar la base de conocimientos.

- ✓ KNN: El algoritmo los K vecinos más cercanos (KNN) es un algoritmo de clasificación supervisada. Este algoritmo trata de clasificar el ejemplo utilizando un conjunto de datos de los cuales ya se conoce su clase. Primero calcula la distancia entre el ejemplo que se quiere clasificar y todos los elementos del conjunto y selecciona los K ejemplos con menor distancia. La clase para el ejemplo que se quiere clasificar será la clase que más se repita en esos K ejemplos. (Uriz, 2015)
- ✓ K *: Es un clasificador basado en instancia, la clase de prueba se basa en las instancias de clasificación similares a ella, según lo determinado por alguna función de similitud. Se diferencia de otros basados en instancia en que utiliza una función de distancia basada en la entropía. (Cleary, et al., 1995)
- ✓ LWL: Utiliza un algoritmo basado en instancia para asignar los pesos de instancia que luego son utilizados por un método (*WeightedInstancesHandler)*. Puede hacer la clasificación utilizando Bayes ingenuo o regresión. (Ebie, et al., 2003)

1.2.7 Aplicaciones de los sistemas expertos

Las aplicaciones de un SE son muy diversas, pero muchos autores concuerdan en que el mayor número de SE se encuentran implantados en el ámbito empresarial pues representa una herramienta potencial para manejar grandes volúmenes de información y realizar operaciones numéricas para luego tomar decisiones. Dentro de las áreas más comunes donde se pueden encontrar SE pueden ser: la medicina, la meteorología, la aeronáutica, la informática, entre otras. A continuación en la Tabla 2 se muestran algunos ejemplos de SE utilizados en la predicción en distintas esferas de trabajo y en la Tabla 3 se pueden observar sistemas expertos desarrollados en la UCI.

Tabla 2. Ejemplo de sistemas expertos utilizados en la predicción

Sistema	***Descripción***
Sistema Experto de predicción de Cáncer Prostático a través de muestras de sangre por examen de Antígeno Prostático Específico	Sistema experto de diagnóstico de cáncer de próstata mediante niveles de antígeno prostático específico (PSA) en sangre. La construcción de software contó con el apoyo de los conocimientos de un experto urólogo colombiano.
SAIH	Sistema Automático de Información Hidrológica. Red de recogida de datos de precipitación y de control de caudales especialmente útiles en la zona del mediterráneo.
MEDEX	Pronóstico de vientos en el Mediterráneo. Sistema que usa tecnologías de inteligencia artificial y sistemas expertos para la predicción de vientos huracanados.
PROMETEO	Sistema que utiliza predicciones a baja resolución de modelos numéricos y busca configuraciones análogas.

Tabla 3. Ejemplo de Sistemas Expertos desarrollados en la UCI

Sistema	***Descripción***
Sistema Experto para la gestión de la Base de Conocimientos de NovaDesk	Sistema Experto basado en reglas que permite identificar y diagnosticar los problemas con los que se enfrenta un usuario en su trabajo con el sistema Operativo Nova, mediante la gestión de la base de conocimientos de NovaDesk.
Sistema basado en casos para predecir la ocurrencia de reacciones adversas a medicamentos en la consulta médica	Para predecir una posible Reacciones Adversas a Medicamentos se realiza un Sistema Basado en Casos que le permite al médico consultar qué Reacciones Adversas a Medicamentos puede ocurrir en el paciente ante un nuevo tratamiento.
Sistema basado en casos para la gestión de errores en el Proyecto ERP-Cuba	Sistema que tiene como primicia dar de forma rápida mediante consultas, una solución a problemas que se

	presenten durante el proceso de desarrollo del Proyecto ERP-Cuba.

Tras realizar el estudio de los sistemas expertos que se describieron anteriormente se puede concluir que los desarrollados internacionalmente aportaron una visión más amplia al conocimiento teórico de la construcción de SE de los diplomantes. Los desarrollados en la UCI, ejemplo, el Sistema Experto para la gestión de la Base de Conocimiento de NovaDesk, no es aplicable en el problema de la presente investigación pues la base de conocimiento está representada en reglas, y la base de conocimiento de PredictNC debe ser representada por casos, aunque si contribuyó junto a los demás sistemas a tener una visión de la implementación de la arquitectura de un SE. Los otros dos sistemas (Sistema basado en casos para predecir la ocurrencia de reacciones adversas a medicamentos en la consulta médica y Sistema basado en casos para la gestión de errores en el Proyecto ERP-Cuba) aunque son basados en casos no cuentan con la misma cantidad de rasgos lo que implicaría igualmente modificar la base de conocimiento y adaptarla a la nueva información, además de modificar sus interfaces gráficas.

1.3 Herramientas, tecnologías y lenguaje de programación

1.3.1 Lenguaje de modelado: UML 2.0

El Lenguaje Unificado de Modelado (UML, del inglés *Unified Modeling Language*) se define como un *"lenguaje que permite especificar, visualizar y construir los artefactos de los sistemas de software (…)"* (Booch, et al., 1997). Es un sistema notacional destinado a los sistemas de modelado que utilizan conceptos orientados a objetos y está diseñado para ser usado en herramientas interactivas de modelado visual, que tengan generadores de informes y de código, por ejemplo, Visual Paradigm. UML es utilizado en la fase de adquisición del conocimiento y diseño del SE para realizar modelos conceptuales, diseño de interfaces, diagramas de clases, y modelos de diseño de la base de datos.

1.3.2 Herramienta CASE: Visual Paradigm for UML 8.0

Visual Paradigm for UML es una herramienta CASE (del inglés *Computer Aided Software Engineering*) que utiliza UML como lenguaje de modelado bajo el paradigma Programación Orientada a Objetos, para la ayuda en el proceso de desarrollo de software. Brinda confiabilidad y estabilidad en el desarrollo orientado a objetos a ingenieros de software, analistas y arquitectos que están interesados en la construcción de sistemas a gran escala.

La versión 8.0 soporta el estándar UML en su versión 2.0 y es compatible con equipos de desarrollo de software en la captura de requisitos, análisis de casos de uso, ingeniería de código, modelado de clase y el modelado de datos. (Visual Paradigm, 2013).

1.3.3 Herramienta para la confección de la base de conocimientos: WEKA 3.7.10

WEKA, del inglés *Waikato Environment for Knowledge Analysis* es una plataforma de software para el aprendizaje automático y la minería de datos escrito en Java y desarrollado en la Universidad de Waikato. Se encuentra distribuido bajo la licencia GNU-GPL. El paquete WEKA contiene una colección de herramientas de visualización y algoritmos para análisis de datos y modelado predictivo, unidos a una interfaz gráfica de usuario para acceder fácilmente a sus funcionalidades. (WEKA, 2016).

1.3.4 Lenguaje de programación: Java 1.8

Java es un lenguaje de programación de propósito general, concurrente, basado en clases y orientado a objetos. Las aplicaciones desarrolladas usando Java son compiladas a código bytes y este es interpretado por la Máquina Virtual de Java, lo cual lo define como un lenguaje interpretado (Gosling, et al., 2013). Se escoge debido a que WEKA tiene un fuerte vínculo con Java, y existen bibliotecas que permiten ser exportadas y son compatibles con Java. Además otro aspecto importante es que los desarrolladores poseen experiencias anteriores en el desarrollo de aplicaciones usando Java.

1.3.5 Entorno de desarrollo: NetBeans 8.1

NetBeans es un entorno de desarrollo para que los programadores puedan escribir, compilar, depurar y ejecutar programas. Existe además un número importante de módulos para extender el NetBeans. Es un producto libre y gratuito sin restricciones de uso. La plataforma ofrece servicios comunes a las aplicaciones de escritorio, permitiéndole al desarrollador enfocarse en la lógica específica de su aplicación. Entre las características de la plataforma están: administración de las interfaces de usuario (ej. menús y barras de herramientas), administración de las configuraciones del usuario, administración del almacenamiento (guardando y cargando cualquier tipo de dato) y administración de ventanas. (NetBeans, 2015)

1.3.6 Gestor de Base de datos: PostgreSQL 9.2

PostgreSQL es un sistema de gestión de bases de datos objeto-relacional, distribuido bajo licencia BSD (del inglés*, Berkeley Software Distribution*) y con su código fuente disponible libremente. PostgreSQL utiliza un modelo cliente/servidor y usa multiprocesos en vez de multihilos para garantizar la estabilidad

del sistema. Un fallo en uno de los procesos no afectará el resto y el sistema continuará funcionando. Entre sus principales características se encuentran que posee completa documentación, integridad referencial, replicación asincrónica y sincrónica, además posee funciones y procedimientos almacenados (del inglés, *stored procedures*) en numerosos lenguajes de programación. (Martínez, 2010)

1.3.7 Administrador de Base de datos: PgAdmin III

PgAdmin III es una aplicación gráfica para gestionar el gestor de bases de datos PostgreSQL. Está escrita en C++ usando la librería gráfica multiplataforma wxWidgets, lo que permite que se pueda usan en Linux, FreeBSD, Solaris, Mac OS X y Windows. Es capaz de gestionar versiones a partir de la PostgreSQL 7.3 ejecutándose en cualquier plataforma. Está diseñado para responder a las necesidades de todos los usuarios, desde escribir consultas SQL simples hasta desarrollar bases de datos complejas. La interfaz gráfico soporta todas las características de PostgreSQL y facilita enormemente la administración. (PgAdmin, 2010)

1.4 Patrones de diseño

Patrones GRASP

En los patrones GRASP (del inglés *General Responsibility Assignment Software Patterns*), se plantean los principios generales para la asignación de responsabilidades a objetos (Larman, 1999). Ellos son:

- ✓ Experto: Define cuándo determinado objeto debe cumplir cierta responsabilidad (ejecutar una acción, activar un estado, etc.). Para la ejecución del sistema en su totalidad todas las clases deben colaborar a través de mensajes para realizar determinadas acciones, cuál debe realizar cada acción está determinado por la información que posee cada una. Este patrón ayuda a mantener el encapsulamiento, dando soporte a un bajo acoplamiento, lo que favorece la robustez del sistema y el fácil mantenimiento. Además, con la aplicación de estos principios se obtienen definiciones de clases más sencillas y más cohesivas, dando soporte a una alta cohesión.
- ✓ Creador: Ayuda a establecer quién es el responsable de la creación de una instancia de determinada clase. Plantea que una clase *A* debe ser responsable de instanciar una clase *B* si: *A* agrega, contiene o utiliza específicamente los objetos de *B*; *A* registra las instancias de los objetos de *B* o *A* tiene los datos de inicialización que serán transmitidos a *B* cuando este objeto sea creado.

- ✓ Bajo Acoplamiento: Este patrón ayuda a disminuir el acoplamiento (nivel en que una clase está conectada a otras), proporcionando una dependencia escasa entre clases y un aumento de la reutilización. De esta manera se garantiza que no deban realizarse cambios locales en ellas por cambios producidos en las clases afines, que sean menos difíciles de entender cuando estén aisladas y que puedan ser más reutilizables.
- ✓ Alta Cohesión: Desde la perspectiva del diseño orientado a objetos, la cohesión funcional es una medida de cuan relacionadas están las responsabilidades de una clase. Cuando estas realizan funcionalidades estrechamente relacionadas existe una alta cohesión, proporcionando una mejor reutilización y comprensión de la clase en sí.
- ✓ Controlador: Este patrón define cuándo una clase debe atender un evento del sistema. Teniendo en cuenta la alta cohesión y el bajo acoplamiento se deben diseñar clases controladoras que realicen las operaciones para dar respuesta a las acciones que ejecuten los usuarios y a su vez, puedan ser reutilizadas. Con este principio se logra separar la lógica del negocio (dominio) de otras clases (por ejemplo, interfaces) que rara vez son reutilizables por sus características propias.
- ✓ No Hables con Extraños: Este patrón se usa para no acoplar una clase *A* que consume servicios de otra *B*, que es indirecta. Entiéndase por indirecta que no es un objeto propio de la clase *A*, no es un parámetro de una operación en *A,* no es un atributo propio de *A*, no es un elemento de una colección que sea atributo propio de *A* y no es un objeto creado en el interior de la operación en *A*. Con este principio se garantiza no acoplar la clase *A* al conocimiento del objeto indirecto *B*, pues si un objeto conoce las conexiones internas y las estructuras de otros entonces presenta un acoplamiento alto.

Data Access Object (DAO)

La mayoría de las aplicaciones, tienen que persistir datos en algún momento, ya sea serializándolos, guardándolos en una base de datos relacional, o una base de datos orientada a objetos, etc. Para hacer esto, la aplicación interactúa con la base de datos. El "cómo interactúa" no debe ser asunto de la capa de lógica de negocio de la aplicación, ya que para eso está la capa de persistencia, que es la encargada de interactuar con la base de datos. Sabiendo esto, se puede decir que DAO es un patrón de diseño utilizado para crear esta capa de persistencia. DAO es un patrón de diseño que encapsula el acceso a la base de datos, por lo que cuando la capa de lógica de negocio necesite interactuar con la base de datos, va a hacerlo a través de la API que le ofrece DAO. Generalmente esta API consiste en métodos CRUD (*Create*, *Read*, *Update* y *Delete*) y se generarán tantas clases DAO como entidades existan en el modelo.

