Universidad de las Ciencias Informáticas
Centro de Informatización Universitaria
Facultad 1

Componentes para diagramas de flujo en la herramienta de diagramación del paquete Abad.

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

Autor: Javier Matos Llanes.

Tutores: Ing. Nadiesda Sanz Carmenates
Ing. Sergio René Vazquez Rodriguez.

Ciudad de la Habana, 25 de julio de 201

Frase

La vida es una obra de teatro que no

permite ensayos...

Por eso, canta, ríe, baila, llora

y vive intensamente cada momento

de tu vida...

...antes que el telón baje

y la obra termine sin aplausos

Dedicatoria

Dedico mi tesis:

Especialmente a mi mamá, mi papá, mi hermana y abuelos por haberse sacrificado tanto.

Al resto de mi familia por apoyarme y guiarme por los buenos caminos.

A alguien muy especial en mi vida, por su constancia, apoyo, dedicación y amor; sin ti no hubiera podido lograr tantas cosas buenas.

A mis grandes amigos, los que siempre estuvieron cuando más los necesite.

A una persona que quiero mucho a pesar que ya no se encuentra entre nosotros, a mi abuelita; que no me acostumbro a estar sin ella.

Agradecimientos

Agradezco a mis padres por apoyarme en todo lo que me he propuesto hacer, por guiarme, por tanto sacrificio y por confiar siempre en mí.

A mis abuelas Esperanza y Melba por estar siempre a mi lado y pelearme mucho para que yo entre en razón.

A mis abuelos Tomas y Luis Felipe por apoyarme y enseñarme a enfrentar los problemas.

A todo el resto de mi familia que han brindado afecto, confianza, por haberme enseñado a luchar por lo que quiero y a ser mejor persona.

A mis tutores que han luchado conmigo todo este tiempo, gracias por enseñarme mis prioridades como estudiante, por estar siempre ahí cuando lo necesite, darme apoyo y fuerzas para llegar hasta aquí.

A Yanicet que gracias a sus consejos, a su apoyo y a toda las guerras que me daba en el proyecto me hizo demostró que si se puede lograr lo que te propongas.

A mis amigos de la zona, que son extraordinarios, borrachines y amigos sobre todas las cosas.

A mis hermanos y hermanas de mi vida: a mí tata Mile, a mi Arle, Jessi, Alieski, a Yoha que pocos instantes después de conocernos se ganó un pedacito de mi corazón, a Jose, Rodny, a Tito, a Did, a mi hermanito NEIL, a Belkita, a Vel que tanto me han ayudado y enseñado, a Rick, a Pedro, a Mora, a Betty, a mis hijos Robert y Suyin. Que 6 años de universidad han sido poco para compartir, los quiero mucho a todos y gracias por ser parte de mi familia.

A mis cositas lindas Yanet, Isied, Anisley, Raisa, Karo, Ailin, Chacón, Gleydis, Ari y Yeni.

A Leo, Manu, Roger, Kilmer, Dashiel, Grettel, a los chicos de Cubiamba e Infodaz que han sido mi refugio cuando no tenía a donde ir o quién acudir; Nunca los olvidaré.

Declaración de autoría

Declaro ser el único autor de la presente tesis y reconozco a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo. Para que así conste firmo la presente a los 30 días del mes de julio del año 2012.

_____

Javier Matos Llanes

Firma del autor

_____ _____

Ing. Sergio R. Vazquez Rodriguez Ing. Nadiesda Sanz Carmenates

Firma del tutor Firma del tutor

Datos de contacto

Ing. Nadiesda Sanz Carmenate

Graduada de Ingeniería en Ciencias Informáticas en julio del 2007. Profesor Asistente, 5 años de experiencia en la gestión de proyectos informáticos, Asesora de Gestión de Proyectos del Centro de Informatización Universitaria (CENIA), Facultad 1, Universidad de las Ciencias Informáticas.

Correo: nsanz@uci.cu

Ing. Sergio René Vazquez Rodriguez

Graduado de Ingeniero en Ciencias Informáticas en el curso 2009-2010. Se ha desempeñado como desarrollador y jefe en proyectos relacionados con la gestión de procesos en la Universidad de las Ciencias Informáticas. Actualmente es Jefe de la Línea Soluciones para Ciudad Digital, del centro de Informatización Universitaria (CENIA).

