Universidad de las Ciencias Informáticas

Facultad 15

Título: Análisis y diseño de un sistema para la realización no presencial del Análisis de Secuencia

Trabajo de Diploma para optar por el título de

Ingeniero en Ciencias Informáticas

Autor: *Norisbel González Sánchez*

Tutores: *Ing. Geykel Raúl Moreno Ceballos*

Lic. Asdrúbal Ramírez Hernández

Ciudad de la Habana, Junio 2010

DECLARACIÓN DE AUTORÍA

“Es mejor adaptar la tecnología al usuario,

que forzar al usuario a adaptarse a la tecnología”

Larry Marine

Agradecimientos

A mi mamita Milagros, a mi segundo padre Javier, Daya mi hermana la que siempre quise tener les agradezco por todo su apoyo, su amor, su paciencia y el calor familiar que me han dado los quiero muchoooo.

A mi tutor Geykel que con mucha paciencia me ha guiado y encaminado en la confección de este trabajo, siendo comprensivo y muy buen amigo, mil gracias.

A alguien bien especial en mi vida mi Chenchitino.

A mis amistades y todas las personas que de una forma u otra tienen parte en este trabajo a Addy y Ernesto (Machete), a Aida y Leandro, a Eliober que lo moleste cantidad a Dayron un amigo, a Carlos Felipe me ayudó con sus críticas constructivas, a Vlamir, a Daymí, a Diana, en especial a Asdrubal quien me mostró el camino en el mundo de la AI. Muchísimas gracias a todos sin su ayuda no se habría podido hacer este trabajo.

Dedicatoria

A mis padres y mi hermano que los adoro, por ser mi patrón a seguir en la vida, mi razón de ser, mi orgullo lo más grande que tengo en este mundo.

A JCS y MVC por estar conmigo siempre, ayudarme, protegerme y no abandonarme nunca.

A mis familiares en especial a mi abuelitos Norma y Gerardo que los quiero muchísimo y no soporto más tiempo lejos de ellos. Y a mí otros padres Mily, Javi y mis hermanitos mayores Yasser y Dayi (Doryta) los quiero un montón.

A mis amistades que han estado siempre ahí incondicionalmente Manu, Ali, Jovamna, Addy, Machete, Geykel.

A todos los que realmente me quieren y me tienen en un lugar especial de su corazón.

Resumen

Como parte del desarrollo del proceso de Arquitectura de Información (AI) de un proyecto de software, se realiza el Análisis de Secuencia, el cual tiene como objetivo proveer un orden a las etiquetas seleccionadas para formar parte del producto. Por lo general este proceso involucra al arquitecto de información y a los clientes finales y se realiza de forma presencial. Este proceso puede tornarse agotador si la muestra de usuario potenciales seleccionados por el arquitecto debe ser de gran variedad en cuanto a parámetros como por ejemplo (edad, sexo, profesión, entre otros).

El presente trabajo describe un procedimiento para la realización de forma no presencial y con la ayuda de las nuevas tecnologías de información del proceso de Análisis de Secuencia que se lleva a cabo como parte de desarrollo de la Arquitectura de Información de un producto. Luego se realizan el análisis y diseño de software con el objetivo de obtener los artefactos necesarios para guiar la etapa de implementación del sistema.

Índice

Índice de Figuras

Índice de Tablas

Introducción

La Arquitectura de la Información se origina en la década de 1970 con Richard Saul Wurman aunque hubo que esperar a su libro Information Architects para encontrar una formulación más detallada.

En la misma, Wurman define la arquitectura de la información como "una combinación de la organización del contenido del sitio en categorías y la creación de una interfaz para sostener esas categorías" o también como " el proceso de organizar los contenidos y presentarlos en el mejor formato para una audiencia particular, se ocupa del mejoramiento y la claridad de los Sitios Web" y Kahn simplemente expresa que en su sentido más básico, es sólo "la construcción de la organización de la información". Como pueden ver la Arquitectura de la Información debe su salud y tiene sus armas en el tema de los contenidos que pueden ser organizados y administrados de mejor manera, sobre todo cuando se usan nuevas tecnologías, para que los usuarios puedan encontrar lo que buscan en una cantidad desordenada de información. (Tramullas Saz, 2002)

La organización y representación de términos en el etiquetado de productos electrónicos, díganse principalmente sitios web o multimedia interactivas en CD-ROM o DVD, es una actividad inherente al proceso de arquitectura de información.

Esta actividad conlleva análisis de información a partir de las necesidades y requerimientos de los usuarios potenciales del producto. Entre uno de estos requerimientos se encuentra la organización secuencial, por ejemplo, el orden a dar a las etiquetas; de forma tal que represente de la mejor manera posible las estructuras mentales de los usuarios finales. Es por ello que la organización secuencial de los términos en una barra de navegación o un menú desplegable, reviste gran importancia para el posterior desempeño satisfactorio del producto.

En contadas ocasiones se enfrenta el dilema de cuál término va primero y cuál va segundo… así consecutivamente hasta el último. Es cierto que las formas de organización secuencial más conocidas son: la numérica (1, 2, 3, 4…), la alfabética (a, b, c…) o la cronológica (1954, 1955, 1956…), pero ¿Qué pasa cuando los términos no son susceptibles a estos tipos de organización? ¿Se ubican de acuerdo con el criterio personal del arquitecto de información? Esta podría ser una solución, pero no habría seguridad de que los usuarios del sitio web o la multimedia interactiva se identifiquen con la selección realizada.

Otra posible solución es la de organizar secuencialmente los términos siguiendo una lógica funcional, temporal o estructural. Por ejemplo:

- ✓ una organización funcional sería: agua, azúcar, limón, limonada (siguiendo un principio lógico de funcionamiento)
- ✓ una organización temporal sería: bisabuelos, abuelos, padres, hijos (siguiendo un principio temporal de secuencia)
- ✓ una organización estructural sería: gerente, subgerente, jefe de grupo, guardia de seguridad (siguiendo un principio estructural de una empresa)

Puede haber muchas más formas de organizar secuencialmente siguiendo distintos principios basados en la lógica, pero en ocasiones estos principios no son suficientes.

Con el objetivo de obtener una información lo más cercana posible a la necesidad del usuario final, y definir la secuencia de acorde con esto, se propone complementar los métodos que existen, llegando así al un Análisis de Secuencia.

El Análisis de Secuencia consiste en la realización de una serie de pruebas a usuarios potenciales del producto, y el posterior análisis cualitativo y cuantitativo de esos resultados; para ayudar a definir la secuencia de las etiquetas en el producto electrónico. (Ronda León, y otros, 2005)

Este tipo de análisis se realiza hoy en día de forma presencial, es decir, con la participación directa de los usuarios finales del producto de información que se desea desarrollar, lo cual no tiene inconvenientes para soluciones desarrolladas a la medida para una determinada organización, pero, ¿qué sucedería si el producto que se desea desarrollar es dirigido a un cliente genérico, como lo son, por citar algunos, los paquetes ofimáticos, los antivirus y los sistemas operativos? En estos casos reunir en un lugar específico a estos clientes, logrando una gran diversidad entre ellos (niños, jóvenes, adultos, doctores, ingenieros, constructores) es algo que se torna realmente complejo.

La **problemática** expuesta da origen a la siguiente interrogante:

¿Cómo realizar el análisis de secuencia a un producto de información de forma no presencial con la ayuda de las tecnologías de información?

Definiéndose así, la Arquitectura de la Información como el **objeto de investigación**, la cual enmarca dentro de sí como una técnica de gran importancia en el estudio de los usuarios finales al Análisis de Secuencia, siendo el **campo de acción**.

Para dar solución a la problemática planteada, se trazó el siguiente **objetivo**:

Realizar el análisis y diseño de una aplicación web que permita la realización de forma no presencial del análisis de secuencia.

De aquí, se derivan los siguientes **objetivos específicos**:

- ✓ Definir los requisitos funcionales y no funcionales.
- ✓ Seleccionar la metodología adecuada para el proceso de desarrollo (análisis y diseño).
- ✓ Representar los procesos del negocio.
- ✓ Obtener los artefactos relacionados a los modelos de análisis y diseño del sistema concerniente al proceso de realización del Análisis de Secuencia y procesamiento de resultados cuantitativos.

Se plantea como **hipótesis**: con la elaboración del análisis y diseño de una aplicación web que permita realizar el análisis de secuencia a un producto de información de forma no presencial se podrá implementar un sistema que ayude al arquitecto de información a obtener resultados cuantitativos del análisis de secuencia con un menor esfuerzo así, como realizar este proceso de forma no presencial.

El presente documento se estructura en tres capítulos y varios anexos, que incluye lo relacionado con el trabajo investigativo realizado, así como los Casos de Uso y diagramas generados.

El ***Capítulo1***: Fundamentación Teórica, comienza con una breve introducción a la Arquitectura de Información, profundiza en el Análisis de Secuencia. El capítulo continúa describiendo las tendencias y tecnologías actuales para realizar el análisis y modelado de software, díganse metodologías, herramientas y lenguaje de modelado.

En el ***Capítulo 2***: Características del Sistema, se establecerán los requisitos funcionales y no funcionales para crear un sistema automatizado que cumpla con las expectativas de los usuarios finales y así comenzar con el modelado del mismo. Se hace, además, una descripción de los casos de uso y se muestra el diagrama de casos de uso del sistema.

En *el* ***Capítulo 3***: Análisis y Diseño, se realiza el modelo del análisis y de diseño del sistema. Se obtiene, a través de los casos de uso del sistema, el diagrama de análisis. Como resultado del análisis se obtiene el diseño de la aplicación, el cual queda expresado en el diagrama de clases del diseño, los diagramas de secuencia y la descripción de las clases. Referente a la validación, se habla acerca de la validación de requisitos, métricas de diseño y patrones de diseño aplicados.

Capítulo 1: Fundamentación teórica

Con el fin de alcanzar el objetivo trazado con la realización de este trabajo, es preciso efectuar una serie de estudios preliminares que permitan obtener los mejores resultados posibles. Por tanto, en el presente capítulo se realiza una breve introducción a la Arquitectura de Información así como algunos de sus principales procesos para profundizar en el Análisis de Secuencia, siendo el campo de acción de esta investigación. Además se hace un estudio de las tecnologías y/o plataformas para el desarrollo de aplicaciones web. Otro aspecto importante tratado en este capítulo son las metodologías de desarrollo de software, donde se realiza un estudio de las principales tendencias tanto en la industria a nivel internacional como en nuestra universidad centrándose principalmente en las fases de análisis y diseño de aplicaciones web.

1.1. Aspectos sobre la Arquitectura de Información

La Arquitectura de la Información es un campo de estudio que surge a partir de la necesidad de dar solución a problemas derivados de la organización y estructuración de grandes volúmenes de información.

El surgimiento de la Web y la popularización del hipertexto, la complejidad de los nuevos sistemas de información, así como la gran diversidad de usuarios y contextos de uso, han originado la necesidad de hacer frente a nuevos retos de diseño a los que ninguna disciplina actual puede dar solución por sí sola (Rosenfeld, 2002). Es por ello por lo que ha sido en la era Web cuando la Arquitectura de la Información ha experimentado su mayor auge.

1.1.1. ¿Qué es la Arquitectura de Información?

A lo largo de los años varios autores han dado su propia definición de Arquitectura de Información. Según (Dickson, y otros, 1985) AI es “un gran mapa de los requerimientos de información de una organización. Es una representación independiente, de las principales categorías de información, del personal, la organización y la tecnología dentro de una empresa”. Por su parte (Brancheau, y otros, 1986) amplían este concepto con: “La representación muestra cómo las categorías de información están relacionadas a los procesos de negocio y cómo deben estar conectadas para facilitar la toma de decisiones”.

“Una arquitectura de información es un diseño o plano para modelar los requerimientos globales de información de una empresa. Proveyendo una forma de representar las necesidades de información de

una organización, relacionando a ellas los procesos específicos de negocio y documentando sus interrelaciones. Este mapa de procesos de información es usado entonces para guiar el desarrollo de aplicaciones y para facilitar integrar y compartir datos".

(Wurman, 1996) Planteó AI se define como "una combinación de la organización de la información del contenido del sitio en categorías y la creación de una interfaz para sostener esas categorías".

(Rosenfeld, y otros, 1998) Definen a la AI como:

La combinación de la organización, etiquetado y los esquemas de navegación dentro de un sistema de información.

El diseño estructural de un espacio de información para facilitar las tareas de acabado y acceso intuitivo a los contenidos.

El arte y ciencia de estructurar y clasificar sitios web e intranets para ayudar a los usuarios a encontrar y administrar su información.

Una disciplina emergente y una comunidad práctica enfocada en traer los principios de diseño y arquitectura a los entornos digitales.

El Instituto Asilomar para la Arquitectura de Información AIFLA (Asilomar Institute for Information Architecture) la define como:

"El diseño estructural de ambientes de información compartidos. Es el arte y la ciencia de organizar y etiquetar sitios web, intranets, comunidades en línea y programas computacionales, para apoyar las capacidades de uso y búsqueda".

Independientemente de que existen diferentes definiciones sobre la Arquitectura de Información, todos tienen un punto común, que es organizar la información en un producto informático y garantizar que el resultado final cumpla los requerimientos de sus usuarios potenciales. Logrando una mayor calidad del producto de información (mejor búsqueda y recuperación, usabilidad, experiencia)

Existen otras definiciones de Arquitectura de Información aunque se han seleccionado las más aceptadas o usadas entre los profesionales de este sector.

La arquitectura de información de un sitio web, entendida como el resultado de la actividad de clasificar, describir, estructurar y etiquetar los contenidos del sitio si bien no es percibida directamente por el usuario, como indica (Gullikson, y otros, 1999), tiene un claro impacto en la usabilidad del sitio web, es decir, en la eficacia, eficiencia y satisfacción de uso. Usabilidad y Arquitectura de la Información no son lo mismo (Lash, 2002) , aunque para los fines de este trabajo serán englobadas en una visión "amplia" de la Arquitectura de la Información.

Esta "amplia" Arquitectura de la Información, por su carácter multidisciplinario, se nutre de técnicas, metodologías y teorías de una gran variedad de áreas de conocimiento (diseño gráfico, psicología cognitiva, ciencias de la documentación y otras), así como de prácticas profesionales y estudios de casos reales.

1.1.2. Diseño Centrado en el Usuario

Entre las aportaciones metodológicas que ha recibido la Arquitectura de la Información por parte de disciplinas como la Interacción Persona-Ordenador o HCI (Human-Computer Interaction), cabe destacar el Diseño Centrado en el Usuario (DCU) (Norman, y otros, 1986) .Se trata de un marco metodológico en el que se asume que las características, necesidades y objetivos del usuario deben ser las que conduzcan todo el proceso de diseño. El DCU se basa en un continuo e iterativo proceso de "Diseño-Prototipado-Evaluación", lo que permite, desde tempranas etapas del desarrollo, evaluar lo diseñado y por tanto validar la arquitectura del producto. El DCU, además de proporcionar métodos para la evaluación y el prototipado de lo diseñado, también ofrece un conjunto de técnicas para la toma de decisiones por parte del arquitecto de información sobre qué diseño y organización de la información resultaría más acorde con las necesidades y características del usuario. Entre estas técnicas se encuentra la de "Card Sorting". Esta se basa en la observación de cómo los usuarios agrupan y asocian entre sí un número predeterminado de tarjetas etiquetadas con las diferentes categorías temáticas del sitio web. De esta forma, partiendo del comportamiento de los propios usuarios, es posible organizar y clasificar la información de un espacio web conforme a su modelo mental. Esta técnica, en DCU, ha sido utilizada con éxito durante años con otros fines, tales como en psicología clínica (Obonsawin, y otros, 1999), o en la adquisición de conocimiento en Sistemas Expertos (Wagner, y otros, 2002).

Existen varias técnicas las cuales se clasifican en búsqueda de información, funcionamiento, diseño, organización. Dentro de esta última clasificación se encuentra:

- ✓ Organización de tarjetas (Card Sorting)
  - Agrupación
  - Posición de bloques
  - Análisis de Secuencia
- ✓ Tabulación de contenidos.
- ✓ Diagramas de afinidad.
- ✓ Validación de términos.

Organización de tarjetas (Card Sorting) es una técnica de interacción con el usuario, en la que se le entrega un grupo de tarjetas con palabras escritas y se les pide que las organicen. Se observa el desempeño de los usuarios y se obtienen los resultados cualitativos y después se entran los datos en un software y se obtienen los resultados cuantitativos.

Análisis cualitativo

1. ¿Cuánto tiempo se demora el usuario en comenzar a organizar las tarjetas?
2. ¿Cuál tarjeta es la primera en escoger y cuál la última?
3. ¿Cuál(es) término(s) no comprende y pide aclarar?