1.5 Conclusiones del capítulo

Tras el estudio realizado de la inteligencia artificial y los sistemas expertos se logró tener una mejor visión de la arquitectura de los SE, así como de los algoritmos de IA. Para la creación del sistema PredictNC se definió como metodología de desarrollo la propuesta por John Durkin ya que se ajusta a las características del equipo y del sistema. Se seleccionó el lenguaje de modelado UML porque mediante dicho lenguaje se pueden crear diagramas que modelen cualquier sistema orientado a objetos. Se determinó emplear la herramienta WEKA 3.7.10 para confeccionar la base de conocimientos, dicha herramienta se encuentra desarrollada en Java lo que permite una integración entre las mismas. Para el desarrollo de las interfaces fue utilizado el lenguaje de programación Java en su versión 1.8 y el entorno de desarrollo NetBeans 8.1. Se seleccionaron como Sistema Gestor de Base de Datos PostgreSQL 9.2 y PgAdminIII como administrador de base de datos.

Capítulo 2: Análisis y diseño del sistema experto

En este capítulo se realiza una evaluación del sistema experto a desarrollar reflejado en el estudio de viabilidad. Se expone la manera en que se adquiere el conocimiento y las técnicas aplicadas para su análisis e interpretación reflejados en el modelo conceptual y la identificación de los requisitos del sistema. Se diseñan y describen los diagramas de clases del diseño y el modelo de datos. Además, se seleccionan las técnicas de representación del conocimiento y de control aplicadas.

2.1 Evaluación

2.1.1 Motivación para el esfuerzo

La motivación para el esfuerzo consiste en determinar si la solución es conducida por el problema o por la solución. Es conducida por el problema cuando la organización trata de resolver un problema que ya se ha identificado, y conducida por la solución es cuando una organización desea explorar nuevas tecnologías por un interés general o curiosidad. En el caso específico de PredictNC la motivación para el esfuerzo es conducida por el problema donde la UCI desea tener más que una visión del comportamiento de las no conformidades de un proyecto.

2.1.2 Estudio de viabilidad

Una vez identificada la motivación para el esfuerzo se hace necesario determinar cuán viable resulta el desarrollo del proyecto para la construcción del SE, para esto apoyado en la metodología seleccionada se cumplen dos pasos. El primer paso consiste en la verificación de una lista de requerimientos que debe poseer el proyecto los cuales ya fueron mencionados en la descripción de la metodología en el Capítulo 1 de la presente investigación, de los cuáles todos se cumplen. El segundo paso es basado en la técnica de estimación propuesta por John Durkin donde se empieza por formar una lista de temas importantes para considerar. Cada tema es luego asignado un peso (entre 0 y 10) que refleja la importancia de cada tema durante la evaluación de un proyecto dado, los números (entre 0 y 10) son atribuidos a cada tema que refleja el grado de creencia en el tema. Este valor es luego multiplicado por el valor del tema para establecer un puntaje por el tema. Todos los puntajes son luego añadidos y divididos por la suma de los pesos del tema. Este número es limitado entre 0 y 10, y proporciona una estimación de determinación de viabilidad del proyecto. Los valores de “peso” son resultados de la experiencia de consulta de Durkin sobre los esfuerzos de determinación de proyectos anteriores. (Ver Anexo 1).

Una vez aplicada la técnica el proyecto arrojó una viabilidad de 7,53 demostrando que si podía ser desarrollado el SE.

2.2 Adquisición del conocimiento

La adquisición del conocimiento es una de las fases más difíciles para la creación de un SE, debido a todos los problemas que puede presentar la información o el experto a la hora de comunicarla. Esta fase cuenta con tres tareas principales: recolección del conocimiento, interpretación del conocimiento y análisis del conocimiento.

2.2.1 Recolección del conocimiento

Para la creación del SE que aborda el presente trabajo, se debe realizar la elaboración de una BC que almacene todos los casos a ser usados para realizar las predicciones de las posibles no conformidades. Para la creación de la BC que usará el sistema, los especialistas de calidad juegan un papel importante, pues son los que poseen la experiencia y la información relacionada con los casos almacenados en el sistema. Para llevar a cabo el proceso de obtención del conocimiento se contó, específicamente, con la información brindada por el Ing. Alberto Mendoza Garnache, Ing. Dairys Febles Pérez, la herramienta GESPRO y los procesos descritos en http://mejoras.uci.cu. Para la obtención del conocimiento se efectuaron reuniones informales y entrevistas entre el experto y el equipo de desarrollo, donde se quedó especificado que para realizar la predicción de no conformidades era necesario identificar cuales características de un proyecto y de su desarrollo podían aportar un valor significativo para la predicción.

2.2.2 Interpretación del conocimiento

La interpretación de la información recolectada envuelve la identificación de piezas claves de conocimientos, como conceptos y sus relaciones. Para tener un mejor entendimiento del problema se describe mediante un modelo conceptual (ver Fig. 3) los elementos más importantes que están involucrados en el proceso para la predicción de las posibles no conformidades de un proyecto. Un modelo conceptual es una presentación de conceptos en un dominio del problema.

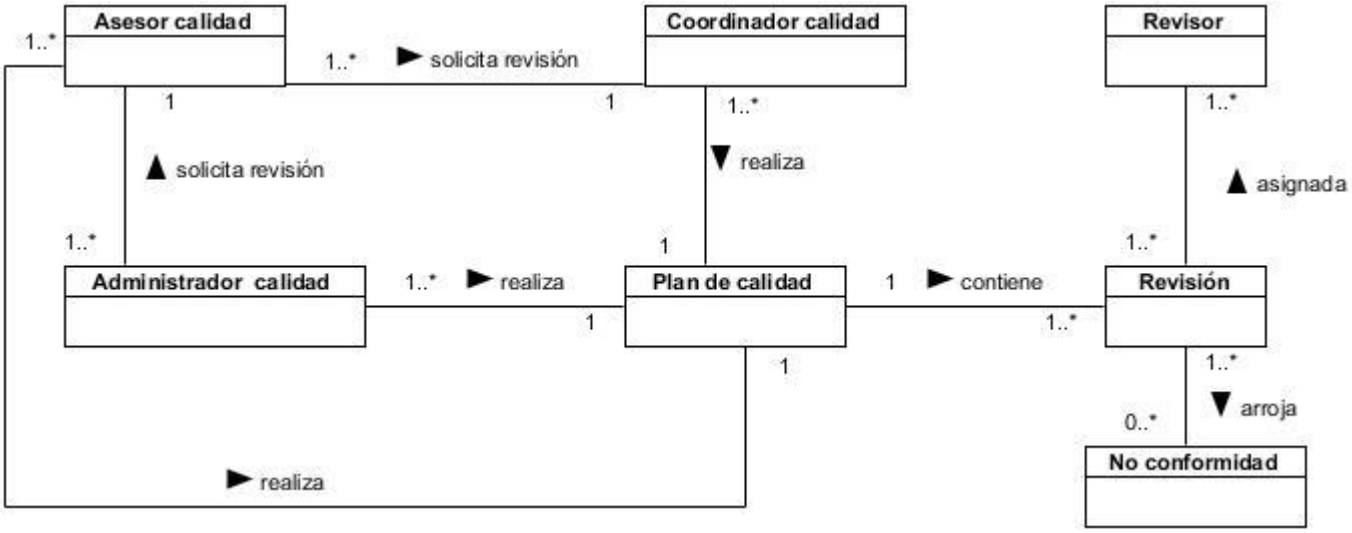

Fig. 3. Modelo conceptual

A continuación, se explican cada uno de los conceptos representados:

- ✓ Coordinador de calidad: Concepto asociado al responsable principal de la Dirección de Calidad encargado de la gestión de todas las actividades relacionadas con el aseguramiento de la calidad a procesos y productos a nivel de Universidad. Tiene como principales responsabilidades: planear las evaluaciones e incluirlas en el Plan de Evaluaciones Gerencial y en el Registro de Evaluaciones Gerencial, asignar recursos humanos para el desarrollo de las evaluaciones, recepcionar solicitudes de escalamiento, realizar los Informes de Tendencias e informar a la Alta Gerencia sobre las solicitudes de mejora, las evaluaciones de desempeño de los revisores y la tendencia de calidad, entre otras.
- ✓ Asesor de calidad: Concepto asociado al encargado de dirigir las actividades relacionadas con el aseguramiento de la calidad de los procesos y productos en la Entidad Desarrolladora. Tiene como principales responsabilidades: planificar las evaluaciones a nivel de la entidad desarrolladora, monitorear el cumplimiento de las acciones correctivas y el estado de las no conformidades, escalar no conformidades que no hayan podido resolverse a nivel de proyecto, entre otras.
- ✓ Administrador de calidad: Concepto asociado a la persona responsable de dar seguimiento a la ejecución de las evaluaciones. Tiene como principales responsabilidades: elaborar el plan de aseguramiento de la calidad, elaborar el plan de pruebas, guiar el diseño y ejecución del control de las pruebas internas, realiza las revisiones de inconsistencia y monitorea las no conformidades hasta su cierre (REQM), entre otras.

- ✓ Plan de calidad: Concepto asociado a un documento que contiene las evaluaciones planificadas al proyecto las cuáles pueden ser elaboradas por el coordinador de calidad, el asesor de calidad y el administrador de calidad dependiendo del nivel en que sean planificadas. Dichas evaluaciones se mantendrán recogidas y planificadas en la herramienta de gestión de proyectos (GESPRO) en el módulo Alcance y Calidad.
- ✓ Revisión: Concepto asociado a las revisiones del software, conjunto de actividades que suceden como resultado del análisis, el diseño y la codificación y que sirven para depurar las actividades de ingeniería del software. Las mismas están contenidas dentro del plan de calidad. Una revisión, tiene como objetivos: señalar la necesidad de mejoras en el producto, continuar las partes de un producto en las que no es necesaria o no es deseable una mejora, conseguir un trabajo técnico de una calidad más uniforme, o al menos más predecible, que la que puede ser conseguida sin revisiones, con el fin de hacer más manejable el trabajo técnico. Existen varios tipos de revisiones dentro de las que se encuentran las Revisiones Técnicas Formales.
- ✓ Revisor: Concepto asociado a la persona encargada de llevar a cabo las revisiones a los proyectos con el objetivo de detectar no conformidades.
- ✓ No conformidad: Concepto asociado a un incumplimiento de un requisito del sistema, sea este especificado o no, como resultado de una revisión aplicada.

2.2.3 Análisis del conocimiento

Requisitos del sistema

Según Craig Larman (Larman, 2004): *"Los requisitos son una descripción de las necesidades o deseos de un producto. La meta primaria de la fase de requisitos es identificar y documentar lo que en realidad se necesita, en una forma que claramente se lo comunique al cliente y a los miembros del equipo de desarrollo. El reto consiste en definirlos de manera inequívoca, de modo que se detecten los riesgos y no se presenten sorpresas al momento de entregar el producto".* Los requisitos pueden ser funcionales o no funcionales. Los requisitos funcionales *"(…) son declaraciones de los servicios que debe proporcionar el sistema, de la manera en que este debe reaccionar a entradas particulares y de cómo debe comportarse en determinadas situaciones particulares (…)"* (Sommerville, 2005). Por otra parte, los requisitos no funcionales "*son restricciones de los servicios o funciones ofrecidos por el sistema (…)"* (Sommerville, 2005) y pueden ser restricciones sobre el proceso de desarrollo, restricciones de tiempo y estándares. (Leal et al., 2014)

Requisitos funcionales

RF1. Realizar predicción del proyecto

RF2. Mostrar cantidad de no conformidades.

RF3. Insertar proyecto

RF4. Modificar proyecto

RF5. Listar proyectos

RF6. Eliminar proyecto

RF7. Filtrar proyecto

RF8. Insertar NC

RF9. Listar NC

RF10. Modificar NC

RF11. Eliminar NC

Requisitos no funcionales

Los requerimientos no funcionales especifican las propiedades del sistema, como confiabilidad y seguridad. Contribuyen considerablemente a la aceptación del producto por parte del cliente. Normalmente están vinculados a requerimientos funcionales, es decir, una vez que se conozca lo que el sistema debe hacer se puede determinar cómo ha de comportarse, qué cualidades debe tener o cuán rápido o grande debe ser.

Existen múltiples categorías para clasificar a los requerimientos no funcionales, siendo las siguientes, representativas de un conjunto de aspectos que se deben tener en cuenta:

- ✓ **Usabilidad.**
  RNF-1. En la interfaz principal debe haber un menú principal con las opciones que brinda el sistema.
- ✓ RNF-2. En todas las interfaces debe existir la posibilidad de que el usuario pueda cerrar o minimizar la interfaz.
- ✓ RNF-3. En la interfaz principal las opciones del menú sólo se deben activar una vez que se valide la contraseña ingresada por el usuario.

- ✓ RNF-4. Los íconos que presente el sistema deben estar etiquetados para que el usuario se oriente.
- ✓ **Rendimiento.**

  RNF 5. Los tiempos de respuesta deben ser los siguientes: para las predicciones no mayores de 2 segundos y para el filtrado no mayor de 22 segundos.
- ✓ **Legales.**

  RNF-6. Las herramientas y las tecnologías en que estará basada la aplicación informática deben cumplir con las licencias de software libre.
- ✓ **Software.**

  RNF-7. La computadora donde se despliegue la aplicación debe tener instalado el sistema operativo Windows (8, 8.1 o 10) y/o Linux y la Máquina Virtual de Java (1.6 como mínimo).
- ✓ **Hardware.**

  RNF 8. La computadora donde se despliegue la aplicación debe poseer 1 GB de memoria RAM como mínimo, microprocesador Intel(R) Pentium(R) 4 o superior, y 1GB en disco duro disponible.
- ✓ **Seguridad**

  RNF 9. El usuario para acceder al sistema deberá introducir su contraseña.

