
# Sistema basado en casos para apoyar el proceso de toma de decisiones en las organizaciones frente a una mejora de procesos de software

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

**Autor:** Karel Riverón Escobar

**Tutor:** Ing. Ana Marys Garcia Rodríguez

3 de Junio de 2013

La Habana

*"The popular definition of artificial intelligence research means designing computers that think as people do, and who needs that? There is no commercial reason to duplicate human thought because there is no market for electronic people, although it might be nice if everyone could have a maid and butler. There are plenty of organic people, and computer vendors can't compete with the modern low-cost technology used in making people."*

William A. Taylor,
what every engineer should know about Artificial Intelligence

**DECLARACIÓN DE AUTORÍA**

Declaro ser autor del presente trabajo de diploma y reconozco a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

Para que así conste se firma la presente a los 3 días del mes de Junio del año 2013.

| | |
|---|---|
| Karel Riverón Escobar | Ing. Ana Marys Garcia Rodríguez |
| _____ | _____ |
| Firma del autor | Firma del tutor |

**RESUMEN**

Actualmente el desarrollo de las tecnologías de la información y las comunicaciones ha desencadenado una fuerte competencia entre las instituciones que desarrollan software por acaparar el mercado mundial. A pesar del incremento de la demanda de software en la sociedad, un gran número de organizaciones fracasa en el cumplimiento de sus objetivos.

Con motivo de contrarrestar esta situación, muchos países e instituciones privadas han enfocado el desarrollo de software en la mejora continua de procesos; sin embargo se refleja un número considerablemente de fracasos debido a que las iniciativas de mejora de procesos de software no contemplan los aspectos sociales de las organizaciones.

La presente investigación persigue como objetivo principal el desarrollo de un sistema informático para apoyar el proceso de toma de decisiones en las organizaciones que se enfrentan a una mejora de procesos de software a partir del diagnóstico de un conjunto de indicadores que inciden en el diseño y ejecución de las mismas. Además, el sistema implementado permite a las organizaciones conocer el estado en que se encuentran para enfrentar una mejora de procesos de software a partir de las experiencias de organizaciones similares y almacena estos resultados con el propósito de procesarlos mediante el razonamiento basado en casos como técnica de inteligencia artificial.

## TABLA DE CONTENIDOS

## INTRODUCCIÓN

En los últimos años el desarrollo de las Tecnologías de la Información y las Comunicaciones (TIC) se ha enfocado en la informatización de los distintos sectores de la sociedad. Gobiernos e instituciones privadas se han centrado en la industria del software con motivo de mejorar sus procesos, por lo que se ha desencadenado una fuerte competencia por acaparar el mercado mundial y vender cada día productos de mayor calidad. Sin embargo la correcta ejecución de proyectos de software está muy por debajo respecto a la progresiva demanda que existe actualmente (1).

En los últimos 2 años comprendidos en el Chaos Manifesto 2011 de Standish Group (1), el 37% de los proyectos de software resultaron exitosos, un 42% presentó problemas de retrasos en los entregables y el 21 % fracasaron totalmente. Estas cifras se deben a un conjunto de problemas que aquejan a la producción de software de manera general, los cuales se reflejan en los altos presupuestos pagados por las organizaciones, el esfuerzo desmedido empleado en el desarrollo, la entrega de productos finales fuera de tiempo y la insuficiente calidad de dichos productos (2). Estos problemas desencadenan una difícil situación para la industria del software y la informatización de la sociedad, por lo que se han tomado medidas para hacerles frente.

Con el propósito de mejorar el desempeño organizacional, en los últimos años se ha reflejado una tendencia por parte de las instituciones a la definición de procesos. Destacados investigadores como Bauer y algunos colaboradores coinciden en que el resultado final de un proyecto de desarrollo de software depende en gran medida del enfoque a procesos (2) (3) (4) (5) (6). Por otro lado, otros autores sostienen que establecer un enfoque dirigido a procesos no resulta suficiente para alcanzar una madurez robusta de la organización; también consideran que es importante definir iniciativas que contribuyan a la mejora continua de estos procesos, permitiendo a las organizaciones brindar productos y servicios de calidad en forma competitiva (2) (7).

La Mejora de Procesos de Software (SPI)[1] se desarrolla de manera gradual transitando de un nivel de madurez a otro y tiene como objetivo analizar y definir cómo mejorar las prácticas de desarrollo de software de una organización (8).

Cada proyecto de software posee características intrínsecas que lo distinguen del resto. Muchos de los problemas asociados al desarrollo del software radican en que no se aplica una correcta gestión del

1 Software Process Improvement

conocimiento para la reutilización de las experiencias adquiridas, buenas prácticas y lecciones aprendidas resultantes de proyectos anteriores. Con vistas a mejorar los productos de software muchas organizaciones han optado por la introducción de estas buenas prácticas en la ejecución de sus procesos elevando la madurez y capacidad de los mismos.

Importantes organizaciones, instituciones y comunidades científicas han optado por la aplicación de normas, guías y estándares en función de la mejora de procesos de software como: el Modelo de Capacidad de Madurez Integrada (CMMI)[2], las normas de la Organización Nacional para la Estandarización (ISO)[3], la Guía de Fundamentos para la Dirección de Proyectos (PMBOK)[4], entre otros. Estas normas especifican qué hacer para mejorar los procesos de la organización pero no establecen el cómo hacerlo, lo que afecta el proceso de toma de decisiones (9) (10) (11). Sin embargo, estudios realizados sobre los resultados de la aplicación de iniciativas de SPI en las organizaciones, reflejan que un 70 % fracasa (12). Buena parte de este porcentaje se debe a que los programas de mejora no contemplan los aspectos sociales asociados a las organizaciones. Esto afecta en gran medida la ejecución de los procesos definidos como parte del programa, provocando inconsistencias y dificultades de entendimiento por parte del equipo de desarrollo que los ejecuta.

Destacados investigadores han definido elementos para adaptar las iniciativas de SPI a las características particulares de las organizaciones. Estos proponen indicadores que influyen en el diseño y ejecución de estos programas a partir de: literatura consultada (13), entrevistas y encuestas a consultores de SPI (14), datos y experiencia acumuladas de la ejecución de SPI (15) (16), y el estudio de casos (17). El análisis de los indicadores que influyen en las iniciativas de SPI, permite inferir que su uso en función de los contextos organizacionales contribuye al éxito de las iniciativas de SPI (13) (14). Algunos de estos indicadores a tener en cuenta para la aplicación de iniciativas SPI según varios autores son: las relaciones interpersonales, la formación y experiencia del personal, la motivación y el compromiso del mismo, la atención al capital humano, el apoyo de la alta gerencia, el funcionamiento y la disponibilidad de recursos (18) (19) (20) (21).

Actualmente, las organizaciones deben emplear gran cantidad de recursos humanos y financieros para

2 Capability Maturity Model Integration

3 International Standard Organization

4 Project management body of knowledge

iniciar una mejora de procesos de software. Estos programas se realizan sin tener en cuenta los resultados alcanzados por otras organizaciones donde los indicadores anteriormente mencionados pudieran comportarse de manera similar debido a la similitud de sus culturas. Además, resulta engorroso realizar el procesamiento de esta información cuando existe un gran número de indicadores que inciden sobre las iniciativas de SPI.

Actualmente, tomando como punto de partida los indicadores mencionados, se ha definido un conjunto de variables que inciden en el resultado de un programa de mejora y los mecanismos que establecen cómo evaluar dichas variables en una organización (22). Por otro lado se plantea que estas variables presentan un nivel de impacto, aunque el mismo aún no se ha sido definido. Teniendo en cuenta que el gran cúmulo de información generada por las organizaciones frente a una SPI no se utiliza para beneficio de las mismas, investigaciones anteriores han diseñado un proceso para apoyar el proceso de toma de decisiones en estas organizaciones (23).

Por tanto, se persigue con esta investigación, implementar un sistema que informatice el proceso mencionado y sirva de soporte al proceso de toma de decisiones en las organizaciones que se enfrentan a programas de mejora de procesos de software. Se propone utilizar en dicho sistema el razonamiento basado en casos como técnica de inteligencia artificial, con el propósito de utilizar las experiencias pasadas de organizaciones frente a una SPI, definiéndose como rasgos predictores los indicadores citados anteriormente, los cuales representan los aspectos sociales de dichas organizaciones.

El análisis anterior conduce al siguiente diseño de investigación:

**Problema de la investigación:**

¿Cómo contribuir a la toma de decisiones en organizaciones que se enfrentan a una mejora de procesos de software a partir de la información del estado de las mismas desde un enfoque social?

**Objeto de estudio de la investigación:**

Gestión del conocimiento.

**Campo de acción:**

Gestión del conocimiento en la mejora de proceso de software.

**Idea a defender:**

El desarrollo de un sistema de razonamiento basado en casos para procesar la información que generan las organizaciones desde un enfoque social frente a una mejora de procesos de software, servirá de apoyo al proceso de toma de decisiones.

**Objetivo general de la investigación:**

Desarrollar un sistema de razonamiento basado en casos para apoyar el proceso de toma de decisiones en las organizaciones a partir del diagnóstico de los indicadores que influyen, desde una perspectiva social, en la mejora de procesos de software.

**Objetivos específicos:**

1. Elaborar el marco teórico conceptual relacionado con la ingeniería y la calidad de software asociado a la mejora de procesos de software y al razonamiento basado en casos como técnica de inteligencia artificial para apoyar la toma de decisiones en las organizaciones.
2. Desarrollar un sistema de razonamiento basado en casos para apoyar el proceso de toma de decisiones en las organizaciones que enfrentan una mejora de procesos de software a partir de la información obtenida de una base de casos con las experiencias pasadas.
3. Validar el sistema propuesto.

**Tareas de la investigación:**

1. Análisis de la bibliografía disponible acerca del estado del arte de la ingeniería y la calidad de software haciendo énfasis en la mejora de procesos de software, las dificultades que enfrentan las organizaciones que ejecutan programas de mejora de procesos de software y las recomendaciones para apoyar el proceso de toma de decisiones.
2. Análisis y selección de los métodos y técnicas necesarios para supervisar la solución propuesta.
3. Diseño e implementación de un sistema de razonamiento basado en casos para procesar el volumen de información que generan las organizaciones frente a una mejora de procesos de software a partir del diagnóstico de los indicadores a considerar y brindar posibles escenarios que servirán de apoyo a la toma de decisiones en dicha organización.
4. Realizar un experimento con casos de estudio para validar la correcta implementación del ciclo del

razonamiento basado en casos.

5. Validar el sistema propuesto a partir de la realización de pruebas unitarias y pruebas de aceptación.

Entre los **métodos de trabajo científico** utilizados se destacan los siguientes:

- Métodos lógicos: se empleó el método histórico-lógico para el análisis crítico de los trabajos anteriores; el método sistémico para el análisis de las iniciativas de mejora de procesos de software y la relación causa-efecto de las problemática abordada y el método inducción-deducción para la identificación de la problemática y sus variantes de solución.
- Métodos empíricos: se empleó la observación participante para la obtención de la información necesaria para el planteamiento del problema y la idea a defender; el método coloquial para la presentación de los resultados en diferentes sesiones científicas; el método experimental para la comprobación de la utilidad del modelo obtenido; y la triangulación metodológica para contrastar los resultados.

**Aportes de la Investigación:**

El valor práctico de la investigación lo constituye el desarrollo de un sistema de razonamiento basado en casos para procesar el gran volumen de información que generan las organizaciones frente a una mejora de procesos de software. Dicho sistema brinda información respecto a aspectos sociales de las organizaciones, la cual puede ser utilizada en el proceso de toma de decisiones.

La **estructura del trabajo** quedó constituida en tres capítulos, conclusiones, anexos y bibliografía.

- Capítulo 1. Fundamentación Teórica: Se presentan elementos teóricos-conceptuales vinculados con las problemáticas y nuevos retos de la ingeniería y la calidad de software. Además se realiza un estudio del estado del arte de la mejora de procesos de software, los indicadores que influyen en la misma. Al mismo tiempo se realiza un análisis de diferentes técnicas de inteligencia artificial que permitan procesar los indicadores mencionados anteriormente para inferir soluciones.
- Capítulo 2. Características del sistema: Se presenta la valoración de los indicadores a considerar que influyen en la SPI en torno a los valores de rango. Se presenta además la descripción de la solución propuesta para apoyar el proceso de toma de decisiones en las organizaciones que se

enfrentan a una mejora de procesos de software a partir del diagnóstico de los indicadores de experiencias anteriores.

- Capítulo 3. Implementación y pruebas: Se presenta la valoración del correcto funcionamiento del sistema propuesto a partir de pruebas unitarias y pruebas de aceptación. Además, se realiza una validación mediante un experimento a partir de datos generados de manera aleatoria.
- Conclusiones: Reflejan el cumplimiento de los objetivos propuestos en la investigación.
- Referencias bibliográficas: Se incluyen las referencias bibliográficas y materiales consultados para el desarrollo de la investigación.
- Recomendaciones: Se presentan de manera general las primeras ideas para darle continuidad a la investigación, apoyándose en otras técnicas de inteligencia artificial que ofrecen un valor agregado al sistema propuesto.
- Anexos: Se incluyen un grupo de figuras y tablas que contribuyen a la compresión del documento.

# 1. CAPÍTULO 1. FUNDAMENTACIÓN TEÓRICA

## 1.1 Introducción

En la producción de software las organizaciones tienen hoy como necesidad inherente a su perfeccionamiento, el aprovechamiento de las buenas experiencias resultantes del proceso de desarrollo de software con el objetivo de minimizar en gran medida los indicadores de tiempo, costo y esfuerzo en la mejora continua de sus procesos, para la obtención de productos y servicios de calidad que satisfaga las necesidades del cliente.

El este capítulo se discuten los principales elementos para la comprensión del trabajo. Se realiza un estudio del estado del arte de la mejora de procesos de software y los indicadores que influyen en su éxito. También se presenta un análisis de diferentes técnicas de inteligencia artificial con el objetivo de seleccionar la más adecuada para resolver la problemática propuesta y se analiza el modelo matemático implementado a partir de la técnica seleccionada. Por último se profundiza en la metodología y las tecnologías escogidas para el desarrollo de la solución.

### 1.1.1 Gestión de la calidad de software

La industria del software involucra la investigación, desarrollo, distribución y comercialización de software. Sus principales problemas se atañen a: la planificación irreal, los cambios no controlados, los presupuestos sobrepasados, la inadecuada gestión de requisitos e insuficiente control de la calidad de los entregables. A raíz de estos problemas el desarrollo de proyectos de software coexistió en un desorden caótico dando lugar a lo que se conoce como “crisis del software”. Muchas han sido las investigaciones en torno a este panorama y todas convergen en la necesidad de la aplicación de métodos más efectivos de ingeniería de software (2) (3) (4) (5) (6).

La gestión de la calidad proporciona un chequeo independiente en el proceso de desarrollo del software. El proceso de gestión de la calidad chequea los entregables del proyecto para garantizar que son consecuentes con los estándares y objetivos organizacionales.

La gestión de la calidad del software puede ser estructurada en tres actividades fundamentales (6):

- Aseguramiento de la calidad: El establecimiento de un marco de trabajo de procedimientos y estándares organizacionales que guíen a una alta calidad del software.

- Planeación de la calidad: La selección de los procedimientos y estándares apropiados desde el marco de trabajo, adaptados a un proyecto de software específico.
- Control de la calidad: La definición y promulgación de procesos que aseguren que el equipo de desarrollo de software ha seguido los procedimientos y estándares de calidad del proyecto.

Una suposición fundamental en la gestión de la calidad es que la calidad del proceso de desarrollo afecta directamente la calidad de la entrega del producto (2) (6) (9).

### 1.1.2 Calidad del proceso y el producto

La ingeniería de software abarca un amplio espectro de disciplinas, entre las cuales se encuentra el dominio en la aplicación de modelos y estándares de calidad. Estos permiten que las empresas puedan implementar dicha calidad en una doble vertiente: a nivel proceso y a nivel producto. Cada modelo o estándar puede tener una aplicación concreta o limitada orientada a lograr mejorar determinados objetivos.

La mayor responsabilidad del éxito para una organización productiva no depende solo de los procesos definidos para la misma; la adherencia a los mismos por parte del personal y los conocimientos que este pueda adquirir e incorporar a la organización, también influyen notablemente. Aunque se defina correctamente un proceso de desarrollo, su implementación determina en gran medida la calidad del proceso y del producto.

En el desarrollo de software la relación entre la calidad del proceso y la calidad del producto es más compleja. Experiencias han demostrado que la calidad del proceso tiene una influencia significativa en la calidad de software. La gestión y mejora de la calidad del proceso pueden llevar a un menor número de defectos en el software entregado. La gestión de la calidad del proceso involucra (2) (6):

- Definición de estándares de proceso, de cuándo y cómo las revisiones deberán ser conducidas.
- Monitoreo del proceso de desarrollo para asegurar que los estándares se están siguiendo.
- Reportes del proceso de software a la administración del proyecto y al cliente del software.

Muchos autores sostienen que si bien es importante establecer una perspectiva dirigida a procesos para el cumplimiento de los objetivos estratégicos con un enfoque de calidad, no resulta suficiente para alcanzar una madurez robusta en la organización. Constituye un elemento primordial la definición de iniciativas que

contribuyan a la mejora continua de dichos procesos, permitiendo a la organización de software mejorar sus capacidades para proporcionar servicios de calidad en forma competitiva (2) (7) (8).

### 1.2 Mejora de proceso de software

La mejora continua significa la eficacia constante del Sistema de Gestión de la Calidad mediante el uso de políticas de calidad, los resultados de las auditorías, el análisis de los datos, las acciones correctivas y preventivas y la revisión de la dirección (24).

Para Mathiassen y Pourkomeylian la mejora del proceso software constituye un enfoque estructurado que permite a una organización de software mejorar continuamente sus capacidades para proporcionar servicios de calidad en forma competitiva (7).