El **análisis cuantitativo**, en cambio, se basa en la aplicación de técnicas estadístico-automáticas sobre los grupos resultantes de la agrupación realizada por los usuarios. Aunque el análisis cualitativo puede ofrecer información adicional al cuantitativo, el número de participantes en la prueba debe ser necesariamente bastante reducido, con el fin de poder realizar un seguimiento de cada uno de los participantes y su modo de actuar. Esto implica una pérdida en la representatividad de los resultados y por lo tanto en su validez. En el análisis cuantitativo, ya que el número de participantes será bastante más elevado, se hace imprescindible el uso de técnicas estadísticas que faciliten la comprensión e interpretación de los resultados, normalmente a través de representaciones gráficas que permitan visualizar las relaciones de agrupación y distancia entre categorías. (Hassan Montero, et al., 2004)

Existen tres técnicas dentro de la Arquitectura de Información que utilizan el Card Sorting, una de ellas es el Análisis de Secuencia, la cual se profundiza a continuación ya que es la estudiada en el presente trabajo.

1.1.3. ¿Qué es el Análisis de Secuencia?

El Análisis de Secuencia consiste en la realización de una serie de pruebas a usuarios potenciales del producto, y el posterior análisis cualitativo y cuantitativo de esos resultados; para ayudar a definir la secuencia de las etiquetas en el producto electrónico. (Ronda León, y otros, 2005)

Para hacer una Arquitectura de Información con calidad se debe hacer una buena aplicación de las técnicas dentro de los procesos y en las etapas requeridas. La aplicación de los procesos y las técnicas se adaptan a cada proyecto específico. Según sea el producto, así serán los procesos y las técnicas aplicadas al mismo.

Para la organización de secuencia u ordenación secuencial de objetos homogéneos o pertenecientes al mismo rango la misma se basa en criterios de organización secuencial:

- ✓ Numérica (1, 2, 3, 4,5)
- ✓ Alfabética (a, b, c, d, e, f…)
- ✓ Cronológica (1998, 1999, 2000, 2001…)
- ✓ Forma (tamaño, peso, tono de color)
- ✓ Funcional

Un ejemplo de organización secuencial funcional sería: agua, azúcar, limón - limonada (siguiendo un principio lógico de funcionamiento)

Una organización secuencial cronológica sería: bisabuelos, abuelos, padres, hijos (siguiendo un principio temporal de secuencia)

1.2. Plataformas y frameworks existentes para el desarrollo de aplicaciones web

Existen varios enfoques a la hora de decidirse por una determinada plataforma y luego framework de trabajo o viceversa para el desarrollo de un sistema. Para el caso del sistema que se describe en el presente trabajo, se realiza la selección del framework y con el mismo la plataforma basándose en la arquitectura del sistema. Aunque la arquitectura del sistema está fuera del alcance del trabajo es necesario para un correcto diseño del producto tener en cuenta varios aspectos de la misma ya que, a fin de cuentas, son procesos que están muy relacionados.

1.2.1. Modelo-Vista-Controlador

Para la arquitectura base de la aplicación se seleccionó el patrón Modelo-Vista-Controlador (MVC), el cual separa el modelado del modelo del dominio, la presentación, y las acciones basadas en entradas del usuario en tres clases separadas (Burbeck, 1992).

- ✓ **Modelo:** El modelo maneja el comportamiento y datos del dominio de la aplicación, responde a los pedidos de información sobre su estado (generalmente desde la vista), y responde a instrucciones para cambiar el estado (usualmente desde el controlador).
- ✓ **Vista:** La vista se encarga del visualizado de información.
- ✓ **Controlador:** Interpreta las entrada del mouse y teclado realizadas por el usuario, informando al modelo y/o a la vista que debe cambiar según sea apropiado.

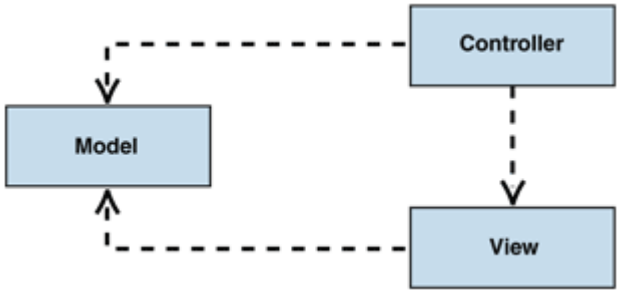

Figura 1: Estructura de clases del MVC.

Es importante destacar que tanto la vista como el controlador dependen del modelo. Sin embargo, el modelo no depende ni de la vista ni del controlador. Este es el principal beneficio de la separación, el cual permita que el modelo sea construido y probado independientemente de la vista. MVC es un patrón, desde el punto de vista de diseño, fundamental para la separación de la lógica de presentación de la lógica de negocio.

Beneficios:

- ✓ **Múltiples vistas:** Debido a que la vista está separada del modelo y no existe una dependencia directa del modelo a la vista, la interfaz de usuario puede mostrar múltiples vistas del mismo dato a la misma vez. Por ejemplo, varias páginas en una aplicación web pueden utilizar el mismo modelo.

Otro ejemplo es una aplicación web que permita al usuario cambiar la apariencia de las páginas, estas páginas muestran la misma información del modelo compartido pero lo hacen de formas diferentes.

- ✓ **Adecuar cambios:** Los requerimientos de interfaz de usuario tienden a cambiar con mayor rapidez que los de las reglas del negocio. Los usuarios pueden preferir nuevos colores, tipos de letras, soporte para nuevos dispositivos. Debido a que el modelo no depende de la vista, adicionar nuevos tipos de vistas al sistema generalmente no afecta al modelo. Como resultado el ámbito de cambio se confina solo a la vista.

1.2.2. MVC y la Web

Debido a la selección de MVC como arquitectura base para el sistema y teniendo en cuenta que el mismo es una aplicación web, es necesario seleccionar un framework que implemente este patrón con el objetivo de ganar en productividad y a su vez realizar el modelo de diseño en base al funcionamiento del framework escogido. A continuación se mencionan algunos de los frameworks que implementan dicho patrón orientados a la web:

Ruby on Rails

Rails, como también es conocido, es uno de los frameworks MVC más populares creado para la web. Funciona de acuerdo a dos grandes principios:

- ✓ **Convención sobre Configuración:** La idea es: *"como desarrolladores hemos estado desarrollando aplicaciones web desde hace años, vamos a ponernos de acuerdo en algo que funciona y hagámoslo un framework"*.
- ✓ **No se repita usted mismo (DRY[1]):** Esto se aplica a la idea de centralizar tanto la lógica de la aplicación como el código. Con esta idea en mente puede escribirse menos código.

Una de las mayores fortalezas de Rails es su lenguaje Ruby, el cual es dinámico, no es fuertemente tipado y es compilado "on the fly" por el interpretador.

Django y Python

[1] Don't Repeat Yourself

Django es un framework para la web que utiliza a Python como lenguaje de programación y valora el diseño limpio. Se enfoca principalmente en automatizar tanto como sea posible y hace uso del principio DRY.

Incluye un ORM [2]que permite describir la base de datos utilizando código Python. Al igual que Rails, permite general una interfaz administrativa. Django es capaz de integrarse con un sistema de cacheo muy poderoso llamado "memcached" el cual también es muy frecuentemente utilizado en aplicaciones Rails.

Spring, Structs y Java

En el espacio de Java, existen tres grandes frameworks MVC: Apache Structs, Spring Framework, y JSF, que es el J2EE framework MVC estándar. Han existido otros pero estos tres son los más utilizados incluso JSF está en una distante tercera posición.

- ✓ Spring es un framework Java/J2EE que valora el diseño orientado a objetos, interfaces sobre clases, y pruebas sobre cualquier otra consideración. Es más que solo un framework MVC, es un framework de objetos de negocio. Es considerado más ligero que Structs.
- ✓ Structs, es considerado un framework estándar, cada vez más es acoplado y utilizado con Spring.
- ✓ JSF es un framework de desarrollo basado en Java, provee un conjunto de estándares y componentes de interfaz reusables para la web. Incluye validación del lado del servidor, conversión de datos y manejo de navegación entre páginas.

A pesar de sus diferencias, todos estos frameworks de Java comparten los mismos conceptos, Modelo, Vista y Controlador. Difieren un poco en el ciclo de vida y las cosas que valoran pero esencialmente el objetivo es el mismo, clara separación de intereses o más conocido por el inglés *(clean separation of concerns)*.

ASP.NET MVC

ASP.NET MVC se basa en muchas de las mismas estrategias que las demás plataformas MVC utilizan, y ofrece además los beneficios de código compilado y manejado, y explota las nuevas características de

[2] Object-Relational Mapper

VB9, C#3 y C#4, como expresiones lambdas, linq, y tipos y métodos anónimos. Ofrece control absoluto sobre el HTML que se genera, no deniega la existencia de HTTP y fue diseñado desde el principio teniendo en cuenta una alta capacidad de prueba. ASP.NET MVC valora la extensibilidad y la flexibilidad, es posible sustituir básicamente cualquier parte del mismo con otros frameworks.

ASP.NET MVC fue el framework escogido para el desarrollo del sistema, debido a sus características ya explicada pero esencialmente por su gran productividad y flexibilidad.

Es importante destacar, para un mejor entendimiento del modelo de diseño, que estos frameworks *"sirven métodos y no ficheros"*, o sea, en sus inicios los servidores web servían HTML guardado en ficheros estáticos en el disco. A medida en que las páginas web dinámicas fueron ganando prominencia, los servidores web servían HTML dinámico a partir de scripts dinámicos que también estaban guardados en disco. Con MVC es un poco diferente, la URL le dice al mecanismo de ruteo cual controlador debe instanciar y que método acción ejecutar y provee los parámetros a dicho método. Estos métodos deciden que vista usar y entonces esta vista genera el HTML a visualizar. En lugar de tener una relación directa entre la URL y un fichero en el disco del servidor web, existe una relación entre la URL y un método en un objeto controlador. ASP.NET MVC implementa la variante *"front controller"* del patrón MVC, donde el controlador esta por delante de todo excepto del sistema de ruteo.

1.3. Tendencias actuales en la realización del análisis y diseño de software

En la actualidad, a raíz del desarrollo del intelecto humano, las necesidades de modelado y avance tecnológico, existen diferentes denominaciones, díganse: tendencias, metodologías, lenguaje de modelado, patrones; las cuales se encuentran contempladas en la Ingeniería de Software y permiten al Analista de Sistemas como especialista, tratar y modelar los procesos a los cuales se enfrenta cuando está inmerso en el análisis para el desarrollo de productos de Software.

Los productos de software tienen diferentes modalidades y características:

Productos de Software (Sommerville, 2005):

- ✓ Productos genéricos: Son producidos por una organización para ser vendidos al mercado.
- ✓ Productos hechos a medida: Sistemas que son desarrollados bajo pedido, a un desarrollador específico.

Características de los productos de software (Sommerville, 2005):

- ✓ Mantenibles: Debe ser posible que el software evolucione y que siga cumpliendo con sus especificaciones.
- ✓ Confiabilidad: El software no debe causar a la información que maneja daños físicos o económicos en el caso de fallos.
- ✓ Eficiencia: El software no debe desperdiciar los recursos del sistema.
- ✓ Utilización adecuada: El software debe contar con una interfaz de usuario adecuada y su documentación.

En el desarrollo de soluciones informáticas, la Ingeniería del Software comprende pasos generales por los cuales se debe transitar si se desea obtener un producto de calidad, estos se listan a continuación (Pressman, 2001):

Identificación del Entorno

La primera actividad es comprender el entorno para el cual se desarrollará la Solución Informática, conocer el negocio, modelarlo para su mejor comprensión, identificar los principales procesos, el orden secuencial de estos, son actividades bien importantes en esta primera etapa de construcción del software.

Captura de requisitos

Extraer los requisitos de un producto de software es la segunda etapa para crearlo y es una de las más importantes pues a partir de esta es donde se identifican los requerimientos tanto funcionales como no funcionales que debe tener el producto que será creado. Para la Captura de Requisitos se aplican diferentes técnicas de recopilación y análisis de información, por ejemplo, entrevistas con los clientes, cuestionarios. Se requiere de habilidad y experiencia en la Ingeniería de Software para reconocer requisitos incompletos, ambiguos o contradictorios.

Análisis y Especificación

Se analizan los requisitos que se describieron en la Captura de Requisitos, refinándolos y estructurándolos. El objetivo de esta fase es conseguir una comprensión más precisa de los requisitos y una descripción de los mismos que sea fácil de mantener y que ayude a estructurar el sistema a

desarrollar. Se describe utilizando el lenguaje de los desarrolladores, introduciendo un mayor formalismo y permite razonar sobre los funcionamientos internos del sistema. Es una primera aproximación al diseño y por tanto una entrada fundamental cuando se da forma al sistema en la fase de diseño e implementación.

Diseño y Arquitectura

Se modela el sistema, alcanzando de este modo su forma (incluida la arquitectura) para que soporte todos los requisitos, incluyendo los requisitos no funcionales y restricciones relacionadas con los lenguajes de programación, componentes reutilizables, sistemas operativos, tecnologías de distribución y concurrencia. Una entrada fundamental en el diseño es el análisis (modelo de análisis) el cual proporciona una comprensión detallada de los requisitos. En esta fase se descompone los trabajos de implementación en partes más manejables (subsistemas y sus interfaces) que puedan ser llevadas a cabo por diferentes equipos de desarrollo lo cual contribuye a una arquitectura estable y sólida.

Implementación

En esta fase se implementan (traduce a lenguaje de programación) las especificaciones de los requisitos, los modelos de análisis y diseño, obteniéndose ejecutables que cumplen con las funcionalidades indicadas.

Prueba

En esta fase se diseñan Casos de Uso de Prueba acorde a los requerimientos capturados y las especificaciones detalladas, con el fin de comprobar que el software realice correctamente las acciones indicadas en la especificación. Las pruebas se realizan a los requerimientos funcionales y no funcionales especificados, primeramente en los módulos del sistema y luego al sistema integrado.

Mantenimiento

En esta fase se le da soporte, o sea mantenimiento y realizan mejoras al software con el fin de enfrentar errores descubiertos y/o nuevos requisitos. Esto puede llevar más tiempo incluso que el desarrollo inicial del software. Alrededor de 2/3 de toda la Ingeniería de Software tiene que ver con dar mantenimiento. Una pequeña parte de este trabajo consiste en arreglar errores. La mayor parte consiste en extender el sistema para hacer nuevas actividades.

Documentación

La documentación de un software tiene gran importancia, pues en esta es donde se describe en lenguaje natural las funcionalidades del software a través de los diferentes documentos que se van generando por cada una de las fases por las que se transita. Los documentos quedan como constancia y sirven luego para una mejor comprensión e identificación de las tareas acometidas y las funcionalidades de cada una de las actividades que se implementan. Sirven posteriormente, para la etapa de soporte, mantenimiento y mejora del producto, y no solo en estas etapas, sino también durante la etapa de uso del software o sea en la etapa de trabajo con la herramienta informática. Se pueden citar ejemplos de documentos como el manual de usuario, y el manual técnico, los cuales tienen el propósito de detallar en un lenguaje asequible a todo tipo de usuario del sistema el modo de operar del mismo.

Para guiar el desarrollo de software, han surgido de acuerdo a las necesidades afrontadas, diferentes metodologías que definen estrategias de desarrollo de soluciones informáticas, promoviendo estas, prácticas adoptativas centradas en las personas o los equipos, dirigidas hacia la funcionalidad y la entrega, de comunicación intensiva entre los clientes y el equipo de desarrollo de la solución. (Arboleda Jiménez, 2005)

1.3.1. Metodologías de desarrollo de software

La disciplina de Ingeniería de Software ha evolucionado en gran medida, trayendo consigo propuestas diferentes para mejorar los resultados del proceso de construcción de soluciones informáticas, teniéndose como herramienta la realización de diagramas de modelado y de prototipos, ambos tienen el objetivo de mostrar, graficar las actividades a implementar según los requerimientos capturados, así como especificar el camino a seguir y exponer el resultado que se obtendrá de la implementación de cada uno de estos requerimientos del cliente una vez terminado el desarrollo, brindando además la posibilidad de obtener retroalimentación de manera temprana.

Para el desarrollo de soluciones informáticas existen diferentes metodologías, pero siempre escoger una u otra se torna difícil.

Se identifican en la actualidad la existencia de dos clasificaciones de metodologías en ciclo de vida de desarrollo de software, las cuales son:

- ✓ Las metodologías tradicionales, haciendo énfasis en la planeación.
- ✓ Las metodologías ágiles, haciendo énfasis en la adaptabilidad del proceso.

Metodologías Tradicionales, Pesadas (Arboleda Jiménez, 2005):

- ✓ Se caracterizan por exponer procesos basados en planeación exhaustiva.
- ✓ La planeación se realiza esperando que el resultado de cada proceso sea determinante y predecible.