Diagrama de caso de uso del sistema

El caso de uso (CU) es un documento narrativo que describe la secuencia de eventos de un actor (agente externo) que utiliza un sistema para completar un proceso. (Jacobson, 1992)

Los patrones de casos de usos permitieron agrupar los requisitos funcionales del sistema, con este propósito se aplicó el patrón CRUD Total en el CU Gestionar proyectos y Gestionar no conformidades y Extensión concreta entre los mismos CU, como se muestra en la Fig. 4.

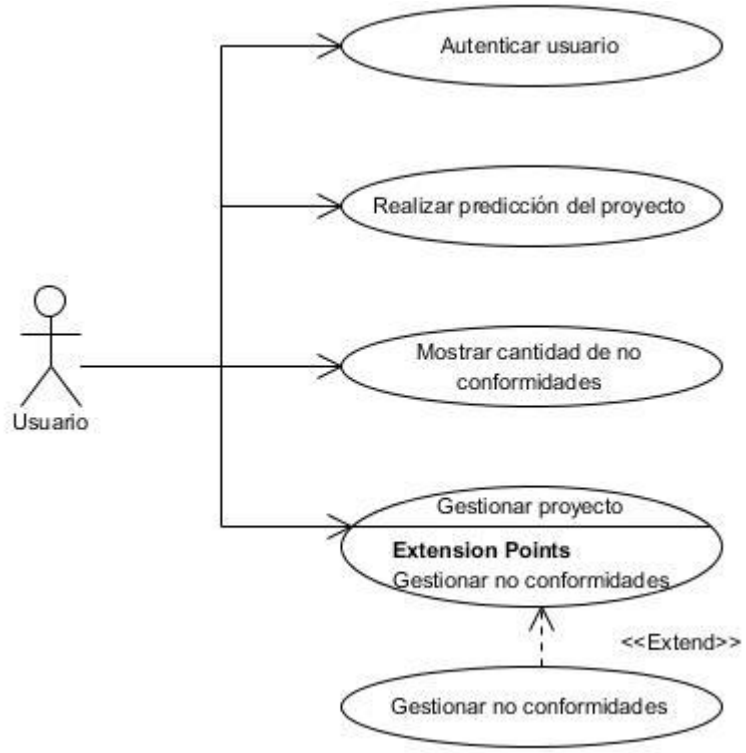

Fig. 4. Modelo de caso de uso

En el diagrama anterior el Usuario inicia los CU *Autenticar usuario, Realizar predicción del proyecto*, *Mostrar cantidad de no conformidades, Gestionar proyecto*, *Gestionar no conformidades y Autenticar usuario.* El primer caso de uso inicia cuando el actor se autentica en el sistema. El segundo se inicia cuando el actor solicita realizar la predicción del proyecto. El tercer CU inicia cuando el usuario realiza una búsqueda de la cantidad no conformidades por tipo y por áreas de proceso. El cuarto CU se inicia cuando el *Usuario* desea gestionar un proyecto permitiéndosele las siguientes acciones: insertar, listar, eliminar, modificar el proyecto seleccionado y además de realizar una búsqueda de proyectos mediante un filtro compuesto de los siguientes atributos: mes inicio, mes final, entidad desarrolladora, estado, modalidad, clasificación según el tipo de cliente y clasificación según su naturaleza. El quinto CU representa la acción del *Usuario* cuando este solicita al sistema *Gestionar No conformidades* permitiéndole al *Usuario* insertar, listar, modificar y eliminar no conformidades de un proyecto deseado. En la Tabla 4 se realiza la especificación formal del CU arquitectónicamente significativo *Realizar predicción*.

Descripción textual de los Casos de Uso

Tabla 4. Descripción textual. CU "Realizar predicción del proyecto"

Actores	Usuario
Resumen	El caso de uso se inicia cuando el usuario selecciona la opción “Realizar predicción”, la cual consiste en determinar las no conformidades que más probabilidades tienen de ocurrir según las respuestas que haya seleccionado en la lista de verificación y finaliza el caso de uso cuando el sistema muestra un listado con las posibles no conformidades que pudiera tener el proyecto.
Complejidad	Alta
Prioridad	Crítico

Flujo de eventos

Flujo básico “Realizar predicción”

Actor	**Sistema**
1. El caso de uso inicia cuando el actor selecciona la opción “Realizar predicción”.	2. El sistema muestra una interfaz con las preguntas de la lista de verificación.
3. El actor selecciona las respuestas a las preguntas de la lista de verificación y luego selecciona la opción “Predecir”.	4. El sistema realiza el proceso de inferencia sobre la BC y clasifica los casos que más se asemejan.
	5. El sistema muestra un listado de los proyectos más semejantes y brinda la posibilidad de conocer sus datos y las no conformidades de cada proyecto semejante.

Prototipo de Interfaz

Fig. 5. Pantalla principal

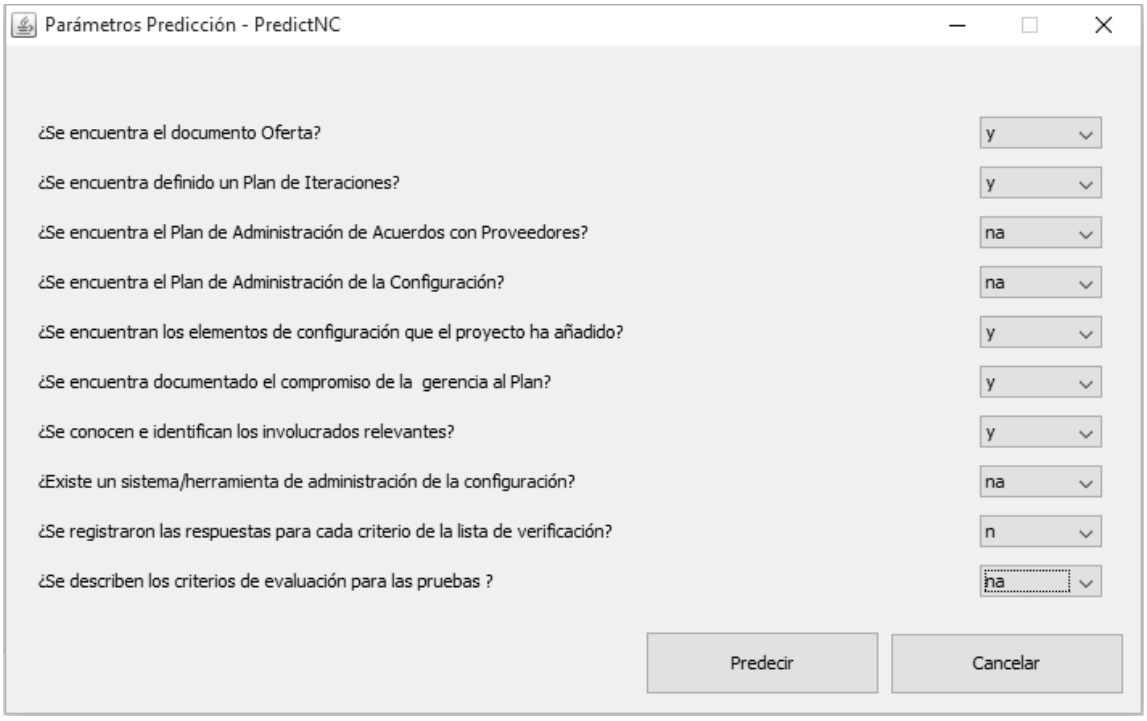

Fig. 6. Prototipo de Formulario de captura de datos

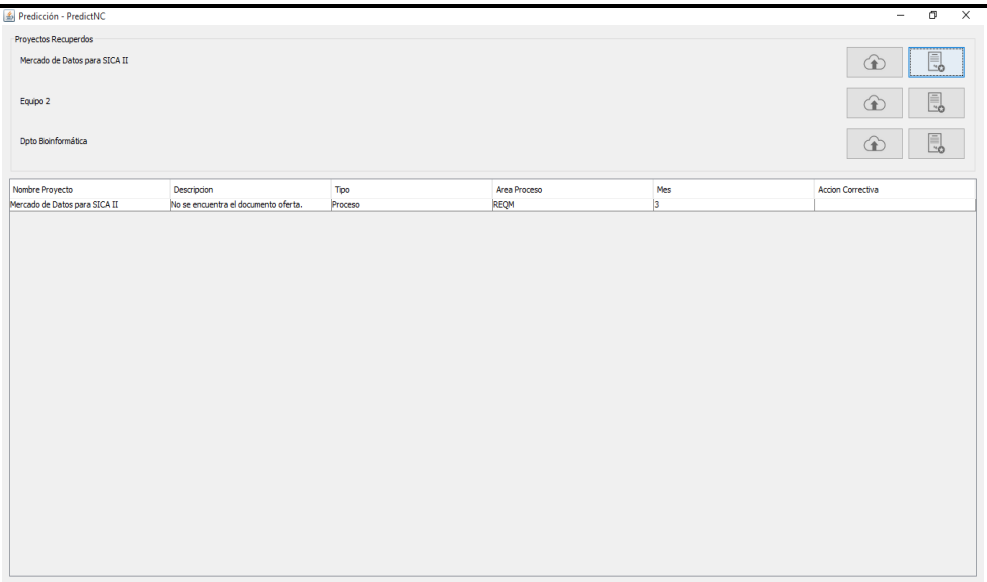

Fig. 7. Prototipo de formulario para el resultado de la predicción

Flujo alterno al paso 3: "Cancelar predicción"	
Actor	**Sistema**
3 a. El actor selecciona las respuestas a las preguntas de la lista de verificación y luego selecciona la opción "Predecir".	3 b. El sistema regresa a la interfaz principal (Ver Fig. 5)

2.3 Diseño

Existen muchas definiciones de arquitectura de software y no es novedad que ninguna definición de la arquitectura de software es respaldada unánimemente por la totalidad de los arquitectos.

Según (Clements, 1996): *"La arquitectura de software es, a grandes rasgos, una vista del sistema que incluye los componentes principales del mismo, la conducta de esos componentes según se la percibe desde el resto del sistema y las formas en que los componentes interactúan y se coordinan para alcanzar la misión del sistema. La vista arquitectónica es una vista abstracta, aportando el más alto nivel de comprensión y la supresión o diferimiento del detalle inherente a la mayor parte de las abstracciones* (…)".

En el presente trabajo se asume como arquitectura de software la planteada por Microsoft:

"*La arquitectura de software abarca el conjunto de decisiones significantes acerca de la organización de un sistema de software incluyendo la selección de elementos estructurales y sus interfaces, por la cual el sistema está compuesto; el comportamiento especificado en la colaboración entre esos elementos; la composición de estos en cuanto a estructura y comportamiento dentro de grandes subsistemas; y un estilo arquitectónico que guía esta organización. La arquitectura de software también involucra funcionalidad, usabilidad, resistencia, rendimiento, reúso, comprensibilidad, restricciones tecnológicas y económicas, equilibrio y preocupaciones estéticas*" (Microsoft, 2016).

2.3.1 Patrón arquitectónico utilizado

Según (Buschmann et al., 1996), *"un patrón arquitectónico expresa esquemas para la organización estructural fundamental para sistemas de software. Ellos proveen un conjunto de subsistemas predefinidos, especifican sus responsabilidades, e incluye reglas y guías para organizar la relación entre ellos (…)"*. También Pressman (Pressman, 2010) plantea que los patrones arquitectónicos para el software definen un enfoque específico para el manejo de alguna característica de comportamiento del sistema.

Patrón arquitectónico N-Capas

Las capas son agrupaciones horizontales lógicas de componentes de software que forman la aplicación o servicio. Ofrece un diseño que maximiza la reutilización y especialmente la mantenibilidad. Se trata de aplicar el principio de separación de responsabilidades (*SoC* del inglés *Separation of Concerns principle*) dentro de una arquitectura. (De la Torre et al., 2010)

La clave de una aplicación N-Capas está en la gestión de dependencias. En una arquitectura N-Capas tradicional los componentes de una capa solo pueden interactuar entre ellos o con otros componentes de capas inferiores. La utilización de una arquitectura en capas posee grandes beneficios (De la Torre et al., 2010):

- ✓ El mantenimiento de mejoras en una solución será mucho más fácil porque las funciones están localizadas y además las capas deben estar débilmente acopladas entre ellas y con alta cohesión internamente, lo cual posibilita variar diferentes implementaciones o combinaciones de capas.
- ✓ Otras soluciones deberían poder reutilizar funcionalidades expuestas por las diferentes capas, especialmente si se han diseñado para ello.

En la construcción del sistema se aplicó el patrón arquitectónico N-Capas, donde se definieron 3 niveles de abstracción (capas): Modelo, Lógica y Presentación. A continuación se explican cada una de ellas (ver Fig. 8) mediante la vista lógica general del sistema.