El término mejora de procesos significa comprender los procesos existentes y modificarlos para incrementar la calidad del producto y reducir los costos y tiempos de desarrollo. La mayor parte de la literatura sobre la mejora de procesos se centra en el perfeccionamiento de los procesos para mejorar la calidad de los productos y en particular reducir el número de defectos de los entregables. Una vez alcanzado esto, el principal objetivo sería la reducción de los costos y el cronograma de trabajo (6).

Por su parte, la mejora de procesos de software tiene por cometido analizar y definir cómo mejorar las prácticas de desarrollo software de una organización, partiendo de una evaluación del proceso en uso, cuyo objetivo es poner de manifiesto el estado actual de dicho proceso. No es un evento de un solo paso, sino que se desarrolla gradualmente mediante transiciones desde un nivel de madurez a otro (8). Se centran en mejorar el rendimiento, la utilidad y la efectividad de los procesos de una manera disciplinada (25).

La mejora de procesos es una actividad cíclica que envuelve tres estados principales:

- Medición del proceso: Atributos del proyecto y el producto son medidos. El objetivo es mejorar las mediciones de acuerdo a los objetivos de la organización involucrada en la mejora de procesos.
- Análisis del proceso: El proceso es evaluado y se identifican las debilidades y riesgos del proceso. Los modelos que describen el proceso son usualmente desarrollados durante este estado.
- Cambio del proceso: Los cambios al proceso que deben ser identificados durante el análisis son introducidos en este estado.

El término mejora de proceso de software implica (2):

- Elementos de un proceso eficaz de software pueden ser definidos de manera efectiva.
- Un enfoque organizacional de desarrollo de software puede ser evaluado contra estos elementos.
- Se debe definir una estrategia de mejora significativa que transforme el enfoque organizacional de desarrollo de software en algo más enfocado, más repetible y seguro.

Por tanto, se entiende por mejora de proceso de software un patrón a seguir, mediante el cual, una empresa intenta modificar sus procesos para ser más eficaces y reducir costes. Mediante el seguimiento de este patrón, que en la mayor parte de los casos implica reorganizaciones internas, se persigue implícitamente la mejora en la calidad de los productos desarrollados.

En su investigación para ayudar a las organizaciones a desarrollar y a mantener productos y servicios de calidad, el Instituto de Ingeniería de Software ha identificado varias dimensiones, sobre las que una organización puede enfocarse para mejorar su actividad. Existen tres dimensiones críticas sobre las cuales típicamente se concentran las organizaciones: las personas, los métodos y procedimientos, y las herramientas y equipamiento (Figura 1.2.1) (26).

El sustento de estas tres dimensiones lo constituyen los procesos utilizados en la organización, los cuales permiten alinear el modo de operar de la organización, evolucionar e incorporar los conocimientos de cómo mejorar el trabajo (26).

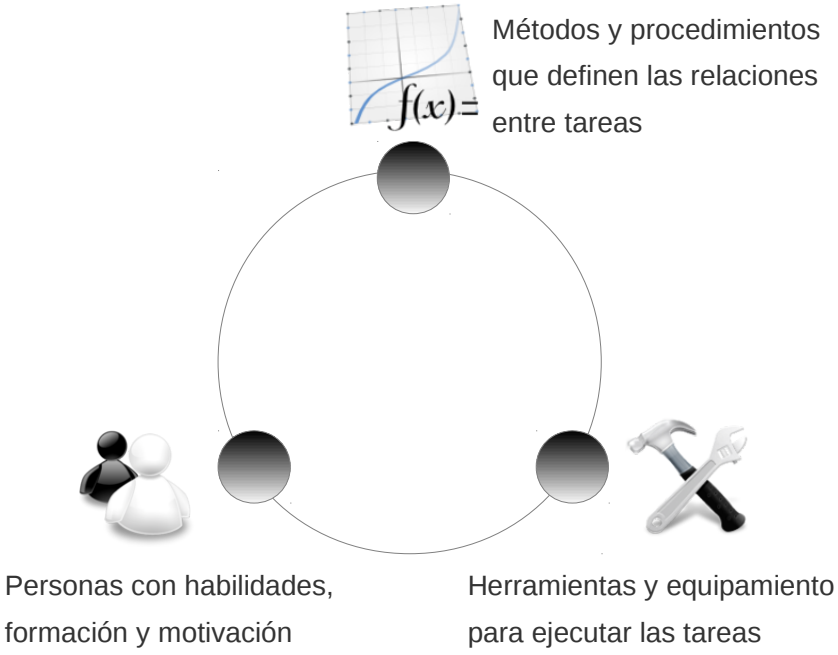


Figura 1.2.1: Las tres dimensiones críticas de las organizaciones (26).

**Indicadores que influyen en la mejora de procesos de software**

Varios autores aportan elementos para adaptar las iniciativas de SPI a las características particulares de las organizaciones, para ello proponen variables que influyen en el diseño y ejecución de iniciativas de SPI a partir de: literatura consultada (13), entrevistas y encuestas a consultores de SPI (14), datos y experiencia acumuladas de la ejecución de SPI (15) (16), y el estudio de casos (17).

El análisis en torno a las variables que influyen positiva o negativamente en las iniciativas SPI, permite inferir que su uso en función de los contextos organizacionales contribuye al éxito de las iniciativas SPI (13) (14).

A partir de las fuentes consultadas se identificaron 12 variables a considerar en la investigación, agrupadas en tres indicadores principales. Estas variables se desglosan en 41 subvariables, que

representan los elementos que serán ponderados a partir de las características de las empresas donde se apliquen las iniciativas de SPI. Las mismas son:

1. **Influencia del personal**
    1. Relaciones interpersonales
        - Colaboración - Competencia
        - Relaciones Individuo - Individuo
        - Relaciones intergrupales
    2. Formación del personal
        - Formación para la mejora de proceso
        - Capacidad de aprendizaje
        - Capacidad de adaptación y autorrenovación
    3. Experiencia del personal
        - Experiencias en la producción
        - Experiencias en roles
    4. Efectividad del programa de reconocimiento y remuneración
        - Reconocimientos y castigos
        - Satisfacción con la política de retribuciones
        - Satisfacción con la política de estimulaciones
    5. Motivación y compromiso del personal
        - Motivación por el trabajo
        - Satisfacción con el trabajo
        - Identificación con la organización
2. **Influencia de la alta gerencia**
    1. Orientación estratégica
        - Orientación a la mejora continua
        - Orientación a la satisfacción del cliente
        - Orientación a procesos

- Gestión del cambio

2. Administración estratégica
    - Planeación estratégica
    - Establecimiento y dominio de los objetivos organizacionales
    - Establecimiento y delimitación de roles organizacionales
3. Atención al capital humano
    - Selección de personal e inducción a la organización
    - Programas de desarrollo y planes de superación
    - Evaluación del desempeño
    - Protección e higiene del trabajo
4. Apoyo de la alta gerencia
    - Confianza en la dirección
    - Competencia de los directivos
    - Supervisión
    - Estilo de dirección
    - Relaciones Jefe - Subordinados

**3. Características de la organización**

1. Comunicación
    - Participación
    - Información
    - Comunicación
2. Funcionamiento
    - Perspectivas de la organización
    - Eficiencia
    - Eficacia
    - Estabilidad interna de la organización
    - Trabajo en equipo

3. Disponibilidad de recursos
    - Disponibilidad de las personas
    - Disponibilidad de tiempo
    - Disponibilidad de infraestructura

Actualmente se ha definido un conjunto de variables que inciden en el resultado de un programa de mejora y los mecanismos que establecen cómo evaluar dichas variables en una organización. Por otro lado se plantea que estas variables presentan un nivel de impacto, aunque aún no se ha definido una medida para ponderar las mismas. Esto genera un gran cúmulo de información por parte de las organizaciones que han intentado iniciarse en mejoras de procesos de software. Sin embargo esta experiencia no ha sido utilizada en función de mejorar los procesos en dichas organizaciones.

### 1.3 Toma de decisiones

Actualmente en el marco de las organizaciones muchas decisiones se toman sin considerar explícitamente las etapas de ese proceso o los métodos cuantitativos y cualitativos existentes en las distintas ramas. En ocasiones tampoco se toman en cuenta las tradiciones, los hábitos, las costumbres, la propia intuición y experiencia de un directivo en la forma en que los problemas se solucionan.

Entre las obligaciones que impone la función gerencial se encuentra tomar decisiones. Con frecuencia, son escasos aquellos individuos que realmente se detienen a considerar el proceso secuencial y sistemático que implica tomar una decisión con el objetivo de obtener realmente la efectividad necesaria a partir de la decisión tomada.

Con respecto al concepto "toma de decisiones", Schein, plantea: "la toma de decisiones es el proceso de identificación de un problema u oportunidad y la selección de una alternativa de acción entre varias existentes, es una actividad clave en todo tipo de organización" (27).

El proceso de toma de decisiones demanda identificar todas las alternativas disponibles, pronosticar sus consecuencias y evaluarlas según los objetivos y metas trazadas. Para ello, se requiere: "En primer lugar, información actualizada sobre qué alternativas se encuentran disponibles en el presente o cuáles se deben considerar. En segundo lugar, se necesita información sobre el futuro: cuáles son las consecuencias de actuar según cada una de las diversas opciones. En tercer lugar, es indispensable la información sobre como pasar del presente al futuro: cuáles son los valores y las preferencias que se

deben utilizar para seleccionar, entre las alternativas que, según los criterios establecidos, conducen del mejor modo a los resultados deseados" (28).

La toma de decisiones puede definirse entonces como una actividad imprescindible en las organizaciones, con un significado especial para todos sus niveles, porque es parte fundamental inherente a todas las demás actividades de la empresa. Este procedimiento ideal, en muchas ocasiones, debido a la escasez de tiempo y recursos para alcanzar este estado de conocimiento, resulta complejo aplicarlo en entornos tradicionales, por ello, la necesidad de sistemas automatizados que posibiliten el análisis y la interpretación de la información disponible.

### 1.3.1 Sistemas de apoyo a la decisión

Un sistema de apoyo a las decisiones (DSS)[5] en términos muy generales, es "un sistema computacional que ayuda en el proceso de toma de decisiones" (29). En términos más específicos, según Turban, un DSS es "un sistema de información basado en un computador interactivo, flexible y adaptable, especialmente desarrollado para apoyar la solución de un problema de gestión no estructurado para mejorar la toma de decisiones. Utiliza datos, proporciona una interfaz amigable y permite la toma de decisiones en el propio análisis de la situación" (30).

Un DSS es un "conjunto de procedimientos basados en modelos para procesar datos y juicios para asistir a un gerente en su toma de decisiones" (31). Moore y Chang consideran que un DSS es un "sistema extensible capaz de apoyar el análisis de datos y el modelado de decisiones, orientado a la planificación futura y utilizado a intervalos irregulares, no planificados" (32).

Un DSS "combina recursos intelectuales individuales con las capacidades de un ordenador para mejorar la calidad de las decisiones. Son un apoyo informático para los encargados de tomar decisiones sobre problemas semi-estructurados." (33). Keen afirma además que es imposible dar una definición precisa incluyendo todas las facetas de la toma de decisiones ya que "no puede haber una definición de los sistemas de apoyo a la decisión, sino sólo del apoyo a la decisión" (34).

Teniendo en cuenta estos conceptos podemos definir los sistemas de apoyo a las decisiones como un conjunto de programas, tecnologías y herramientas que permiten obtener oportunamente la información requerida durante el proceso de la toma de decisiones, en un ambiente de incertidumbre. La información

5 Decision Support System

que generan sirve de apoyo a los mandos intermedios y a la administración de la empresa en el proceso de toma de decisiones. Suelen ser intensivos en cálculos y escasos en entradas y salidas de información.

**Problemas de decisión**

Los sistemas de tomas de decisiones pueden clasificarse en: multicriterios, multiobjetivos y multiatributos.

**Multicriterios:**

Un problema de decisión puede considerarse con múltiples criterios si existen al menos dos criterios en conflicto y como mínimo dos alternativas de solución. La toma de decisión con múltiples criterios (MCDM)[6] abarca las decisiones con "Múltiples Objetivos" (MODM)[7] y con "Múltiples Atributos" (MADM)[8].

- Multiobjetivos: Los multiobjetivos (MODM) son problemas donde se definen las alternativas por medio de restricciones y en general, son infinitas. Los diversos criterios se expresan a partir de variables de decisión numéricas. Al incluir tantas alternativas, se le reconoce como un problema de diseño que emplea para su solución técnicas matemáticas tradicionales de optimización (35).
- Multiatributos: Los multiatributos (MADM) trabaja con un número finito y generalmente pequeño de alternativas (número discreto). Se considera como un problema de selección en el que no se pueden aplicar las herramientas clásicas para resolverlo (35).

### 1.3.2 Razonamiento basado en casos

Los DSS requieren del empleo del conocimiento como materia base para su procesamiento y transformación, lo cual trae consigo algunas limitantes dadas por propiedades del conocimiento como: ser voluminoso, ser difícil de caracterizar y modelar con precisión, estar cambiando constantemente. Por ello se requiere de técnicas de inteligencia artificial para explotar dicho conocimiento (36).

Una técnica de inteligencia artificial es un método para explotar el conocimiento, que debería ser representado de manera que (36):

- Capte generalizaciones: No es una base de datos. No debe ser necesario representar cada situación individual, sino que se agrupen las situaciones que compartan propiedades importantes.

6 Multiple Criteria Decision Making

7 Multiple Objective Decision Making

8 Multiple Attribute Decision Making

Si no tiene esta característica se necesitaría más espacio del disponible y más tiempo del que se tiene para mantenerlo actualizado.

- Sea comprendido por los especialistas que lo proporcionan.
- Sea modificable fácilmente.
- Sea usado en muchas situaciones diversas, incluso si no es totalmente preciso o completo.
- Sea usado para extenderse a sí mismo.

Las técnicas más comúnmente usadas para la implementación de DSS son:

- Sistemas expertos: Sistemas informáticos que simulan el proceso de aprendizaje, de memorización, de razonamiento, de comunicación y de acción de un experto humano en una determinada rama de la ciencia o campo, suministrando de esta forma, un consultor que puede sustituirle con unas ciertas garantías de éxito (37).
- Redes neuronales artificiales: Una red neuronal artificial puede definirse como un sistema de procesamiento de información compuesto por un gran número de elementos de procesamiento (neuronas), profusamente conectados entre sí a través de canales de comunicación. Estas conexiones establecen una estructura jerárquica y permiten la interacción con objetos del mundo real tratando de emular al sistema nervioso biológico. La computación neuronal permite desarrollar sistemas que resuelvan problemas complejos cuya formalización matemática es sumamente difícil. Esto se logra gracias a los principios de funcionamiento de las redes neuronales: aprendizaje adaptativo, auto-organización, tolerancia a fallos, operación en tiempo real y fácil inserción en la tecnología existente (38).
- Redes bayesianas: Es un grafo dirigido acíclico en el que cada nodo tiene la información de su probabilidad cuantitativa. Se utiliza para conocer la probabilidad a posteriori de cierta variable de interés dado un conjunto de hallazgos (39).
- Algoritmos genéticos: Es una variante de búsqueda estocástica en la que los estados sucesores son generados por combinaciones de dos estados padres, más que mediante la modificación de un solo estado. Intervienen procesos basados en la genética natural como la selección natural, el cruzamiento y la mutación (39). Combinan la supervivencia de los más compatibles entre las estructuras de cadenas, con una estructura de información ya aleatorizada, intercambiada para

construir un algoritmo de búsqueda con algunas de las capacidades de innovación de la búsqueda humana (40).

- Lógica difusa: Es un método para el razonamiento con expresiones lógicas que describen la pertenencia a conjuntos difusos (39).
- Razonamiento basado en casos (CBR)[9]: El razonamiento basado en casos consiste en utilizar experiencias pasadas para comprender y resolver nuevos problemas. Es una manera de razonar haciendo analogías. En CBR, un razonador recuerda una situación anterior similar a la actual y la usa para resolver el nuevo problema (41).

Se ha argumentado que el razonamiento basado en casos no sólo es un método poderoso para el razonamiento de computadoras, sino que es usado por las personas para solucionar problemas cotidianos. Más radicalmente se ha sostenido que todo razonamiento es basado en casos porque está basado en la experiencia previa. Según Kolodner, CBR es una tecnología derivada de la inteligencia artificial que representa el conocimiento como casos que fueron usados para resolver problemas pasados (42).

El enfoque de CBR se basa en dos principios sobre la naturaleza del mundo. El primer principio es que el mundo es regular: problemas similares tienen soluciones similares. En consecuencia, las soluciones para problemas similares anteriores son un buen punto de partida para la nueva resolución de problemas. El segundo principio es que los tipos de problemas de un agente de encuentros tienden a repetirse. En consecuencia, problemas en el futuro es probable que sean similares a los problemas actuales. El CBR es una estrategia de razonamiento efectivo (43).

Leake y Kolodner definen el CBR como un enfoque que aborda nuevos problemas tomando como referencia problemas similares resueltos en el pasado (44) (45) (46). De modo que problemas similares tienen soluciones similares, y la similitud juega un rol esencial (47).

Este sistema de razonamiento se basa en una unidad mínima llamada caso, el cual tiene dos componentes: rasgos predictores (descripción del problema) y rasgos objetivos (solución del problema). Un caso es una pieza contextualizada de conocimiento, la cual representa una experiencia que enseña una lección fundamental para el logro de los objetivos del razonador (44).

El CBR es un paradigma para la resolución de problemas nuevos partiendo del análisis y adaptación de

9 Case-Based Reasoning

soluciones que fueron dadas a problemas previos, que se encuentran almacenados y organizados en una biblioteca o base de casos. Cuando un nuevo problema es encontrado, CBR recuerda casos similares y adapta las soluciones que funcionaron en el pasado al problema actual. Problemas cuya solución posea forma similar presentan soluciones similares. En consecuencia con lo anterior, soluciones de problemas previos similares al actual son un punto de partida útil para las soluciones de un nuevo problema (48).
Por consiguiente, el razonamiento basado en casos es una técnica de inteligencia artificial que se basa en la utilización de experiencias previas para resolver nuevos problemas mediante la hipótesis “problemas similares tienen soluciones similares”.