Los resultados de los procesos no son siempre predecibles y sobre todo, es difícil predecir desde el comienzo del proyecto cada resultado. Sin embargo, es posible por medio de la recolección y estudio de métricas de desarrollo lograr realizar estimaciones acertadas en contextos de desarrollo repetibles.

Las metodologías tradicionales son referenciadas por diferentes nombres, por ejemplo, pesadas, ortodoxos, la más significativa y empleada en la actualidad en los procesos de desarrollo de software es RUP. (Navarra, 2005)

RUP (Rational Unified Process) (Jacobson, y otros, 2002).

RUP es un proceso iterativo e incremental de Ingeniería de Software el cual designa tareas y responsabilidades. Asegura la producción de software de alta calidad, capaz de ajustarse a las necesidades de los usuarios finales con un costo y un calendario predecible.

RUP tiene tres características esenciales:

- ✓ Está dirigido por los Casos de Uso: Son una herramienta para especificar los requisitos del sistema. También guían su diseño, implementación y prueba. Los Casos de Uso constituyen un elemento integrador y una guía del trabajo.
- ✓ Está centrado en la arquitectura: Toma en consideración elementos de calidad del sistema, rendimiento, reutilización y capacidad de evolución por lo que debe ser flexible durante todo el proceso de desarrollo. La arquitectura se ve influenciada por la plataforma software, sistema operativo, gestor de bases de datos, protocolos, consideraciones de desarrollo como sistemas heredados.

- ✓ Es iterativo e incremental: Consta de una secuencia de iteraciones. Cada iteración aborda una parte de la funcionalidad total, pasando por todos los flujos de trabajo relevantes y refinando la arquitectura. Cada iteración se analiza cuando termina. Se puede determinar si han surgido nuevos requisitos o han cambiado los existentes, afectando a las iteraciones siguientes.

RUP consta de un conjunto de fases para el desarrollo de software las cuales tienen objetivos específicos. A continuación se mencionan las mismas y se identifican los hitos que se obtienen en cada una de ellas.

- ✓ Inicio: Se determinar la visión del proyecto.
- ✓ Elaboración: Se determinar la arquitectura óptima.
- ✓ Construcción: Se trata de obtener la capacidad operacional inicial.
- ✓ Transición: Se logra obtener el lanzamiento (release) del proyecto.

Las ***fases de Inicio y Elaboración*** se enfocan hacia la comprensión del problema y la tecnología, la delimitación del ámbito del proyecto, la eliminación de los riesgos críticos, y al establecimiento de una línea base (base line) de la arquitectura.

Durante la fase de inicio las iteraciones ponen mayor énfasis en actividades modelado del negocio y de requisitos.

En la ***fase de Elaboración***, las iteraciones se orientan al desarrollo de la línea base de la arquitectura, abarcan más los flujos de trabajo de requerimientos, modelo de negocios (refinamiento), análisis, diseño y una parte de implementación orientado a la línea base de la arquitectura.

En la ***fase de Construcción***, se lleva a cabo la construcción del producto por medio de una serie de iteraciones.

Para cada iteración se selecciona algunos Casos de Uso, se refina su análisis y diseño y se procede a su implementación y pruebas. Se realiza una pequeña cascada para cada ciclo. Se realizan tantas iteraciones hasta que se termine la implementación de la nueva versión del producto.

En la ***fase de Transición*** se pretende garantizar que se tiene un producto preparado para su entrega a la comunidad de usuarios.

Cada unas de las iteraciones son clasificadas y ordenadas según su prioridad, y cada una se convierte luego en un entregable al cliente. Esto trae como beneficio la retroalimentación que se tendría en cada entregable o en cada iteración.

Vale mencionar que el ciclo de vida que se desarrolla por cada iteración, es llevada bajo dos disciplinas (Mendoza Sánchez, 2004):

Disciplina de Desarrollo

- ✓ Ingeniería de Negocios: Entendiendo las necesidades del negocio.
- ✓ Requerimientos: Trasladando las necesidades del negocio a un sistema automatizado.
- ✓ Análisis y Diseño: Trasladando los requerimientos dentro de la arquitectura de software.
- ✓ Implementación: Creando software que se ajuste a la arquitectura y que tenga el comportamiento deseado.
- ✓ Pruebas: Asegurándose que el comportamiento requerido es el correcto y que todo lo solicitado está presente.

Disciplina de Soporte

- ✓ Configuración y administración del cambio: Guardando todas las versiones del proyecto.
- ✓ Administrando el proyecto: Administrando horarios y recursos.
- ✓ Ambiente: Administrando el ambiente de desarrollo.
- ✓ Distribución: Hacer todo lo necesario para la salida del proyecto.

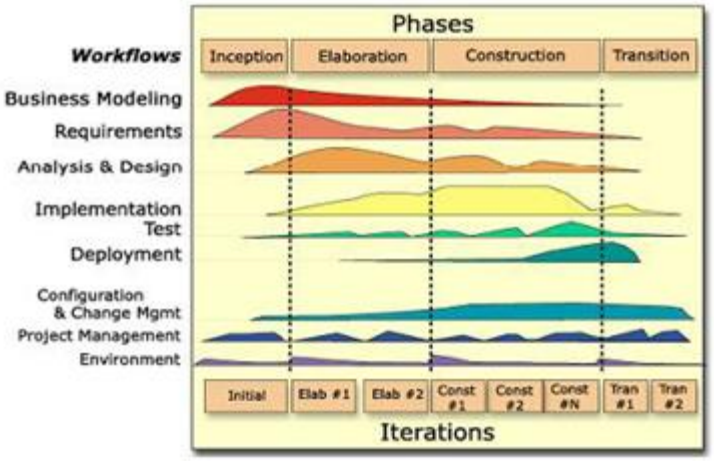

Figura 2: Metodología RUP (Rational Unified Process)

RUP es el resultado de varios años de desarrollo y uso práctico en el que se han unificado técnicas de desarrollo, a través del UML, y trabajo de muchas metodologías utilizadas por los clientes. Como RUP es un proceso en su modelación define como sus principales elementos:

- ✓ Trabajadores (“Quién”): Define el comportamiento y responsabilidades (rol) de un individuo, grupo de individuos, sistema automatizado o máquina, que trabajan en conjunto como un equipo. Ellos realizan las actividades y son propietarios de elementos.
- ✓ Actividades (“Cómo”): Es una tarea que tiene un propósito claro, es realizada por un trabajador y manipula elementos.
- ✓ Artefactos (“Qué”): Productos tangibles del proyecto que son producidos, modificados y usados por actividades. Pueden ser modelos, elementos dentro del modelo, código fuente y ejecutables.
- ✓ Flujo de Actividades (“Cuándo”): Secuencia de actividades realizadas por trabajadores y que produce un resultado de valor observable.

Una particularidad de esta metodología es que, en cada ciclo de iteración, se hace exigente el uso de artefactos, siendo por este motivo, una de las metodologías más importantes para alcanzar un grado de certificación en el desarrollo del software (Mendoza Sánchez, 2004) .

Metodologías Ágiles

- ✓ Fueron formalizadas en el manifiesto para el desarrollo de software ágil, en febrero del 2001 en UTA-EEUU.
- ✓ No están en contra de administrar procesos de desarrollo.
- ✓ Promueve la formalización de procesos adaptables.
- ✓ Su principal característica es la habilidad de responder al cambio y su peso inicialmente ligero.

Las metodologías ágiles tienen los siguientes basamentos:

1. La prioridad es satisfacer al cliente mediante tempranas y continuas entregas de software que le aporte un valor.
2. Dar la bienvenida a los cambios. Se capturan los cambios para que el cliente tenga una ventaja competitiva.
3. Entregar frecuentemente software que funcione desde un par de semanas a un par de meses, con el menor intervalo de tiempo posible entre entregas.
4. Los Funcionarios del negocio y el equipo de desarrollo deben trabajar juntos a lo largo del proyecto.
5. Construir el proyecto en torno a individuos motivados. Darles el entorno y el apoyo que necesitan y confiar en ellos para conseguir finalizar el trabajo.
6. El diálogo cara a cara es el método más eficiente y efectivo para comunicar información dentro de un equipo de desarrollo.
7. El software que funciona es la medida principal de progreso.
8. Los procesos ágiles promueven un desarrollo sostenible. Los promotores, desarrolladores y usuarios deberían ser capaces de mantener una paz constante.
9. La atención continua a la calidad técnica y al buen diseño mejora la agilidad.
10. La simplicidad es esencial.
11. Las mejores arquitecturas, requisitos y diseños surgen de los equipos organizados por sí mismos.
12. En intervalos regulares, el equipo reflexiona respecto a cómo llegar a ser más efectivo, y según esto ajusta su comportamiento (Canós, y otros, 2003).

Entre las metodologías más conocidas figuran:

- ✓ XP (Extreme Programing).
- ✓ Scrum.
- ✓ Crystal Metodologies.
- ✓ MSF (Microsoft Solution Framework).

XP (Extreme Programing)

Es una de las metodologías de desarrollo de software más exitosas en la actualidad, utilizadas para proyectos de corto plazo, equipo pequeño y cuyo plazo de entrega era ayer (entrega inmediata). La metodología consiste en una programación rápida o extrema, cuya particularidad es tener como parte del equipo, al usuario final, pues es uno de los requisitos para llegar al éxito del proyecto (Mendoza Sánchez, 2004) .

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. XP se define como especialmente adecuada para proyectos con requisitos imprecisos y muy cambiantes, y donde existe un alto riesgo técnico.

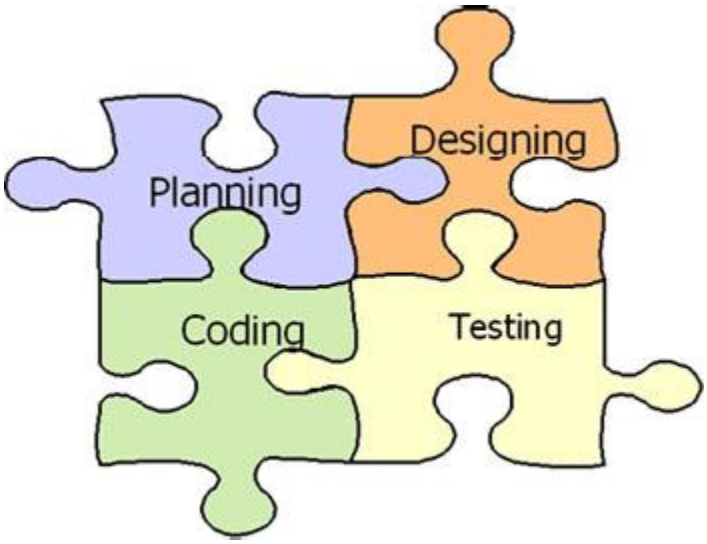

Figura 3: Metodología Extreme Programing.

Características de XP, la metodología se basa en:

- ✓ Pruebas Unitarias: se basa en las pruebas realizadas a los principales procesos, de tal manera que adelantándonos en algo hacia el futuro, se puedan realizar pruebas de las fallas que pudieran ocurrir. Es como adelantarse a obtener los posibles errores.
- ✓ Re fabricación: se basa en la reutilización de código, para lo cual se crean patrones o modelos estándares, siendo más flexible al cambio.
- ✓ Programación en pares: una particularidad de esta metodología es que propone la programación en pares, la cual consiste en que dos desarrolladores participen en un proyecto en una misma estación de trabajo. Cada miembro lleva a cabo la acción que el otro no está haciendo en ese momento. Es como el chofer y el copiloto: mientras uno conduce, el otro consulta el mapa.

  ¿Qué es lo que propone XP?
  - ✓ Empieza en pequeño y añade funcionalidad con retroalimentación continua.
  - ✓ El manejo del cambio se convierte en parte sustantiva del proceso.
  - ✓ El costo del cambio no depende de la fase o etapa.
  - ✓ No introduce funcionalidades antes que sean necesarias.
  - ✓ El cliente o el usuario se convierte en miembro del equipo.

Scrum (Canós, y otros, 2003)

Desarrollada por Ken Schwaber, Jeff Sutherland y Mike Beedle. Define un marco para la gestión de proyectos, que se ha utilizado con éxito durante los últimos 10 años. Está especialmente indicada para proyectos con un rápido cambio de requisitos. Sus principales características se pueden resumir en dos. El desarrollo de software se realiza mediante iteraciones, denominadas sprints, con una duración de 30 días. El resultado de cada sprint es un incremento ejecutable que se muestra al cliente. La segunda característica importante de esta metodología, son las reuniones durante el período de vida del proyecto, destacándose la reunión diaria de 15 minutos del equipo de desarrollo para coordinación e integración.

Crystal Methodologies (Aguilar Sierra, 2003)

- ✓ Da vital importancia a las personas que componen el equipo de proyecto.
- ✓ Sus puntos de estudio son:
  - Aspecto humano del equipo.
  - Tamaño de un equipo (número de componentes).
  - Comunicación entre los componentes.
  - Distintas políticas a seguir.
- ✓ Espacio físico de trabajo.
  - Aconseja que el tamaño del equipo sea reducido.
  - Comunicación entre los miembros del equipo del proyecto.

    Mismo lugar de trabajo -> Disminuye el coste de la comunicación.
    Mejora individual -> Mejora global del equipo.

  - Establece una estructura en el equipo de trabajo indicando roles a cada miembro dentro del proyecto.

MSF (Microsoft Solution Framework)

Es una metodología flexible e interrelacionada con una serie de conceptos, modelos y prácticas de uso, que controlan la planificación, el desarrollo y la gestión de proyectos tecnológicos. MSF se centra en los modelos de proceso y de equipo dejando en un segundo plano las elecciones tecnológicas.

Figura 4: Metodología MSF (Microsoft Solution Framework)

MSF tiene las siguientes características (Mendoza Sánchez, 2004):

- ✓ Adaptable: es parecido a un compás, usado en cualquier parte como un mapa, del cual su uso es limitado a un específico lugar.
- ✓ Escalable: puede organizar equipos tan pequeños entre 3 o 4 personas, así como también, proyectos que requieren 50 personas a más.
- ✓ Flexible: es utilizada en el ambiente de desarrollo de cualquier cliente.
- ✓ Tecnología Agnóstica: porque puede ser usada para desarrollar soluciones basadas sobre cualquier tecnología.

Parámetros	**Metodologías Ágiles.**	**Metodologías Tradicionales (No Ágiles).**
Basamentos.	Basadas en heurísticas provenientes de prácticas de producción de código.	Basadas en normas provenientes de estándares seguidos por el entorno de desarrollo.
Resistencia a cambios.	Especialmente preparados para cambios durante el proyecto.	Cierta resistencia a los cambios.
Determinación.	Impuestas internamente (por el equipo).	Impuestas externamente.
Nivel de Control.	Proceso menos controlado, con pocos principios.	Proceso mucho más controlado, con numerosas políticas/normas.
Contrato.	No existe contrato tradicional o al menos es bastante flexible.	Existe un contrato prefijado.
Interacción Cliente-Equipo de desarrollo.	El cliente es parte del equipo de desarrollo.	El cliente interactúa con el equipo de desarrollo mediante

		reuniones.
Cantidad de personas.	Grupos pequeños (menos de 10 integrantes) y trabajando en un mismo sitio.	Grupos grandes y posiblemente distribuidos.
Cantidad de Artefactos.	Pocos Artefactos.	Más Artefactos.
Roles	Pocos roles.	Más roles.
Énfasis en la Arquitectura.	Menos énfasis en la arquitectura del software.	La arquitectura del software es esencial y se expresa mediante modelos.

Tabla 1: Comparación entre Metodologías Tradicionales y Metodologías Ágiles. (Canós, y otros, 2003)

No basta con la selección de la metodología y seguir los pasos que estas sugieren para poder llevar a cabo el desarrollo de un software, es preciso graficar las acciones que se identifican durante la captura de requisitos para una mejor comprensión de estas, es por ello que se requiere de un lenguaje para poder modelar estas funciones, el cual pueda ser comprendido por todas las personas que intervienen en el proceso de desarrollo del software.

1.3.2. ¿Por qué se seleccionó RUP?

Fue necesario realizar un estudio sobre las metodologías de desarrollo de software existentes en la actualidad, como se puede apreciar en los epígrafes anteriores, para decidir que metodología era la más apropiada a seguir para el desarrollo de la aplicación. Dado que se necesita una amplia y clara documentación de los procesos a desarrollar, se decide que la metodología idónea a utilizar es RUP, debido a que es la estudiada, más conocida y aplicada en la universidad, por ser un proceso iterativo e incremental, flexible, que divide el trabajo en fases teniendo bien definidas las tareas a realizar en cada una de ellas, además establece roles de trabajo dentro del proyecto con el fin de definir una organización del equipo de trabajo asignando tareas y responsabilidades a cada uno de sus miembros. Además de proponer un número de artefactos para cada flujo de trabajo organizando así el proceso de desarrollo.