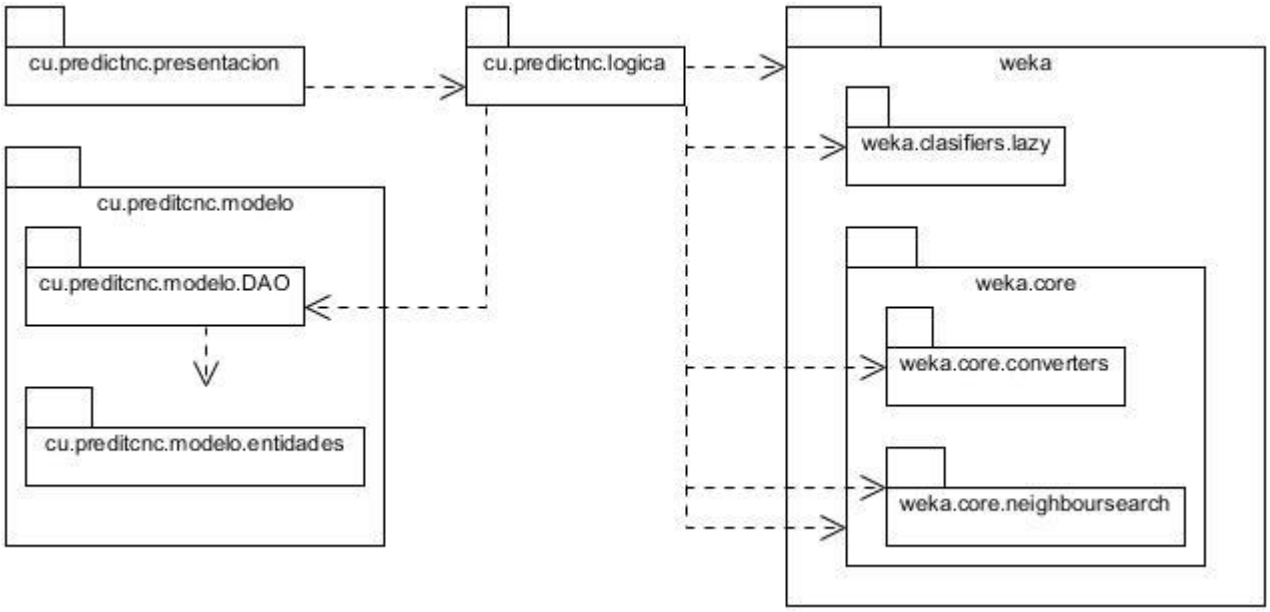

Fig. 8. Vista lógica general del sistema.

En la figura anterior se representaron los siguientes elementos:

- ✓ Capa de Presentación (*cu.predictnc.presentacion*): En este nivel se encuentran las interfaces gráficas de usuario estas acceden a los procedimientos definidos por la capa inferior (Lógica) para obtener los servicios que la misma brinda y mostrarle al usuario las respuestas requeridas una vez realizadas las peticiones al sistema.
- ✓ Capa Lógica (*cu.predictnc.logica*): Esta capa contiene las clases y procedimientos necesarios para dar cumplimiento a los requisitos. Utiliza las funcionalidades brindadas por la librería WEKA para realizar el proceso de predicción, además usa los servicios de la capa Modelo para obtener datos de las no conformidades y de los proyectos cuando se ejecutan procedimientos de búsquedas. También proporciona funcionalidades a la capa Presentación.
- ✓ Capa Modelo *(cu.predictnc.modelo*): Este nivel está compuesto por las clases que representan o manejan la información persistente del sistema, los datos de los proyectos y sus no conformidades, que son mostrados al usuario una vez que se realiza alguna búsqueda, así como los

procedimientos necesarios para acceder a dicha información. Esta capa brinda servicios a la superior (Capa Lógica) y está estructurada en los siguientes paquetes:

cu.predictnc.modelo.DAO: Contiene las clases que interactúan con la Base de Datos ejecutando las consultas necesarias para administrar los proyectos y sus no conformidades.

cu.predictnc.modelo.entidades: Contiene las clases entidades, es decir las clases que son utilizadas para encapsular la información persistente de la base de datos.

- ✓ Librería WEKA: La librería WEKA no constituye una capa en la estructura de la aplicación, pero es usada en la predicción de las no conformidades.

2.3.2 Diagrama de clases del diseño

Un diagrama de clases del diseño (DCD) representa las clases que serán utilizadas dentro del sistema y las relaciones que existen entre ellas. Se utiliza para visualizar las relaciones entre las clases que involucran el sistema, las cuales pueden ser asociativas, de herencia, de uso y de convencimiento. Un diagrama de clases está compuesto por los siguientes elementos: Clase: atributos, métodos y visibilidad. Relaciones: Herencia, Composición, Agregación, Asociación y Uso. A continuación, en la Fig.. 9 se muestra el DCD del CU *Realizar predicción del proyecto.*

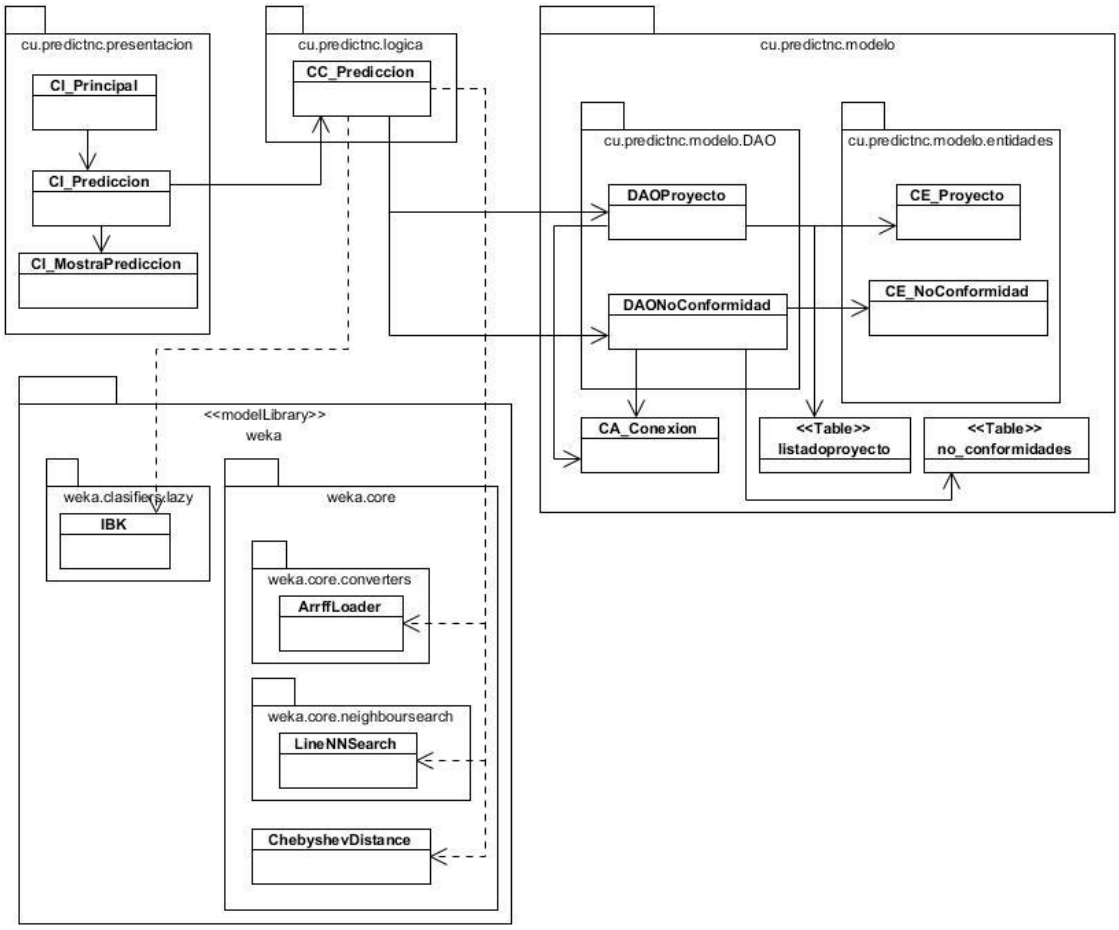

Fig. 9. Diagrama de clases del diseño. CU "Realizar predicción del proyecto"

La clase interfaz *CI_Prediccion* crea un objeto de tipo CC_Prediccion que mediante el método *Write()* guarda los valores de inferencia (respuestas de la lista de verificación) y el método *KNNPrediccion()* devuelve el id de los proyectos utilizando el algoritmo KNN (IBK) de la librería WEKA. Una vez obtenido estos id y el proceso de inferencia haya terminado, se buscan los datos de los proyectos y de las no conformidades en la base de datos a través de las clases *DAOProyecto y DAONoConformidad*. Posteriormente se devuelven en forma de objetos mediante las clases *CE_Proyecto y CE_NoConformidad*. Después de obtenida la información, en la clase interfaz *CI_MostrarPrediccion* se muestran los proyectos que fueron recuperados por el proceso de inferencia y las no conformidades relacionadas a los mismos.

En el anterior diagrama no se incluyeron las operaciones de las clases, para consultar una descripción más detallada del mismo, así como de los restantes diagramas de clases del diseño, debe remitirse al expediente de proyecto, específicamente a la plantilla *Modelo_de_diseño_PredictNC.doc*.

Patrones aplicados

Patrones GRASP

Como buenas prácticas de desarrollo de software, para la construcción de los diagramas, se tuvieron en cuenta los patrones GRASP y GoF, garantizando un correcto diseño de la interacción de los objetos del sistema, así como la asignación de responsabilidades. A continuación, se mencionan los patrones aplicados y su impacto en el sistema.

- ✓ Experto: CC_Proyecto, tiene asignada la responsabilidad de efectuar las operaciones relacionadas con insertar, modificar, listar y filtrar proyectos.
- ✓ Creador: CC_Proyecto es creadora de DAOProyecto, dado que en CC_Proyecto se crea una instancia de la clase DAOProyecto. Garantizando así un bajo acoplamiento.
- ✓ Bajo Acoplamiento: En el sistema se logra un bajo acoplamiento aplicando los principios de este patrón y gracias a la aplicación de los demás patrones relacionados (Experto, Creador).
- ✓ Alta Cohesión: En el sistema se tuvo en cuenta los principios planteados por este patrón, pues las clases realizan solamente funciones relacionadas entre sí o enmarcadas en un área específica, por ejemplo la clase *CC_Prediccion*, presenta funcionalidades relacionadas con el manejo de la librería WEKA, la clase *DAOProyecto* posee funcionalidades relacionadas con las consultas a la base datos, cada clase está destinada a la realización de tareas afines. Esto mejora la claridad y facilidad para entender el diseño, propicia que se simplifique el mantenimiento y la mejora de funcionalidades y genera además un bajo acoplamiento.
- ✓ Controlador: En el SE diseñado existen cuatro clases controladoras: *CC_Proyecto, CC_NoConformidad, CC_Tendencia* y *CC_Prediccion*, la primera y la segunda se encargan de administrar los proyectos y las no conformidades, la tercera atiende las tareas para mostrar la cantidad de no conformidades por tipo y por áreas de proceso y la última es la encarga de interactuar con la librería para realizar la predicción.
- ✓ No Hables con Extraños: La clase CI_Administrador, que es una interfaz gráfica, para realizar una consulta a la base de datos, en lugar de invocar directamente un procedimiento de la clase DAOProyecto, usa un objeto de la clase CC_Proyecto como intermediario, porque es un objeto directo de ella, ya que CI_Administrador agrega a CC_Proyecto.

DAO

- ✓ Se aplicó por cada clase entidad, de esta manera quedaron dos clases DAO: *DAOProyecto* y *DAONoConformidades.* En la Fig. 10 se puede observar un fragmento de código del método *insertarProyecto* perteneciente a la clase DAOProyecto.

```
public DAOProyecto()
{
   objetoProyecto = null;
}

 public void insertarProyecto(CE_Proyecto proyectoInsertar) throws SQLException
 {
        //Conexion
        this.coneccion = new CA_Conexion();
        this.coneccionTrabajo = this.coneccion.Conectar();

        //Consulta Insertar en la base de datos
        PreparedStatement stmt = coneccionTrabajo.prepareStatement("INSERT "
        + "INTO listadoproyecto(prioridad , nombre_original ,estado,  cliente,"
        + " naturaleza, modalidad, entidad, mes) VALUES(?,?,?,?,?,?,?,?)");
        stmt.setInt(1,proyectoInsertar.getPrioridad());
        stmt.setString(2,proyectoInsertar.getNombreOficial());
        stmt.setString(3,proyectoInsertar.getEstado());
        stmt.setString(4,proyectoInsertar.getCliente());
        stmt.setString(5,proyectoInsertar.getNaturaleza());
        stmt.setString(6,proyectoInsertar.getModalidad());
        stmt.setString(7,proyectoInsertar.getEntidad());
        stmt.setInt(8,proyectoInsertar.getMes());

        stmt.executeUpdate();

        //Desconexion
        coneccionTrabajo.close();
    }
```

Fig. 10. Ejemplo de código de la clase DAOProyecto

2.3.3 Modelo de datos

Diagrama Entidad-Relación

En un diagrama entidad-relación se reflejan las entidades que constituyen información del mundo real a nivel conceptual; en ellas se reflejan los objetos (o entidades persistentes) primarios que se procesan en un sistema y su composición, de los cuales se almacena información persistente, pero desde el punto de vista de cómo se reflejan estos en tablas en la base de datos. Para PredictNC se realizó un modelo de este tipo para reflejar la estructura de la base de datos que contiene toda la información de los proyectos y sus no conformidades, como se observa en la Fig. 11.