**Ciclo de vida de un sistema basado en casos**

Un sistema basado en casos tiene tres componentes principales: una base de casos, un analizador de problemas y un recuperador de casos. La base de casos contiene las descripciones de los problemas resueltos previamente. Cada caso puede describir un episodio particular o una generalización de un conjunto de episodios relacionados. En el estilo de solución de problemas se recupera un caso semejante al nuevo y la solución del problema recuperado se propone como solución potencial del nuevo problema. Esto se deriva de un proceso de adaptación en el cual se adecua la vieja solución a la nueva situación (42) (44).
Un ciclo de vida CBR está formado esencialmente por los cuatro procesos siguientes (Figura 1.4.2.1) (42) (49):

1. Recuperar el caso o casos pasados más similares (Retrieve). Esto es, retomar la experiencia de un problema anterior que se cree similar al nuevo.
2. Reutilizar la información y conocimiento de este caso o casos recuperados para resolver el nuevo problema (Reuse). Esto es, copiar o integrar la solución del caso o casos recuperados.
3. Revisar la solución propuesta (Revise).
4. Guardar la nueva solución una vez ha sido confirmada o validada (Retain). Se guardan aquellas partes de la experiencia de una manera tal que sea útil para resolver problemas futuros.

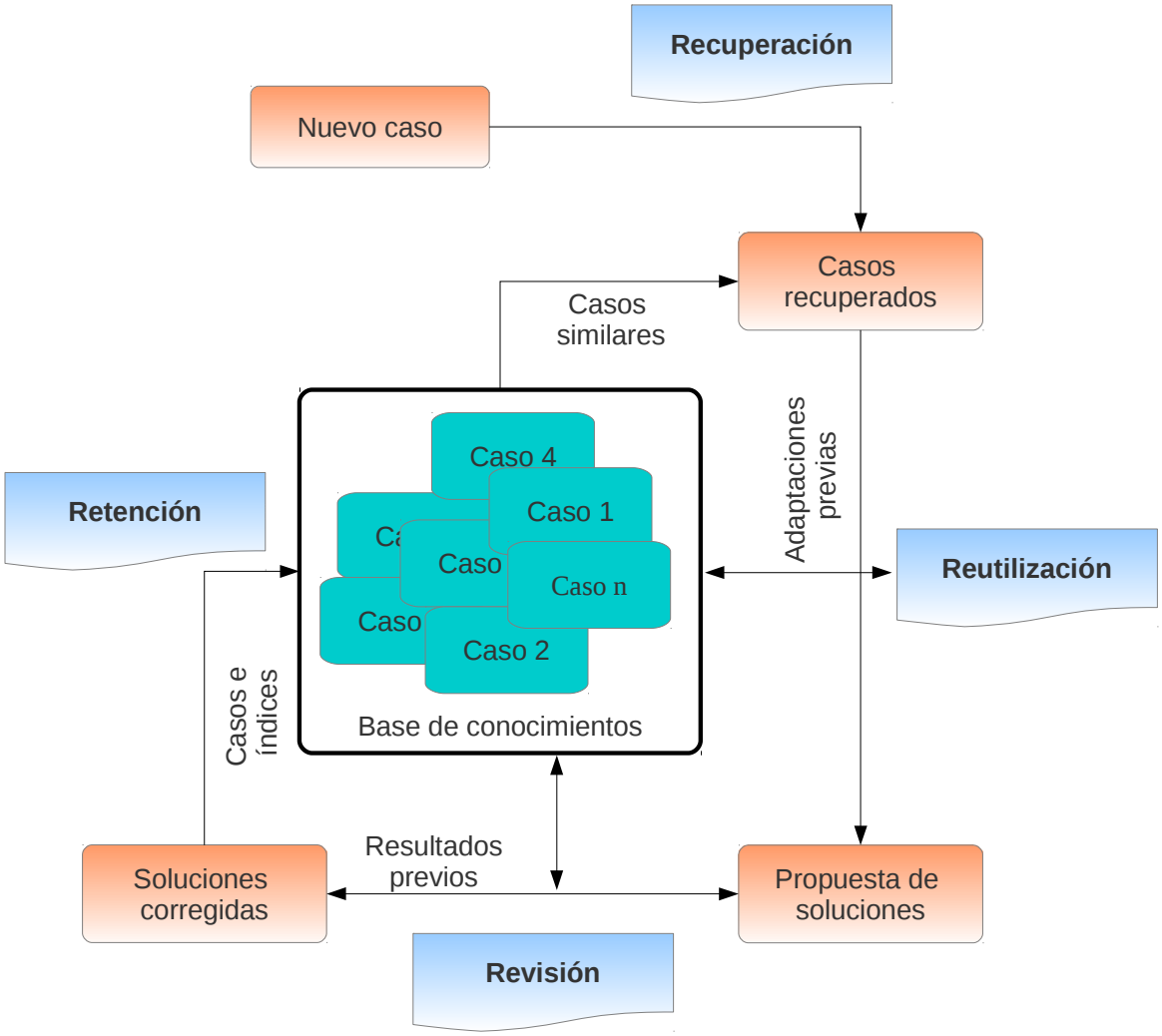


Figura 1.3.2.1: Ciclo básico de un sistema de Razonamiento Basado en Casos (42) (50) (51) (52) (53) (54).

**Arquitectura de un DSS basado en casos**

A la hora de definir la arquitectura de un sistema de apoyo a la decisión aplicando como técnica de inteligencia artificial el CBR, diferentes autores identifican múltiples componentes para solucionar el problema. Sprague y Carlson, identifican tres componentes básicos que son explicados con más detalles por Haag (55) (56):

- El sistema de gestión de base de datos: Almacena información de diversos orígenes, puede proceder de los repositorios de datos de una organización tradicional, de fuentes externas (como Internet), o del personal (de ideas y experiencias de los usuarios individuales).
- El sistema gestor de modelos: Se ocupa de las representaciones de los acontecimientos, hechos o situaciones utilizando varios tipos de modelos (dos ejemplos serían modelos de optimización y modelos de búsqueda-objetivo).
- El sistema gestor y generador de diálogos: Se trata de la interfaz de usuario; es, por supuesto, el componente que permite a un usuario interactuar con el sistema.

Según Power un DSS tiene cuatro componentes fundamentales (57):

- La interfaz de usuario.
- La base de datos.
- Las herramientas analíticas y de modelado.
- La red y arquitectura del DSS.

### 1.4 Proceso para apoyar la toma de decisiones en la SPI

Investigaciones anteriores han definido un proceso para apoyar la toma de decisiones frente a una mejora de procesos de software a partir del razonamiento basado en casos (23).

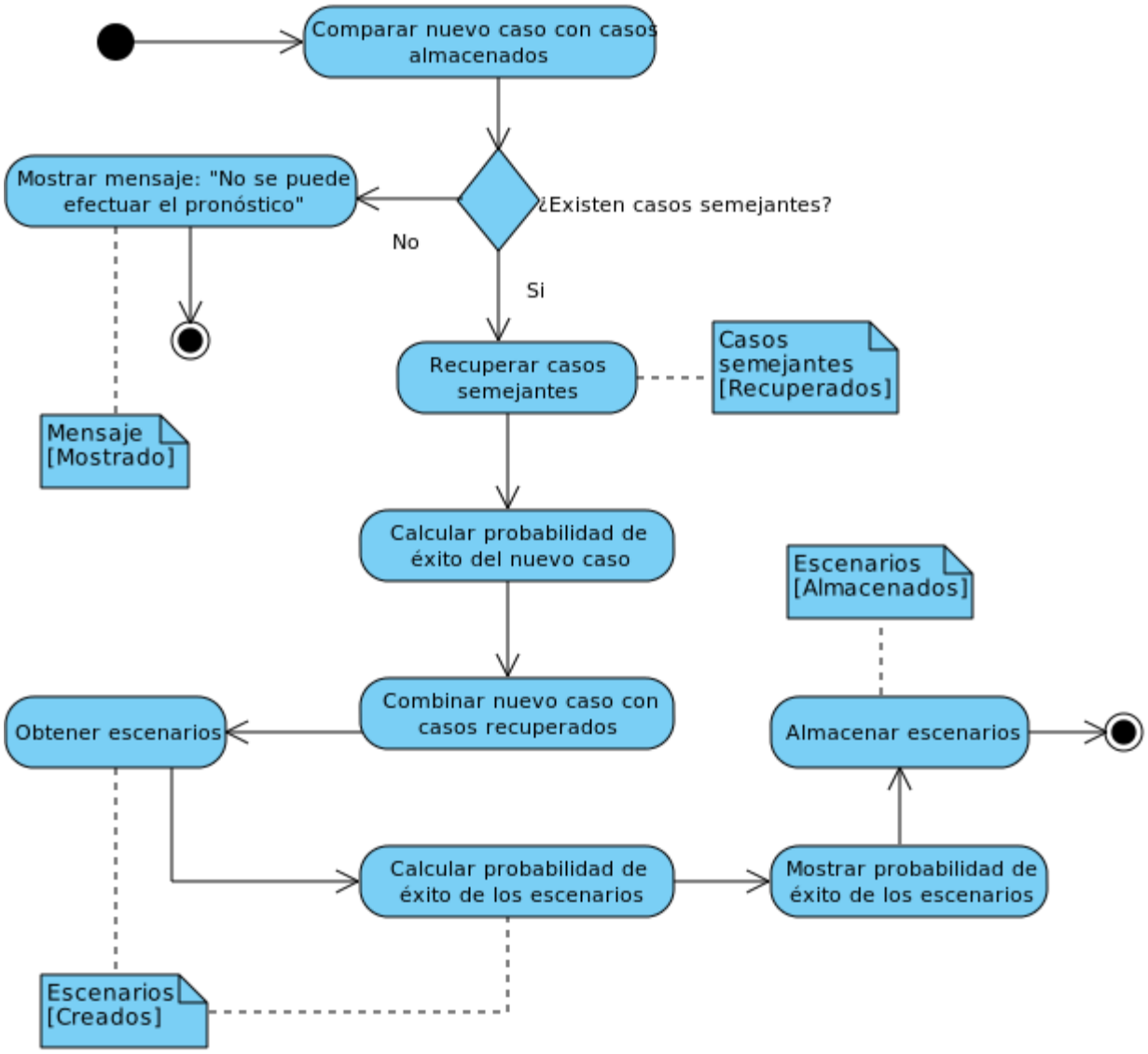


Tabla 1.4.1: Proceso para apoyar la toma de decisiones en una SPI

A continuación se describen las fases que componen el ciclo así como el proceso para el cálculo de la probabilidad de éxito con la descripción de sus actividades, entradas y salidas.

**Fase de recuperación**

1. Comparar nuevo caso con casos almacenados.

En esta actividad el sistema realiza una comparación del caso nuevo proporcionado por la organización y los casos almacenados en la base de casos. Para ello se emplea una función de semejanza propuesta por Fix y Hodges en 1951 y analizada por Silverman y Jones en 1989, entre un nuevo problema a resolver $O_0$ y un caso $O_t$ de la base (58).

$$\beta(O_0, O_t) = \frac{\sum_{k=i_1}^{i_n} p_k * \left[1 - \frac{|X_i(O_0) - X_i(O_t)|}{max_i - min_i}\right]}{\sum_{k=i_1}^{i_n} p_k} \quad (1)$$

La variable n indica el número de rasgos predictores. La variable $p_k$ constituye el peso o relevancia del rasgo i. Los valores $max_i$ y $min_i$ son los umbrales que alcanza el rasgo i. En el caso de las variables $X_i(O_0)$ y $X_i(O_t)$ representan los valores del rasgo i en los casos $O_0$ y $O_t$ respectivamente. Si $\beta(O_0, O_t)$ es mayor o igual a 0.75 se considera que los casos $O_0$ y $O_t$ son semejantes.

2. Recuperar casos semejantes.

   La actividad se ejecuta cuando el sistema encuentra casos semejantes al proporcionado por la organización dentro de la base de casos. A partir de la comparación realizada anteriormente el sistema obtiene los casos semejantes.

**Fase de reutilización**

3. Calcular probabilidad de éxito del nuevo caso.

   El sistema calcula la probabilidad de éxito del nuevo caso proporcionado por la organización mediante la ecuación:

$$\rho = \frac{\sum_{i=c_1}^{c_n} f(i)}{n}$$

La variable $\rho$ es la probabilidad de éxito resultante de la división entre la cantidad de casos recuperados con resultados de éxito y la cantidad total de casos recuperados. La variable *n* constituye el número de casos recuperados. La función $f(i)$ obtiene los casos recuperados con resultados de éxito.

4. Combinar caso nuevo con recuperados.

   El sistema realiza combinaciones entre los rasgos que puedan ser utilizados de los casos recuperados y el nuevo caso para mostrar alternativas de solución a la organización con mejores probabilidades de éxito. Es válido especificar que las combinaciones se realizan sobre la base de mejorar los rasgos que reflejan problemas en la organización, a partir de la superioridad de los mismos en los casos semejantes recuperados.

5. Obtener escenarios.

   Se obtienen los posibles escenarios a ser aplicados por la organización para mejorar su probabilidad de éxito.

6. Calcular probabilidad de éxito de escenarios.

   El sistema calcula la probabilidad de éxito de los escenarios obtenidos anteriormente, utilizando la ecuación (1) antes definida.

7. Mostrar probabilidad de éxito (caso nuevo y escenarios).

   El sistema muestra la probabilidad de éxito del caso nuevo proporcionado por la organización, así como los escenarios obtenidos con sus respectivas probabilidades de éxito.

8. Guardar escenarios.

   El sistema guarda en memoria los escenarios mostrados, para la posterior evaluación y almacenamiento del escenario empleado como caso real.

**Fase de evaluación**

9. Iniciar la SPI.

   A partir de la información brindada por el sistema la organización puede elegir o no la implementación de un escenario para iniciarse en la mejora de proceso de software.

10. Especificar resultado obtenido y si aplicó o no algún escenario.

Luego de la puesta en práctica de la SPI en la organización, esta especifica si utilizó o no algún escenario de los propuestos por el sistema y el resultado obtenido a partir de ello.

11. Comparar resultado de aplicación de escenarios con el pronóstico del sistema.
    Esta tarea se ejecuta cuando la organización especifica que aplicó un escenario de los propuestos por el sistema. El sistema a partir del escenario aplicado por la organización, establece una comparación entre el resultado obtenido y el resultado pronosticado.

**Fase de almacenamiento**

12. Guardar como caso real.
    Se guarda el caso aplicado por la organización en la base de casos como un caso real.
13. Desechar escenarios no usados.
    Se desechan los escenarios que la organización no aplicó.

## 1.5 Plataforma de desarrollo

### 1.5.1 Patrones de diseño y arquitectónicos

En la implementación del sistema se emplearon varios patrones de diseño. Los mismos son propuestos por Symfony 2, que fue el marco de trabajo sobre el cuál se trabajó. A continuación se ejemplificará cuáles son y cómo son utilizados.

**Patrones generales de software para asignación de responsabilidades (GRASP)[10]**

- Creador: En las clases controladoras se encuentran todas las acciones definidas para el módulo HefextoBundle. En las acciones se crean los objetos de las clases que representan las entidades, evidenciando de este modo que las clases controladoras son *creadoras* de dichas entidades.
- Experto: Es uno de los más utilizados, puesto que Symfony 2 utiliza el ORM Doctrine 2 para realizar su capa de abstracción en el modelo, encapsula toda la lógica de los datos y son generadas las clases con todas las funcionalidades comunes de las entidades.

10 General Responsibility Assignment Software Patterns

- Alta cohesión: Symfony 2 permite asignar responsabilidades con una alta cohesión ya que los controladores definen las acciones para las plantillas y colaboran con otras clases para realizar diferentes operaciones, instanciar objetos y acceder a las propiedades, es decir, está formada por diferentes funcionalidades que se encuentran estrechamente relacionadas proporcionando que el software sea flexible frente a grandes cambios.
- Controlador: Todas las peticiones web son manejadas por un solo controlador frontal (app.php), que es el punto de entrada único de toda la aplicación en un entorno determinado. Cuando el controlador frontal recibe una petición, utiliza el sistema de enrutamiento para asociar el nombre de una acción y el nombre de un módulo con la URL solicitada por el cliente.
- Bajo acoplamiento: Las clase controladoras heredan solamente de *BaseController.php* para lograr un bajo acoplamiento de clases.

**Banda de los cuatro (GOF)**[11]

Esta clase de patrones recibe este nombre tan particular en honor a los 4 autores del libro *Design Patterns*, los autores Gamma, Helm, Johnson y Vlissides.

Creacionales:

- Abstract factory (Fábrica abstracta): Permite trabajar con objetos de distintas familias de manera que las familias no se mezclen entre sí y haciendo transparente el tipo de familia concreta que se esté usando. Cuando el marco de trabajo necesita crear un nuevo objeto para una petición, busca en la definición de la fábrica el nombre de la clase que se debe utilizar para esta tarea (59).
- Builder (Constructor): La intención es abstraer pasos de construcción de objetos de modo que diferentes implementaciones de estos pasos pueden construir diferentes representaciones de objetos. (60).

Estructurales:

- Composite (Objeto compuesto): Permite tratar objetos compuestos como si de uno simple se tratase. Sirve para construir objetos complejos a partir de otros más simples y similares entre sí, gracias a la composición recursiva y a una estructura en forma de árbol. Esto simplifica el

11 Gand of four

tratamiento de los objetos creados, ya que al poseer todos ellos una interfaz común, se tratan todos de la misma manera (61) (60).