El tiempo de desarrollo, en el caso del producto que se propone en el presente trabajo, no es el parámetro de más peso, debido fundamentalmente a que es una solución pequeña que incluso utilizando RUP no se invertiría un tiempo de desarrollo grande aunque sin lugar a dudas utilizando una metodología ágil este tiempo se reduciría. El aspecto más importante en este producto es lograr una documentación detallada que tribute a una futura certificación y RUP garantiza esto, según (Mendoza Sánchez, 2004) una particularidad de esta metodología es que, en cada ciclo de iteración, se hace exigente el uso de artefactos, siendo por este motivo, una de las metodologías más importantes para alcanzar un grado de certificación en el desarrollo del software.

1.3.3. Lenguaje de Modelado de sistema

La creciente complejidad de los sistemas informáticos representa un reto importante para los ingenieros y arquitectos del software. De la preocupación inicial sobre la definición de la estructura y calidad del código final, se ha pasado a dedicar cada vez más tiempo, atención y esfuerzo al análisis, diseño y modelado de los sistemas. Los modelos proporcionan un mayor nivel de abstracción, permiten trabajar con sistemas mayores y más complejos, facilitando el proceso de codificación e implementación del sistema de forma distribuida y en distintas plataformas.

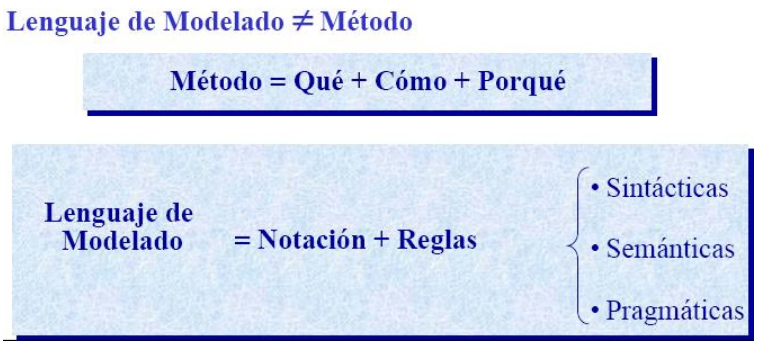

Figura 5: Lenguaje de Modelado.

Entre los lenguajes de modelado que define OMG (Object Management Group) el más conocido y usado es sin duda UML (Unified Modeling Language) (ISO/IEC, 2005).

UML es un lenguaje gráfico para especificar, construir y documentar los artefactos que modelan un sistema, ofrece un estándar para describir un modelo del sistema, incluyendo aspectos tales como

procesos de negocios y funciones del sistema, así como aspectos concretos, dígase expresiones de lenguajes de programación, esquemas de bases de datos y componentes de software reutilizables.

UML fue diseñado para ser un lenguaje de modelado de propósito general, por lo que puede utilizarse para especificar la mayoría de los sistemas basados en objetos o en componentes, para modelar aplicaciones de muy diversos dominios de aplicación (telecomunicaciones, comercio, sanidad.), plataformas de objetos distribuidos (como por ejemplo J2EE20, .NET21 o CORBA22) y además se puede aplicar en una gran variedad de formas para soportar una metodología de desarrollo de software como por ejemplo el Proceso Unificado de Rational, pero no especifica en sí mismo, qué metodología o proceso utilizar.

En UML existen tipos diferentes de diagramas, y se organizan jerárquicamente para una mejor comprensión.

Diagramas de estructura enfatizan en los elementos que deben existir en el sistema modelado:

- ✓ Diagrama de clases.
- ✓ Diagrama de componentes.
- ✓ Diagrama de objetos.
- ✓ Diagrama de estructura compuesta (UML 2.0).
- ✓ Diagrama de despliegue.
- ✓ Diagrama de paquetes.

Diagramas de comportamiento, enfatizan en lo que debe suceder en el sistema modelado:

- ✓ Diagrama de actividades.
- ✓ Diagrama de casos de uso.
- ✓ Diagrama de estados.

Diagramas de Interacción, un subtipo de diagramas de comportamiento, que enfatiza sobre el flujo de control y de datos entre los elementos del sistema modelado:

- ✓ Diagrama de secuencia.
- ✓ Diagrama de comunicación.

- ✓ Diagrama de tiempos (UML 2.0).
- ✓ Diagrama de vista de interacción (UML 2.0).

Para el modelado de UML es preciso el empleo de Herramientas CASE (Computer Aided Software Engineering), las cuales están destinadas a aumentar la productividad en el desarrollo de software reduciendo el coste de las mismas en términos de tiempo y de dinero. Estas herramientas ayudan en todos los aspectos del ciclo de vida de desarrollo del software en tareas como el proceso de realizar captura de requisitos, análisis y modelado, diseño del proyecto, cálculo de costes, implementación de parte del código automáticamente con el diseño dado, compilación automática, documentación o detección de errores, entre otras.

1.3.4. Herramientas para Modelado de sistema

Las Herramientas CASE tienen como principales objetivos (Herramienta CASE, 2003) :

1. Mejorar la productividad en el desarrollo y mantenimiento del software.
2. Aumentar la calidad del software.
3. Mejorar el tiempo y coste de desarrollo y mantenimiento de los sistemas informáticos.
4. Mejorar la planificación de un proyecto.
5. Aumentar la biblioteca de conocimiento informático de una empresa ayudando a la búsqueda de soluciones para los requisitos.
6. Automatizar, desarrollo del software, documentación, generación de código, pruebas de errores y gestión del proyecto.
7. Ayuda a la reutilización del software, portabilidad y estandarización de la documentación.
8. Gestión global en todas las fases de desarrollo de software con una misma herramienta.
9. Facilitar el uso de las distintas metodologías propias de la ingeniería del software.

Las herramientas CASE se pueden clasificar en base a los parámetros siguientes:

- ✓ Las plataformas que soportan.
- ✓ Las disciplinas del ciclo de vida del desarrollo de sistemas que cubren.
- ✓ La arquitectura de las aplicaciones que producen.
- ✓ Su funcionalidad.

La siguiente clasificación es la más habitual, basada en las fases del ciclo de desarrollo que cubren:

- ✓ Upper CASE, herramientas que ayudan en las fases de planificación, análisis de requisitos y estrategia del desarrollo, usando, entre otros diagramas UML.
- ✓ Middle CASE, herramientas para automatizar tareas en el análisis y diseño de la aplicación.
- ✓ Lower CASE, herramientas que semi-automatizan la generación de código, crean programas de detección de errores, soportan la depuración de programas y pruebas. Además documentación completa de la aplicación. Aquí pueden incluirse las herramientas de Desarrollo rápido de aplicaciones.

Por funcionalidad se podrían diferenciar algunas como:

- ✓ Herramientas de generación semiautomática de código.
- ✓ Editores UML.
- ✓ Herramientas de Refactorización de código.
- ✓ Herramientas de mantenimiento como los sistemas de control de versiones.

Ejemplo de estas **herramientas** que soportan el Lenguaje de Modelado UML son:

Rational Rose

- ✓ Herramienta de modelación visual que provee el modelado basado en UML.
- ✓ Es en la actualidad, una de las herramientas CASE más potentes, es la herramienta que comercializan los desarrolladores de la Corporación Rational.
- ✓ Soporta de forma completa la especificación de UML.
- ✓ Basada principalmente en el nivel de integración que tiene este con el resto de las herramientas que lo acompañan en la Suite entre las que aparecen:
  - Rational Clear CASE, para el control de versiones.
  - Rational Clear Quest, para el control de cambios.
  - Rational Model Integrator, para la integración de los artefactos.
  - Rational Requisite Pro, herramienta de administración de requerimientos.

- ✓ Brinda la posibilidad de generar y realizar ingeniería inversa (IBM Rational Software, 2007) en una buena cantidad de lenguajes de programación en su versión XDE y el número de framework que vienen predefinidos, entre los cuales se pueden citar .Net, J2EE, C++ , Visual Basic 6.
- ✓ Permite que haya varias personas trabajando a la vez en el proceso iterativo controlado, para ello posibilita que cada desarrollador opere en un espacio de trabajo privado que contiene el modelo completo y tenga un control exclusivo sobre la propagación de los cambios en ese espacio de trabajo.

Visual Paradigm for UML

- ✓ Permite los principales diagramas UML, aunque restringe a un solo tipo de diagrama en cada modelo.
- ✓ Validación en tiempo real del modelo.
- ✓ Interfaz de usuario configurable.
- ✓ Estructuración u organización automática de diagramas.
- ✓ Copia de diagramas como imágenes.
- ✓ Soporta subdiagramas para todos los modelos UML.
- ✓ Provee pleno soporte para ingeniería directa e inversa.

Poseidón

- ✓ Poseidón Community Edition: Esta versión permite generar los diagramas básicos de UML, aunque en un ambiente no profesional, tiene restringidas opciones como copia y pega al Clipboard de Windows, de modo que no permite copiar los diagramas como imágenes a presentaciones o documentos. Permite la generación de código en Java pero no soporta la ingeniería inversa.
- ✓ Poseidón Edición Estándar: Permite entre sus funcionalidades principales la generación de documentación automática, ingeniería directa e inversa para código Java, soporta plataformas como Windows, Linux y Mac y posee un mecanismo de plugins que permite una configuración alta en función de las necesidades del usuario.
- ✓ Poseidón Edición Profesional: Permite Ingeniería en modelos de ida y vuelta manteniendo perfecta coherencia entre la modelación y el código, y permite además generación de código de alto nivel en un gran número de lenguajes.

1.3.5. ¿Por qué UML y Visual Paradigm?

Se decide emplear el lenguaje de modelado UML, de propósito general, el más reconocido y usado en la actualidad, lenguaje gráfico para especificar, construir y documentar los diferentes artefactos, decidiéndose emplear para ello la herramienta informática de modelado Visual Paradigm la cual permite diseñar todos los artefactos que se generan a través del ciclo de vida del producto. La organización estándar que provee para guardar la información está muy relacionada con el proceso de desarrollo seleccionado. Posibilita el modelado del negocio con notación BPMN.

1.4. Conclusiones

En este capítulo se definieron conceptos importantes relacionados con el objeto de estudio, se presentaron las principales características de las metodologías y herramientas de desarrollo de software, así como el lenguaje de modelado, además se expuso un estudio de las tecnologías y/o plataformas para el desarrollo de aplicaciones web. Luego de dicho estudio se determinó aplicar RUP como metodología de desarrollo y usar el lenguaje de modelado UML sobre la herramienta Visual Paradigm. Esta selección fue basada en los diferentes criterios que se explican a lo largo del capítulo.

Capítulo 2: Características del sistema

Las metodologías de desarrollo de software, como se mostró en el capítulo anterior definen los pasos a seguir para el desarrollo de un software y RUP, metodología que se acordó seguir para guiar el proceso de desarrollo del software, en sus disciplinas de desarrollo plantea como primer paso el entendimiento del negocio, o sea, la Ingeniería del Negocio. En el capítulo que se desarrolla a continuación se realiza el Modelado del Negocio del análisis de secuencia identificando así los principales procesos que tienen lugar en el entorno, luego se procederá a la captura de requisitos o también denominados requerimientos con el propósito de trasladar las necesidades del Negocio hacia el sistema. Posteriormente se da inicio a la próxima fase (Análisis de Sistema), donde se analizan las exigencias que se describieron en la Captura de Requisitos; se refinan y estructuran estos, con el fin de comprender detalladamente los requerimientos y obtener una descripción de los mismos que sea fácil de mantener y que ayude a estructurar el sistema entero. (Jacobson, y otros, 2002). He aquí los temas que se abordarán en el presente capítulo, se exponen los Casos de Uso del Sistema que obedecen a la temática de este trabajo.

2.1. Descripción general de la propuesta de sistema

En el presente epígrafe se describe el funcionamiento del sistema, como va a interactuar con los usuarios y el Arquitecto de Información. El sistema va a permitir al Arquitecto iniciar sesión introduciendo su nombre y contraseña luego aparecerá un panel de control el cual consta de varias opciones:

- ✓ Crear nuevo análisis.
- ✓ Ver análisis (listado).
- ✓ Cambiar contraseña.

2.1.1. Crear nuevo análisis

Al entrar en esta opción el arquitecto se va a encontrar con tres pasos para crear el análisis, el primer paso es un formulario nombrado datos generales donde introduce el nombre del análisis y el tipo de análisis a realizar, el cual puede ser abierto o cerrado. El análisis es abierto cuando el usuario tiene la posibilidad de sugerir otros términos a ordenar agregándolo en una nueva tarjeta, mientras el análisis cerrado se realiza únicamente con las tarjetas presentes sin posibilidad de agregar nuevas al mismo. Este formulario contiene un vínculo que le permite continuar con el siguiente paso.

El segundo paso contiene un nuevo formulario donde se confeccionan las tarjetas; para ello se tiene la opción nueva tarjeta donde se introduce el nombre de la misma, el cual sería la etiqueta a ordenar seleccionada por el arquitecto y los usuarios potenciales del producto de información a desarrollar. Después de adicionada la tarjeta, esta es visualizada en una lista de tarjetas adicionas mostrando los números y términos de las mismas. Se tendrá la posibilidad de marcar cuales quiera de estas tarjetas con el fin de eliminarlas y/o editarlas. Este formulario contiene vínculos los cuales permiten regresar al formulario anterior o continuar con el tercer y último paso para crear el nuevo análisis.

El tercer paso de este asistente permitirá introducir la información de los usuarios a los cuales se desea invitar a realizar el análisis de secuencia. Para ello de cada usuario se introduce su nombre completo y su dirección de correo electrónico, a medida que se va introduciendo estos datos, se va creando una lista de participantes, la cual es mostrada en este mismo paso con el objetivo de eliminar y/o editar cualquier información que haya sido introducida incorrectamente.

Una vez finalizado el último paso, el sistema guarda la información del nuevo análisis de secuencia creado. De forma transparente al arquitecto, se le asigna un número de identificación al análisis y de igual forma se realiza esta operación para cada uno de los usuarios que han sido seleccionados como muestra. El sistema envía un correo electrónico de invitación a dichos usuarios, el cual entre otras informaciones, contiene un enlace a un módulo del sistema donde realizará finalmente la organización de tarjetas. Este enlace contiene dos parámetros con el objetivo de identificar al usuario que accede al módulo y qué análisis de secuencia debe ser cargado por el mismo. Un ejemplo de vínculo sería: http://analisissecuencia.uci.cu/Tablero?IdAnalisis=1&IdUsuario=3 cuando el usuario que recibe este enlace accede al mismo, el sistema interpreta que el usuario con número de identificación **3** ha accedido al sistema para realizar el análisis de secuencia con número de identificación **1**.

2.1.2. Ver análisis (listado)

Esta funcionalidad permite al arquitecto ver los análisis de secuencia creados por él. Para ello el sistema muestra un listado con el número de identificación y el nombre de los análisis. A partir de este listado se pueden realizar dos funcionalidades, una es eliminar uno o varios análisis, para ello solo basta con seleccionarlos y oprimir la opción eliminar y la otra es visualizar el detalle de un análisis en específico, *(ver epígrafe 2.1).*

2.1.2.1. Ver detalles

A partir del listado de análisis explicado en el epígrafe anterior, se puede ver el detalle de un análisis de secuencia determinado, para ello se selecciona el análisis en el listado y se oprime la opción “Ver detalles”. Una vez realizada esta operación se muestra una interfaz, la cual muestra el nombre y el tipo de análisis (abierto o cerrado), visualiza la lista de los usuarios invitados seleccionados como muestra, presentando de los mismos el número de identificación, el correo electrónico y el estado, o sea, si realizó el análisis o no.

Esta pantalla muestra además las tarjetas a utilizar en el análisis seleccionado (número de identificación y nombre/etiqueta).

Desde esta interfaz se puede importar el resultado del análisis realizado por un usuario en otra aplicación *(ver epígrafe 2.1.1)*, especificando en la lista de usuarios, que este resultado fue importado y no realizado en el sistema y a su vez se puede eliminar un usuario determinado.

2.1.2.1.1. Importar

Para importar el resultado del análisis realizado por un usuario en otra aplicación se selecciona la opción importar desde la interfaz explicada en el epígrafe anterior la cual muestra un cuadro de diálogo que permite seleccionar el archivo a importar, el mismo debe seguir el formato que se explica en el epígrafe 2.2 lo cual garantiza la interoperabilidad entre sistemas.

2.1.2.1.2. Eliminar usuario

Para eliminar uno o varios usuarios del análisis de secuencia se realiza la selección de los mismos y se oprime la opción eliminar.