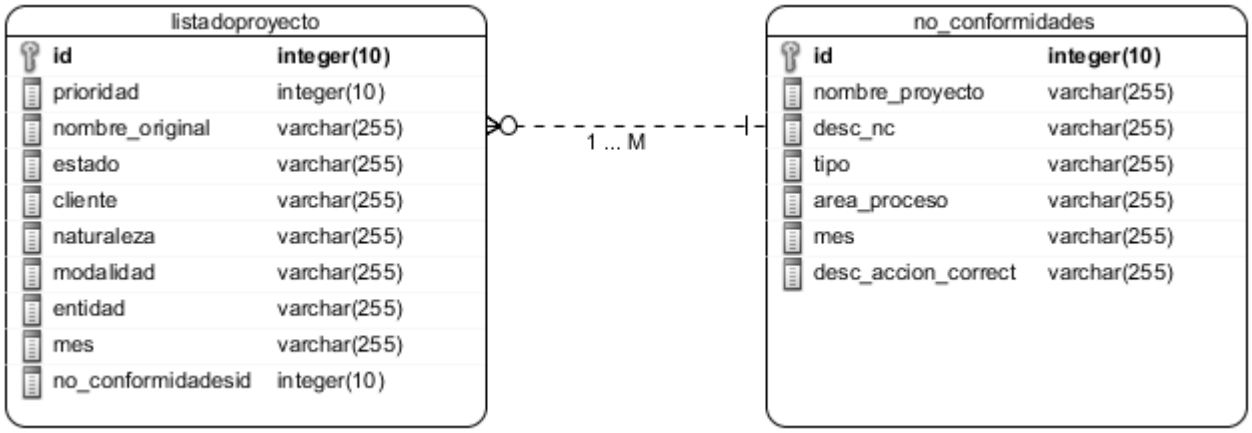

Fig. 11. Diagrama entidad relación del sistema PredictNC.

A continuación, se describen cada una de las tablas:

Tabla 5. Descripción de la tabla listadoproyecto en la base de datos

Nombre: listadoproyecto		
Descripción: Almacena los datos de los proyectos		
Atributo	**Tipo**	**Descripción**
id	integer(10)	Identificador de la entidad en la base de datos.
prioridad	integer(10)	Prioridad del proyecto.
nombre_original	varchar(255)	Nombre original del proyecto.
estado	varchar(255)	Estado en el que se encuentra el proyecto.
cliente	varchar(255)	Tipo de cliente para el cuál se desarrolla el proyecto.
naturaleza	varchar(255)	Naturaleza del proyecto
modalidad	varchar(255)	Modalidad del proyecto
entidad	varchar(255)	Entidad desarrolladora del proyecto
mes	varchar(255)	Mes en que se inicia el proyecto

Tabla 6. Descripción de la tabla no_conformidades en la base de datos

Nombre: no_conformidades		
Descripción: Almacena los datos de las no conformidades de un proyecto		
Atributo	**Tipo**	**Descripción**
id	integer(10)	Identificador de la entidad en la base de datos.
nombre_proyecto	varchar(255)	Nombre del proyecto al cual pertenece la no conformidad
desc_nc	varchar(255)	Descripción de la no conformidad
tipo	varchar(255)	Tipo de no conformidad
area_proceso	varchar(255)	Área de proceso a la que pertenece la no conformidad
mes	varchar(255)	Mes en que fue detectada la no conformidad
desc_accion_correct	varchar(255)	Descripción de la acción correctiva para la no conformidad

2.3.4 Técnicas de representación del conocimiento

Una vez que se adquiere de los especialistas el conocimiento es necesario encontrar un representación simbólica, clara, precisa y concreta del mismo. Existen diversas formas de representar el conocimiento, como las siguientes:

- ✓ Un método basado en marcos es apropiado si el experto describe el problema referenciando los objetos importantes y sus relaciones, particularmente si el estado de un objeto afecta a otro objeto. Esta situación es encontrada en problemas tipo simulación o algunas donde las relaciones causales son importantes. Otra señal que un método basado en marcos puede ser bien escogido es que el experto considere varios objetos similares cuando resuelve el problema. Un sistema basado en marcos puede razonar sobre objetos similares usando solo unas pocas reglas del modelo de emparejamiento que trabajan a través una clase de objetos. Esto proporciona un método eficaz al codificar los objetos y las reglas.

- ✓ Un método basado en reglas es conveniente si el experto discute el problema principalmente usando declaraciones tipo IF/THEN.
- ✓ El método de la inducción es de valor si existen ejemplos pasados del problema. La inducción también es apropiada si no existe ningún experto real en el problema, pero si un historial de información del problema está disponible que puede usarse para derivar los procedimientos de toma de decisión automáticamente.

Por lo anteriormente expuesto se decide utilizar como técnica de representación del conocimiento el método de la inducción ya que es la que se ajusta a las características del problema.

2.3.5 Técnicas de control

Existen dos tipos de encadenamiento, el encadenamiento hacia delante y el encadenamiento hacia atrás. En el encadenamiento hacia atrás se comienza con la definición de las metas del sistema, por lo tanto, se puede decir que no es más que la estrategia de inferencia que intenta probar una hipótesis recolectando información de apoyo. Por otra parte, el encadenamiento hacia delante es apropiado si el experto primero recolecta información sobre el problema y luego ve qué puede ser concluido. Por lo anteriormente descrito se selecciona como técnica de control el encadenamiento hacia adelante.

2.4 Conclusiones del capítulo

El estudio de viabilidad realizado arrojó como resultado que el sistema experto para la predicción de las no conformidades podía ser desarrollado exitosamente. El conocimiento adquirido de los especialistas de calidad permitió la elaboración de casos que conformaron la base de conocimiento, además permitió la identificación de los requisitos del sistema. Se seleccionó el patrón 3-Capas como arquitectura de PredictNC, quedando la misma compuesta por las capas de presentación, lógica de negocio y acceso a datos. Mediante el diseño se logró mostrar gráficamente cómo los objetos se comunican entre ellos a fin de cumplir con los requisitos del sistema, y se modelaron las relaciones entre las tablas y sus atributos mediante el modelo de datos. Además, se seleccionaron las técnicas de representación del conocimiento y de control para el desarrollo del SE.

Capítulo 3: Implementación y prueba

En este capítulo se presenta la organización de los componentes del sistema mediante el diagrama de componentes y son descritos cada uno de estos. Se selecciona el estándar de codificación para guiar la implementación. Se describen los elementos que conforman la base de conocimiento y se muestran algunas interfaces del sistema. Se exponen las pruebas realizadas a la base de conocimiento y los resultados arrojados por las mismas, así como las pruebas realizadas al sistema, la cantidad de no conformidades, el tipo de no conformidad, y las corregidas.

3.1 Diagrama de componentes

Un diagrama de componentes representa cómo un sistema de software es dividido en componentes y muestra las dependencias entre estos. Los componentes físicos incluyen archivos, cabeceras, bibliotecas compartidas, módulos, ejecutables o paquetes. Estos son utilizados para modelar la vista estática y dinámica de un sistema, muestra la organización y las dependencias entre un conjunto de componentes (Ambler, 2014). No es necesario que un diagrama incluya todos los componentes del sistema, normalmente se realizan por partes. En él se situarán librerías, tablas, archivos, ejecutables y documentos que formen parte del sistema.

Uno de los usos principales es que puede servir para ver qué componentes pueden compartirse entre sistemas o entre diferentes partes de un sistema, como se muestra en la Fig. 12.

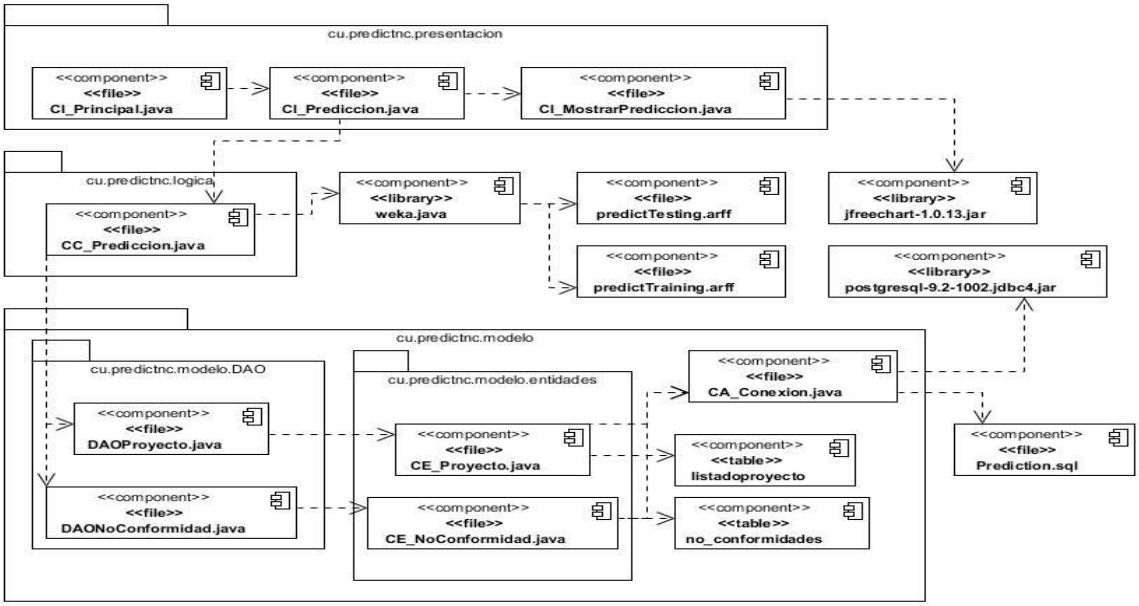

Fig. 12. Diagrama de componente. CU "Realizar predicción del proyecto"

A continuación, se describen cada uno de los componentes:

Clases interfaces

- ✓ CI_Principal.java: contiene la clase interfaz CI_Principal.
- ✓ CI_Predicción.java: contiene la clase interfaz CI_Prediccion.
- ✓ CI_MostrarPrediccion.java: contiene la clase interfaz CI_MostrarPrediccion.

Clases controladoras:

- ✓ CC_Prediccion.java: contiene la clase controladora CC_Prediccion.

Clases entidades:

- ✓ CE_Proyecto.java: contiene la clase entidad CE_Proyecto.
- ✓ CE_NoConformidad.java: contiene la clase entidad CE_NoConformidad.

Otras clases:

- ✓ DAOProyecto.java: contiene la clase DAOProyecto.
- ✓ DAONoConformidad.java: contiene la clase DAONoConformidad.
- ✓ CA_Conexion.java: contiene la clase auxiliar CA_Conexion.

Tablas:

- ✓ listadoproyecto: contiene la tabla listadoproyecto almacenada en la base de datos.
- ✓ no_conformidades: contiene la tabla no_conformidades almacenada en la base de datos.

Archivos y librerías:

- ✓ predictTesting.arff: contiene el archivo generado por WEKA que almacena el conjunto de prueba.
- ✓ predictTraining.arff: contiene el archivo generado por WEKA que almacena el conjunto de entrenamiento.
- ✓ jfreechart-1.0.13.jar: contiene la librería jfreechart encargada de genera los gráficos en Java.

Base de datos:

- ✓ Prediction.sql: contiene la base de datos Prediction generada en el lenguaje PostgeSQL.

3.2 Estándar de codificación

"Un código fuente completo debe reflejar un estilo armonioso, como si un único programador hubiera escrito todo el código de una sola vez (…)" (Microsoft, 2016). Por lo anteriormente expuesto se hace necesaria la selección de un estándar de codificación para el desarrollo de PredictNC; para el cual se eligió el estándar de codificación definido para el lenguaje de programación Java que plantea:

- ✓ Evitar las líneas de más de 80 caracteres.
- ✓ Siempre que sea posible inicializar las variables locales donde se declaran.
- ✓ Poner las declaraciones solo al principio de los bloques.
- ✓ Cada línea debe contener como máximo una sentencia simple (ej.: *c++).*
- ✓ El prefijo del nombre de un paquete se escribe siempre con letras ASCII en minúsculas, y debe ser uno de los nombres de dominio de alto nivel (com, gov) o uno de los códigos ingleses de dos letras que identifican a cada país (cu, es, en). Los subsecuentes componentes del nombre del paquete varían de acuerdo a las convenciones de nombres internas de cada organización.
- ✓ Los nombres de las clases e interfaces deben ser sustantivos, cuando son compuestos tendrán la primera letra de cada palabra que lo forma en mayúsculas.
- ✓ Los nombres de los métodos deben ser verbos, cuando son compuestos tendrán la primera letra en minúscula, y la primera letra de las siguientes palabras que los forman en mayúscula.

En la Fig. 13 se muestra un fragmento del código en lenguaje Java implementado aplicando el estándar de codificación.

```
public Integer[] knnPrediccion(){

//Listado de los id proyectos
Integer[] listadoID = new Integer[3];

try {
            ArffLoader loaderTraining = new ArffLoader();
            ArffLoader loaderTesting = new ArffLoader();

            //Carga de los ficheros de prueba y de entrenamiento
            loaderTraining.setFile(new File(keyDirPredictTraining));
            loaderTesting.setFile(new File(keyDirPredictTesting));

            //Inicializar el DataSet de prueba y de entrenamiento
            Instances train = loaderTraining.getDataSet();
            train.setClassIndex(train.numAttributes() - 1);
            Instances test = loaderTesting.getDataSet();
            test.setClassIndex(test.numAttributes() - 1);

            LinearNNSearch knn = new LinearNNSearch(train);
            knn.setDistanceFunction(new ChebyshevDistance(train));
```

Fig. 13. Ejemplo de código fuente aplicando los estilos de codificación

3.3 Desarrollo del prototipo

El conocimiento obtenido de los expertos se representó en forma de casos mediante la estructura definida por WEKA. Cada pregunta de la Lista de Verificación representa un rasgo de los casos almacenados. Inicialmente se constituyó un prototipo de la Base de Conocimiento (BC) con 11 casos compuestos por 253 rasgos equivalentes a las preguntas de impacto alto (A) de la lista de verificación, para optimizar la predicción del sistema se aplicó el algoritmo CfsSubsetEval permitiendo identificar cuáles rasgos de los 253 inciden directamente con la correcta clasificación de un caso, quedando conformada la BC con un total de 50 casos y cada caso compuesto por 10 rasgos. El resultado final es un conjunto de casos almacenados en un fichero de extensión arff (Ver Anexo 2). Para la clasificación de un nuevo caso se utilizó el algoritmo de clasificación KNN producto de los resultados arrojados en la validación de la base de conocimientos (Ver Tabla 7 y 8).