- Flyweight (Peso mosca): Este patrón es un objeto que minimiza el uso de memoria mediante el intercambio de datos tanto como sea posible con otros objetos similares; es una forma de usar objetos cuando una simple representación reiterada usaría una inaceptable cantidad de memoria. A menudo algunas partes del estado de los objetos puede ser compartida, y es una práctica común para para mantenerlos en estructuras de datos externas y pasarlos a objetos moscas temporalmente para ser usados (60).
- Dependency Injection (Inyección de dependencia): Permite la eliminación de las dependencias incluidas en el código, por lo que es posible cambiarlas, ya sea en tiempo de ejecución o en tiempo de compilación. Este patrón puede ser usado como una forma simple de cargar componentes dinámicamente o elegir objetos simulados en entornos de prueba frente a objetos reales en entornos de producción (62).
- Facade (Fachada): Una fachada es un objeto que proporciona una interfaz simplificada de un bloque de código más grande, tal como una biblioteca de clases. Una fachada permite:
  - Usar, entender y probar una biblioteca de software.
  - Hacer la biblioteca más legible.
  - Reducir la dependencia de código (59).

**Modelo-Vista-Controlador**

El propósito de realizar el diseño arquitectónico del software es estructurar los componentes del sistema, sus interrelaciones, y los principios y reglas que gobiernan su implementación y evolución en el tiempo. Aporta además una visión abstracta de alto nivel, postergando el detalle de cada uno de los módulos definidos a pasos posteriores del diseño. Establece los fundamentos para que analistas, diseñadores y programadores trabajen en una línea común que permita alcanzar los objetivos y necesidades del sistema (63) (64).

El patrón de arquitectura Modelo-Vista-Controlador (MVC)[12] separa la lógica de negocio (el modelo) y la

12 Model-View-Controller

presentación (la vista) por lo que se consigue un mantenimiento más sencillo de las aplicaciones. El controlador se encarga de aislar al modelo y a la vista de los detalles del protocolo utilizado para las peticiones. El modelo se encarga de la abstracción de la lógica relacionada con los datos, haciendo que la vista y las acciones sean independientes del tipo de gestor de bases de datos utilizado por la aplicación (65).

### 1.5.2 Lenguaje de modelado

El Lenguaje Unificado de Modelado (UML)[13] es un lenguaje de modelado visual que se usa para especificar, visualizar, construir y documentar artefactos de un sistema de software. Se usa para entender, diseñar, configurar, mantener y controlar la información sobre los sistemas a construir (66) (67).

UML capta la información sobre la estructura estática y el comportamiento dinámico de un sistema. Este lenguaje de modelado pretende unificar la experiencia pasada sobre técnicas de modelado e incorporar las mejores prácticas actuales en un acercamiento estándar. UML es una notación con la cual se construyen sistemas por medio de conceptos orientados a objetos. Este prescribe un conjunto de notaciones y diagramas estándares, y describe la semántica esencial de lo que estos diagramas y símbolos significan (68) (69).

### 1.5.3 Herramientas de desarrollo

#### Visual Paradigm 5.0

Durante la etapa de modelación y para diseñar el sistema se utilizó Visual Paradigm, una herramienta de modelado multiplataforma que no se inclina por ninguna metodología específica. Además ofrece un entorno de creación de diagramas para UML, con soporte para los 13 diagramas de la última versión (UML 2.0). Por otro lado el diseño es centrado en casos de uso y enfocado al negocio, con soporte para los Diagramas de Procesos de Negocios (BPD)[14] y Diagramas de Flujos de Datos (DFD)[15], lo cual genera un software de mayor calidad (70).

Usa un lenguaje estándar común para todo el equipo de desarrollo que facilita la comunicación y esta

13 Unified Modeling Language

14 Business Process Diagram

15 Data Flow Diagram

capacitado para la ingeniería directa e inversa en Java, C++ y PHP, además de la capacidad de generación de código en estos lenguajes (71).

Presenta dos tipos de diagramas de modelado de bases de datos: entidad-relación (ERD)[16] y mapeo objeto-relacional (ORM)[17]. Los diagramas ERD modelan la base de datos a nivel físico y los ORM muestran la relación entre las clases y la entidades (70).

**NetBeans 7.2**

NetBeans es un entorno de desarrollo integrado (IDE)[18] de código abierto para desarrolladores. Posee todas las herramientas necesarias para crear aplicaciones web, de escritorio y para teléfonos móviles con los lenguajes de programación Java, C, C++ e incluso lenguajes dinámicos como: PHP, Javascript, Groovy, y Ruby. Es fácil de usar e instalar. Puede ser ejecutado en múltiples plataformas como Windows, Linux, Mac OS X y Solaris (72).

NetBeans IDE 7.2 soporta todas las características estándar, como el autocompletado de código, el resaltado de sintaxis, refactorización, plantillas de código, documentación de pop-up, navegación de código, las advertencias de editor y lista de tareas (73) (72).

Entre sus principales características podemos mencionar:

- Brinda completamiento automático de código PHP, así como coloreado de código sintáctico y semántico.
- Permite depurar el código usando Xdebug.
- Genera fragmentos de código para bases de datos MySQL.

Dadas las características, ventajas y soporte que brinda este potente entorno de desarrollo y sobre todo por tener Licencia Pública General (GPL)[19] versión 2, se eligió como IDE a utilizar para el desarrollo del sistema a implementar.

16 Entity-Relationship Diagram

17 Object-Relational Mapping

18 Integrated Development Environment

19 General Public License

### 1.5.4 Metodologías de desarrollo

Desde los inicios del desarrollo de software se ha querido encontrar las mejores prácticas y distribuir la experiencia obtenida a través de la práctica. Esto se ha hecho tradicionalmente mediante metodologías impulsadas por universidades y grandes empresas de tecnología (6).

Existen dos tipificaciones de metodologías: las metodologías ágiles y las metodologías tradicionales. Entre las primeras podemos mencionar algunas bien conocidas como XP[20], DSDM[21] y ASD[22]. Están especialmente preparadas para para realizar cambios durante el desarrollo del proyecto, el cliente forma parte del equipo de desarrollo y este es menor de 10 personas. Dentro de la segunda categoría podemos ubicar a RUP[23] y MSF[24] (2). Este tipo de metodologías presenta resistencia a los cambios durante el desarrollo, el cual es controlado mediante numerosas normas y políticas y el elemento principal es la arquitectura del software. En esta investigación se decidió emplear la metodología ágil Programación Extrema, conocida como XP.

**Extreme Programming**

Programación Extrema es una metodología ágil centrada en potenciar las relaciones interpersonales como clave para el éxito en desarrollo de software, promoviendo el trabajo en equipo, preocupándose por el aprendizaje de los desarrolladores, y propiciando un buen clima de trabajo. XP se basa en realimentación continua entre el cliente y el equipo de desarrollo, comunicación fluida entre todos los participantes, simplicidad en las soluciones implementadas y coraje para enfrentar los cambios. Se define como especialmente adecuada para proyectos con requisitos muy cambiantes, y donde existe un alto riesgo técnico (74) (75).

Los principios y prácticas son de sentido común pero llevadas al extremo, de ahí proviene su nombre. Beck, el padre de XP, describe la filosofía de su metodología sin cubrir los detalles técnicos y de implantación de las prácticas (75). A continuación presentaremos las características esenciales de XP

20 Extreme Programming

21 Dynamic Systems Developmemt Method

22 Adaptive Software Development

23 Racional Unified Process

24 Microsoft Solution Framework

organizadas en los 4 apartados siguientes: historias de usuario, roles, proceso y prácticas.

**Proceso**

Un proyecto XP tiene éxito cuando el cliente selecciona el valor de negocio a implementar basado en la habilidad del equipo para medir la funcionalidad que puede entregar a través del tiempo. El ciclo de desarrollo consiste en los siguientes pasos (76):

1. El cliente define el valor de negocio a implementar.
2. El programador estima el esfuerzo necesario para su implementación.
3. El cliente selecciona qué construir, de acuerdo con sus prioridades y las restricciones de tiempo.
4. El programador construye ese valor de negocio.
5. Vuelve al paso 1.

En todas las iteraciones de este ciclo tanto el cliente como el programador aprenden. No se debe presionar al programador a realizar más trabajo que el estimado, ya que se perderá calidad en el software o no se cumplirán los plazos. De la misma forma el cliente tiene la obligación de manejar el ámbito de entrega del producto, para asegurarse que el sistema tenga el mayor valor de negocio posible con cada iteración.

**Historias de usuario**

Las historias de usuario son la técnica utilizada en XP para especificar los requerimientos del software. Se trata de tarjetas de papel en las cuales el cliente describe brevemente las características que el sistema debe poseer.

El tratamiento de las historias de usuario es muy dinámico y flexible, en cualquier momento historias de usuario pueden romperse, reemplazarse por otras más específicas o generales, añadirse nuevas o ser modificadas. Cada historia de usuario es lo suficientemente comprensible y delimitada para que los programadores puedan implementarla en unas semanas (76).

**Roles**

A continuación se describen los roles de acuerdo con la propuesta original de Beck (75).

- **Programador:** El programador escribe las pruebas unitarias y produce el código del sistema. Debe

existir una comunicación y coordinación adecuada entre los programadores y otros miembros del equipo.

- Cliente:El cliente escribe las historias de usuario y las pruebas funcionales para validar su implementación. Además, asigna la prioridad a las historias de usuario y decide cuáles se implementan en cada iteración centrándose en aportar mayor valor al negocio. El cliente es sólo uno dentro del proyecto pero puede corresponder a un interlocutor que está representando a varias personas que se verán afectadas por el sistema.
- **Encargado de pruebas (Tester):** El encargado de pruebas ayuda al cliente a escribir las pruebas funcionales. Ejecuta las pruebas regularmente, difunde los resultados en el equipo y es responsable de las herramientas de soporte para pruebas.
- **Encargado de seguimiento (Tracker):** El encargado de seguimiento proporciona realimentación al equipo en el proceso XP. Su responsabilidad es verificar el grado de acierto entre las estimaciones realizadas y el tiempo real dedicado, comunicando los resultados para mejorar futuras estimaciones. También realiza el seguimiento del progreso de cada iteración y evalúa si los objetivos son alcanzables con las restricciones de tiempo y recursos presentes. Determina cuándo es necesario realizar algún cambio para lograr los objetivos de cada iteración.
- **Entrenador (Coach):** Es responsable del proceso global. Es necesario que conozca a fondo el proceso XP para proveer guías a los miembros del equipo de forma que se apliquen las prácticas XP y se siga el proceso correctamente.
- **Consultor:** Es un miembro externo del equipo con un conocimiento específico en algún tema necesario para el proyecto. Guía al equipo para resolver un problema específico.
- **Gestor (Big boss):** Es el vínculo entre clientes y programadores, ayuda a que el equipo trabaje efectivamente creando las condiciones adecuadas. Su labor esencial es de coordinación.

**Prácticas**

La principal suposición que se realiza en XP es la posibilidad de disminuir la mítica curva exponencial del costo del cambio a lo largo del proyecto, lo suficiente para que el diseño evolutivo funcione. XP apuesta por un crecimiento lento del costo del cambio y con un comportamiento asintótico. Esto se consigue gracias a las tecnologías disponibles para ayudar en el desarrollo de software y a la aplicación disciplinada

de las prácticas que describiremos a continuación.

### 1.5.5 Programación del lado del cliente

**Chromium Browser 24.0**

Chromium es el proyecto de software libre con el que se ha desarrollado Google Chrome y es de participación comunitaria para fundamentar las bases del diseño y desarrollo del navegador Chrome, además del sistema operativo Google Chrome OS (77).

La porción realizada por Google está amparada por la licencia de uso BSD, con otras partes sujetas a una variedad de licencias de código abierto permisivas que incluyen MIT License[25], Ms-PL[26] y la triple licencia MPL[27]/GPL/LGPL[28] (78).

En esencia, los aportes hechos por el proyecto Chromium fundamentan el código fuente del navegador base sobre el que está construido Chrome y por tanto tendrá sus mismas características (79), pero con un logotipo ligeramente diferente y sin el apoyo comercial o técnico de la compañía Google (80) (81).

**Javascript 1.8**

JavaScript es un lenguaje de programación del lado del cliente, interpretado y utilizado principalmente en páginas web, con una sintaxis semejante a la del lenguaje Java y el lenguaje C. Es un lenguaje orientado a objetos, ya que dispone de herencia, aunque la realiza siguiendo el paradigma de programación basada en prototipos, pues las nuevas clases se generan clonando las clases base y extendiendo su funcionalidades (82).

Todos los navegadores modernos interpretan el código Javascript integrado en las páginas web. Para interactuar con una página web se provee al lenguaje Javascript de una implementación del DOM[29] (83).

Gran parte de la programación en este lenguaje está centrada en describir objetos, escribir funciones que respondan a movimientos del ratón, aperturas, utilización de teclas, cargas de páginas entre otros (84).

25 Massachusetts Institute of Technology License

26 Microsoft Public License

27 Mozilla Public License

28 Lesser General Public License

29 Document Object Model

**Ext JS 4.0.7**

Ext.JS es un marco de trabajo o librería construido con Javascript que concentra su potencia en la rica colección de componentes para el diseño de la interfaz gráfica de usuario complejas y dinámicas del lado del cliente, haciendo uso extensivo de Ajax[30] (85).

Entre los componentes que ofrece este marco de trabajo encontramos: cuadros de diálogo, menús, tablas editables, capas, paneles, pestañas y todo lo necesario para construir atractivos desarrollos al estilo de Web 2.0 (86). La misma da soporte para comunicar datos de forma asíncrona con el servidor y manejarlos aunque sean de distinta índole de una manera simple (87).

Amparado bajo GPL versión 3, es un marco de trabajo completo y extremadamente avanzado. Ext JS basa toda su funcionalidad en Javascript a través de librerías ya muy conocidas: YUI, jQuery y Prototype y un núcleo interno poderoso. Así, en tiempo de ejecución carga y crea todos los objetos HTML[31] a través del uso intenso de DOM (85).

### 1.5.6 Programación del lado del servidor

**Apache 2.0**

El servidor Apache es un servidor web HTTP[32] de código abierto para plataformas Unix, Microsoft Windows, Macintosh y otras, que implementa el protocolo HTTP/1.1 y la noción de sitio virtual (88) (89). Es el servidor web por excelencia. Su usabilidad, robustez y estabilidad hacen que cada vez millones de servidores reiteren su confianza en este programa (89).

Es un software que está estructurado en módulos, así los programadores asumen que el software puede ser ampliado por otros desarrolladores, los cuales pueden escribir pequeñas partes del código que se integrará en Apache de una manera fácil. Esto se lleva a cabo al haber creado un API modular y una serie de fases bien definidas por las que cada petición al servidor debe atravesar. Estas fases van desde la inicialización del servidor (cuando Apache lee los ficheros de configuración), hasta la traducción de una URL en un nombre de fichero del servidor, o registrar los resultados de la transacción (90).

30 Asynchronous JavaScript And XML

31 Hypertext Markup Language

32 Hypertext Transfer Protocol

**PHP 5.3**

PHP[33] es un lenguaje interpretado del lado del servidor, de propósito general, ampliamente usado y que está diseñado especialmente para desarrollo web y puede ser embebido dentro de código HTML. Generalmente se ejecuta en un servidor web, tomando el código en PHP como su entrada y creando páginas web como salida (91).

Es un lenguaje multiplataforma con capacidad de conexión con la mayoría de los manejadores de base de datos que se utilizan en la actualidad, tiene capacidad de expandir su potencial utilizando la enorme cantidad de módulos, llamados extensiones, es libre, por lo que se presenta como una alternativa de fácil acceso para todos, además de que permite la utilización de las técnicas de Programación Orientada a Objetos.

**Symfony 2.0.10**

Symfony es un marco de trabajo para el desarrollo de aplicaciones web escrito en PHP. Fue concebido originalmente por la agencia interactiva Sensio Labs para el desarrollo de sitios web para sus propios clientes. Symfony fue publicado por la agencia en 2005 bajo la licencia de código abierto MIT y hoy es uno de lo mejores marcos de trabajo disponibles para el desarrollo en PHP (93).

Con el apoyo de Sensio Labs y una gran comunidad de usuarios Symfony tiene muchos recursos: abundante documentación, soporte comunitario mediante listas de correos, soporte profesional mediante consultantes y mucho más (93).

### 1.5.7 Base de datos

**Doctrine 2.0**

Doctrine es un ORM escrito en PHP que proporciona una capa de persistencia para objetos de este lenguaje. Es una capa de abstracción que se sitúa justo encima de un sistema gestor de bases de datos (94).

Una característica de Doctrine es el bajo nivel de configuración que necesita para empezar un proyecto. Doctrine puede generar clases a partir de una base de datos existente y después el programador puede especificar relaciones y añadir funcionalidad extra a las clases autogeneradas. No es necesario generar o

33 PHP Hypertext Preprocessor

mantener complejos esquemas XML de base de datos como en otros marcos de trabajo. Otra característica importante de Doctrine es la posibilidad de escribir consultas de base de datos utilizando un dialecto de SQL[34] denominado DQL[35] que está inspirado en HQL[36] y en JPQL[37] (95).

**PostgresSQL 9.1**

PostgreSQL es un sistema de bases de datos relacional potente y de código abierto. Tiene más de 15 años de desarrollo activo y una arquitectura probada que ha ganado una sólida reputación de fiabilidad, integridad de datos y exactitud. Se ejecuta en los sistemas operativos más importantes, incluyendo Linux, Unix, y Windows. Tiene soporte completo para llaves foráneas, uniones, vistas, disparadores, y procedimientos almacenados (en múltiples lenguajes).

PostgreSQL incluye la mayor parte de los tipos de datos de SQL 2008, incluyendo INTEGER, NUMERIC, BOOLEAN, CHAR, VARCHAR, DATE, INTERVAL y TIMESTAMP. También soporta el almacenamiento de objetos binario grandes, incluyendo imágenes, sonidos y videos. Tiene interfaces nativas de programación para C/C++, Java, .Net, Perl, Python, Ruby, Tcl, ODBC, entre otros y una excepcional documentación (96).

PostgreSQL, como base de datos de clase empresarial, cuenta con características avanzadas como Control de Concurrencia Multiversión, espacios de tablas, replicación asincrónica, transacciones anidadas, salvas en línea, un planificador de consultas sofisticadas y se provee de tolerancia a fallos. Es altamente escalable tanto en la enorme cantidad de datos que puede manejar y en el número de usuarios simultáneos que puede acomodar. Hay sistemas activos de PostgreSQL en entornos de producción que manejan más de 4 terabytes de datos (96).