2.1.2.1.3. Resultados del usuario

Para acceder a los resultados de un usuario en específico el arquitecto de información selecciona al usuario desde la lista de usuarios en los detalles del análisis y oprime la opción ver resultados del usuario. Luego el sistema visualiza la interfaz de resultado mostrando el nombre del usuario, el nombre del análisis de secuencia y las tarjetas ordenadas según el criterio de dicho usuario, de las mismas se presenta el número en un extremo y el nombre de la etiqueta en el centro. Al colocar el mouse encima de las tarjetas, el sistema muestra la hora en que fue seleccionada por el usuario para organizarla y la hora en que fue

colocada. La pantalla muestra además la hora en que se inició y terminó el análisis y brinda la posibilidad de funcionalidades como “Ver Sesión” y “Exportar” explicadas a continuación.

2.1.2.1.3.1. Ver sesión

Esta interfaz tiene como objetivo simular el modo en que el usuario colocó las tarjetas, aparecerá una línea de tiempo y un video donde se puede observar la expresión facial del usuario en el momento de organizar cada una de las tarjetas, también se puede apreciar qué términos ofrecieron dificultad, el tiempo que demora en ordenar cada tarjeta entre otros aspectos que contribuyen al análisis cualitativo del análisis de secuencia.

2.1.2.1.3.2. Exportar

Permite exportar el resultado del análisis de secuencia para un usuario en un archivo que luego puede ser importado por otras aplicaciones. Este archivo utiliza el mismo formato de la funcionalidad importar explicada en el epígrafe 2.1.2.1.1.

2.1.2.1.4. Ver resultados

Ver resultados es una de las funcionalidades más importantes del sistema debido a que muestra, como lo dice el nombre de dicha funcionalidad, los resultados cuantitativos generales del análisis de secuencia. Se baso en una secuencia de pasos los cuales son los realizados por los arquitectos de información tradicionalmente

2.1.3. Cambiar contraseña

La funcionalidad cambiar contraseña permite al arquitecto cambiar su contraseña de inicio de sesión. Para ello introduce una nueva contraseña y confirma dicha contraseña introduciéndola nuevamente en otro campo para evitar errores de escritura.

2.1.4. Interacción de los usuarios con el sistema

Como se menciona al principio del capítulo además del arquitecto de información, el usuario también interactúa con el sistema. Luego de acceder al enlace recibido en el correo de invitación se encontrará con una página que le permitirá realizar su sesión de análisis de secuencia. Esta presenta datos generales al usuario como son el nombre del análisis de secuencia y el tipo de análisis que realiza, los datos del usuario (nombre y correo electrónico), y constará además de un espacio de trabajo donde estarán las tarjetas y un lugar donde se pueden ubicar las mismas en el orden que estime conveniente el usuario.

En caso de ser un análisis abierto el usuario cuenta con la opción de crear una tarjeta nueva y conjuntamente con la tarjeta se genera un espacio para la ubicación una tarjeta más. Para eliminar una tarjeta simplemente se selecciona la misma y esta mostrará una opción para ser eliminada, al seleccionar dicha opción se elimina la tarjeta seleccionada. Solamente presentarán la posibilidad de ser eliminada aquellas que han sido adicionadas por el usuario.

2.2. Estructura y formato de la información manejada por el sistema

Como se explica en las funcionalidades del sistema, es posible tanto importar como exportar análisis de secuencias con el objetivo de adicionar en la aplicación análisis creados en aplicaciones de terceros y viceversa, para contribuir en cierta medida a la interoperabilidad. Dentro de la bibliografía consultada no se pudo constatar la existencia de algún estándar de información referente a análisis de secuencia, es por ello que en este epígrafe se explica el formato escogido para tal fin en el sistema que se diseña en este trabajo así como su estructura.

XML como formato de archivo de información de análisis de secuencia

Anteriormente se mencionaba que una de las características que se persigue es que el sistema sea interoperable con aplicaciones de terceros, para lo cual se brinda la funcionalidad de importar y exportar análisis de secuencia. Ante la no existencia de un estándar de información de análisis de secuencia, al menos dentro de la bibliografía consultada, se decide utilizar un estándar para el formato del archivo de intercambio.

El Lenguaje de Etiquetado Extensible (XML del inglés Extensible Markup Language) es un formato de datos independiente y común a través de empresas y organizaciones, además de esto posibilita:

- ✓ Estructuras y tipos de datos estándares, independiente de cualquier lenguaje de programación, entorno de desarrollo, o software.
- ✓ Tecnología dominante para la definición de documentos e intercambio de información de negocio, incluyendo estándares de información de muchas industrias.

XML provee un gran conjunto de características aunque solo se han mencionada aquellas que lo hacen un excelente formato para el intercambio de información y a su vez garantizar interoperabilidad entre

sistemas informáticos. Es por ello que se decide seleccionar este estándar como formato de los archivos de intercambio de información utilizados en el sistema.

Estructura del archivo de información de análisis de secuencia

Para describir la estructura del archivo de información se decide utilizar XML Schema, teniendo en cuenta que el formato seleccionado es XML. Esta selección trae varias ventajas de acuerdo a varios puntos de vista, en primer lugar provee un estándar para la descripción de la estructura y brinda una herramienta para la fase de implementación ya que a partir del esquema se puede determinar la validez del documento XML, tanto en estructura como en tipo de datos utilizados.

La siguiente figura, para una mejor comprensión, muestra un gráfico que representa al esquema de un archivo de información de análisis de secuencia.

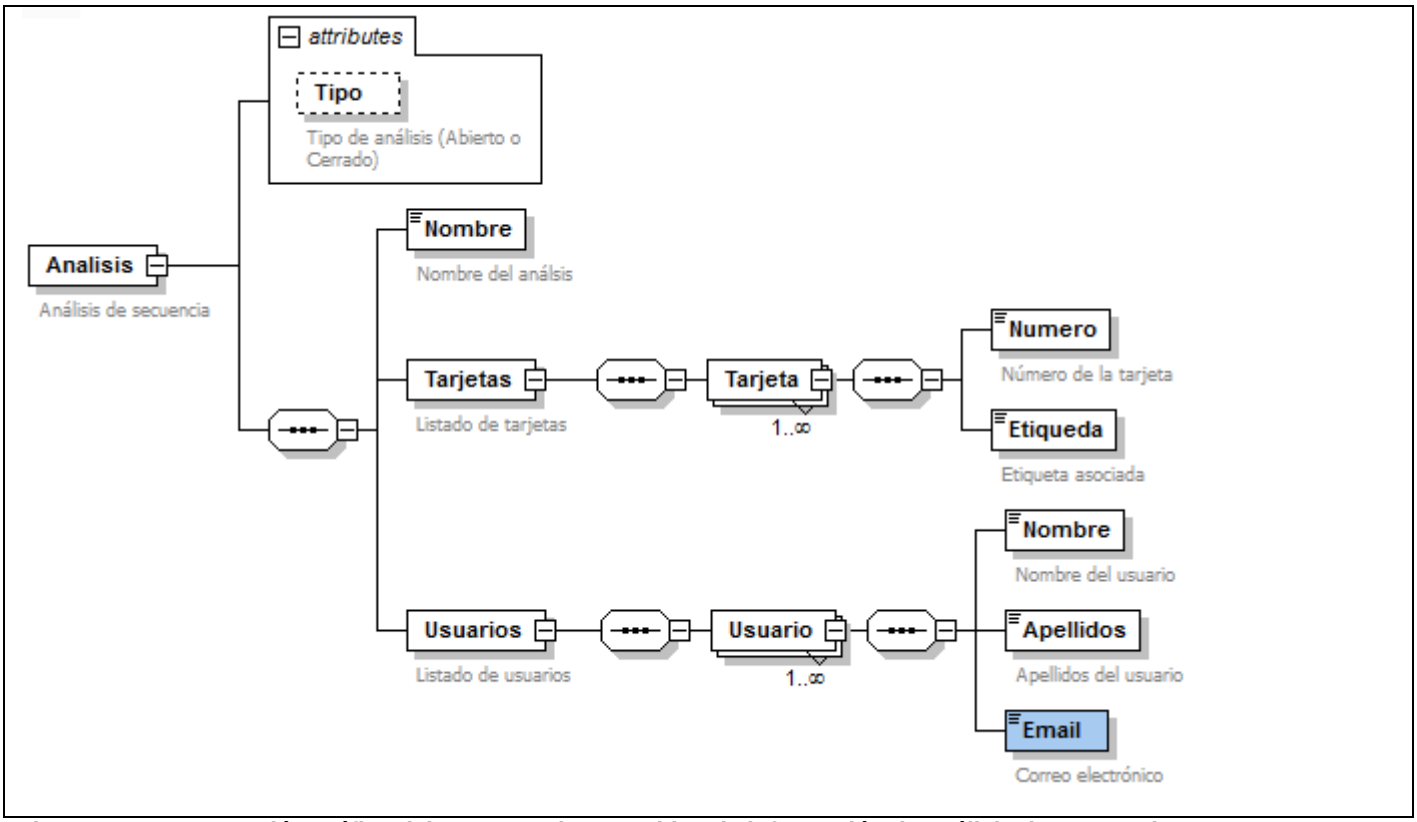

Figura 6: Representación gráfica del esquema de un archivo de información de análisis de secuencia.

Como se puede apreciar en la figura 6, el esquema especifica como elemento raíz a la etiqueta Analisis, la cual tiene un atributo para definir el tipo de análisis llamado “tipo” el cual solo acepta los valores (Abierto o Cerrado). Además contiene otras tres etiquetas:

- ✓ Nombre: valor de tipo cadena para guardar el nombre del análisis.
- ✓ Tarjetas: valor de tipo “tipo complejo o complexType” debido a que su contenido es el listado de tarjetas (elementos Tarjeta) pertenecientes al análisis de secuencia.
- ✓ Usuarios: similar a Tarjetas, pero en este caso contiene el listado de usuarios (elementos Usuario) del análisis.

Por su parte las etiquetas Tarjeta y Usuario son a su vez de tipo complejo ya que contienen elementos para guardar la información de cada una de estas entidades como por ejemplo (Nombre, Apellidos y Email) en el caso de Usuario. Para un estudio más detallado del esquema ver anexo 1.

2.3. Procesos del negocio

¿Qué es un Modelo?

Según (DRAE, 1992) un modelo es la expresión de una realidad o sistema complejo mediante algún lenguaje formal o simbolismo gráfico que facilita su comprensión y el estudio de su comportamiento.

Un modelo es una invención: algo que se concibe para explicar una serie de datos que se desea interpretar. (Pressman, 2001)

Para que un modelo sea útil, tiene que permitir que todos los datos "encajen" de forma coherente, es decir, tiene que poder explicar lo que pasa de una manera lógica. (Martínez, y otros, 2002)

Por su propia definición, un modelo debe cumplir con tres requisitos básicos:

- ✓ General, es decir, debe ser válido para cualquier aplicación del campo que formaliza.
- ✓ Abstracto, ya que con esto se puede separar las características particulares del objeto de estudio para extraer su esencia.
- ✓ Consistente, para lograr que cada elemento tenga una única definición, acorde con la función que se espera que represente y coherente con el resto de componentes del modelo. (Koutras, 1990)

¿Qué es BPMN?

BPMN (Business Process Modeling Notation) es un nuevo estándar de modelado de procesos de negocio, en donde se presentan gráficamente las diferentes etapas del proceso del mismo. La notación ha sido diseñada específicamente para coordinar la secuencia de procesos y los mensajes que fluyen entre los diferentes procesos participantes.

Define un Business Process Diagram (BPD), que se basa en una técnica de grafos de flujo para crear modelos gráficos de operaciones de procesos de negocio. Un modelo de procesos de negocio, es una red de objetos gráficos, que son actividades (trabajo) y controles de flujo que definen su orden de rendimiento.

El modelado de procesos de negocio se usa para comunicar una amplia variedad de información a diferentes audiencias. BPMN está diseñado para cubrir muchos tipos de modelados y para permitir la creación de segmentos de proceso así como procesos de negocio, con diferentes niveles de fidelidad.

Objetivos de BPMN

BPMN proporciona a los negocios la capacidad de entender sus procedimientos internos en una notación gráfica, facilitando a las organizaciones la habilidad para comunicar esos procedimientos de una manera estándar. Por tanto sus principales objetivos son:

- ✓ Proveer una notación que sea fácilmente entendida por todos los usuarios, desde el analista de negocio, el desarrollador técnico y hasta la propia gente del negocio.
- ✓ Crear un puente estandarizado para el vacío existente entre el diseño del proceso de negocio y su implementación.
- ✓ Asegurar que los lenguajes para la ejecución de los procesos de negocio puedan ser visualizados con una notación común.
- ✓ BPMN es usado para comunicar una amplia variedad de información a una amplia variedad de audiencias

Diagramas de Procesos de Negocio

Son una representación abstracta (gráfica) de los procesos de una organización, que muestran principalmente cómo y por quién son llevadas a cabo las actividades que generan valor para la organización. Muestran también:

- ✓ Los actores involucrados en los procesos.
- ✓ ¿Cuáles son las actividades operativas distinguibles?
- ✓ ¿Qué actividades son ejecutables y por quien?
- ✓ ¿Cuáles son las entradas y salidas de actividades?
- ✓ ¿Cuál es la secuencia de las actividades?
- ✓ Los recursos consumidos, y los eventos que dirigen el proceso.

Las cuatro categorías básicas de elementos que se pueden encontrar en un Diagrama de Proceso de Negocio son:

- ✓ Objetos de Flujo
- ✓ Objetos de Conexión
- ✓ Roles (swimlane)
- ✓ Artefactos

Los **diagramas de procesos del negocio** se encuentran en el artefacto “Modelo del Negocio con BPMN”.

2.4. Especificación de los requisitos de software

2.4.1. ¿Qué son los Requerimientos?

De las muchas definiciones que existen para requerimiento, a continuación se presenta la definición que aparece en el glosario de la IEEE.

Requerimientos son una condición o necesidad de un usuario para resolver un problema o alcanzar un objetivo. Una condición o capacidad que debe estar presente en un sistema o componentes de sistema para satisfacer un contrato, estándar, especificación u otro documento formal. Una representación documentada de una condición o capacidad, como las condiciones antes mencionadas. (Saiden, 1999)

Para obtener bien detallados los requerimientos del sistema es preciso el empleo de técnicas para su captura, pues en la mayoría de los casos los clientes solicitan los servicios para el desarrollo de

soluciones informáticas que les sirvan de soporte para desarrollar y agilizar su trabajo, pero no tienen bien definido las funcionalidades que ellos desean que tenga el software y no son capaces de explicarse bien o hacerse entender en la mayoría de los casos, previendo razones como estas, es que se aplican técnicas para la Captura de Requisitos. Los requerimientos son importantes y es donde las técnicas del UML son especialmente provechosas. (UML, 2000)

2.4.2. Principales técnicas para la Captura de Requisitos

Se listan a continuación algunas de las técnicas que emplean en la actualidad los Analistas de Sistemas de los equipos de desarrollo de software para identificar los requerimientos del sistema a desarrollar.

Entrevistas y Cuestionarios (Saiden, 1999) (Hofmann, 1993)

Las entrevistas y cuestionarios se emplean para reunir información proveniente de personas o de grupos. Durante la entrevista, el Analista conversa con el encuestado; el cuestionario consiste en una serie de preguntas relacionadas con varios aspectos de un sistema.

Por lo común, los encuestados son usuarios de los sistemas existentes, trabajadores del entorno o usuarios en potencia del sistema propuesto. En algunos casos, son gerentes o empleados que proporcionan datos para el sistema propuesto o que serán afectados por él.

Las preguntas que deben realizarse en esta técnica, deben ser preguntas de alto nivel y abstractas que se realizan generalmente al inicio del proyecto para obtener información sobre aspectos globales del problema del usuario y soluciones potenciales.

Las preguntas pueden ser enfocadas a un elemento del sistema, tales como usuarios, procesos, etc.

Análisis de sistemas existentes (Hofmann, 1993) (Saiden, 1999)

Consiste en el análisis de los diferentes sistemas existentes ya desarrollados que estén relacionados con el sistema a ser construido.

Con esta técnica se pretende:

- ✓ Analizar las diferentes interfaces y la información que se maneja en ellas.
- ✓ Analizar las salidas (listas, consultas) de los sistemas.

- ✓ Importante analizar el porqué de la información que se maneja en las diferentes interfaces.

Grabaciones de Video

Son empleadas como registro y apoyo de las entrevistas, además para analizar algún proceso en específico. Apoyo de las entrevistas pues permite concentrarse en la entrevista en sí y no tomar nota de cuanta cosa se diga. Importante es que se debe iniciar la grabación con preguntas poco importantes que permitan relajar al entrevistado.

Arqueología de Documentos (Hofmann, 1993) (Saiden, 1999)

Se tratan de determinar posibles requerimientos sobre la base de inspeccionar la documentación utilizada y generada en la empresa. Sirve más que nada como complemento de las demás técnicas y ayuda a obtener información que de otra manera sería sumamente difícil obtener por ejemplo, información que no se pensaba manejar, tipos de codificaciones empleadas en la empresa.