3.4 Desarrollo de interfaces

Como resultado de la implementación se muestran algunas interfaces del sistema. En la Fig. 14 se muestra los parámetros con los cuales se realiza el proceso de tendencia.

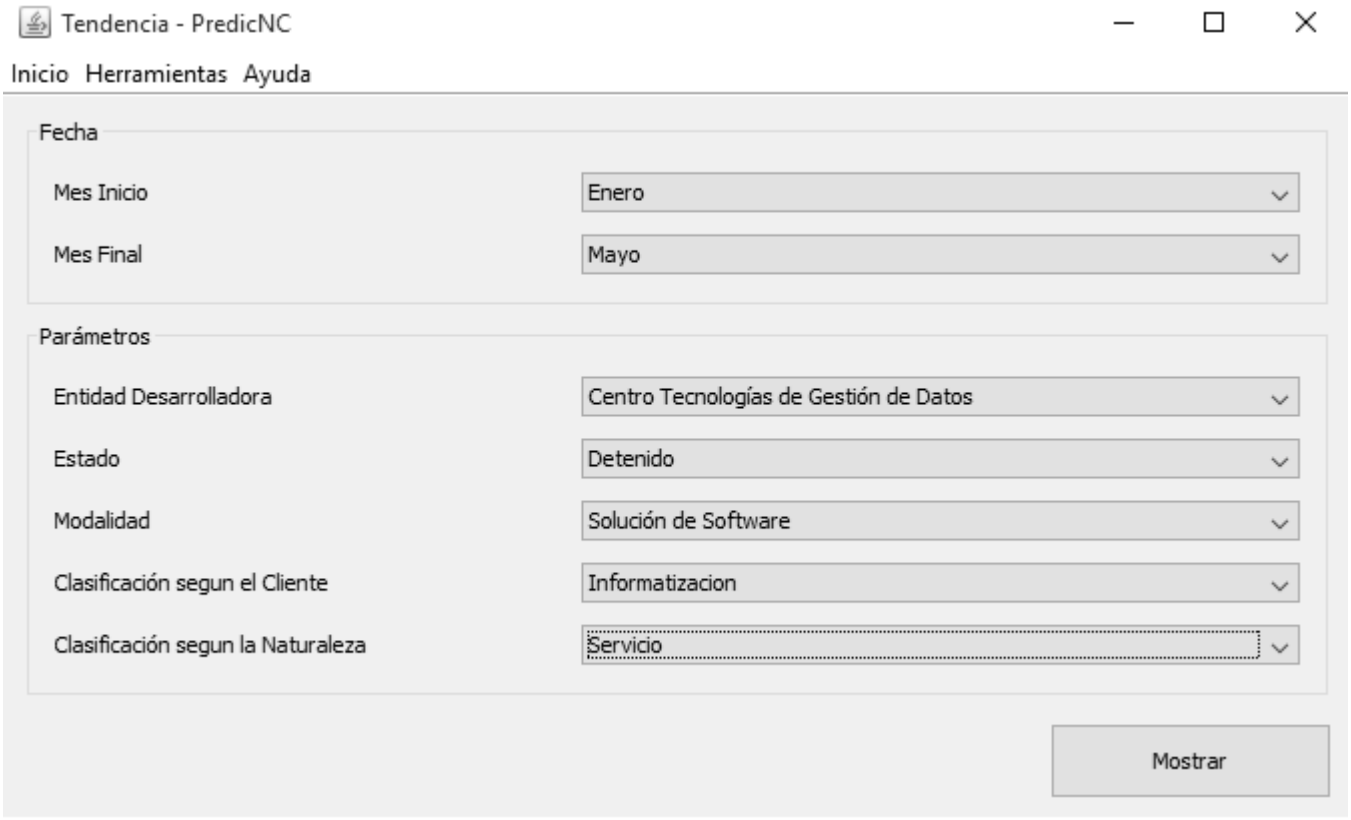

Fig. 14. Interfaz de Tendencia

Luego de seleccionar la fecha y los parámetros se realiza el proceso de tendencia y se muestran la cantidad de no conformidades por tipo y por área de proceso (ver Fig. 15, 16 y 17).

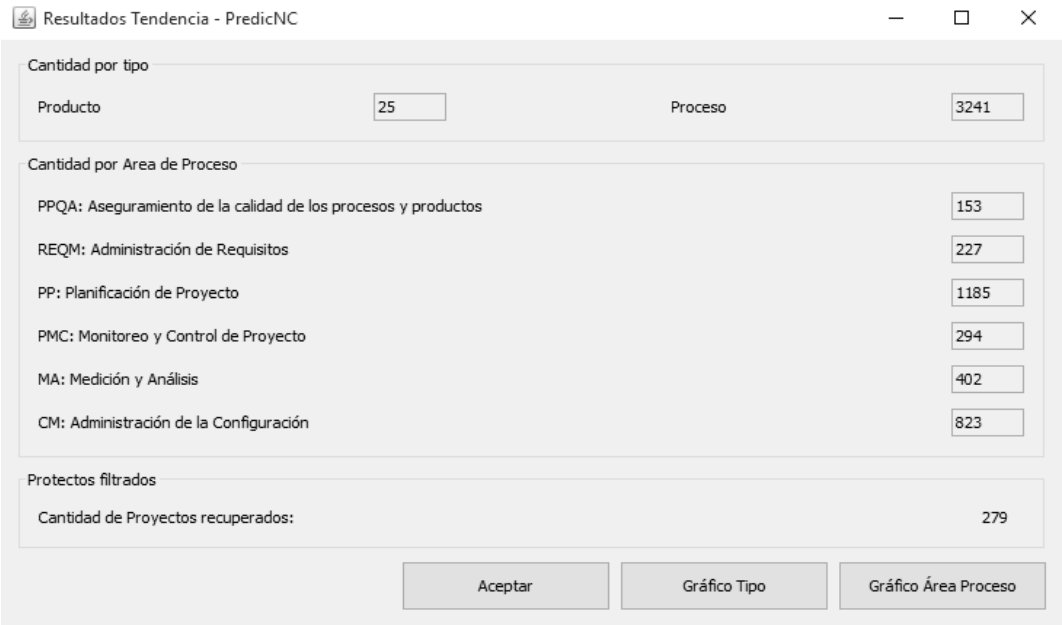

Fig. 15. Resultado del proceso de tendencia

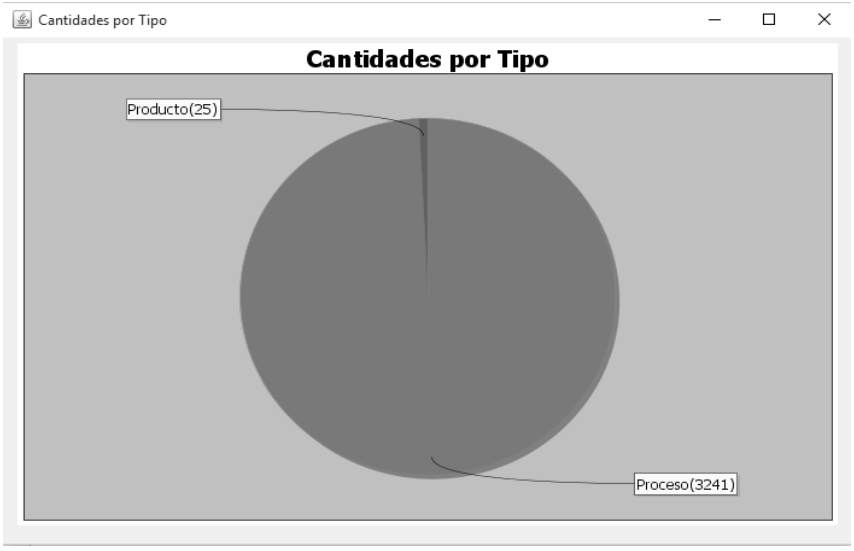

Fig. 16. Gráfica de no conformidades por tipo

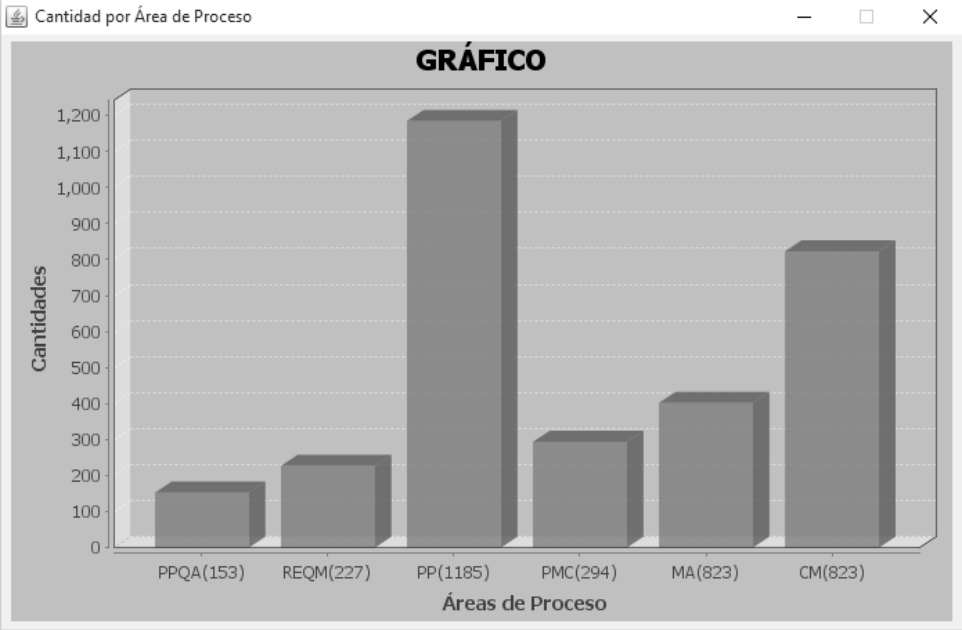

Fig. 17. Gráfico de no conformidades por área de procesos

3.5 Pruebas

Durante el ciclo de desarrollo del software vienen implícitas actividades de producción en las que las posibilidades en que aparezca un fallo humano son enormes. De ahí la relevancia de realizar un proceso de pruebas con calidad. Existen muchos autores que definen sus conceptos de pruebas, así como los niveles, tipos y métodos. Según Pressman (Pressman, 2010): *"La prueba del software es un elemento crítico para la garantía de calidad del software y representa una revisión de las especificaciones, del diseño y de la codificación (…)".*

Para validar el resultado obtenido se verifica el correcto razonamiento de la base de conocimientos mediante la técnica de Validación cruzada y se aplican las pruebas de software en los niveles de unidad y sistema, definiendo además el tipo de prueba realizada, los métodos y sus técnicas correspondientes.

3.5.1 Validación de la base de conocimiento

Para la validación de la base de conocimiento se utilizó la técnica Validación Cruzada (del inglés, *Cross Validation*) la cual permite evaluar los resultados y garantizar que son independientes entre la partición de datos de entrenamiento y prueba. Consiste en repetir y calcular la media aritmética obtenida de las medidas de evaluación sobre diferentes particiones. Se utiliza en entornos donde el objetivo principal es la

predicción y se quiere estimar cómo de preciso es un modelo que se llevará a cabo a la práctica (Schneider, 1997).

En la Tabla 7 y 8 se muestra el resultado de la aplicación de dicha técnica.

Tabla 7. Casos clasificados correctamente para los algoritmos de selección CfsSubsetEval y CorrelationAttributeEval

Algoritmos de clasificación	CfsSubsetEval		CorrelationAttributeEval	
	Probabilidad de error	Casos clasificados correctamente	Probabilidad de error	Casos clasificados correctamente
KNN	**0.3783**	**56%**	**0.3561**	**52%**
K*	0.4478	30%	0.3713	48%
LWL	0.4706	32%	0.3784	48%

Tabla 8. Casos clasificados correctamente para los algoritmos de selección ReliefFAttributeEval y WrapperSubsetEvalAtt

Algoritmos de clasificación	ReliefFAttributeEval		WrapperSubsetEval	
	Probabilidad de error	Casos clasificados correctamente	Probabilidad de error	Casos clasificados correctamente
KNN	0.4787	30%	**0.4160**	**40%**
K*	0.4637	32%	0.4177	42%
LWL	**0.4467**	**32%**	0.4374	26%

En la Tabla 7 se obtuvo como resultado que el algoritmo de clasificación más eficiente es KNN dado que la probabilidad de error es la menor y el porciento de casos clasificados correctamente es el mayor para los algoritmos de selección utilizados. En la Tabla 8 resultó ser más eficiente LWL para el algoritmo de selección ReliefFAttributeEval y KNN para el algoritmo WrapperSubsetEval; de forma tal que los

algoritmos seleccionados fueron KNN para la clasificación de los casos y CfsSubsetEval para la selección de atributos.