34 Sentence Query Language

35 Doctrine Query Language

36 Hibernate Query Language

37 Java Persistence Query Language

# 2. CAPÍTULO 2. CARACTERÍSTICAS DEL SISTEMA

## 2.1 Introducción

A continuación se realiza una descripción detallada de la solución propuesta, que incluye el diseño del software con todas las herramientas y tecnologías a utilizar; el modelado completo de la solución que comprende el análisis de la propuesta, la definición de los requerimientos funcionales y no funcionales, entre otros; la vista arquitectónica de sistema, que contempla: la línea base definida y los principales componentes que integran la solución, así como los algoritmos y fórmulas que utiliza el sistema en su funcionamiento interno.

## 2.2 Modelado de la solución

### 2.2.1 Requerimientos funcionales de software

| No | Funcionalidades | Prioridad | Esfuerzo |
|---|---|---|---|
| 1 | Adicionar indicador | Alta | 4 |
| 2 | Modificar indicador | Alta | 4 |
| 3 | Eliminar indicador | Alta | 4 |
| 4 | Listar indicadores | Alta | 4 |
| 5 | Adicionar variable | Alta | 4 |
| 6 | Modificar variable | Alta | 4 |
| 7 | Eliminar variable | Alta | 4 |
| 8 | Listar variables | Alta | 4 |
| 9 | Adicionar subvariable | Alta | 4 |
| 10 | Modificar subvariable | Alta | 4 |
| 11 | Eliminar subvariable | Alta | 4 |
| 12 | Listar subvariables | Alta | 4 |
| 13 | Adicionar organismo | Alta | 4 |
| 14 | Modificar organismo | Alta | 4 |
| 15 | Eliminar organismo | Alta | 4 |

| | | | |
|---|---|---|---|
| 16 | Listar organismos | Alta | 4 |
| 17 | Adicionar organización | Alta | 4 |
| 18 | Modificar organización | Alta | 4 |
| 19 | Eliminar organización | Alta | 4 |
| 20 | Listar organizaciones | Alta | 4 |
| 21 | Adicionar caso real | Alta | 8 |
| 22 | Modificar caso real | Alta | 4 |
| 23 | Eliminar caso real | Baja | 4 |
| 24 | Listar casos reales | Media | 8 |
| 25 | Adicionar caso escenario | Alta | 8 |
| 26 | Modificar caso escenario | Alta | 4 |
| 27 | Eliminar caso escenario | Baja | 4 |
| 28 | Listar casos escenarios | Media | 8 |
| 29 | Calcular probabilidad de éxito | Media | 10 |
| 30 | Listar propuesta de escenarios | Media | 10 |

Tabla 2.2.1.1 Requerimientos funcionales de software

### 2.2.2 Requerimientos no funcionales de software

Los requerimientos no funcionales son condiciones que debe cumplir un sistema para satisfacer un contrato o una especificación. Están regidos por las necesidades del usuario para poder resolver un problema o conseguir un beneficio determinado. Se refieren a las propiedades emergentes del sistema como la fiabilidad, el tiempo de respuesta, la capacidad de almacenamiento, la capacidad de los dispositivos de entrada/salida, y la representación de datos que se utilizan en las interfaces del sistema.
Estos requerimientos son de gran significación en la aceptación del software, debido a que representan las ventajas más visibles al usuario y repercuten en el óptimo funcionamiento y mantenimiento del sistema.

**Apariencia o interfaz externa**

La interfaz debe ser lo más sencilla posible, para que pueda ser manejada por cualquier tipo de usuario.

Debe presentar una correcta combinación de colores.

**Usabilidad**

El sistema debe poder ser usado por cualquier persona que tenga conocimientos básicos de informática y del programa de mejora de software.

**Seguridad**

- Autenticación
- Autorización
- Implementación de auditoría

**Rendimiento**

Los tiempos de respuesta y velocidad de procesamiento de la información serán rápidos, no mayores de 5 segundos para las actualizaciones y 15 para las recuperaciones.

**Soporte**

- Servidor de aplicaciones: Se requiere que esté instalado un intérprete de ficheros PHP rápido y con las últimas actualizaciones del lenguaje.
- Servidor de base de datos: Se requiere que esté instalado un gestor de base de datos que soporte grandes volúmenes de datos, maneje la concurrencia y transacciones.
- Cliente: Se requiere que esté instalado un navegador que interprete Javascript.

**Portabilidad**

El sistema debe ser multiplataforma haciéndose énfasis en la plataforma Linux.

**Software**

Para el cliente:

1. Navegador Mozilla Firefox o Chromium Browser.

2. Sistema operativo Windows XP o superior o Linux.

**Hardware**

Requerimientos mínimos para la conexión del cliente:

1. Intel(R) Core(TM) i3-2120 CPU @ 3.30GHz.
2. 2 GB RAM o superior

**Restricciones de diseño**

El producto de software final debe diseñarse sobre una arquitectura MVC. Emplear los estándares establecidos (diseño de interfaces, base de datos y codificación). Emplear como lenguaje del lado del servidor, PHP y del lado del cliente Javascript.

### 2.3 Historias de usuarios

Las historias de usuario son la forma en que se especifican en XP los requerimientos funcionales del sistema. Estas se escriben desde la perspectiva del cliente aunque los desarrolladores pueden brindar también su ayuda en la identificación de las mismas. El contenido de estas debe ser concreto y sencillo. Durante la fase de exploración se realizaron 30 historias de usuario. A continuación se detallan 2 de ellas y el resto, en los anexos.

| **Historia de usuario** | |
|---|---|
| **Número:** 29 | **Usuarios:** Administrador |
| **Nombre de historia:** Calcular probabilidad de éxito | |
| **Prioridad en negocio:** Alta | **Riesgo en Desarrollo:** Bajo |
| **Puntos Estimados:** 8 | **Iteración asignada:** 1 |
| **Programador responsable:** Karel Riverón Escobar | |
| **Descripción:** El administrador del sistema calcula la probabilidad de éxito que tendrá un caso escenario en función de los valores de sus subvariables y recarga la tabla con el nuevo valor de probabilidad. | |
| **Observaciones:** | |

Tabla 2.3.1 Historia de usuario Calcular probabilidad de éxito

| Historia de usuario | |
|---|---|
| **Número:** 30 | **Usuarios:** Administrador |
| **Nombre de historia:** Listar propuesta de escenarios | |
| **Prioridad en negocio:** Alta | **Riesgo en Desarrollo:** Bajo |
| **Puntos Estimados:** 8 | **Iteración asignada:** 1 |
| **Programador responsable:** Karel Riverón Escobar | |
| **Descripción:** El administrador del sistema muestra una tabla con la propuesta de escenarios y una gráfica para visualizar los valores de probabilidad de éxito de estos escenarios. | |
| **Observaciones:** | |

Tabla 2.3.2 Historia de usuario Listar propuesta de escenarios

## 2.4 Diseño

### 2.4.1 Descripción de la arquitectura

Para la implementación de la solución se propone utilizar el estilo en capas (capa de presentación, capa de negocio, capa de acceso a datos y capa de datos) y el patrón Modelo – Vista – Controlador (MVC) el cual está formado por tres niveles:

- El modelo representa la información con la que trabaja la aplicación, es decir, su lógica de negocio. Integra las clases que contienen las consultas a la base de datos, así como la definición de las tablas, sus relaciones y atributos.
- La vista transforma el modelo en una página web que permite al usuario interactuar con ella. Además, determina la interfaz que se muestra finalmente al cliente para su intercambio con la aplicación.
- El controlador se encarga de procesar las interacciones del usuario y realiza los cambios apropiados en el modelo o en la vista. Gestiona todas las peticiones del usuario y se encarga de darles respuesta.

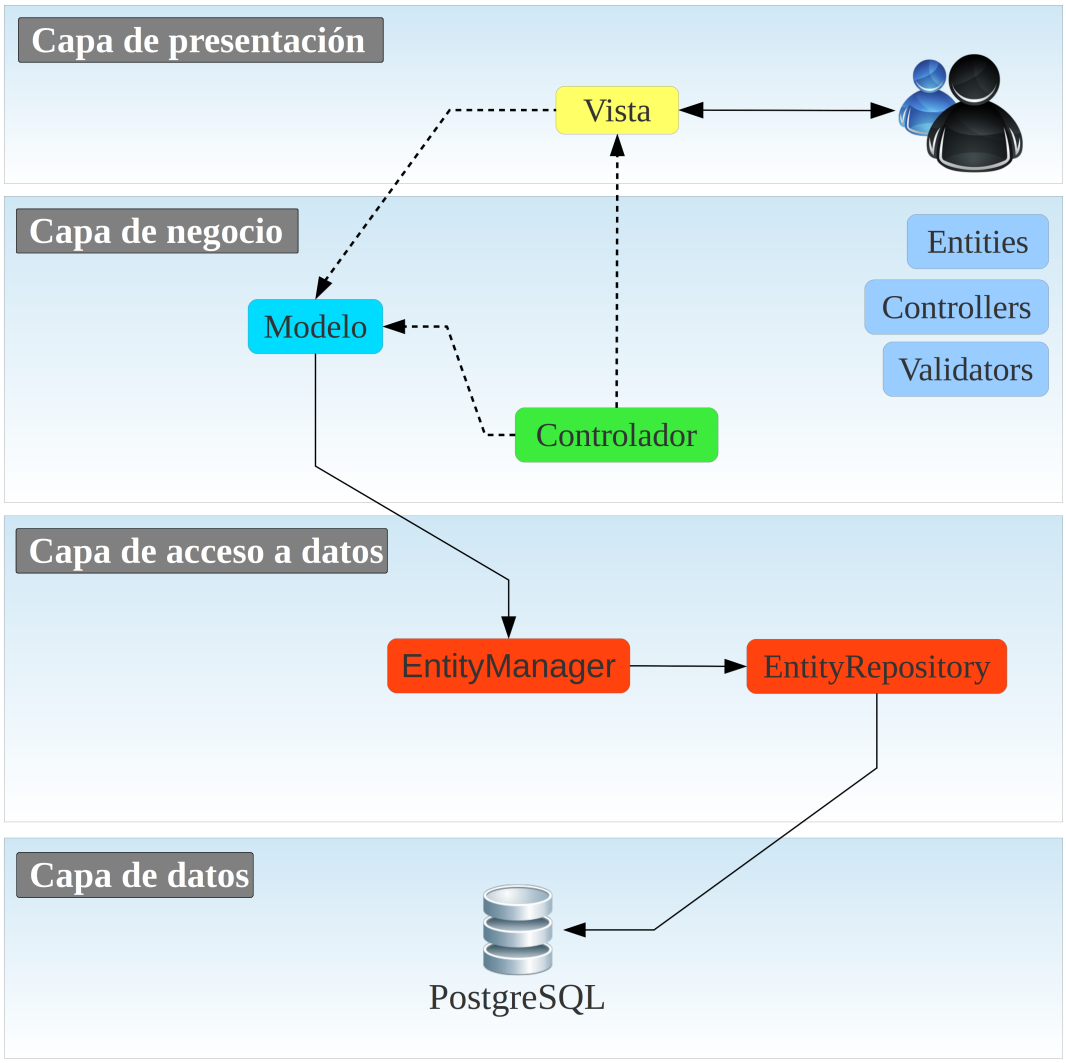


Figura 2.4.1.1: Diseño arquitectónico

### 2.4.2 Plan de iteraciones

| Iteración | Historia de usuario | Prioridad | Esfuerzo | Tiempo de duración |
|---|---|---|---|---|
| Iteración 1 | Adicionar indicador | Alta | 4 | 32 |
| | Modificar indicador | Alta | 4 | |
| | Eliminar indicador | Alta | 4 | |

| | | | | |
|---|---|---|---|---|
| | Listar indicadores | Alta | 4 | |
| | Adicionar variable | Alta | 4 | |
| | Modificar variable | Alta | 4 | |
| | Eliminar variable | Alta | 4 | |
| | Listar variables | Alta | 4 | |
| Iteración 2 | Adicionar subvariable | Alta | 4 | 64 |
| | Modificar subvariable | Alta | 4 | |
| | Eliminar subvariable | Alta | 4 | |
| | Listar subvariables | Alta | 4 | |
| | Adicionar organismo | Alta | 4 | |
| | Modificar organismo | Alta | 4 | |
| | Eliminar organismo | Alta | 4 | |
| | Listar organismos | Alta | 4 | |
| | Adicionar organización | Alta | 4 | |
| | Modificar organización | Alta | 4 | |
| | Eliminar organización | Alta | 4 | |
| | Listar organizaciones | Alta | 4 | |
| Iteración 3 | Adicionar caso real | Alta | 8 | 20 |
| | Modificar caso real | Alta | 4 | |
| | Adicionar caso escenario | Alta | 4 | |
| | Modificar caso escenario | Alta | 4 | |
| Iteración 4 | Listar casos reales | Media | 8 | 36 |
| | Listar casos escenarios | Media | 8 | |
| | Calcular probabilidad de éxito | Media | 10 | |
| | Listar propuesta de escenarios | Media | 10 | |
| Iteración 5 | Eliminar caso real | Baja | 4 | 8 |
| | Eliminar caso escenario | Baja | 4 | |

Tabla 2.4.2.1 Plan de iteraciones

### 2.4.3 Diagramas de clases persistentes

#### Diagrama de clases persistentes

A través del diagrama de clases persistente se definen los conceptos que se manejan en el sistema y que sirven para describir la estructura de la base de datos. Es decir, en dicho modelo se representan las entidades, sus atributos y tipos, sus relaciones y las restricciones que deben cumplirse sobre ellos.

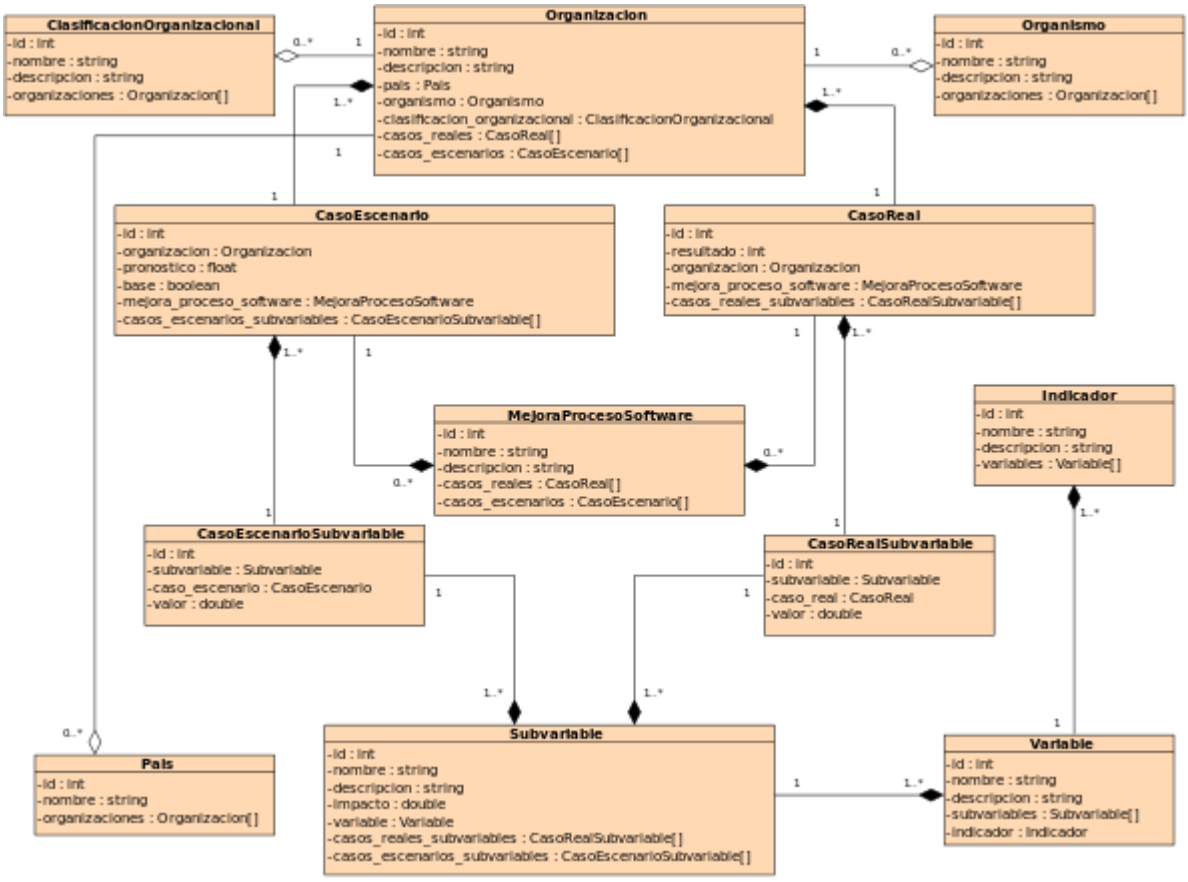


Figura 2.4.3.1: Diagrama de clases

**Descripción de las clases**

Se identificaron 12 clases persistentes. A continuación se detallan 2 de ellas y el resto, en los anexos.

| **Nombre: CasoReal** | |
|---|---|
| **Tipo de clase:** Entidad | |
| **Atributo** | **Tipo** |
| id | Entero |
| resultado | Entero |
| organizacion | Organizacion |
| mejora_proceso_software | MejoraProcesoSoftware |
| casos_reales_subvariable | CasoRealSubvariable [] |
| **Para cada responsabilidad:** | |
| Nombre | InsertarCasoReal |
| Descripción | Busca el objeto CasoReal recibido como parámetro en la base de datos. En caso de no encontrarlo, lo registra. |
| Nombre | ActualizarCasoReal |
| Descripción | Busca el objeto CasoReal que coincida con los parámetros recibidos en la base de datos y lo actualiza con los nuevos datos. |
| Nombre | EliminarCasoReal |
| Descripción | Elimina el caso real cuyo identificador coincide con el recibido como parámetro. |
| Nombre | ListarCasosReales |
| Descripción | Devuelve una lista con todas los casos reales. |
| Nombre | ListarCasosRealesConTodo |
| Descripción | Devuelve una lista con todos los casos reales dentro de un rango determinado con el objeto de paginar dicha lista. |
| Nombre | UltimoCasoRealInsertado |

| | |
|---|---|
| Descripción | Devolver el último caso real insertado en la base de datos. |
| Nombre | RegistrarCasosReales |
| Descripción | Crea objetos de tipo Caso Real con los datos de cada una de los casos reales y los registra en la base de datos. |

Tabla 2.4.3.2 Descripción de la clase CasoReal

| **Nombre: CasoEscenario** | |
|---|---|
| **Tipo de clase:** Entidad | |
| **Atributo** | **Tipo** |
| id | Entero |
| pronostico | Real |
| organizacion | Organizacion |
| base | Boolean |
| mejora_proceso_software | MejoraProcesoSoftware |
| casos_escenarios_subvariables | CasoEscenarioSubvariable [] |
| **Para cada responsabilidad:** | |
| Nombre | InsertarCasoEscenario |
| Descripción | Busca el objeto CasoEscenario recibido como parámetro en la base de datos. En caso de no encontrarlo, lo registra. |
| Nombre | ActualizarCasoEscenario |
| Descripción | Busca el objeto CasoReal que coincida con los parámetros recibidos en la base de datos y lo actualiza con los nuevos datos. |
| Nombre | ListarCasosEscenarios |
| Descripción | Devuelve una lista con todas los casos escenarios. |
| Nombre | UltimoCasoEscenarioInsertado |

| | |
|---|---|
| Descripción | Devolver el último caso esceario insertado en la base de datos. |
| Nombre | RegistrarCasosEscenariosSeleccionados |
| Descripción | Registrar los casos escenarios seleccionados por el usuario. |
| Nombre | CalcularProbabilidadDeExito |
| Descripción | Calcula la probabilidad de éxito ejecutando un procedimiento almacenado que realiza el proceso. |
| Nombre | ListarCasosRecuperados |
| Descripción | Devuelve una lista con los casos recuperados a partir del procesamiento del algoritmo. |
| Nombre | ListarCasosEscenariosCruzados |
| Descripción | Devuelve una lista con los escenario después de haber realizado un proceso de cruzamiento para obtener mejores resultados. |

Tabla 2.4.3.3 Descripción de la clase CasoEscenario

### 2.4.4 Diseño de la base de datos

**Diagrama Entidad - Relación**

A través del modelo de datos se definen los conceptos que se manejan en el sistema y que sirven para describir la estructura de la base de datos diseñada. Es decir, en dicho modelo se representan los datos, sus atributos y tipos, sus relaciones y las restricciones que deben cumplirse sobre ellos.