Para el análisis de los documentos se deben tener presente algunos objetivos como:

- ✓ Propósito del documento.
- ✓ Persona que lo usa. Por qué y Para qué.
- ✓ Tareas que se realizan con ese documento.
- ✓ Relación o dependencia con otros documentos.
- ✓ Proceso que realiza la conexión entre los documentos.
- ✓ Documento más conflictivo para los usuarios.

Observación (Hofmann, 1993) (Saiden, 1999)

Consiste en observar cómo se realizan las actividades en la empresa, esta es una buena manera de entender lo que estos requieren. Conectarse íntimamente con la cultura de la organización, vivirla, es una herramienta que debe ser tomada en cuenta.

Se pueden realizar filmaciones del lugar de trabajo, para luego observarlos cuidadosamente y obtener resultados del análisis.

Siempre se debe estar atento a lo que sucede en el entorno de la organización, ver cómo resuelven un problema que surge, modo en que fluyen las operaciones.

Prototipos (Hofmann, 1993) (Saiden, 1999)

Desarrolladores y clientes se reúnen y definen los objetivos globales del software, identifican todos los requerimientos que son conocidos, y señalan áreas en las que será necesaria la profundización en las definiciones. Luego de esto, tiene lugar un "diseño rápido". El diseño rápido se centra en una representación de aquellos aspectos del software que serán visibles al usuario (por ejemplo, entradas y formatos de las salidas). El prototipo es evaluado por el Cliente – Usuario, es utilizado para refinar los requerimientos del software a ser desarrollado. Un proceso de iteración tiene lugar a medida que el prototipo es "puesto a punto" para satisfacer las necesidades del cliente y permitiendo al mismo tiempo una mejor comprensión del problema por parte del desarrollador.

Lluvia de Ideas (Hofmann, 1993) (Saiden, 1999)

Básicamente se busca que los involucrados en un proyecto desarrollen su creatividad, promoviendo la introducción de los principios creativos.

A esta técnica se le conoce también como torbellino de ideas, tormenta de ideas, desencadenamiento de ideas, movilización verbal, bombardeo de ideas, sacudidas de cerebros, promoción de ideas, tormenta cerebral, avalancha de ideas, tempestad en el cerebro y tempestad de ideas, entre otras.

Principios de la lluvia de ideas

- ✓ Aplazar el juicio y no realizar críticas, hasta que no agoten las ideas, ya que actuaría como un inhibidor. Se ha de crear una atmósfera de trabajo en la que nadie se sienta amenazado.
- ✓ Cuantas más ideas se sugieren, mejores resultados se conseguirán: "la cantidad produce la calidad". Las mejores ideas aparecen tarde en el período de producción de ideas, será más fácil que encontremos las soluciones y tendremos más variedad sobre la que elegir.
- ✓ La producción de ideas en grupos puede ser más efectiva que la individual.

2.4.3. Requisitos funcionales y no funcionales

Como resultado de las operaciones anteriormente descritas se logran identificar Requerimientos Funcionales y No Funcionales del software a desarrollar, ver artefacto “Especificación de requisitos”.

Definición de requerimientos funcionales y no funcionales.

Los **requerimientos funcionales** son capacidades o condiciones que el sistema debe cumplir. (Pressman, 2001)

Los **requerimientos no funcionales** son propiedades o cualidades que el producto debe tener. Debe pensarse en estas propiedades como las características que hacen al producto atractivo, usable, rápido o confiable. En muchos casos los requerimientos no funcionales son fundamentales en el éxito del producto. (Pressman, 2001)

2.5. Actores del sistema

Los actores son todos los agentes externos que provocan un evento y un cambio de estado del sistema, cosas y personas externas que interaccionan con el sistema y no son parte de él. Como se ha venido describiendo el proceso de análisis de secuencia tanto en la etapa del Negocio como en el Sistema, se identifican Actores, a continuación se enumeran y describen los correspondientes al Modelo de Sistema. En este trabajo se identificaron dos actores que se describen en el artefacto “Modelo del sistema”.

Figura 7: Representación gráfica de los actores del sistema.

2.6. Definición de los casos de uso del sistema

Los casos de uso son artefactos narrativos que describen, bajo la forma de acciones y reacciones, el comportamiento del sistema desde el punto de vista del usuario. Por lo tanto, establece un acuerdo entre

clientes y desarrolladores sobre las condiciones y posibilidades que debe cumplir el sistema. (Pressman, 2001)

2.6.1. Patrones

El planteamiento de formalizar soluciones a distintas situaciones, de modo que puedan ser entendidas por otros profesionales, es lo que se llama patrones. Por tanto, un patrón no es más que la descripción detallada de una solución adecuada a un problema concreto.

Un patrón es un modelo que podemos seguir para realizar algo. Los patrones surgen de la experiencia de seres humanos de tratar de lograr ciertos objetivos. Los patrones capturan la experiencia existente y probada para promover buenas prácticas . Los patrones permiten y han permitido en diferentes áreas del conocimiento humano rehusar la esencia de la solución de un problema al enfrentar nuevos problemas similares.

2.6.2. Patrones de casos de uso utilizados

Con motivo de capturar técnicas para que el modelo sea reusable y entendible, se utilizó el patrón de caso de uso **CRUD** y otros patrones adicionales como **Include** y **Extend**.

CRUD se utiliza en los casos donde se quiere realizar altas, bajas, cambios y consultas a alguna entidad del sistema, su nombre es acrónimo de la palabra en inglés, Create, Read, Update, Delete. (Cuesta Rodriguez, 2005)

CRUD Parcial: Indica, que en caso de que solo algunas de las cuatro operaciones sean simples mientras que otras son complejas, se puede agrupar las operaciones simples en un caso de uso y dejar las otras modeladas como un caso de uso separado.

CRUD Completo: Consiste en un caso de uso para administrar la información (CRUD Información), permite modelar las diferentes operaciones para administrar una entidad de información, tales como crear, leer, cambiar, y dar baja. Este patrón deberá ser usado cuando todas las operaciones contribuyen al mismo valor de negocio y todas son cortas y simples.

Include: En términos muy simples, cuando relacionamos dos casos de uso con un “include”, se está diciendo que el primero (el caso de uso base) incluye al segundo (el caso de uso incluido). Es decir, el

segundo es parte esencial del primero. Sin el segundo, el primero no podría funcionar bien; pues no podría cumplir su objetivo. (Berrocal, y otros, 2009)

Extend: En ciertos escenarios el caso de uso base no podría cumplir su objetivo si no se ejecutara la extensión. Pero, una de las diferencias básicas es que en el caso del "extend" hay situaciones en que el caso de uso de extensión no es indispensable que ocurra, y cuando lo hace ofrece un valor extra (extiende) al objetivo original del caso de uso base.

Después de analizar cada uno de los requerimientos funcionales identificados y aplicando los patrones de casos de uso explicados en el epígrafe anterior, se obtuvieron diez casos de usos que darán respuesta a las necesidades del sistema:

CU-1. Autenticar arquitecto

CU-2. Crear nuevo análisis

CU-3. Gestionar tarjeta

CU-4. Gestionar usuario

CU-5. Gestionar datos análisis

CU-6. Ver resultados usuario

CU-7. Ver resultado general del análisis

CU-8. Realizar análisis de secuencia

CU-9. Ordenar tarjetas

CU-10. Cambiar contraseña

2.6.3. Descripción de los casos de uso del sistema

A continuación aparece una breve descripción de los casos de uso más importantes del sistema. Para ver una descripción más detallada de todos los casos de uso, ver el documento "Modelo del sistema":

CU-2	Crear nuevo análisis
Actor	Arquitecto de Información
Descripción	El caso de uso comienza cuando el arquitecto solicita introducir los datos generales del análisis de secuencia, luego confecciona las tarjetas y posteriormente crea la muestra de usuarios invitándolos a realizar el análisis de secuencia.
Referencia	RF 2, RF 3, CU-3, CU-4

Tabla 2: Descripción de los casos de uso del sistema <Crear nuevo análisis>

CU-3	Gestionar tarjetas
Actor	Arquitecto de Información
Descripción	El caso de uso comienza cuando el arquitecto inserta los datos en el Sistema con el fin de confeccionar las tarjetas a utilizar en el análisis de secuencia que está creando. La tarjeta puede ser modificada, eliminada o mostrada al arquitecto.
Referencia	RF 4, RF 5, RF 6, RF 7, CU-2

Tabla 3: Descripción de los casos de uso del sistema <Gestionar tarjetas>

CU-4	Gestionar usuarios
Actor	Arquitecto de Información
Descripción	El caso de uso se inicializa cuando el arquitecto inserta en el sistema los datos de los usuarios mostrándose el listado de los mismos, este usuario se crea, se muestra, puede ser modificado o eliminado por el arquitecto.
Referencia	RF 8, RF 9, RF 10, RF 11, RF 12, CU-2

Tabla 4: Descripción de los casos de uso del sistema <Gestionar usuarios>

CU-5	Gestionar datos análisis

Actor	Arquitecto de Información
Descripción	El caso de uso comienza cuando arquitecto desea consultar el listado de análisis de secuencia, puede eliminar un análisis así como importar un análisis, además puede visualizar los detalles del mismo, puede modificar un análisis, además puede ver resultados generales del análisis o resultados de un análisis en particular.
Referencia	RF 13, RF 14, RF 15, RF 16, RF 17, RF 18, CU-6

Tabla 5: Descripción de los casos de uso del sistema <Gestionar datos análisis>

CU-6	Ver resultados usuario
Actor	Arquitecto de Información
Descripción	El caso de uso comienza cuando el arquitecto desde los detalles del análisis desea consultar el resultado de un usuario seleccionado.
Referencia	RF 19, RF 20, RF 21, CU-5

Tabla 6: Descripción de los casos de uso del sistema <Ver resultados usuario>

CU-7	Ver resultado general del análisis de secuencia
Actor	Arquitecto de Información
Descripción	El caso de uso se inicia cuando el arquitecto desde el listado de análisis y/o detalles del análisis solicita consultar el resultado general.
Referencia	RF 28, CU-5

Tabla 7: Descripción de los casos de uso del sistema <Ver resultado general del análisis>

CU-8	Realizar análisis de secuencia

Actor	Usuario
Descripción	El caso de uso comienza cuando el usuario entra a la aplicación mediante la dirección enviada por correo desde el sistema y así puede realizar el análisis de secuencia.
Referencia	RF 23, RF 24, CU-9

Tabla 8: Descripción de los casos de uso del sistema <Realizar análisis de secuencia>

2.6.3.1. Diagrama de casos de uso del sistema

Un diagrama de casos de uso del sistema representa gráficamente las funcionalidades principales del sistema y su interacción con los actores. (Ver el artefacto “Modelo del sistema”)

2.7. Conclusiones

En el presente capítulo se inició el desarrollo de la propuesta de solución para el problema planteado. Se generó un listado de requerimientos funcionales, lo que permitió posteriormente la definición de las funcionalidades del sistema. Se identificaron los principales elementos del negocio para lograr un acercamiento a las clases que intervendrán en el modelo de análisis y diseño. A partir de este momento se puede comenzar a trabajar en el análisis y diseño del mismo teniendo en cuenta que cumpla con todos los requisitos planteados en el capítulo.

Capítulo 3: Análisis y diseño del sistema

3.1. Introducción

En el presente capítulo se mantiene la trazabilidad de los requerimientos identificados en el capítulo anterior, realizando su documentación en las etapas de análisis y diseño planteadas por la metodología de desarrollo escogida. Esta documentación consta de los artefactos necesarios que contribuyen a la implementación del sistema. Estos artefactos son el Modelo de Análisis y el Modelo de Diseño, los cuales se encargan de describir en términos de clases cómo deben funcionar los casos de uso del sistema, se presenta el diseño de la base de datos del sistema y un prototipo de cómo debe quedar la interfaz de la aplicación para su interacción con los usuarios. Aquí también se logra representar la colaboración entre estas clases y se dan a conocer los patrones de diseño que se utilizan en la elaboración de las mismas.

3.2. Modelo de Análisis

En la etapa de Análisis se realiza un modelo de análisis donde se incluyen las descripciones textuales de los casos de usos, así como los diagramas de clases. Esto proporciona la estructura de una vista interna del sistema, estructurado por clases estereotipadas:

Entidad: Modelan información que posee larga vida y que es a menudo persistente.

Interfaz: Modelan la interacción entre el sistema y sus actores.

Controladora: Coordinan la realización de uno o unos pocos casos de uso coordinando las actividades de los objetos que implementan la funcionalidad del caso de uso.

El Modelo de Análisis debe lograr tres objetivos primarios:

- ✓ Describir lo que requiere el cliente.
- ✓ Establecer una base para la creación de un diseño de software.
- ✓ Definir un conjunto de requisitos que se pueda validar una vez que se construye el software.

El propósito del análisis es definir todas las clases que son relevantes al problema que se va a resolver, las operaciones y atributos asociados, las relaciones y comportamientos asociadas con ellas. (Pressman, 2001) Este modelo es usado para representar la estructura global del sistema, describe la realización de

casos de uso y sirve como una abstracción del Modelo de Diseño. El Modelo de Análisis puede contener: las clases y paquetes de análisis, las realizaciones de los casos de uso, las relaciones y los diagramas. “Durante el análisis, analizamos los requisitos que se describen en la captura de requerimientos, refinándolos y estructurándolos. El objetivo de hacerlo es conseguir una comprensión más precisa de los requisitos y una descripción de los mismos que sea fácil de mantener y que nos ayude a estructurar el sistema entero, incluyendo su arquitectura”. (Jacobson, y otros, 2002)

3.2.1. Diagramas de clases del Análisis

Los diagramas de clases del análisis representan la relación entre las clases que intervienen en los casos de uso. Proporcionan una vista interna del sistema. Es utilizado para comprender de forma general la estructura del sistema y sirve como entrada para la etapa de diseño.

3.2.2. Diagramas de Interacción

Los diagramas de interacción se utilizan para modelar los aspectos dinámicos de un sistema, lo que conlleva modelar instancias concretas o prototípicas de clases interfaces, componentes y nodos, junto con los mensajes enviados entre ellos, todo en el contexto de un escenario que ilustra un comportamiento. En el contexto de las clases describen la forma en que grupos de objetos colaboran para proveer un comportamiento. Se dividen en dos categorías los diagramas de colaboración y los diagramas de secuencia. Un diagrama de secuencia es un diagrama de interacción que destaca la ordenación temporal de los mensajes. Muestran las interacciones expresadas en función de secuencias temporales. Un diagrama de colaboración es un diagrama de interacción que destaca la organización estructural de los objetos que envían y reciben mensajes. Muestran las relaciones entre los objetos y los mensajes que intercambian.

Los diagramas de clases del análisis de los casos de uso críticos del sistema propuesto y los diagramas de interacción (diagramas de secuencia) correspondiente a estos casos de uso (*ver documento Modelo del Sistema)*:

3.3. Modelo de Diseño

El diseño consiste en el refinamiento de los Modelos de Análisis para crear especificaciones adicionales que enriquecen el modelo de análisis con detalles próximos a la implementación. Una solución lógica, de forma que se cumplan los requerimientos (asignación de responsabilidades, interacciones entre objetos y

otros.) Sirve de abstracción de la implementación y es utilizada como entrada fundamental de las actividades de implementación. Este modelo puede contener: los diagramas, las clases, paquetes, subsistemas, cápsulas, protocolos, interfaces, relaciones, colaboraciones, atributos, las realizaciones de los casos de uso, entre otros que se puedan considerar para el sistema en desarrollo. “El Modelo de Diseño es un modelo de objetos que describe la realización física de los casos de uso centrándose en cómo los requisitos funcionales y no funcionales, junto con otras restricciones relacionadas con el entorno de implementación, tienen impacto en el sistema a considerar. Además, el Modelo de Diseño sirve de abstracción de la implementación del sistema y es, de ese modo, utilizada como una entrada fundamental de las actividades de implementación”. (Jacobson, y otros, 2002)

3.3.1. Diagramas de clases de diseño

A través del flujo de diseño, uno de los artefactos más importantes a obtener son los diagramas de clases de diseño, donde se exponen las clases que intervienen en las realizaciones de los casos de uso del sistema. En este tipo de diagrama se representa un nivel de detalle más alto que los diagramas de clases del análisis, relacionándose con el lenguaje de programación del cual se hará uso en la implementación del sistema. El diagrama de clases de diseño describe gráficamente las especificaciones de las clases de software y de las interfaces en una aplicación. Normalmente contiene la siguiente información:

- ✓ Clases, asociaciones y atributos
- ✓ Interfaces, con sus operaciones y constantes
- ✓ Métodos
- ✓ Información sobre los tipos de los atributos, navegabilidad y dependencias

3.3.2. Diseño de la base de datos

La descripción del diseño de la base de datos se encuentra en el documento “Modelo del Diseño”, en el epígrafe 5.1 Diagrama de entidad relación y 5.2 Descripción de las tablas.