Correo: srvazquez@uci.cu

Resumen

El presente trabajo de diploma se centra en el diagrama de flujo como parte de la técnica de diagramación que aplican los arquitectos de información en su trabajo. El objetivo está orientado a implementar los componentes visuales que se utilizan en este diagrama integrándolos finalmente a una herramienta que permita realizar todos los diagramas de esta técnica (presentación, funcionamiento y organización). En estos momentos las soluciones existentes en la actualidad no realizan estos diagramas con el nivel de acabado necesario y en muchos de los casos estas soluciones son propietarias, obligando al arquitecto de información a utilizar varias herramientas para diagramar. Por esto fue necesario desarrollar una solución, este proceso fue guiado por la metodología ágil de desarrollo SXP híbrido de SCRUM y XP y modelado a través del Lenguaje Unificado de Modelado (UML), haciendo uso de la herramienta Visual Paradigm. Se escogió para su implementación el lenguaje de programación Java y como entorno de desarrollo integrado NetBeans 6.9.

Palabras clave: Arquitectura de información, diagramación, diagramas de funcionamiento y componentes visuales.

Índice de contenidos

Índice de contenidos

Índice de contenidos

Índice de figuras

Índice de tablas

Introducción

El proceso de Arquitectura de Información (AI) es la disciplina encargada del estudio, análisis, organización, disposición y estructuración de los contenidos en espacios de información, a partir de las necesidades y preferencias de la audiencia, con el objetivo de garantizar la calidad final del producto y la plena satisfacción de los usuarios. En el proceso se utilizan diferentes técnicas que ayudan a desarrollar al mismo, entre estas técnicas se encuentra la diagramación, que agrupa y organiza la información deseada por los usuarios para un mejor entendimiento, así como la representación, el funcionamiento básico y la ubicación de estos contenidos en la interfaz (León, 2007).

Para el desarrollar esta técnica se realizan diagramas de organización, flujo y presentación, usados por el arquitecto de información para proponer cómo será el producto final. Específicamente, el diagrama de flujo tiene un nivel de acabado superior al de organización y complementa al mismo. Debe ser el que muestre los niveles de navegación[1] así como los tipos de navegación en el producto, emplean símbolos gráficos para representar o describir la secuencia de los pasos o etapas de un proceso y su interacción. Es una forma esquemática de representar ideas y conceptos en relación. A menudo, se utiliza para especificar algoritmos de manera gráfica. Se conoce como diagramas de flujo a aquellos gráficos representativos que se utilizan para esquematizar conceptos vinculados a la programación, la economía, los procesos técnicos y/o tecnológicos, la psicología, la educación y casi cualquier temática de análisis (León, 2007).

Actualmente existen soluciones informáticas que realizan estos tres tipos de diagramas (organización, flujo y presentación) aunque no los hacen todos, estas soluciones generalmente se distribuyen bajo licencias propietarias o no realizan una correcta interacción entre los contenidos, su principal dificultad consiste en que obligan al arquitecto de información a navegar entre varias herramientas para realizar los diferentes diagramas correspondientes a la técnica de diagramación de la arquitectura de información, lo que hace que este sea más lento y engorroso, impidiendo al arquitecto de información tener una vista global o general de la representación visual de la propuesta a desarrollar.

En el Centro de Informatización Universitaria (CENIA) de la Facultad 1, se encuentra el departamento Universidad Digital, en el que se incluye el proyecto “Paquete de herramientas para diseñar experiencia de

[1] La estructura jerárquica que posee el producto para mejorar la interacción con el usuario.

usuario" (nombrado también "Abad"). Este proyecto busca desarrollar una solución integradora para la gestión de los flujos de experiencia de usuario en los proyectos de desarrollo de soluciones informáticas de la UCI, facilitando el trabajo a aquellos que desempeñan el rol de arquitecto de información en dichos proyectos. La solución en general pretende integrar los diagramas de organización, flujo y presentación de la técnica de diagramación. Dentro de esa solución se encuentra el paquete que recoge la implementación de componentes visuales[2] correspondientes solamente al diagrama de flujo en el que se enmarca esencialmente este trabajo.

Teniendo en cuenta la problemática anterior se plantea como **problema a resolver*:*** ¿Cómo mejorar la representación visual, que realiza el arquitecto de información, de la interacción entre contenidos de un producto determinado?

De ahí que se defina como **objeto de estudio:** el proceso de arquitectura de la información, centrándose en el **campo de acción:** en la diagramación dentro del proceso de la arquitectura de información**.**

Respondiendo la interrogante anterior se define como **objetivo general:** Implementar los componentes visuales para los diagramas de flujo en la herramienta de diagramación del paquete Abad, contribuyendo a mejorar la representación visual que realiza el arquitecto de información de la interacción entre contenidos de un producto.

Como **objetivos específicos** se definen:

- Analizar los aspectos teóricos-conceptuales relacionados con los diagramas de flujo de contenido dentro de la arquitectura de información.
- Implementar los componentes para diagramas de flujo de contenidos en la herramienta de diagramación.
- Probar el correcto funcionamiento del sistema de acuerdo a los requisitos planteados.