3.5.2 Validación del sistema

Los niveles en los cuales fue probado PredictNC fueron Unidad, Sistema y Aceptación. Cuando se considera el software orientado a objeto (OO) el nivel de unidad comprueba el correcto funcionamiento de las clases u objetos encapsulado, ya que la encapsulación conduce a la definición de clases y cada instancia de una clase, envuelven atributos y operaciones que manipulan estos datos. (Pressman, 2010).

Para probar en dicho nivel se utilizó el NetBeans IDE donde se crearon casos de prueba JUnit (del inglés, *JUnit Test Case*), donde se verificó el correcto funcionamiento del procedimiento que responde al CU crítico *"Realizar predicción del proyecto"* mediante la clase *CC_Prediccion*. Se aplicaron valores límites y dentro de los límites y el sistema respondió correctamente (Ver Anexo 3).

A nivel de sistema los detalles de conexiones de clases desaparecen. La validación del software OO se centra en las acciones visibles al usuario y salidas reconocibles desde el sistema, permitiendo comprobar el correcto funcionamiento de las interfaces de usuario y el flujo de entrada/salida de datos. El tipo de prueba aplicado en este nivel fue el de funcionalidad, el método caja negra y la técnica partición de equivalencia. El método de caja negra consiste en elaborar casos de pruebas partiendo de las funcionalidades descritas en los casos de uso del sistema y sus descripciones. En cada caso de prueba se refleja la especificación de un caso de uso, dividido en secciones y escenarios, donde se detallan las funcionalidades descritas en él y se describen las variables utilizadas. En la aplicación de este método se empleó la técnica partición de equivalencia para elaborar los casos de prueba, donde se dividieron los datos de entrada en un conjunto de válidos e inválidos (clases de equivalencias). De esta forma se garantiza que para una condición de entrada se evalúen todos los casos posibles. En la Tabla 9 se describe el caso de prueba para el CU "*Administrar proyecto"* la sección *Insertar Proyecto* y en la Tabla 10 se describen la variable para el mismo caso de prueba.

Tabla 9. Caso de prueba "Administrar proyecto". Sección 1 Insertar Proyecto

Escenario	**Descripción**	**Nombre**	**Respuesta del sistema**	**Flujo central**
EC 1.1 El actor	En este escenario	V	El sistema inserta	1-Adminstrador/Nuevo

inserta todos los datos del proyecto	el actor Inserta un nuevo proyecto con todos los datos en la base de datos	alasSIM	correctamente el nuevo proyecto en la base de datos	Proyecto. 2-Ingresa todos los datos. 3-Selecciona la opción "Guardar".
EC 1.3 El actor no ingresa correctamente todos los datos del proyecto	En este escenario el actor introduce todo los datos pero el campo Nombre incorrecto	I 25	El sistema muestra un mensaje de notificación informando al usuario que ingresó un nombre de proyecto erróneo	1-Adminstrador/Nuevo Proyecto. 2-Ingresa todos los datos. 3-Selecciona la opción "Guardar".

Tabla 10. Caso de prueba "Administrar proyecto". Descripción de las variables

No	Nombre de campo	Clasificación	Valor Nulo	Descripción
1	Datos	Lista	No	Listado de todos los datos del proyecto.

Después de realizadas las pruebas funcionales los errores encontrados fueron redactados como no conformidades para darle seguimiento y solución por el equipo de desarrollo. Se realizaron tres iteraciones en total y se detectaron 10 no conformidades siendo todas resueltas en cada iteración donde se encontraron como se puede observar en la Fig. 18. En la Tabla 11 se muestran algunas de las no conformidades encontradas.

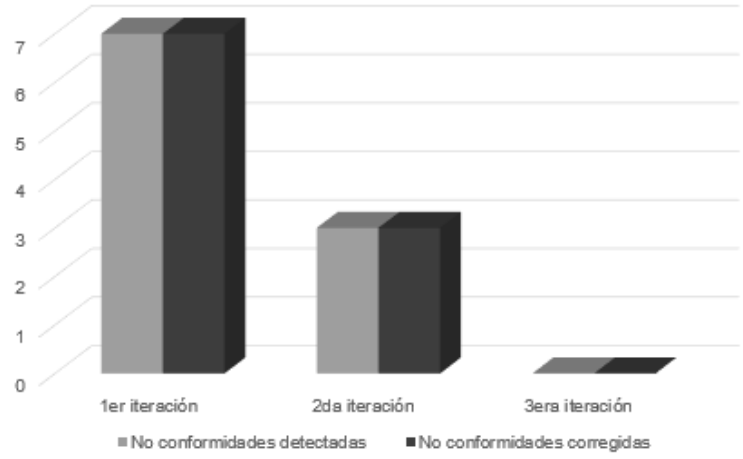

Fig. 18. Gráfica de No conformidades detectadas y corregidas en cada iteración

Tabla 11. Ejemplo de no conformidades encontradas

Elemento	No.	No Conformidad	Ubicación	Etapa de detección	Clasificación
Aplicación	1	El campo “Nombre de proyecto” admite números cuando solo debe admitir nombres de proyectos que empiecen con letras.	Administrador/Nuevo Proyecto	Pruebas de funcionalidad	Significativa
Aplicación	2	Error ortográfico (Ausencia de tildes) en la palabra parámetro.	Administrador	Pruebas de funcionalidad	Significativa
Aplicación	3	Error ortográfico (Ausencia de tildes) en la palabra descripción.	Administrador/Nueva no Conformidad.	Pruebas de funcionalidad	Significativa
Aplicación	4	El campo “Descripción” admite números cuando solo debe admitir descripciones de proyectos que comiencen con letras.	Administrador/Insertar Nueva no Conformidad.	Pruebas de funcionalidad	Significativa
Aplicación	5	En el campo “Área de proceso” la selección por defecto es vacía no siendo así en los demás campos de la ventana.	Administrador/Nueva no Conformidad.	Pruebas de funcionalidad	Significativa

Además, para probar el rendimiento del sistema se realizaron pruebas donde se midió el tiempo de respuesta al ejecutar alguna búsqueda o realizar algún diagnóstico. En dependencia de las características de las computadoras donde se probó, en cuanto a cantidad de memoria RAM y a prestaciones del

microprocesador, se obtuvieron distintos resultados, lo que ayudó a comprobar los requisitos mínimos de hardware que debe poseer la PC donde se instale el sistema.

Las predicciones de las no conformidades se ejecutaron de forma normal, probando las distintas combinaciones posibles para realizarlas y los tiempos de respuesta siempre fueron menores de 1 segundo. En la Tabla 12 se muestran los resultados obtenidos al ejecutar los filtros con el máximo número de criterios posibles (6), garantizando así una cota máxima de tiempo.

Tabla 12. Tiempo de respuesta del sistema

Memoria RAM	**Microprocesador**	**Tiempo de respuesta en segundos del filtrado**
2 GB	Intel Celeron LGA775 Wolfdale 2,60GHz	10
2 GB	Intel Pentuim 4 HT(Prescott) mpGA478b 1,3GHz	15
1 GB	Intel Pentuim D 820 2,66 GHz	12
2 GB	AMD A8-3850 2,9GHz	12
4 GB	Intel i5-2500 3,3Ghz	8

De la anterior tabla se deduce que el sistema funcionará brindando tiempos de respuesta menores a 22 segundos para los filtrados y menores de 2 segundos para las predicciones, si la PC donde se instale tiene un microprocesador Intel(R) Pentium(R) 4 o superior con 1GB de memoria RAM como mínimo. En condiciones de hardware inferiores a las mencionadas el sistema pudiera funcionar correctamente, pero los tiempos de respuestas podrían ser mayores a los especificados.

3.6 Conclusiones del capítulo

Mediante los casos definidos se representó formalmente la base de conocimiento quedando compuesta por 50 casos y los 10 rasgos más significativos una vez aplicado el algoritmo de selección de atributos CfsSubsetEval, además para la clasificación de los nuevos casos se seleccionó el algoritmo de clasificación KNN. Se implementaron todos los requisitos funcionales definidos en el lenguaje de programación Java utilizando su mismo estándar de codificación. Se realizaron pruebas de software a las funcionalidades del sistema mediante la librería JUnit a nivel de unidad y a nivel de sistema las pruebas de caja negra mediante la técnica de Partición de equivalencia, arrojando un total de 10 no conformidades, todas corregidas satisfactoriamente. También se realizaron pruebas de rendimiento donde se midieron los tiempos de respuestas de las principales funcionalidades del sistema.

CONCLUSIONES

Después de realizada la investigación y de haberse construido el Sistema Experto, transitando por todas las fases desde el análisis hasta las pruebas, se puede arribar a las siguientes conclusiones:

- ✓ El estudio de los sistemas expertos realizado en el marco teórico permitió una correcta selección de las herramientas, lenguajes y metodología para guiar el desarrollo del sistema experto para la predicción de las no conformidades en la Actividad de Desarrollo-Producción UCI.
- ✓ El estudio de viabilidad realizado arrojó como resultado que el sistema experto para la predicción de las no conformidades en la Actividad de Desarrollo-Producción UCI podía ser desarrollado exitosamente.
- ✓ El conocimiento adquirido de los especialistas de calidad permitió la elaboración de casos que conformaron la base de conocimiento, además permitió la identificación de los requisitos del sistema.
- ✓ Mediante el diseño se logró mostrar gráficamente cómo los objetos se comunican entre ellos a fin de cumplir con los requisitos del sistema.
- ✓ Se verificó el correcto funcionamiento de todas las funcionalidades del sistema mediante el uso de pruebas de software, corrigiéndose todas las no conformidades detectadas.
- ✓ PredictNC constituye una herramienta para la “Actividad de Desarrollo-Producción UCI” pues brinda la posibilidad de conocer las posibles no conformidades que pudieran existir en un proyecto, además de contar con un gestor y filtrado de proyectos y no conformidades.

RECOMENDACIONES

- ✓ Expandir la BC del sistema para que sea capaz de predecir con una mayor cantidad de casos, incorporando la nueva información en la base de datos.
- ✓ Implementar un subsistema de aprendizaje para incorporar conocimiento que no se tenga almacenado hasta el momento.

REFERENCIAS BIBLIOGRAFICAS

Aha D., Kilber D. 1991. *Instance-based learning algorithms. Machine Learning .* 1991.

Ambler, Scott W. 2014. Agile Modeling. *UML 2 Component Diagramming Guidelines.* [En línea] 2014. [Citado el: 25 de 3 de 2016.] http://www.agilemodeling.com/style/componentDiagram.htm.

Bello Pérez, R. E. 2002. *Aplicaciones de la Inteligencia Artificial.* Guadalajara, Jalisco : Ediciones de la Noche, 2002.

Booch, G., Jacobonson, I. y Rumbaugh, J. 1997. The UML specification documents. [En línea] 1997. http://www.rational.com.

Buschmann, Frank, y otros. 1996. *PATTERN-ORIENTED SOFTWARE ARCHITECTURE. Vol 1.* [PDF] New York : John Wiley & Sons, 1996. ISBN 0 471 95889 7.

Caseres Prieto, O. 2011. Tesis de diplomado. *Sistema Inteligente para el Diagnóstico de Hipertensión Arterial.* La Habana : Universidad de las Ciencias Informáticas. , 2011.

Cleary, John G., Trigg, Leonard E. 1995. *K*: An Instansed-based learner Using an Entropic distance Measure.* 1995.

Clements, P. 1996. *"Coming attractions in Software Architecture". Technical Report.* 1996.

Criado Briz, J. M. 2015. Introducción a los Sistemas Expertos. [En línea] 2015. http://www.ingenieroseninformatica.org/.

De Ávila Ramos, J. 2008. Sistemas Expertos. *Sistemas Expertos.* [En línea] 2008. http://www.lafacu.com/apuntes/informatica/sist_expe/...

De laTorre, C. 2010. *Guía de Arquitectura N-Capas orientada al dominio .NET.* s.l. : Microsoft, 2010.

Deming, W. E. 1996. *La salida de la crisis. Calidad, productividad y competitividad.* Madrid : Díaz de Santos, 1996.

Mestizo Gutiérrez, S. L. , Guerra Hernández, Dr. y Parra Loera, Dr. 2012. *Desarrollo de un centro de ayuda inteligente mediante el uso de tecnologías de Internet.* Veracruz : s.n., 2012, Maestría en Inteligencia Artificial, pág. 90.

Durkin, J. 1994. *EXPERT SYSTEMS: DESIGN AND DEVELOPMENT.* New York : Maxwell Macmilan, 1994.

Ebie, F., Hall, M. 2003. *Locally Weighted Navie Bayes.* 2003.

Bello Pérez, R. E. 2005. *Explicación basada en casos utilizando conjuntos borrosos para un sistema experto conexionista.* Santa Clara : Universidad Central de las Villas, 2005.

Gálvez Lio, Dr. 2008. *Sistemas Basados en el Conocimiento. Especialización en Inteligencia Artificial.* Santa Clara : Universidad Central “Martha Abreu” , 2008.

Garcia, E. 2001. *Calidad de servicio en hoteles de sol y playa.* Madrid : Síntesis, 2001.

Garlan, David y Shaw, Mary. 1994. *An Introduction to Software Architecture.* [PDF] Pittsburg : World Scientific Publishing Company, 1994.