3.3.2. ASP.NET MVC

Es de vital importancia destacar que para el correcto entendimiento del diseño del sistema (diagramas de clases del diseño y diagramas de secuencia) es necesario conocer el funcionamiento del framework ASP.NET MVC para lo cual se recomienda la lectura de los libros Professional MVC 1.0 (Conery, y otros, 2009) y Pro ASP.NET Framework (Sanderson, 2009). En los diagramas se hace referencia a varias clases

y/o interfaces que son parte de Microsoft .NET Framework para acceder a una detallada información de las mismas consultar la documentación en línea de Microsoft Developer Network (MSDN) (Microsoft Corp., 2009).

A pesar de haberse realizado una descripción de las principales características de ASP.NET MVC en el capítulo 1, en los siguientes epígrafes se desea hacer énfasis en dos aspectos importantes del mismo, los cuales son ActionResult, y ViewDataDictionary y Model.

3.3.2.1. ActionResult

En el patrón Modelo Vista Controlador (MVC) el propósito de las clases controladoras, entre otras responsabilidades, es responder a la interacción del usuario con el sistema. En ASP.NET MVC, un método acción o más conocido por el inglés “action method” es la unidad básica de respuesta a la interacción del usuario, estos action methods no son más que los métodos públicos de las clases controladoras encargados de manejar las peticiones tanto HTTP-GET como HTTP-POST realizadas por el usuario y crear y devolver la respuesta que es mostrada al mismo, la cual generalmente es HTML.

El patrón que un action method sigue es: realizar cualquier operación que se le haya sido solicitada, y al final, devolver una instancia de un tipo que derive de la clase abstracta ActionResult. ActionResult representa la implementación del patrón de diseño comando o “command” (Gamma, y otros, 1994) y no es más que el comando u operación que el action method desea que el framework realice en tu nombre. ActionResult se encarga generalmente de llevar a cabo el trabajo a nivel de framework, mientras que los action methods se encargan de la lógica de la aplicación.

ASP.NET MVC, como se puede apreciar en la siguiente tabla, incluye varios tipos de ActionResult para llevar a cabo tareas comunes.

Tipo ActionResult	Descripción
EmptyResult	Representa una respuesta nula o vacía.
ContentResult	Escribe un contenido específico directamente a la respuesta en formato de texto.

JsonResult	Serializa los objetos recibidos hacia JSON[3] y escribe el JSON resultante a la respuesta.
RedirectResult	Envía al usuario hacia una determinada URL.
RedirectToRouteResult	Envía al usuario hacia una URL especificada a través de parámetros de ruta.
ViewResult	Invoca al motor de vistas para generar una vista hacia la respuesta.
PartialViewResult	Similar a ViewResult, excepto que genera una vista parcial hacia la respuesta, generalmente en respuesta a una petición AJAX.
FileResult	Sirve como clase base para un conjunto de resultados que escriben una respuesta binaria hacia el flujo de salida. Útil para devolución de ficheros al usuario.
FilePathResult	Deriva de FileResult y devuelve la escritura de un fichero hacia la salida basado en el camino del fichero.
FileContentResult	Deriva de FileResult y devuelve la escritura de un arreglo de bytes hacia la respuesta.
FileStreamResult	Deriva de FileResult y devuelve la escritura de un flujo hacia la respuesta.
JavaScriptResult	Utilizado para ejecutar de forma inmediata código JavaScript en el cliente enviado desde el servidor.

Tabla 9: Varios tipos de ActionResult.

3.3.2.2. ViewDataDictionary y Model

La vista es responsable de proveer una interfaz de usuario. Recibe una referencia del modelo y transforma el mismo a un formato listo para ser presentado al usuario. En ASP.NET MVC este proceso consiste en

[3] JavaScript Object Notation

examinar una instancia de ViewDataDictionary suministrado por el controlador, a través de la propiedad ViewData, y transformar dicha instancia a HTML.

ViewDataDirectory tiene un objeto fuertemente tipado llamado Model, que es al cual la vista está encargada de visualizar. Este objeto puede representar a un objeto del dominio, como por ejemplo una instancia de Usuario, o podría representar un objeto del modelo de presentación específico a la vista.

3.4. Validación

3.4.1. Validación de requisitos

La validación de requisitos examina las especificaciones para asegurar que todos los requisitos del sistema han sido establecidos sin ambigüedad, sin inconsistencias, sin omisiones, que los errores detectados hayan sido corregidos, y que el resultado del trabajo se ajusta a los estándares establecidos para el proceso, el proyecto y el producto.

El primer mecanismo para la validación de los requisitos es la revisión técnica formal. El equipo de revisión incluye ingenieros del sistema, clientes, usuarios, y otros intervinientes que examinan la especificación del sistema buscando errores en el contenido o en la interpretación, áreas donde se necesitan aclaraciones, información incompleta, inconsistencias. Es un problema importante, requisitos contradictorios, o requisitos imposibles o inalcanzables.

Aunque la validación de requisitos puede guiarse de manera que se descubran errores, es útil chequear cada requisito con un cuestionario. Las siguientes cuestiones representan un pequeño subconjunto de las preguntas que pueden plantearse (Pressman, 2001):

- ✓ ¿Está el requisito claramente definido? ¿Puede interpretarse mal?
- ✓ ¿Está identificado el origen del requisito (por ejemplo: persona, norma, documento)? ¿El planteamiento final del requisito ha sido contrastado con la fuente original?
- ✓ ¿El requisito está delimitado en términos cuantitativos?
- ✓ ¿Qué otros requisitos hacen referencia al requisito estudiado? ¿Está claramente identificados por medio de una matriz de referencias cruzadas o por cualquier otro mecanismo?
- ✓ ¿El requisito incumple alguna restricción definida?

- ✓ ¿El requisito es verificable? Si es así, podemos efectuar pruebas (algunas veces llamadas criterios de validación) para verificar el requisito?
- ✓ ¿Se puede seguir el requisito en el modelo del sistema que hemos desarrollado?
- ✓ ¿Se puede localizar el requisito en el conjunto de objetivos del sistema/producto?
- ✓ ¿Está el requisito asociado con los rendimientos del sistema o con su comportamiento y han sido establecidas claramente sus características operacionales? ¿El requisito está implícitamente definido?

Las preguntas planteadas en la lista de comprobación ayudan a asegurar que el equipo de validación dispone de lo necesario para realizar una revisión completa de cada requisito.

Para validar los requisitos del sistema que se desarrolla en el presente trabajo se llevaron a cabo dos acciones:

1. Se le mostró y explicó al cliente el modelo de proceso del negocio para su aprobación.
2. Se le realizaron las preguntas propuestas por (Pressman, 2001) para cada uno de los requisitos, lo cual arrojó una matriz de respuestas (*ver Anexo 2*).

3.4.2. Métricas de validación del diseño

Con el objetivo de validar el diseño del sistema desarrollado, se aplicaron un seleccionado número de métricas perteneciente a dos grupos de métricas conocidas para este fin. A continuación se menciona en qué consisten las mismas y luego se muestra una tabla con el resultado de estas métricas aplicadas a los diagramas de clases de la aplicación.

Serie de métricas CK (Pressman, 2001)

- ✓ **Árbol de profundidad de herencia (APH)**: Esta métrica se define como la máxima longitud del nodo a la raíz del árbol.
- ✓ **Número de descendientes (NDD)**: Las subclases inmediatamente subordinadas a una clase en la jerarquía de clases se denominan sus descendientes.
- ✓ **Respuesta para una clase (RPC)**: El conjunto de respuesta de una clase es una serie de métodos que pueden potencialmente ser ejecutados, en respuesta a un mensaje recibido por un objeto, en la clase.RPC se define como el número de métodos en el conjunto de respuesta.

- ✓ **Carencia de cohesión en los métodos (CCM)**: Cada método dentro de una clase, C, accede a uno o más atributos (también llamados variables de instancia) CCM es el número de métodos que accede a uno o más de los mismos atributos.

Métricas propuestas por Lorenz y Kidd (Pressman, 2001)

- ✓ **Tamaño de clase (TC)**: El tamaño general de una clase puede medirse determinando las siguientes medidas:
  - El total de operaciones.
  - El número de atributos.
- ✓ **Número de operaciones redefinidas para una subclase (NOR)**: Existen casos en que una subclase reemplaza una operación heredada de su superclase por una versión especializada para su propio uso. A esto se le llama redefinición.
- ✓ **Número de operaciones añadidas por una subclase (NOA)**: Las subclases se especializan añadiendo operaciones y atributos privados.

Métrica	Valor	Observaciones
Serie de métricas CK		
APH	2	
NDD	10	De los diagramas de clases de los paquetes, el del paquete Vistas es el de mayor NDD debido a que necesariamente en el framework ASP.NET MVC cada vista debe heredar de la clase **ViewPage<T>**. Sin embargo en los demás paquetes los valores de NDD son 1, 2 y 3.
RPC	1	
CCM	12	Este valor es relativamente alto debido a que es el valor de CCM de la clase **AnalisisRepository**, ya que como parte del patrón Repositorio, todos los métodos deben hacer uso del atributo contexto a través del cual se realizan las consultas a la base de datos. Además es la clase

		repositorio más grande del sistema debido a que manipula la información de la entidad principal del dominio: **Analisis**[4]. El valor CCM máximo entre las demás clases, incluidos los restantes repositorios es 4.
Métricas propuestas por Lorenz y Kidd		
TC	40	Este valor es grande, aunque es preciso destacar que debido a la naturaleza de ASP.NET MVC, cada controlador debe tener asignadas las responsabilidades que le corresponde para que así funcione semánticamente el mecanismo de ruteo. Este valor es el valor del controlador de análisis que como ya se ha mencionado es la entidad principal del sistema, sobre la cual se realizan la mayoría de las acciones. El valor de TC para la clase más grande después de este controlador es 12 para la clase **AnalisisRepository**.
NOR	0	
NOA	12	Este es el caso de la clase **AnalisisRepository**. En las observaciones de la métrica CCM se explica por qué este número es relativamente alto.

Tabla 10: Resultado de la aplicación de las métricas.

Nota: Todos los valores de la tabla representan el valor máximo encontrado para cada métrica dentro de todos los diagramas de clases del diseño del sistema.

3.4.3. Patrones de diseño

Los patrones de diseño posibilitan la reutilización de diseños y arquitecturas en una forma exitosa y sencilla ya que expresan técnicas probadas. Los patrones ayudan a escoger alternativas de diseño que

[4] La palabra “Analisis” en este caso no se acentuó debido a que es el nombre de una clase, aunque el lenguaje de programación a utilizar soporta acentos, se decidió no utilizarlos por si en un futuro se desea migrar la aplicación a otra plataforma.

hacen al sistema más reusable y evitan alternativas que comprometen la reusabilidad. Pueden mejorar la documentación y mantenimiento de sistemas existentes suministrando una especificación explicita de una clase, las interacciones entre los diferentes objetos y su marcada intención. Puesto de una manera sencilla, los patrones de diseño ayudan al diseñador a obtener un diseño de una forma "correcta" y rápida. (Gamma, y otros, 1994)

Teniendo en cuenta lo expresado en el párrafo anterior, se puede afirmar que sin lugar a dudas, el uso de patrones de diseño en el desarrollo de un sistema proporciona un cierto grado de validación al mismo, debido a que son soluciones probadas que se le han ido dando a problemas de diseño conocidos.

Existen varios grupos de patrones conocidos, como por ejemplo el grupo de patrones de la pandilla de los cuatro (Gamma, y otros, 1994) y los patrones generales de software para asignar responsabilidades (GRASP) del inglés "General Responsibility Asignment Software Patterns". Los patrones se clasifica de varias maneras, (Gamma, y otros, 1994) hace referencia a dos criterios, el primero, según el propósito del patrón, los cuales pueden ser: de creación, estructurales y de comportamiento; y el segundo criterio, es según el ámbito, donde se especifica si un patrón se aplica primariamente a una clase o a un objeto.

En el modelo de diseño del trabajo se aplican fundamentalmente los siguientes patrones:

- ✓ **Repositorio:** Mediador entre los objetos del dominio y las capas de mapeo de datos. Encapsula toda la lógica necesaria para obtener referencia a objetos, de esta manera los objetos del dominio no tienen que interactuar directamente con infraestructuras para obtener las referencias que necesiten (Evans, 2006).
- ✓ **Comando:** Encapsula una petición como un objeto, por lo que permite configurar clientes con diferentes peticiones, colas o bitácoras de peticiones y soporta operaciones "deshacer". Este patrón es conocido también como "Acción" y "Transacción" (Gamma, y otros, 1994). Todas las clases controladoras del diseño del sistema descrito en este trabajo forman parte de la implementación del patrón comando del framework ASP.NET MVC, siendo comandos concretos, subclases de la clase abstracta *Controller* de dicho framework.
- ✓ **Experto:** Asignar una responsabilidad al experto en información, la clase que posee la información necesaria para cumplir con la responsabilidad (Larman, 1999). Se aplica en las clases controladores y del modelo (entidades) del sistema diseñado en el presente trabajo.

- ✓ **Controlador:** Asignar la responsabilidad de controlar el flujo de eventos del sistema, a clases específicas. Esto facilita la centralización de actividades (validaciones, seguridad y otros). La mayor parte de los sistemas reciben eventos de entrada externa, por lo cual, si se recurre a un diseño orientado a objetos, hay que elegir los controladores que manejen esos eventos de entrada (Larman, 1999). Se aplica a las clases controladoras del sistema diseñado en el presente trabajo.
- ✓ **Creador:** Asignar a la clase B la responsabilidad de crear una instancia de clase A en alguno de los siguientes casos (B agrega los objetos de A; B contiene a los objetos de A; B registra las instancias de los objetos de A; B utiliza específicamente los objetos de A; B tiene los datos de inicialización que serán transmitidos a A cuando este objeto sea creado). El propósito fundamental de este patrón es encontrar un creador que debemos conectar con el objeto producido en cualquier evento. Al escogerlo como creador, se da soporte al bajo acoplamiento (Larman, 1999). Se aplica a las clases del modelo (entidades) del sistema diseñado en el presente trabajo.

3.5. Conclusiones

El análisis y diseño de un software brinda la primera visión de lo que pudiera ser la solución en el desarrollo del mismo. En este capítulo se han mostrado las funcionalidades que tendrá el sistema que se describe en el presente trabajo de diploma, a través de una serie de modelos que permiten tener un mayor entendimiento de este. Se definieron las principales clases del análisis y el diseño mostrando sus relaciones así como la colaboración entre ellas mediante diferentes diagramas. Para lograr un mejor diseño se utilizaron algunos patrones que ayudan a la optimización del mismo. El resultado del capítulo posibilitará la guía para la futura implementación de la aplicación.

Conclusiones

Con la realización del presente trabajo se arriba a las siguientes conclusiones:

- ✓ Se obtuvieron los artefactos necesarios para la modelación de la aplicación a partir de la selección de RUP como metodología de desarrollo de software, debido a su particularidad de ser iterativo – incremental, facilitó la organización, obtención y perfeccionamiento de los mismos.
- ✓ Se identificaron los principales procesos del negocio a partir del estudio del mismo, lo cual permitió su modelado con BPMN, mostrando así cómo y por quién son llevadas a cabo las actividades de estos procesos.
- ✓ Se logró la definición de requerimientos funcionales y no funcionales acorde a las necesidades del cliente, a partir del intercambio con el mismo y el entendimiento del negocio.
- ✓ Se diseñaron los artefactos relacionados a los modelos de análisis y diseño del sistema concerniente al proceso de realización del Análisis de Secuencia y procesamiento de resultados cuantitativos, lo cual servirán de guía para el proceso de implementación de la aplicación.

Recomendaciones

Se recomienda la implementación de la solución modelada siguiendo el diseño obtenido en el presente trabajo, así como ir adicionando nuevos procesos de la Arquitectura de Información que estén directamente relacionados a los resultados del Análisis de Secuencia. Como otra recomendación se sugiere investigar sobre cómo contribuir aún más a obtener resultados cualitativos a partir del modo en que se realiza el análisis de secuencia descrito en este trabajo e implementarlo en el mismo.

Glosario de términos

A

Análisis (Flujo de Trabajo)

Flujo de trabajo fundamental cuyo propósito principal es analizar los requisitos descritos en la captura de requisitos, mediante su refinamiento y estructuración. El objetivo de esto es: lograr una comprensión más precisa de los requisitos y obtener una descripción de los requisitos que sea más fácil de mantener y que ayude a dar estructura al sistema en su conjunto.

Aplicación (sistema)

Sistema que ofrece a un usuario final un conjunto coherente de casos de uso.