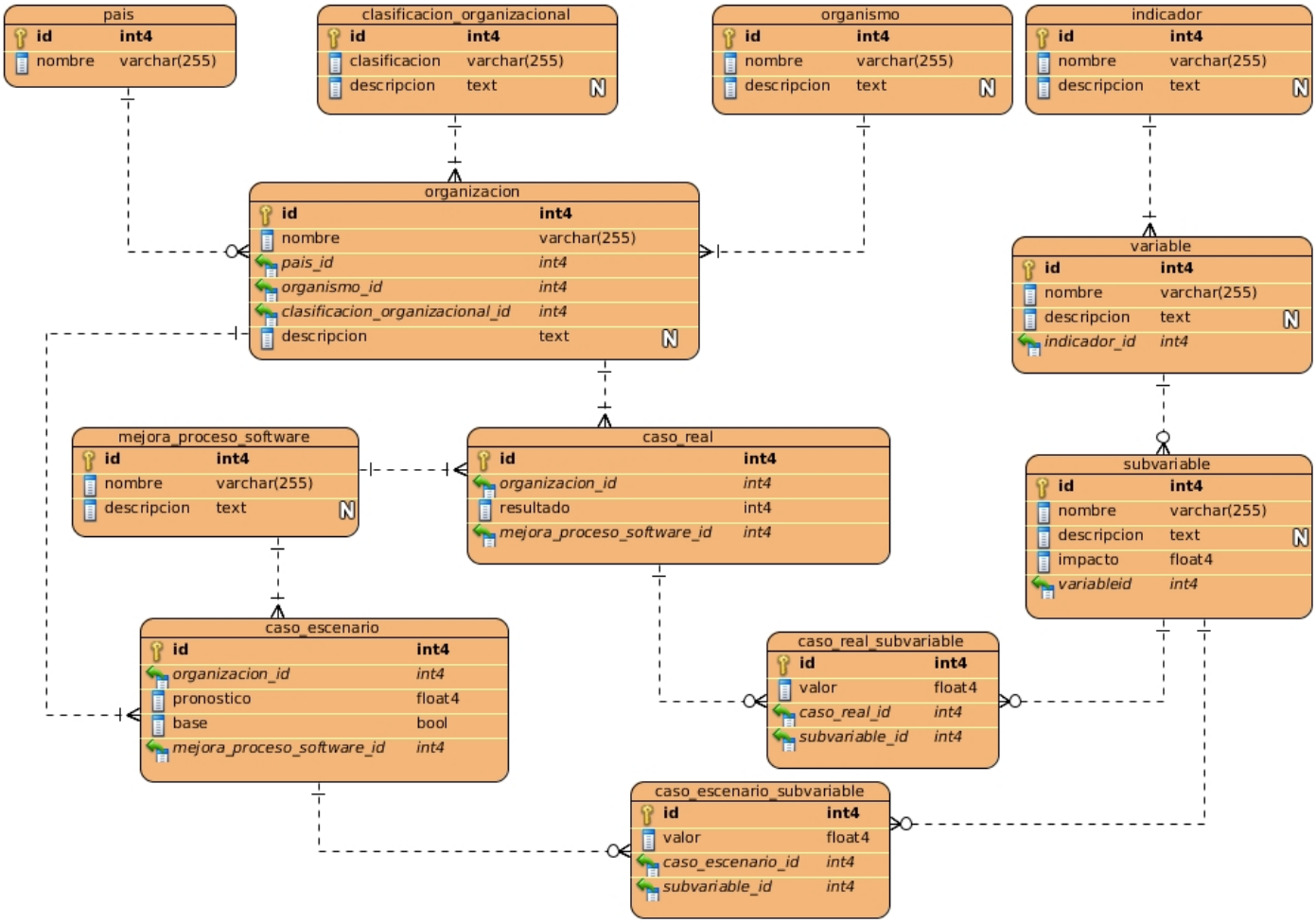


Figura 2.4.4.1: Diagrama Entidad-Relación

### Descripción de las tablas

Se identificaron 12 tablas. A continuación se detallan 2 de ellas y el resto, en los anexos.

| **Nombre: caso_real** | | |
|---|---|---|
| **Descripción:** Almacena todos los casos reales. | | |
| **Atributo** | **Tipo** | **Descripción** |
| id | integer | Identificador. |
| organizacion_id | integer | Llave foránea que referencia al |

| | | |
|---|---|---|
| | | identificador de la tabla organizacion. |
| mejora_proceso_software_id | integer | Llave foránea que referencia al identificador de la tabla mejora_proceso_software. |
| resultado | integer | Resultado del caso (1 si es exitoso, 2 si resultó un fracaso). |

Tabla 2.4.4.2: Descripción de la tabla caso_real

| **Nombre: caso_escenario** | | |
|---|---|---|
| **Descripción:** Almacena todos los casos escenarios. | | |
| **Atributo** | **Tipo** | **Descripción** |
| id | integer | Identificador. |
| organizacion_id | integer | Llave foránea que referencia al identificador de la tabla organizacion. |
| mejora_proceso_software_id | integer | Llave foránea que referencia al identificador de la tabla mejora_proceso_software. |
| pronostico | double | Probabilidad de éxito del caso. |
| base | boolean | Define si es un caso registrado (true) o si es un caso creado a partir de la combinación de casos recuperados (false). |

Tabla 2.4.4.3: Descripción de la tabla caso_escenario

### 2.5 Validación del diseño

El diseño está enfocado a convertir los requerimientos del cliente en un modelo que al ser implementado, se obtenga el producto deseado. Generalmente constituye el punto de partida para el desarrollo de

software una vez especificados los requerimientos del sistema. Realizar una validación del mismo para verificar su calidad y flexibilidad, garantiza una buena base para la implementación. Con este objetivo se utilizan un conjunto de métricas de software orientadas a determinar, entre otros aspectos, qué características del modelo de diseño se pueden estimar para comprobar que el sistema será fácil de implementar en cuanto a organización.

Lorenz y Kidd dividen las métricas basadas en clases en cuatro categorías: tamaño, herencia, valores internos y valores externos. Las métricas orientadas a tamaños para una clase se centran en cálculos de atributos y de operaciones, para luego promediar los valores para el sistema en su totalidad. Las métricas basadas en herencia se centran en la forma en que se reutilizan las operaciones en la jerarquía de clases. Las métricas para valores internos de clase examinan la cohesión y asuntos relacionados con el código así como las métricas orientadas a valores externos examinan el acoplamiento y la reutilización (97).

Para validar el diseño de la presente investigación se seleccionaron las métricas Tamaño Operacional de Clase y Relaciones entre Clases.

**Tamaño Operacional de Clase**

Esta métrica se determina por el número total de operaciones que están encapsuladas dentro de la clase. Grandes valores de esta medida muestran que la clase puede tener demasiada responsabilidad, lo cual reducirá la reusabilidad de la misma y complicará su implementación. Por otro lado, en cuanto menor sea el valor medio para el tamaño más probable es que las clases tengan menos responsabilidad y complejidad y más nivel de reutilización.

Para evaluar las métricas son necesarios los valores de los umbrales para los parámetros de calidad. Algunos especialistas plantean umbrales para esta métrica basándose en el promedio de operaciones por clases obtenidos, estos valores fueron los aplicados en el diseño del sistema y los mismos son reflejados en la siguiente tabla.

| Atributos de calidad | Categoría | Umbrales |
|---|---|---|
| Responsabilidad | Baja | |
| | Media | |
| | Alta | |

| | | |
|---|---|---|
| Complejidad de implementación | Baja | $x \leqslant promedio$ |
| | Media | $promedio \leqslant x \leqslant 2 * promedio$ |
| | Alta | $x \geqslant 2 * promedio$ |
| Reutilización | Alta | $x \leqslant promedio$ |
| | Media | $promedio \leqslant x \leqslant 2 * promedio$ |
| | Baja | $x \geqslant 2 * promedio$ |

Tabla 2.5.1: Umbrales de la métrica Tamaño Operacional de Clase

Tras un análisis de los resultados arrojados por la evaluación bajo los instrumentos de medición de la métrica Tamaño Operacional de Clase, se demuestra que se alcanzaron buenos valores para cada uno de los atributos de calidad evaluados, puesto que, como se puede observar, el 54.17% de las clases del software contienen un número menor que el promedio de procedimientos por clases, lo cual influye positivamente en el hecho de que predomine una responsabilidad baja de las clases y hace que carezcan de mucha complejidad, por lo que se tornan mucho más reutilizables. Solamente un 12.50% de las clases tienen pocas posibilidades de reutilización y una gran complejidad de implementación. Estos valores demuestran que los indicadores de reutilización, complejidad y responsabilidad son aceptables.

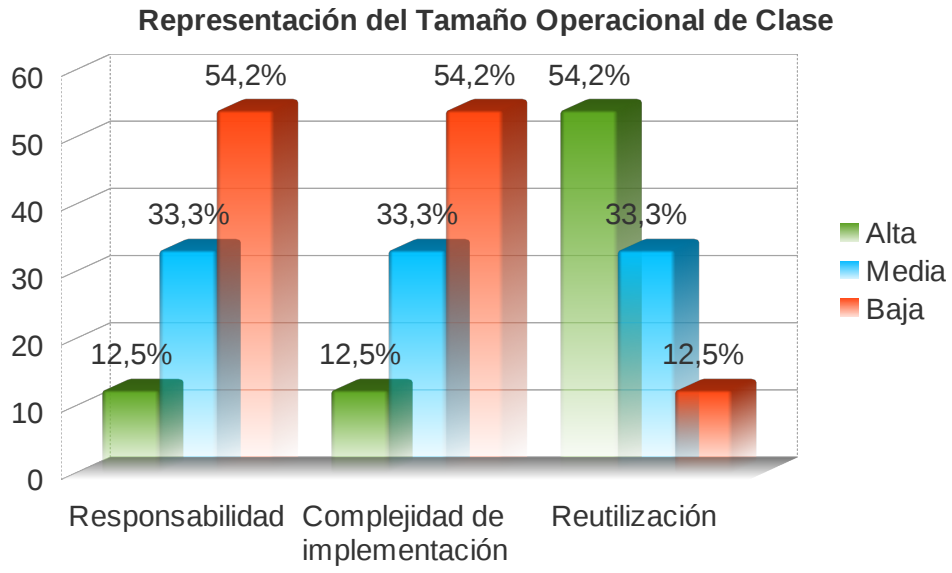


Figura 2.5.2: Representación de la métrica Tamaño Operacional de Clase

**Relaciones entre Clases**

Esta métrica se determina por la cantidad de relaciones existentes entre las clases contenidas en el diseño. El número de dependencias es directamente proporcional al nivel de acoplamiento, a la complejidad del mantenimiento y a la cantidad de pruebas a realizar sobre las clases, y es inversamente proporcional al grado de reutilización de las mismas. La aplicación de esta métrica evidencia que el diseño del sistema es aceptable con respecto a todos los atributos de calidad afectados, arrojando resultados positivos.

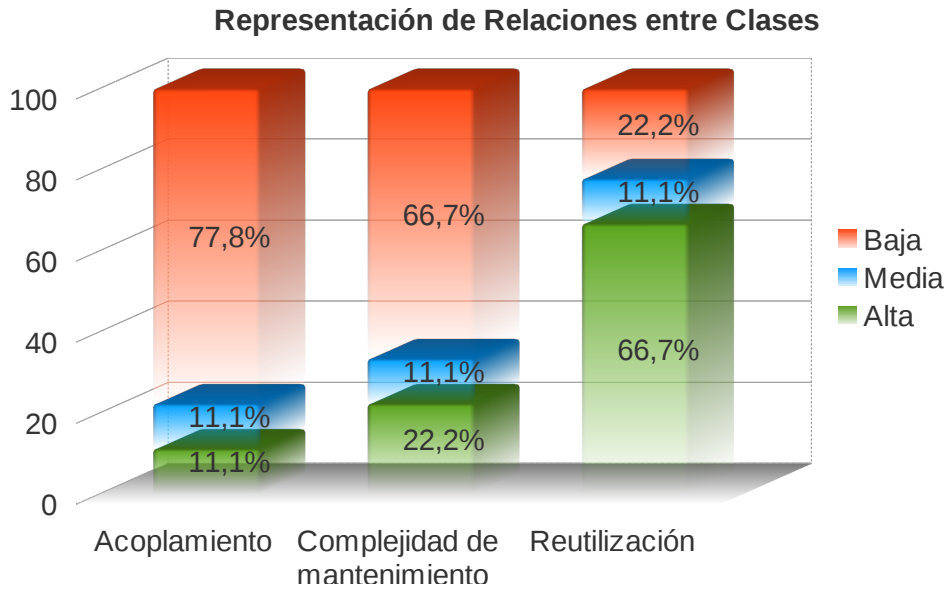


Figura 2.5.3: Representación de la métrica Relaciones entre clases

# 3. CAPÍTULO 3. IMPLEMENTACIÓN Y PRUEBA

## 3.1 Introducción

En este capítulo se aborda los aspectos más importantes relacionados con la implementación del sistema y la validación del mismo. Se realizan el diagrama de despliegue y el diagrama de compontes. También se valida mediante pruebas unitarias y pruebas de aceptación, si el software realizado responde a las necesidades del cliente y se corresponde con los requerimientos funcionales planteados durante la etapa del análisis. Además, se realizó un experimento con casos generados aleatoriamente para comprobar el correcto funcionamiento de los algoritmos.

## 3.2 Implementación

### 3.2.1 Plan de liberaciones

| No. | Historia de usuario | Esfuerzo | Duración | Fecha de inicio | Fecha de fin |
|---|---|---|---|---|---|
| 1 | Adicionar indicador | 4 | 4 | 25/11/2012 | 12/01/2012 |
| | Modificar indicador | 4 | | | |
| | Eliminar indicador | 4 | | | |
| | Listar indicadores | 4 | | | |
| | Adicionar variable | 4 | | | |
| | Modificar variable | 4 | | | |
| | Eliminar variable | 4 | | | |
| | Listar variables | 4 | | | |
| 2 | Adicionar subvariable | 4 | 6 | 13/01/2013 | 23/02/2013 |
| | Modificar subvariable | 4 | | | |
| | Eliminar subvariable | 4 | | | |
| | Listar subvariables | 4 | | | |
| | Adicionar organismo | 4 | | | |
| | Modificar organismo | 4 | | | |
| | Eliminar organismo | 4 | | | |

| | | | | | |
|---|---|---|---|---|---|
| | Listar organismos | 4 | | | |
| | Adicionar organización | 4 | | | |
| | Modificar organización | 4 | | | |
| | Eliminar organización | 4 | | | |
| | Listar organizaciones | 4 | | | |
| 3 | Adicionar caso real | 8 | 3 | 24/02/2013 | 16/03/2013 |
| | Modificar caso real | 4 | | | |
| | Adicionar caso escenario | 4 | | | |
| | Modificar caso escenario | 4 | | | |
| 4 | Listar casos reales | 8 | 5 | 17/03/2013 | 20/04/2013 |
| | Listar casos escenarios | 8 | | | |
| | Calcular probabilidad de éxito | 10 | | | |
| | Listar propuesta de escenarios | 10 | | | |
| 5 | Eliminar caso real | 4 | 1 | 21/04/2013 | 27/04/2013 |
| | Eliminar caso escenario | 4 | | | |

Tabla 3.2.1.1: Plan de liberaciones

### 3.2.2 Diagrama de componentes

Los diagramas de componentes permiten describir los elementos físicos que integran el sistema y las relaciones que existen entre ellos. Muestran además las opciones de realización incluyendo código fuente,binario y ejecutable. Los componentes representan todos los tipos de elementos software que entran en la producción de aplicaciones informáticas. Pueden ser simples archivos, paquetes, y/o bibliotecas cargadas dinámicamente. A continuación se exponen el diagrama de componentes asociado al sistema implementado.