Gosling, J. 2013. *The Java Language Specification. Java SE.* California : Oracle America, 2013.

Gosling, James, y otros. 2013. *The Java Language Specification. Java SE 7 Edition.* [PDF] California : Oracle America, Inc., 2013.

Hall, M. 1998. *Correlation-based feature subset selection for machine learning .* New Zealand : s.n., 1998.

Harrignton, James. 1993. *Mejoramiento de los procesos de la empresa.* Mexico DF : Mc. Graw Hill Interamericana, 1993.

Hommel, Scott. 1999. *Java Code Conventions.* [PDF] 1999.

Jacobson, I. 1992. *Object-Oriented Software Engineering: A Use Case Driven Approach.* 1992.

Jones, Kennet L. 2007. *SMITH. Patrones reconocibles de malformaciones humanas. 6ta Edición.* Barcelona : ELSEVIER SAUNDERS, 2007.

Juran, J. M. 1993. *Manual de Control de la Calidad .* La Habana : MES, 1993.

Kabboul, F. 2011. *Curso de mejoramiento continuo.* 2011.

Kenji, K., Larry A. 2002. *A practical approach to feature selection.* 2002.

Konhavi, R., George, J. 2007. *Wrapper for feature subset selection.* 2007.

Kruchten, Philippe. 1995. *Architectural Blueprints - The “4+1” View.* [PDF] s.l. : IEEE, 1995.

Laboratorio Nacional de Calidad de Software. 2009. *Ingeniería del software: Metodologías y ciclos de vida.* España : INTECO (Instituto de Tecnologías de la Comunicación), 2009.

Larman, Craig. 1999. *UML Y PATRONES. Introducción al análisis y diseño orientado a objetos.* [PDF] México : PRENTICE HALL, 1999. ISBN: 970-17-0261-1.

Leal, A. , Peregrino, V. 2014. *SEDEGEN: Sistema Experto para el diagnóstico de enfermedades Genéticas.* La Habana : Universidad de las Ciencias Informáticas, 2014.

Martínez, Rafael. 2010. PostgreSQL-es. [En línea] 2010. [Citado el: 21 de 2 de 2016.] http://www.postgresql.org.es/sobre_postgresql.

Microsoft . *Revisiones de códigos y estándares de codificación.* [En línea] [Citado el: 16 de 4 de 2016.] https://msdn.microsoft.com/es-es/library/aa291591(v=vs.71).aspx.

Microsoft. 2016. Microsoft. *Microsoft.* [En línea] Marzo de 2016. [Citado el: 2 de Marzo de 2016.] http://msdn.microsoft.com/en-us/library/ee658098.aspx.

NetBeans. **2015**. NetBeans [En línea] 2015. [Citado el: 21 de 2 de 2016.] http://www.netbeans.org.

Palma, J. T., y otros. 2000. *Ingeniería del Conocimiento. De la Extracción al Modelado de Conocimiento.* [PDF] 2000. ISSN: 1988-3064.

Peña Ayala, A. 2006. *Sistemas basados en Conocimiento: Una Base para su Concepción y Desarrollo.* México : Instituto Politécnico Nacional, 2006.

PgAdmin. 2010. PgAdmin. [En línea] 2010. [Citado el: 21 de 2 de 2016.] http://www.pgadmin.org/.

Pressman, Roger S. 2010. *Software Engineering: A Practitioner´s Approach, Seventh Edition.* [PDF] New York : McGrawHill, 2010. ISBN: 978-0-07-337597-7.

Prieto Morales, R. 2014. *Identificando patrones de Predicción y Clasificación de Alarmas por Alto Spread en un Sistema de Combustión de Turbina a Gas.* AntofagastaChile : Univerisadad Católica del Norte, 2014.

Schneider, Jeff. 1997. *Cross Validation.* [En línea] 1997. [Citado el: 16 de 4 de 2016.] https://www.cs.cmu.edu/~schneide/tut5/node42.html.

Sommerville, Ian. 2007. *Software Engineering (Eighth Edition).* [PDF] Hong Kong, Macau, Taiwan : Adison-Wesley Publishers, 2007. ISBN 13: 978-0-321-31379-9; ISBN 10: 0321-31379-8; ISBN 7-111-19770-4.

Uriz Martin, M. X. 2015. *Aprendizaje de distancias basadas en disimilitudes para el algoritmo de clasificación KNN.* [PDF] Pamplona, España : s.n., 2015.

Using Expert Systems and Artificial Intelligence For Real Estate Forecasting. **Rossini, P. 200.** s.l. : Pacific-Rim Real Estate Society, 200, Vol. 6th.

Visual Paradigm. 2015. Visual Paradigm. *Visual Paradigm.* [En línea] 2015. http://www.visual-paradigm.com/.

Witten, I. H., Ebie, F., Trigg L.,Hall, M. 2009. *Weka: Practical Machine Learning Tools and Techniques with Java Implementations.* 2009.

Yogesh, K. y Yogyata, J. 2012. *Research Aspects of Expert System.* s.l. : International Journal of Computing & Business Research, 2012.

BIBLIOGRAFIA

Aha D., Kilber D. 1991. *Instance-based learning algorithms. Machine Learning .* 1991.

Ambler, Scott W. 2014. Agile Modeling. *UML 2 Component Diagramming Guidelines.* [En línea] 2014. [Citado el: 25 de 3 de 2016.] http://www.agilemodeling.com/style/componentDiagram.htm.

Bello Pérez, R. E. 2002. *Aplicaciones de la Inteligencia Artificial.* Guadalajara, Jalisco : Ediciones de la Noche, 2002.

Booch, G., Jacobonson, I. y Rumbaugh, J. 1997. The UML specification documents. [En línea] 1997. http://www.rational.com.

Buschmann, Frank, y otros. 1996. *PATTERN-ORIENTED SOFTWARE ARCHITECTURE. Vol 1.* [PDF] New York : John Wiley & Sons, 1996. ISBN 0 471 95889 7.

Caseres Prieto, O. 2011. Tesis de diplomado. *Sistema Inteligente para el Diagnóstico de Hipertensión Arterial.* La Habana : Universidad de las Ciencias Informáticas. , 2011.

Cleary, John G., Trigg, Leonard E. 1995. *K*: An Instansed-based learner Using an Entropic distance Measure.* 1995.

Clements, P. 1996. *"Coming attractions in Software Architecture". Technical Report.* 1996.

Criado Briz, J. M. 2015. Introducción a los Sistemas Expertos. [En línea] 2015. http://www.ingenieroseninformatica.org/.

De Ávila Ramos, J. 2008. Sistemas Expertos. *Sistemas Expertos.* [En línea] 2008. http://www.lafacu.com/apuntes/informatica/sist_expe/...

De laTorre, C. 2010. *Guía de Arquitectura N-Capas orientada al dominio .NET.* s.l. : Microsoft, 2010.

Deming, W. E. 1996. *La salida de la crisis. Calidad, productividad y competitividad.* Madrid : Díaz de Santos, 1996.

Mestizo Gutiérrez, S. L. , Guerra Hernández, Dr. y Parra Loera, Dr. 2012. *Desarrollo de un centro de ayuda inteligente mediante el uso de tecnologías de Internet.* Veracruz : s.n., 2012, Maestría en Inteligencia Artificial, pág. 90.

Durkin, J. 1994. *EXPERT SYSTEMS: DESIGN AND DEVELOPMENT.* New York : Maxwell Macmilan, 1994.

Ebie, F. , Hall, M. 2003. *Locally Weighted Navie Bayes.* 2003.

Bello Pérez, R. E. 2005. *Explicación basada en casos utilizando conjuntos borrosos para un sistema experto conexionista.***.** Santa Clara : Universidad Central de las Villas, 2005.

Gálvez Lio, Dr. 2008. *Sistemas Basados en el Conocimiento. Especialización en Inteligencia Artificial.* Santa Clara : Universidad Central “Martha Abreu” , 2008.

Garcia, E. 2001. *Calidad de servicio en hoteles de sol y playa.* Madrid : Síntesis, 2001.

Garlan, David y Shaw, Mary. 1994. *An Introduction to Software Architecture.* [PDF] Pittsburg : World Scientific Publishing Company, 1994.

Gosling, J. 2013. *The Java Language Specification. Java SE.* California : Oracle America, 2013.

Gosling, James, y otros. 2013. *The Java Language Specification. Java SE 7 Edition.* [PDF] California : Oracle America, Inc., 2013.

Hall, M. 1998. *Correlation-based feature subset selection for machine learning .* New Zealand : s.n., 1998.

Harrignton, James. 1993. *Mejoramiento de los procesos de la empresa.* Mexico DF : Mc. Graw Hill Interamericana, 1993.

Hommel, Scott. 1999. *Java Code Conventions.* [PDF] 1999.

Jacobson, I. 1992. *Object-Oriented Software Engineering: A Use Case Driven Approach.* 1992.

Jones, Kennet L. 2007. *SMITH. Patrones reconocibles de malformaciones humanas. 6ta Edición.* Barcelona : ELSEVIER SAUNDERS, 2007.

Juran, J. M. 1993. *Manual de Control de la Calidad .* La Habana : MES, 1993.

K., Rich E. y Knight. 1994. *Inteligencia Artificial.* México : McGraw Hill, 1994.

Kabboul, F. 2011. *Curso de mejoramiento continuo.* 2011.

Kenji, K., Larry A. 2002. *A practical approach to feature selection.* 2002.

Konhavi, R., George, J. 2007. *Wrapper for feature subset selection.* 2007.

Kruchten, Philippe. 1995. *Architectural Blueprints - The “4+1” View.* [PDF] s.l. : IEEE, 1995.

Laboratorio Nacional de Calidad de Software. 2009. *Ingeniería del software: Metodologías y ciclos de vida.* España : INTECO (Instituto de Tecnologías de la Comunicación), 2009.

Larman, Craig. 1999. *UML Y PATRONES. Introducción al análisis y diseño orientado a objetos.* [PDF] México : PRENTICE HALL, 1999. ISBN: 970-17-0261-1.

Leal, A. , Peregrino, V. 2014. *SEDEGEN: Sistema Experto para el diagnóstico de enfermedades Genéticas.* La Habana : Universidad de las Ciencias Informáticas, 2014.

Martínez, Rafael. 2010. PostgreSQL-es. [En línea] 2010. [Citado el: 21 de 2 de 2016.] http://www.postgresql.org.es/sobre_postgresql.

Microsoft . *Revisiones de códigos y estándares de codificación.* [En línea] [Citado el: 16 de 4 de 2016.] https://msdn.microsoft.com/es-es/library/aa291591(v=vs.71).aspx.

Microsoft. 2016. Microsoft. *Microsoft.* [En línea] Marzo de 2016. [Citado el: 2 de Marzo de 2016.] http://msdn.microsoft.com/en-us/library/ee658098.aspx.

2015. NetBeans IDE. [En línea] 2015. [Citado el: 21 de 2 de 2016.] http://www.netbeans.org.

Palma, J. T., y otros. 2000. *Ingeniería del Conocimiento. De la Extracción al Modelado de Conocimiento.* [PDF] 2000. ISSN: 1988-3064.

Peña Ayala, A. 2006. *Sistemas basados en Conocimiento: Una Base para su Concepción y Desarrollo.* México : Instituto Politécnico Nacional, 2006.

PgAdmin. 2010. PgAdmin. [En línea] 2010. [Citado el: 21 de 2 de 2016.] http://www.pgadmin.org/.

Pressman, Roger S. 2010. *Software Engineering: A Practitioner´s Approach, Seventh Edition.* [PDF] New York : McGrawHill, 2010. ISBN: 978-0-07-337597-7.

Prieto Morales, R. 2014. *Identificando patrones de Predicción y Clasificación de Alarmas por Alto Spread en un Sistema de Combustión de Turbina a Gas.* AntofagastaChile : Univerisadad Católica del Norte, 2014.

Rusell, Stuar y Norvig, Peter. 2010. *Artificial Intelligence. A Modern Approach. Third Edition.* [PDF] New Jersey : Prentice Hall, 2010. ISBN-13: 978-0-13-604259-4; ISBN-10: 0-13-604259-7.

Schneider, Jeff. 1997. *Cross Validation.* [En línea] 1997. [Citado el: 16 de 4 de 2016.] https://www.cs.cmu.edu/~schneide/tut5/node42.html.

Sommerville, Ian. 2007. *Software Engineering (Eighth Edition).* [PDF] Hong Kong, Macau, Taiwan : Adison-Wesley Publishers, 2007. ISBN 13: 978-0-321-31379-9; ISBN 10: 0321-31379-8; ISBN 7-111-19770-4.

Uriz Martin, M. X. 2015. *Aprendizaje de distancias basadas en disimilitudes para el algoritmo de clasificación KNN.* [PDF] Pamplona, España : s.n., 2015.

Using Expert Systems and Artificial Intelligence For Real Estate Forecasting. **Rossini, P. 200.** s.l. : Pacific-Rim Real Estate Society, 200, Vol. 6th.

Visual Paradigm. 2015. Visual Paradigm. *Visual Paradigm.* [En línea] 2015. http://www.visual-paradigm.com/.

Witten, I. H., Ebie, F., Trigg L.,Hall, M. 2009. *Weka: Practical Machine Learning Tools and Techniques with Java Implementations.* 2009.

Yogesh, K. y Yogyata, J. 2012. *Research Aspects of Expert System.* s.l. : International Journal of Computing & Business Research, 2012.