Arquitectura

Conjunto de decisiones significativas acerca de la organización de un sistema software, la selección de los elementos estructurales a partir de los cuales se compone el sistema, y las interfaces entre ellos, junto con su comportamiento, tal y como se especifica en las colaboraciones entre esos elementos, la composición de estos elementos estructurales y de comportamiento en subsistemas progresivamente mayores, y el estilo arquitectónico que guía esta organización: estos elementos y sus interfaces, sus colaboraciones y su composición. La arquitectura del software se interesa no solo por la estructura y el comportamiento, sino también por las restricciones y compromisos de uso, funcionalidad, flexibilidad al cambio, funcionamiento, reutilización, comprensión, economía y tecnología, así como por aspectos estéticos.

Artefacto

Pieza de información tangible que es creada, modificada y usada por los trabajadores al realizar actividades; representa un área de responsabilidad, y es candidata a ser tenida en cuenta para el control de la configuración. Un artefacto puede ser un modelo, un elemento de un modelo, o un documento.

C

Caso de Uso

Es una descripción de un conjunto de secuencia de acciones, incluyendo variaciones, que un sistema lleva a cabo y que conduce a un resultado observable de interés para un actor determinado.

Computer Aided Software Engineering

Ingeniería de Software Asistida por Ordenador.

CORBA

Common Object Request Broker Architecture (Arquitectura común de intermediarios en peticiones a objetos). Es un estándar que establece una plataforma de desarrollo de sistemas distribuidos, facilitando la invocación de métodos remotos bajo un paradigma orientado a objetos.

C++

Lenguaje de Programación diseñado a mediados de los años 1980, por Bjarne Stroustrup, como extensión del lenguaje de programación C.

E

Etiqueta

Las etiquetas son una forma de representación de la información y se utiliza a lo largo de todo el sitio Web.

Entidades

Representa un contenedor de información, algo físico que se utilice en el proceso del negocio y que sirva para obtener información o para actualizar información. Generalmente tiene estados, en dependencia de en qué momento aparezca como parte del proceso.

F

Fase

Período de tiempo entre dos hitos principales de un proceso de desarrollo.

Framework

Es una estructura de soporte definida, en la cual otro proyecto de software puede ser organizado y desarrollado. Puede incluir soporte de programas, bibliotecas y un lenguaje de scripting entre otros software para ayudar a desarrollar y unir los diferentes componentes de un proyecto.

H

HTML

Acrónimo inglés de HyperText Markup Language (lenguaje de marcas hipertextuales), lenguaje de marcación diseñado para estructurar textos y presentarlos en forma de hipertexto, que es el formato estándar de las páginas web.

I

Ingeniería de Software

Disciplina de la Ingeniería que concierne a todos los aspectos de la producción de software. Es una parte de la Ingeniería de la Ingeniería de Sistemas (concierne a todos los aspectos del desarrollo de sistemas basados en cómputo, que incluyen hardware, software y el proceso de Ingeniería).

IEEE

Instituto de IEEE de los ingenieros electrónicos eléctricos. Fundado en 1963, IEEE es una organización integrada por ingenieros, científicos, y estudiantes. IEEE es el mejor conocido para los estándares que se convierten para la industria del ordenador y del elemento electrónico.

J

J2EE

Java 2 Enterprise Edition. Es una parte de la plataforma de programación Java. Basada en componentes de software modulares ejecutándose sobre un servidor de aplicaciones.

L

Línea base

Conjunto de artefactos revisados y aprobados que representa un punto de acuerdo para la posterior evolución y desarrollo, y solamente puede ser modificado a través de un procedimiento formal, como la gestión de cambios y configuraciones.

Línea base de la arquitectura

Línea base resultado de la fase de elaboración.

M

Metodologías

Se encargan de elaborar estrategias de desarrollo de software que promuevan prácticas adoptativas en vez de predictivas; centradas en las personas o los equipos, orientadas hacia la funcionalidad y la entrega, de comunicación intensiva y que requieren implicación directa del cliente.

Modelos

Es una descripción de (parte de) un sistema, descrito en un lenguaje bien definido. Un lenguaje bien definido es un lenguaje con una sintaxis y semántica precisa y que puede ser interpretado y manipulado por un ordenador.

N

.NET

Plataforma de desarrollo de software creada por Microsoft, con énfasis en transparencia de redes, con independencia de plataforma y permite un rápido desarrollo de aplicaciones.

P

PHP

Hypertext Pre-processor. Lenguaje de programación usado frecuentemente para la creación de contenido para sitios web con los cuales se puede programar las páginas HTML y los códigos fuente.

R

Requisitos

Condición o capacidad que debe cumplir un sistema. Requisitos Funcionales: especifica una acción que debe ser capaz de realizar el sistema. Requisito No Funcional: especifica restricciones físicas sobre un requisito funcional, dígase restricciones del entorno o de implementación, rendimiento, dependencias de la plataforma.

S

Software

Es la suma total de los programas de cómputo, procedimientos, reglas, documentación y datos asociados que forman parte de las operaciones de un sistema de cómputo y que “un producto de software es un producto diseñado para un usuario”.

V

Visual Basic

Lenguaje de programación guiado por eventos, y centrado en un motor de formularios que facilita el rápido desarrollo de aplicaciones gráficas.

X

XDE

Nueva herramienta de Rational, que se integra a los IDEs de Java o a Microsoft Visual Studio .NET, permitiendo modelar e implementar sin necesidad de trabajar con herramientas diferentes.

XML

Extensible Markup Language (lenguaje de Marcas Extensibles). Es un metalenguaje extensible de etiquetas desarrollado por el World Wide Web Consortium (W3C). Permite definir la gramática de lenguajes específicos de la misma manera que HTML.

Bibliografía

Aguilar Sierra, Alejandro. 2003. Las Metodologías Ágiles en la Enseñanza de la Ingeniería de Software. 2003.

Arboleda Jiménez, MSc. Hugo F. 2005. Modelos de ciclo de vida en desarrollo de software. *Asociación Colombiana de Ingenieros de Sistemas (ACIS).* [Online] Julio 2005. http://www.acis.org.co/index.php?id=551.

Berrocal, Javier, García Alonso, José Manuel and Murillo Rodríguez, Juan Manuel. 2009. Patrones para Extracción de Casos de Usos a partir de proceso de negocio. 2009, Vol. 3, 3.

Brancheau, James C and Wetherbe, James C. 1986. *Arquitectura de información: métodos y prácticas.* 1986. pp. 453-463. Vol. 22.

Burbeck, Steve. 1992. Applications Programming in Smalltalk-80(TM): How to use Model-View-Controller (MVC). [En línea] 1992. http://st-www.cs.illinois.edu/users/smarch/st-docs/mvc.html.

Canós, José H., Letelier, Patricio and Penadés, Mª Carmen. 2003. Metodologías ágiles en el desarrollo de Software. [Online] Noviembre 12, 2003. http://issi.dsic.upv.es/tallerma/actas.pdf.

Conery, Rob, y otros. 2009. *Professional ASP.NET MVC 1.0.* Indianapolis : Wiley Publishing, Inc., 2009. 978-0-470-38461-9.

Cuesta Rodriguez, Saul. 2005. Patrones de Casos de Uso. 2005, 6.

Dickson, Gary W and Wetherbe, James C. 1985. *La gestión de sistemas de información.* New York : McGraw-Hill, 1985.

DRAE. 1992. *Diccionario de la Real Academia de la Lengua Española.* Madrid : s.n., 1992.

Evans, Eric. 2006. *Domain-Driven Design.* s.l. : Addison Wesley, 2006. 0-321-12521-5.

Gamma, Erich, y otros. 1994. *Desing Patterns. Elements of Reusable Object-Oriented Software.* s.l. : Addison-Wesley Longman, Inc., 1994.

Gullikson, Shelley, et al. 1999. The impact of information architecture on academic web site usability. 1999, Vol. 17, 5, pp. 293-304.

Hassan Montero, Yusef, et al. 2004. Arquitectura de la Información en los entornos virtuales de aprendizaje. Aplicación de la técnica Card Sorting y análisis cuantitativo de los resultados. marzo-abril 2004, Vol. 13, 2, pp. 93-99.

Herramienta CASE. 2003. *Software-Engineering.* [Online] 2003. http://www.cs.queensu.ca/Software-Engineering/toolcat.html.

Hofmann, Hubert. 1993. *"Requirements Engineering".* Zurich : Institute for Informatics –University of Zurich, 1993.

IBM Rational Software. 2007. Rational. [Online] 2007. http://www.rational.com/uml/.

ISO/IEC. 2005. Unified Modeling Language (UML) Version 1.4.2. [Online] 2005. http://www.iso.org/iso/catalogue_detail.htm?csnumber=32620.

Jacobson, Ivar, Booch, Grady and Rumbaugh, James. 2002. *El Proceso Unificado de Desarrollo.* s.l. : Addison Wesley, 2002.

Koutras, Afrati. 1990. A Hypertext Model Supporing Query Mechanisms". Proceeding European Conference on Hypertext Technology. Noviembre 1990, Vol. 1.

Larman, Craig. 1999. *UML y Patrones. Introducción al análisis y diseño orientado a objetos.* s.l. : Prentice Hall, 1999. 970-17-0261-1.

Lash, Jeff. 2002. Information Architecture is not Usability. *Digital Web Magazine.* [Online] Noviembre 2002. http://www.digital-web.com/articles/information_architecture_is_not_usability.

Martínez, Alicia, Estrada, Hugo and Pastor, Oscar. 2002. *El Modelo de Negocios como origen de especificaciones de requisitos de software: una aproximación metodológica.* España : s.n., 2002.

Mendoza Sánchez, Ing.Informático -UNT María A. 2004. Metodologías de Desarrollo de Software. *Informatizate.com.* [Online] Junio 7, 2004. http://www.informatizate.net/articulos/metodologias_de_desarrollo_de_software_07062004.html.

Microsoft Corp. 2009. Microsoft Developer Network. *Microsoft Developer Network.* [En línea] Microsoft Corp., 2009. http://msdn.microsoft.com.

Navarra, Universidad de. 2005. Escuela Superior de Ingenieros San Sebastian. Campus Tecnológico de la Universidad de Navarra. [Online] 2005. http://www.tecnun.es/asignaturas/ingsoft/pagina_4.html..

Norman, Donald A. and Draper, Stephen W. 1986. *User Centered System Design; New Perspectives on Human-Computer Interaction.* 1986. p. 526 . 0898597811.

Obonsawin, M.C., et al. 1999. Performance on the Modified Card Sorting Test by normal, healthy individuals : Relationship to general intellectual ability and demographic variables. 1999, Vol. 38, 1, pp. 27-41.

Pressman, Roger S. 2001. *Ingenieria Del Software: Un Enfoque Practico.* 5ta Edición. s.l. : Mcgraw-hill, 2001. 8448132149.

Ronda León, Rodrigo and Mesa Rábade, Yaima. 2005. Análisis de Secuencia: una herramienta para la Arquitectura de Información. 2005, 4.

Rosenfeld, Louis and Morville, Peter. 1998. *Information Architecture for the World Wide Web.* Cambridge : O'Reilly, 1998.

Rosenfeld, Louis. 2002. Information Architecture: Looking Ahead. Agosto de 2002, Vol. 53, 10, págs. 874-876.

Saiden, H. 1999. *"Requirements Engineering: Making the connection between the software developer and customer".* Nebraska : s.n., 1999.

Sanderson, Steven. 2009. *Pro ASP.NET MVC Framework.* New York : Apress, 2009. 978-1-4302-1008-5.

Sommerville, Ian. 2005. *Ingeniería del Software.* 7ma. s.l. : Prentice Hall, 2005. p. 687. 8478290745.

Tramullas Saz, Jesús. 2002. Arquitectura de la información:más que diseño, hacia la findability. 2002, 39.

UML. 2000. *The Unified Modeling Language.* 2000.

Wagner, William P., Najdawi, Mohammad K. and Chung, Q.B. 2002. Selection of knowledge acquisition techniques based upon the problem domain characteristics of production and operations management expert systems. Diciembre 16, 2002, Vol. 18, 2, pp. 76 - 87.

Wurman, Richard Saul. 1996. *Information Architects.* Zurich : Graphis Press Corp, 1996.

Anexos

Anexo 1: Esquema de un archivo de información de análisis de secuencia.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
	<xs:element name="Analisis">
		<xs:annotation>
			<xs:documentation>Análisis de secuencia</xs:documentation>
		</xs:annotation>
		<xs:complexType>
			<xs:sequence>
				<xs:element name="Nombre">
					<xs:annotation>
						<xs:documentation>Nombre del análsis</xs:documentation>
					</xs:annotation>
					<xs:simpleType>
						<xs:restriction base="xs:string">
							<xs:maxLength value="255"/>
						</xs:restriction>
					</xs:simpleType>
				</xs:element>
				<xs:element name="Tarjetas">
					<xs:annotation>
						<xs:documentation>Listado de tarjetas</xs:documentation>
					</xs:annotation>
					<xs:complexType>
						<xs:sequence>
							<xs:element name="Tarjeta" maxOccurs="unbounded">
								<xs:complexType>
									<xs:sequence>
<xs:element name="Numero" type="xs:short">
<xs:annotation>
	<xs:documentation>Número de la tarjeta</xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="Etiqueda">
<xs:annotation>
	<xs:documentation>Etiqueta asociada</xs:documentation>
</xs:annotation>
<xs:simpleType>
<xs:restriction base="xs:string">
	<xs:maxLength value="100"/>
</xs:restriction>
</xs:simpleType>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name="Usuarios">
<xs:annotation>
```

```
                                        <xs:documentation>Listado de usuarios</xs:documentation>
                                    </xs:annotation>
                                    <xs:complexType>
                                        <xs:sequence>
                                            <xs:element name="Usuario" maxOccurs="unbounded">
                                                <xs:complexType>
                                                    <xs:sequence>
                                                        <xs:element name="Nombre">
                                                            <xs:annotation>

        <xs:documentation>Nombre del usuario</xs:documentation>
                                                            </xs:annotation>
                                                        </xs:element>
                                                        <xs:element name="Apellidos">
                                                            <xs:annotation>

        <xs:documentation>Apellidos del usuario</xs:documentation>
                                                            </xs:annotation>
                                                        </xs:element>
                                                        <xs:element name="Email">
                                                            <xs:annotation>

        <xs:documentation>Correo electrónico</xs:documentation>
                                                            </xs:annotation>
                                                        </xs:element>
                                                    </xs:sequence>
                                                </xs:complexType>
                                            </xs:element>
                                        </xs:sequence>
                                    </xs:complexType>
                                </xs:element>
                            </xs:sequence>
                            <xs:attribute name="Tipo">
                                <xs:annotation>
                                    <xs:documentation>Tipo de análisis (Abierto o Cerrado)</xs:documentation>
                                </xs:annotation>
                                <xs:simpleType>
                                    <xs:restriction base="xs:string">
                                        <xs:enumeration value="Abierto"/>
                                        <xs:enumeration value="Cerrado"/>
                                    </xs:restriction>
                                </xs:simpleType>
                            </xs:attribute>
                        </xs:complexType>
        </xs:element>
</xs:schema>
```

Anexo 2: Matriz de resultados de la validación de requisitos.

	Preg. 1	Preg. 2	Preg. 3	Preg. 4	Preg. 5	Preg. 6	Preg. 7	Preg. 8	Preg. 9
RF1	si	si	si	no	no	si	si	si	si
RF2	si	si	no	no	no	si	si	si	si
RF3	si	si	no	no	no	si	si	si	si
RF4	si	si	no	no	no	si	si	si	si
RF5	si	si	no	no	no	si	si	si	si
RF6	si	si	no	no	no	si	si	si	si
RF7	si	si	no	no	no	si	si	si	si
RF8	si	si	no	no	no	si	si	si	si
RF9	si	si	no	no	no	si	si	si	si
RF10	si	si	no	no	no	si	si	si	si
RF11	si	si	no	no	no	si	si	si	si
RF12	si	si	no	no	no	si	si	si	si
RF13	si	si	no	no	no	si	si	si	si
RF14	si	si	no	no	no	si	si	si	si
RF15	si	si	no	no	no	si	si	si	si
RF16	si	si	no	no	no	si	si	si	si
RF17	si	si	no	no	no	si	si	si	si
RF18	si	si	no	no	no	si	si	si	si
RF19	si	si	no	no	no	si	si	si	si
RF20	si	si	no	no	no	si	si	si	si
RF22	si	si	no	no	no	si	si	si	si
RF23	si	si	no	no	no	si	si	si	si
RF24	si	si	no	no	no	si	si	si	si
RF25	si	si	no	no	no	si	si	si	si
RF26	si	si	no	no	no	si	si	si	si
RF27	si	si	no	no	no	si	si	si	si
RF28	si	si	no	no	no	si	si	si	si