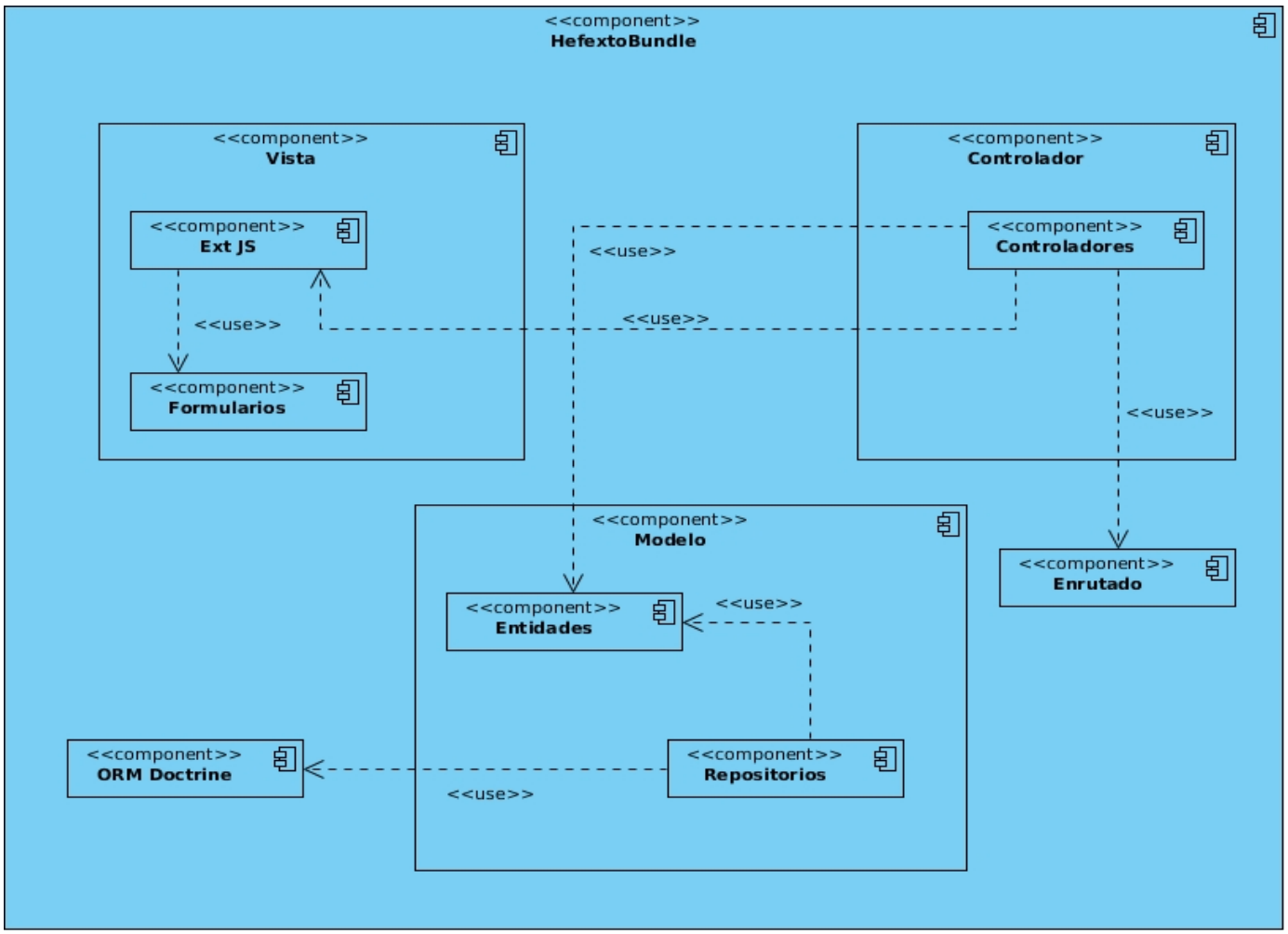


Figura 3.2.2.1: Diagrama de componentes

### 3.2.3 Diagrama de despliegue

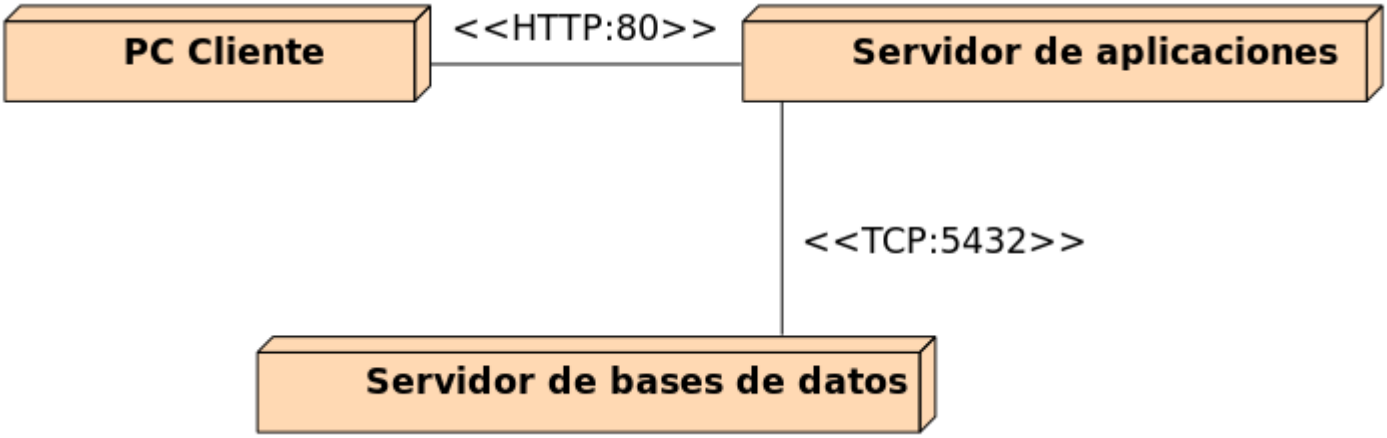


Figura 3.2.3.1: Diagrama de despliegue

## 3.3 Diseño y ejecución de las pruebas de software

Las pruebas de software constituyen un instrumento para determinar el nivel de la calidad de un producto. La metodología XP propone la realización de pruebas de aceptación, las cuales que se crean a partir de las historias de usuario. Durante las iteraciones las historias de usuarios seleccionadas serán traducidas a pruebas de aceptación. En ellas se especifican, desde la perspectiva del cliente, los escenarios para probar que una historia de usuario ha sido implementada correctamente. Una historia de usuario puede tener todas las pruebas de aceptación que necesite para asegurar su correcto funcionamiento. El objetivo final de éstas es garantizar que los requerimientos han sido cumplidos y que el sistema es aceptable.

En esta investigación se decidió realizar también pruebas unitarias, con motivo de verificar el correcto funcionamiento de los procedimientos, agrupados en los distintos componentes.

Durante la realización de las pruebas se generaron un conjunto de casos de prueba, especificados deforma estructurada mediante técnicas de prueba. Los mismos pueden ser consultados en los anexos 3 y 4.

### 3.4 Validación de la solución

Para validar la solución propuesta se decidió emplear la técnica de validación por caso de estudio. Para ello se generaron 3000 casos en la base de casos de manera aleatoria. De ellos el 50% de ellos fueron exitosos y el resto resultaron fracasados.

A continuación se muestra una tabla con las subvariables y sus valores de impacto, los mismos son generados de manera aleatoria entre 0 y 10.

| No. | Nombre de la subvariable | Impacto |
|---|---|---|
| 1 | Colaboración - competencia | 6 |
| 2 | Relaciones individuo – individuo | 5 |
| 3 | Relaciones intergrupales | 8 |
| 4 | Formación para la mejora de proceso | 4 |
| 5 | Capacidad de aprendizaje | 6 |
| 6 | Capacidad de adaptación y autorrenovación | 5 |
| 7 | Experiencias en la producción | 9 |
| 8 | Experiencias en roles | 7 |
| 9 | Reconocimientos y castigos | 4 |
| 10 | Satisfacción con la política de retribuciones | 8 |
| 11 | Satisfacción con la política de estimulaciones | 6 |
| 12 | Motivación por el trabajo | 8 |
| 13 | Satisfacción con el trabajo | 6 |
| 14 | Identificación con la organización | 5 |
| 15 | Orientación a la mejora continua | 10 |
| 16 | Orientación a la satisfacción del cliente | 8 |
| 17 | Orientación a procesos | 9 |
| 18 | Gestión del cambio | 7 |
| 19 | Planeación estratégica | 6 |
| 20 | Establecimiento y dominio de los objetivos organizacionales | 4 |
| 21 | Establecimiento y delimitación de roles organizacionales | 5 |
| 22 | Selección de personal e inducción a la organización | 6 |

| | | |
|---|---|---|
| 23 | Programas de desarrollo y planes de superación | 6 |
| 24 | Evaluación del desempeño | 8 |
| 25 | Protección e higiene del trabajo | 7 |
| 26 | Confianza en la dirección | 7 |
| 27 | Competencia de los directivos | 6 |
| 28 | Supervisión | 9 |
| 29 | Estilo de dirección | 6 |
| 30 | Relaciones Jefe – subordinados | 8 |
| 31 | Participación | 6 |
| 32 | Información | 5 |
| 33 | Comunicación | 3 |
| 34 | Perspectivas de la organización | 7 |
| 35 | Eficiencia | 10 |
| 36 | Eficacia | 8 |
| 37 | Estabilidad interna de la organización | 7 |
| 38 | Trabajo en equipo | 7 |
| 39 | Disponibilidad de las personas | 6 |
| 40 | Disponibilidad de tiempo | 6 |
| 41 | Disponibilidad de infraestructura | 9 |

Tabla 3.4.1: Subvariables y sus correspondientes valores de impacto

**Caso de estudio #1**

Los valores de las subvariables, para los casos exitosos, fueron generados bajo las siguientes condiciones:

- Entre 4 y 5 si el impacto de la subvariable está entre 9 y 10
- Entre 2 y 3 si el impacto de la subvariable está entre 6 y 8
- Entre 0 y 1 si el impacto de la subvariable está entre 3 y 5

Posteriormente se realizó el ingreso de 5 nuevos caso, cuyos valores de subvariables se encontraban

también entre estos intervalos. En la etapa de recuperación de cada uno de los casos el sistema identificó 1500 casos semejantes con resultado de éxito, fijando el umbral de la función de semejanza en 0.75. Por tanto, podemos afirmar que en los 5 casos de prueba se recuperó el 100% de los casos.

**Caso de estudio #2**

Para los casos fracasados, los valores de las subvariables se generaron de manera opuesta a los anteriores:

- Entre 0 y 1 si el impacto de la subvariable está entre 9 y 10
- Entre 2 y 3 si el impacto de la subvariable está entre 6 y 8
- Entre 4 y 5 si el impacto de la subvariable está entre 3 y 5

Luego se realizó el ingreso de 5 nuevos caso, cuyos valores de subvariables se encontraban, al igual que en el caso de estudio anterior, entre los intervalos anteriores. En la etapa de recuperación de cada caso el sistema logró identificar 1500 casos semejantes con resultado de fracaso. En esta ocasión el umbral también se fijó en 0.75. Por tanto, podemos afirmar que en los 5 casos, también se recuperó el 100% de los casos.

## CONCLUSIONES

Como resultados del presente trabajo de investigación se tienen:

1. El estudio sobre la ingeniería y la calidad de software asociados a la mejora de procesos de software y la inteligencia artificial, permitió seleccionar el razonamiento basado en casos como técnica a implementar y un modelo matemático que permite el cálculo de la probabilidad de éxito a partir de la recuperación de los casos más similares.
2. El sistema basado en casos que se obtuvo como resultado de la investigación permite procesar los grandes volúmenes de datos generados por las empresas frente a una mejora de procesos de software a partir del diagnóstico de indicadores que reflejan aspectos sociales de las organizaciones. Dicho sistema permite además la inferencia de un conjunto de escenarios que constituyen la información que las organizaciones pueden tener en cuenta para tomar decisiones frente a una mejora de procesos de software.
3. Con la validación realizada a la solución implementada a partir de pruebas de aceptación y de pruebas unitarias empleando los métodos de caja negra y caja blanca, se pudo constatar después de 4 iteraciones, que el sistema quedó libre de no conformidades, por lo que el cliente puede utilizarlo en un entorno real.
4. El experimento realizado con valores aleatorios generados convenientemente, corroboró la correcta implementación del modelo matemático empleado para el cálculo de la probabilidad de éxito y la inferencia de los escenarios de mejora que se proponen.

## RECOMENDACIONES

A raíz del estudio acerca de las técnicas de inteligencia artificial más utilizadas para resolver problemas de apoyo a la toma de decisiones, se determinó que el empleo de las redes neuronales artificiales permitiría el aprendizaje del sistema en consecuencia con cambios futuros en los umbrales de los indicadores. Se consideró además que el uso de la lógica difusa admitiría cierto grado de incertidumbre en los valores asignados a dichos indicadores. A partir de este análisis, se identificó que la utilización de estas 2 técnicas combinadas con el razonamiento basado en casos le aportarán al sistema un valor agregado para realizar los cálculos con mayor precisión y apoyar aún más el proceso de toma de decisiones.

## REFERENCIAS BIBLIOGRÁFICAS


1. THE STANDISH GROUP. Chaos Manifiesto 2011. En: *The Standish Group* [En línea]. 2011. [Accedido: 25 de septiembre de 2011]. Disponible desde: http://standishgroup.com/newsroom/chaos_manifesto_2011.php.

2. PRESSMAN, ROGER S. *Ingeniería del software, un Enfoque Práctico*. 7ma. S.l.: McGraw-Hill Interamericana de España, SL, 2010. ISBN 9701054733.

3. LUDWIG BAUER, FRIEDRICH. *Software engineering, Information Processing*. Amsterdam: North-Holland, 1972.

4. BOEHM, BARRY. Software Engineering. En: Diciembre de 1976, Vol. 25, no. 12, pp. 1226–1241. DOI 10.1109/TC.1976.1674590.

5. HUMPHREY, WATTS S. *A discipline for software engineering*. 1ra. Boston: Addison-Wesley, 1995.

6. SOMMERVILLE, IAN. *Software Engineering*. 8va. S.l.: Addison- Wesley, 2007.

7. MATHIASSEN, LARS y POURKOMEYLIAN, POUYA. Managing knowledge in a software organization. En: *Journal of Knowledge Management* [En línea]. 2003, Vol. 7, no. 2. Disponible desde: http://vbn.aau.dk/en/publications/managing-knowledge-in-a-software-organization%281fe48eb0-9c2f-11db-8ed6-000ea68e967b%29/export.html.

8. MATTURRO MAZONI, GERARDO. *Modelo para la gestión del conocimiento y la experiencia integrada a las prácticas y procesos de desarrollo software* [En línea]. Madrid: Universidad Politécnica de Madrid, 2010. Disponible desde: http://www.ort.edu.uy/fi/pdf/tesismatturro2010.pdf.

9. CMMI. Overview. En: *Software Engineering Institute* [En línea]. 2000. [Accedido: 4 de mayo de 2011]. Disponible desde: http://www.sei.cmu.edu/cmmi/index.cfm.

10. ISO. Home. En: *ISO - International Organization for Standardization* [En línea]. 2000. [Accedido: 30 de septiembre de 2011]. Disponible desde: http://www.iso.org/iso/home.htm.

11. PMI. Standards Overview. En: *Project Management Institute* [En línea]. 2005. [Accedido: 30 de septiembre de 2011]. Disponible desde: http://www.pmi.org/PMBOK-Guide-and-Standards.aspx.

12. GLASS, ROBERT L. IT Failure Rates - 70 Percent or 10-15 Percent? En: *Computing Trends*. May 2005, Vol. 22, no. 3.

13. DOUNOS, P. y BOHORIS, G. Factors for the Design of CMMI-Based Software Process Improvement Initiatives. En: *14th Panhellenic Conference on Informatics*. Piraeus, Greece: s.n., 10 de septiembre de 2010. pp. 43 – 47.

14. MONTONI, M.A. y ROCHA, A.R. Applying Grounded Theory to Understand Software Process Improvement Implementation. En: *Seventh International Conference on the Quality of Information and Communications Technology (QUATIC)*. Porto, Portugal: s.n., 29 de septiembre de 2010. pp. 25 – 34.

15. BOAS, G.V., DA ROCHA, A.R.C. y PECEGUEIRO DO AMARAL, M. An Approach to Implement Software Process Improvement in Small and Mid Sized Organizations. En: *Seventh International Conference on the Quality of Information and Communications Technology (QUATIC)*. Porto, Portugal: s.n., 2010.

16. SANTOS, G., KALINOWSKI, M., DA ROCHA, A. R, TRAVASSOS, G. H., WEBER, K. C. y ANTONIONI, J. A. MPS.BR: A Tale of Software Process Improvementand Performance Results in the Brazilian SoftwareIndustry. En: *Seventh International Conference on the Quality of Information and Communications Technology (QUATIC)*. Porto, Portugal: s.n., 2010.

17. ALLISON, I. Organizational Factors Shaping Software Process Improvement in Small-Medium Sized Software Teams: A Multi-Case Analysis. En: *Seventh International Conference on the Quality of Information and Communications Technology (QUATIC)*. Porto, Portugal: s.n., 29 de septiembre de 2010.

18. FORRADELLAS, PATRICIA, PANTALEO, GUILLERMO y ROGERS, JUAN. El modelo CMM/CMMI - Cómo garantizar el éxito del proceso de mejoras en las organizaciones, superando los conflictos y tensiones generados por su implementación. En: [En línea]. [Accedido: 28 de septiembre de 2011].

Disponible desde: http://www.it-mentor.com.ar/pdf/CMMCulturaOrg.pdf.

19. ANABALÓN, JUAN RODRIGO. Las Causas más Comunes de Falla en la Implantación de Mejoras en Software. En: [En línea]. 4 de noviembre de 2005, [Accedido: 28 de septiembre de 2011]. Disponible desde: http://www.geocities.ws/trichotecene/doc/papers/fallasimplementacionsw.pdf.

20. ROGERS, JUAN. Programando los Procesos de Mejoras (CMM/CMMI) a partir de la Cultura de las Organizaciones. En: [En línea]. [Accedido: 30 de septiembre de 2011]. Disponible desde: http://www.it-mentor.com.ar/pdf/CMM_CltryCmbOrg.pdf.

21. NGWENYAMA O. y NIELSEN, PETER AXEL. Competing Values in Software Process Improvement: An Assumption Analysis of CMM From an Organizational Culture Perspective. En: *IEEE Transactions on Engineering Management*. February 2003, Vol. 50, no. 1.

22. TRUJILLO, YAIMÍ, TAMAYO, LEANET, RODRÍGUEZ, YULEIDIS, SÁNCHEZ, YIXI, ENAMORADO, ODANNIS, LEÓN, GIRALDO, BETANCOURT, YADAINY y FEBLES , AILYN. Modelo para valorar las organizaciones previo a la mejora de proceso de software. En: *VI Taller de Calidad en las Tecnologías de la Información y las Comunicaciones* [En línea]. La Habana: s.n., 2013. Disponible desde: http://www.informaticahabana.cu/sites/default/files/documentos/2013/3457/CAL074.pdf.

23. TRUJILLO ARMAS, YURIXAY. *Proceso para Pronosticar el Resultado de Iniciativas de Mejora de Procesos de Software* [En línea]. La Habana: Universidad de las ciencias informáticas, 2012. Disponible desde: http://repositorio_institucional.uci.cu/jspui/handle/ident/TD_05497_12.

24. ISO. Standard ISO 9001:2008. Quality management systems - Requirements. En: *International Organization for Standardization* [En línea]. 2000. [Accedido: 2 de octubre de 2012]. Disponible desde: http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=46486.

25. ASHRAFI, NOUSHIN. The impact of software process improvement on quality: in theory and practice. En: *Information & Management* [En línea]. Agosto de 2003, Vol. 40. Disponible desde: http://www.sciencedirect.com/science/article/pii/S0378720602000964.

26. SOFTWARE ENGINEERING INSTITUTE. CMMI Dev-v1.2. Guía para la integración de procesos. En: *Software Engineering Institute* [En línea]. 2012. [Accedido: 3 de octubre de 2012]. Disponible desde: http://www.sei.cmu.edu/.

27. SCHEIN, EDGAR H. *Proccess consultation*. Cambridge: Addison-Wesley Publishing Company, 1988.

28. CHOO C.W. *La organización inteligente: el empleo de la información para dar significado, crear conocimiento y tomar decisiones*. México D.F.: Oxford University Press, 1999.

29. FINLAY, PAUL N. *Introducing decision support systems*. Oxford: Blackwell Publishers, 1994.

30. TURBAN, EFRAIM. *Decision support and expert systems: management support systems*. Englewood Cliffs, New Jersey: Prentice Hall, 1995.

31. LITTLE, JOHN D. C. Models and Managers: The Concept of a Decision Calculus. En: *Management Science*. 1970, Vol. 16, no. 8.

32. CHANG, MICHAEL G. y MOORE, JEFFERY H. Design of Decision Support Systems. En: *Data Base*. 1980, Vol. 12, no. 2.

33. KEEN, PETER G. W. *Decision support systems: an organizational perspective*. Massachusetts: Addison-Wesley Publishing Company, 1978.

34. KEEN, PETER G. W. *Decision support systems: a research perspective. Decision support systems: issues and challenges.* Oxford: Pergamon Press, 1980.

35. GARCÍA PÉREZ, LYNETTE y MARTÍNEZ DELGADO, EDITH. Base conceptual de un sistema inteligente de apoyo a las decisiones multicriterio. En: *Serie Científica UCI* [En línea]. 2009, Vol. 2, no. 4. Disponible desde: http://publicaciones.uci.cu/index.php/SC/article/download/106/102.

36. MONTES, V. *Inteligencia Artificial*. Colombia: Universidad Regional Autónoma de los Andes, 2010.

37. VERDEJO, ADRIÁN JUAN. Algoritmos Genéticos. En: *Inteligencia Artificial*. 2007,

38. REGUEIRO, CARLOS V. *Modelos básicos de redes neuronales artificiales*. España: Universidad de Santiago de Compostela, 2005.

39. RUSSELL, STUART y NORVIG, PETER. *Artificial Intelligence. A Modern Approach*. 3era Edición. New Jersey: Pearson Education, 2009.

40. CARLOS SOTO, MARLENE. Sistema experto de diagnóstico médico del síndrome de Guillian Barre. En: *Biblioteca Central de la UNMSM* [En línea]. 2009, [Accedido: 19 de frebrero de 2012]. Disponible desde: http://ateneo.unmsm.edu.pe/ateneo/handle/123456789/2388.

41. AAMODT, AGNAR y PLAZA, ENRIC. Case-Based Reasoning: Foundational Issues, Methodological Variations, and System Approaches. En: *Artificial Intelligence Communications*. 1994, Vol. 7, no. 1, pp. 39–52.

42. KOLODNER, JANET L. An introduction to case-based reasoning. En: *Artificial Intelligence Review*. 1992, Vol. 6, no. 1, pp. 3–34.

43. LEAKE, DAVID. *CBR in Context: The Present and Future*. S.l.: AAAI Press/MIT Press, 1996.

44. KOLODNER, JANET L. *Case-Based Reasoning*. S.l.: Morgan Kaufmann Publishers, 1993.

45. BELLO, RAFAEL. *Aplicaciones de la Inteligencia Artificial*. Guadalajara, Jalisco: Ediciones de la noche, 2002. ISBN 970-27-0177-5.

46. LÓPEZ DE MÁNTARAS, R. Retrieval, reuse, revision, and retention in case based reasoning. En: *The Knowledge Engineering Review*. 2005, Vol. 00:0, no. 1-2. DOI 10.1017/S000000000000000.

47. RODRÍGUEZ, Y. y GARCÍA, M. *Generalización de la métrica basada en la diferencia de valores (VDM) para variables lingüísticas y su aplicación en sistemas basados en el conocimiento*. Villa Clara: Universidad Central Martha Abreu, 2007.

48. FRANCIS, S. S. *Modelo para diseñar Mapas Conceptuales Inteligentes utilizando el Razonamiento*

*Basado en Casos*. La Habana: Universidad de las Ciencias Informáticas, 2010.

49. BANDYOPADHYAY, SANGHAMITRA, MAULIK, UJJWAL, HOLDER, LAWRENCE B. y COOK, DIANE J. *Advanced Methods for Knowledge Discovery from Complex Data*. S.l.: Springer, 2005. ISBN 1-85233-989-6.

50. AHA, DAVID W. The omnipresence of case-based reasoning in science and application. Knowledge-Based Systems. En: *Seventeenth SGES International Conference on Knowledge Based Systems and Applied Artificial Intelligence* [En línea]. S.l.: s.n., 1998. [Accedido: 13 de mayo de 2013]. Disponible desde: http://www.cs.auckland.ac.nz/~ian/CBR/omnipresence%20of%20cbr.pdf.

51. CÁRDENAS VARGAS, ELENA, OKTABA, D. HANNA, GUARDATI, SILVIA y LAUREANO, ANA LILIA. Agents, Case-Based Reasoning and their relation to the Mexican Software Process Model (MoProSoft). En: *31st Annual International Computer Software and Applications Conference*. S.l.: Computer society, 2007.

52. ZHU, ZHIQIANG, HSU, HUNG-YAO y NAGALINGAM, SEV. Towards Creative Case Based Reasoning. En: *2010 Fourth International Conference on Genetic and Evolutionary Computing*. S.l.: s.n., 2010.

53. WANG, JUN y REN, HUIQIN. Case-based Reasoning Enabling Database Mining for Cryo-Preserving Algae Applications. En: *2011 International Conference on Internet Computing and Information Services*. S.l.: s.n., 2011.

54. DÍAZ-AGUDO, BELÉN, GÓMEZ-MARTÍN, MARCO A., GÓMEZ-MARTÍN, PEDRO P. y GONZÁLEZ-CALDERO, PEDRO A. Formal Concept Analysis for Knowledge Refinement in Case Based Reasoning. En: *25th International Conference on Innovative Techniques and Applications ofArtificial Intelligence*. S.l.: s.n., 2005.

55. SPRAGUE, RALPH H. y CARLSON, ERIC D. *Building effective decision support systems*. Englewood Cliffs, New Jersey: Prentice Hall, 1982.

56. HAAG, S., CUMMINGS, M., MCCUBBREY, D. J., PINSONNEAULT, A. y DONOVAN, R. *Management*

*Information Systems: For the Information Age*. S.l.: McGraw Hill, 2000.

57. POWER, DANIEL J. *Decision Support Systems: Concepts and Resources for Managers*. Westport, Connecticut: Quorum Books, 2002. ISBN 156720497X.

58. SILVERMAN, BERNARD W. y JONES, M.C. (1951): An Important Contribution to Nonparametric Discriminant Analysis and Density Estimation: Commentary on Fix and Hodges (1951). En: *International Statistical Review / Revue Internationale de Statistique*. 1989, Vol. 57, no. 3, pp. 233–238. DOI 10.2307/1403796.

59. FREEMAN, ERIC, FREEMAN, ELISABETH, SIERRA, KATHY y BATES, BERT. *Head First Design Patterns*. S.l.: O'Reilly, 2004. ISBN 978-0-596-00712-6.

60. GAMMA, ERICH, HELM, RICHARD, JOHNSON, RALPH y VLISSIDES, JOHN M. *Design Patterns: Elements of Reusable Object-Oriented Software*. S.l.: Addison-Wesley Publishing Company, 1995. ISBN ISBN 0-201-63361-2.

61. Patrón Composite en PHP y Symfony. En: *symfony.cu* [En línea]. 10 de marzo de 2007. [Accedido: 8 de abril de 2013]. Disponible desde: http://www.symfony.es/noticias/2007/03/10/patron-composite-en-php-y-symfony/.

62. Inyección de dependencias en Symfony 2. En: *symfony.es* [En línea]. 2 de abril de 2009. [Accedido: 8 de abril de 2013]. Disponible desde: http://www.symfony.es/noticias/2009/04/02/inyeccion-de-dependencias-en-symfony-2/.

63. REENSKAUG, TRYGVE y COPLIEN, JAMES O. The DCI Architecture: A New Vision of Object-Oriented Programming. En: *Aritma developer* [En línea]. 20 de marzo de 2009. [Accedido: 23 de abril de 2013]. Disponible desde: http://www.artima.com/articles/dci_vision.html.

64. BURBECK, STEVE. Applications Programming in Smalltalk-80(TM): How to use Model-View-Controller (MVC). En: *UIUC Smalltalk Archive* [En línea]. 1997. [Accedido: 23 de abril de 2013]. Disponible desde: http://st-www.cs.illinois.edu/users/smarch/st-docs/mvc.html.

65. Yii Framework. En: *Best MVC Practices* [En línea]. [Accedido: 23 de abril de 2013]. Disponible desde: http://www.yiiframework.com/doc/guide/1.1/en/basics.best-practices.

66. FOWLER, MARTIN y SCOTT, KENDALL. *UML gota a gota*. S.l.: Pearson Education, 1999.

67. STEVENS, PERDITA y POOLEY, ROB. *Utilización de UML en Ingeniería del Software con Objetos y Componentes*. 2da edición. S.l.: Addison-Wesley Publishing Company, 2002.

68. BOOCH, GRADY. *The Unified Modeling Language User Guide*. S.l.: Pearson Education, 2005.

69. MILUTINOVICH, JOVANA. Introduction To OMG's Unified Modeling Language™. En: *OMG* [En línea]. Julio de 2005. Disponible desde: http://www.omg.org/gettingstarted/what_is_uml.htm.

70. Visual Paradigm for UML 8.0 Released. En: *Visual Paradigm* [En línea]. 16 de agosto de 2010. [Accedido: 23 de abril de 2013]. Disponible desde: http://www.visual-paradigm.com/aboutus/newsreleases/vpuml80.jsp.

71. Round-trip Code Engineering. En: *Visual Paradigm* [En línea]. [Accedido: 23 de abril de 2013]. Disponible desde: http://www.visual-paradigm.com/product/vpuml/features/roundtripcodeengine.jsp#cpproundtrip.

72. Overview. En: *Net Beans IDE* [En línea]. [Accedido: 23 de abril de 2013]. Disponible desde: https://netbeans.org/features/index.html.

73. GEERTJAN WIELENGA. Top 10 New Features in NetBeans IDE 7.2. En: *Geertjan's Blog* [En línea]. 24 de julio de 2012. Disponible desde: https://blogs.oracle.com/geertjan/entry/top_10_new_features_in.

74. MARCHESI, MICHELE, SUCCI, GIANCARLO, WELLS, DON y WILLIAMS, LAURIE. *Extreme Programming Perspectives*. S.l.: Addison-Wesley Publishing Company, 2002.

75. BECK, KENT. *Extreme Programming Explained. Embrace Change*. S.l.: Addison-Wesley Publishing Company, 1999. ISBN 978-0321278654.

76. JEFFRIES, R., ANDERSON, A. y HENDRICKSON, C. *Extreme Programming Installed*. S.l.: Addison-Wesley Publishing Company, 2001.

77. User Experience (Chromium Developer Documentation). En: *Chromium Developer Documentation* [En línea]. 2009. [Accedido: 8 de mayo de 2013]. Disponible desde: http://dev.chromium.org/user-experience.

78. GOODGER, BEN. Welcome to Chromium. En: *The Chromium Blog* [En línea]. 2 de septiembre de 2008. [Accedido: 8 de mayo de 2013]. Disponible desde: http://blog.chromium.org/2008/09/welcome-to-chromium_02.html.

79. FETTE, IAN. Google Chrome, Chromium, and Google. En: *The Chromium Blog* [En línea]. 1 de agosto de 2008. [Accedido: 8 de mayo de 2013]. Disponible desde: http://blog.chromium.org/2008/10/google-chrome-chromium-and-google.html.

80. Differences between Google Chrome and Linux distro Chromium. En: *Google Code* [En línea]. 20 de noviembre de 2012. Disponible desde: https://code.google.com/p/chromium/wiki/ChromiumBrowserVsGoogleChrome.

81. The Chromium Projects. En: *Chromium* [En línea]. [Accedido: 23 de abril de 2013]. Disponible desde: http://www.chromium.org/Home.

82. HOEHRMANN, B. Scripting Media Types. En: *The Internet Society* [En línea]. Abril de 2006, Disponible desde: http://www.apps.ietf.org/rfc/rfc4329.html.

83. JavaScript Introduction. En: *w3schools.com* [En línea]. [Accedido: 8 de mayo de 2013]. Disponible desde: http://www.w3schools.com/js/js_intro.asp.

84. Programming languages on the internet. En: *Web Developers Notes* [En línea]. [Accedido: 8 de mayo de 2013]. Disponible desde: http://www.webdevelopersnotes.com/basics/languages_on_the_internet.php3.

85. Sencha Ext JS JavaScript Framework for Rich Desktop Apps. En: *Sencha* [En línea]. [Accedido: 4 de mayo de 2013]. Disponible desde: http://www.sencha.com/products/extjs/.

86. SPENCER, ED. Countdown to Ext JS 4: Dynamic Loading and New Class System. En: *Sencha* [En línea]. 19 de enero de 2011. [Accedido: 5 de mayo de 2013]. Disponible desde: http://www.sencha.com/blog/countdown-to-ext-js-4-dynamic-loading-and-new-class-system.

87. MAINTZ, TOMMY. Architecting Your App in Ext JS 4, Part 1. En: *Sencha* [En línea]. 21 de julio de 2011. [Accedido: 8 de abril de 2013]. Disponible desde: http://www.sencha.com/learn/architecting-your-app-in-ext-js-4-part-1.

88. Licenses. En: *The Apache Software Foundation* [En línea]. [Accedido: 7 de mayo de 2013]. Disponible desde: http://www.apache.org/licenses/.

89. What is the Apache HTTP Server Project? En: *The Apache Software Foundation* [En línea]. [Accedido: 8 de mayo de 2013]. Disponible desde: http://httpd.apache.org/ABOUT_APACHE.html.

90. Configuration Files. En: *The Apache Software Foundation* [En línea]. [Accedido: 8 de mayo de 2013]. Disponible desde: http://httpd.apache.org/docs/current/configuring.html.

91. What is PHP? En: *PHP* [En línea]. [Accedido: 6 de mayo de 2013]. Disponible desde: http://php.net/.

92. Varias licencias y comentarios acerca de las mismas. En: *Proyecto GNU* [En línea]. [Accedido: 9 de mayo de 2013]. Disponible desde: http://www.gnu.org/licenses/license-list.html#GPLIncompatibleLicenses.

93. About - Symfony. En: *Symfony* [En línea]. [Accedido: 26 de marzo de 2013]. Disponible desde: http://symfony.com/about.

94. Object Relational Mapper. En: *Doctrine* [En línea]. [Accedido: 9 de mayo de 2013]. Disponible desde: http://www.doctrine-project.org/projects/orm.html.

95. Doctrine Query Language. En: *Doctrine* [En línea]. [Accedido: 9 de mayo de 2013]. Disponible desde: http://docs.doctrine-project.org/en/2.1/reference/dql-doctrine-query-language.html.

96. PostgreSQL: About. En: *PostgreSQL* [En línea]. [Accedido: 25 de marzo de 2013]. Disponible desde:

http://www.postgresql.org/about/.

97. LORENZ, MARK y KIDD, JEFF. *Object-Oriented Software Metrics*. Englewood Cliffs, Nueva Jersey: s.n., 1994. Prentice Hall Object-Oriented.