Con el presente trabajo se espera como **posible resultado** la incorporación de diversos componentes para diagramas de flujo, que permite al arquitecto realizar una representación del flujo entre contenido más completa, también favorece a la compresión del proceso al mostrarlo como un dibujo, permite

[2] Los elementos que forman un vocabulario visual, para realizar los diagramas que describen la arquitectura de información.

identificar los problemas y mejoras del proceso representado, brinda la posibilidad de integrar el diagrama de flujo a la herramienta general, y así la universidad contará con un producto capaz de constituir todos los diagramas de la técnica de diagramación de la AI.

Métodos teóricos utilizados:

En el presente trabajo de diploma se utilizaron métodos científicos que se analizan y describen a continuación:

Histórico –lógico:

Este método se utiliza para el análisis de la evolución de la técnica de diagramación. Para establecer una línea lógica en la investigación sobre su desarrollo y funcionamiento, para así tener un correcto orden en los epígrafes que componen este trabajo.

Analítico – sintético

La utilización de este método posibilitó extraer e identificar conceptos, características y otros elementos de la abundante bibliografía consultada que posteriormente ayudaron a establecer una propuesta adecuada a las necesidades de implementación del sistema. Después del análisis de esa información fue necesario organizarla y sintetizarla para la elaboración del presente informe además que permitió realizar un análisis crítico de la propuesta realizada por el analista del sistema.

Introducción

Este trabajo está compuesto por tres capítulos:

Capítulo 1. Base teórica de la investigación y estudio de homólogos: Este capítulo incluye el basamento teórico en el que se insertan los diagramas de flujo, incluyendo el tema de los alfabetos visuales. Además de un estudio del estado del arte de soluciones homólogas, para lograr una solución que cumpla con las necesidades y preferencias del cliente. También la fundamentación teórica de la necesidad de realizar la automatización de los procesos de diagramación para la plataforma de arquitectura de información del paquete Abad.

Capítulo 2. Entorno de desarrollo y valoración del análisis: En este capítulo recoge una valoración crítica del análisis en cuanto a la definición de requisitos, entorno de desarrollo y arquitectura, propuestos por el analista.

Capítulo 3. Descripción de la implementación y estrategia de pruebas: Incluye los estándares de codificación usados para la implementación de la solución, así como los modelos de diseño y de implementación. También es descrita la solución en términos de componentes de software y se describen las pruebas ejecutadas a la aplicación.

Capítulo 1: Base teórica de la investigación y estudio de homólogos

1.1 Introducción al capítulo: Base teórica de la investigación y estudio de homólogos

En este capítulo se exponen los elementos generales sobre la técnica de diagramación, centrando el estudio en los diagramas de flujo pertenecientes a la arquitectura de Información, y el comportamiento de cada uno de los componentes visuales que conforman este tipo de diagrama. Se presenta el estudio de las herramientas homólogas. Además, se explica y valora la metodología usada y el lenguaje de programación empleado para el desarrollo del producto.

1.2 Arquitectura de información:

La arquitectura de la información (AI) es la disciplina encargada del estudio, análisis, organización, disposición y estructuración de los contenidos en espacios de información (León, 2007). En la actualidad es muy utilizada en el diseño de las interfaces para sitios web, aunque en sus inicios fue empleada en otros escenarios. En general permite una fácil organización y representación de la información logrando mejorar su búsqueda y recuperación, influyendo en la calidad y usabilidad del producto, la experiencia de usuario[3] y otros factores según el entorno.

En la AI se trata de organizar la información de forma tal que los usuarios puedan encontrar las respuestas correctas a sus interrogantes. Su objetivo radica en la creación de sistemas de organización de la información que designan o describen una entidad que realmente posean un significado para los usuarios (Bustamante, 2004).

1.3 Diagramación:

La representación se ha usado desde los comienzos del diseño de software, en forma de organigramas, diagramas de flujo de datos, árboles de decisión, etc. Al evolucionar las interfaces gráficas de usuario, las labores de representación se ampliaron con los llamados guiones de navegación y guiones de interacción, los cuales consistían en diagramas que representaban el funcionamiento de los productos electrónicos que se generaban en ese momento.

La diagramación es la técnica de la AI encargada esencialmente de la organización de los contenidos del producto, al funcionamiento básico del mismo, y la ubicación que tendrán estos contenidos en la interfaz (León, 2007). En ella se realizan los diferentes diagramas como son los de organización, flujo y

[3] La sensación, sentimiento, respuesta emocional, valoración y satisfacción del usuario respecto a un producto, resultado del fenómeno de interacción con el producto y la interacción con su proveedor. Yusef, H.M. La Experiencia del Usuario. 2005.

presentación de la AI. La diagramación, a la cual se refiere este trabajo, consiste en la representación de los contenidos que tendrá un producto digital, y las relaciones entre dichos contenidos.

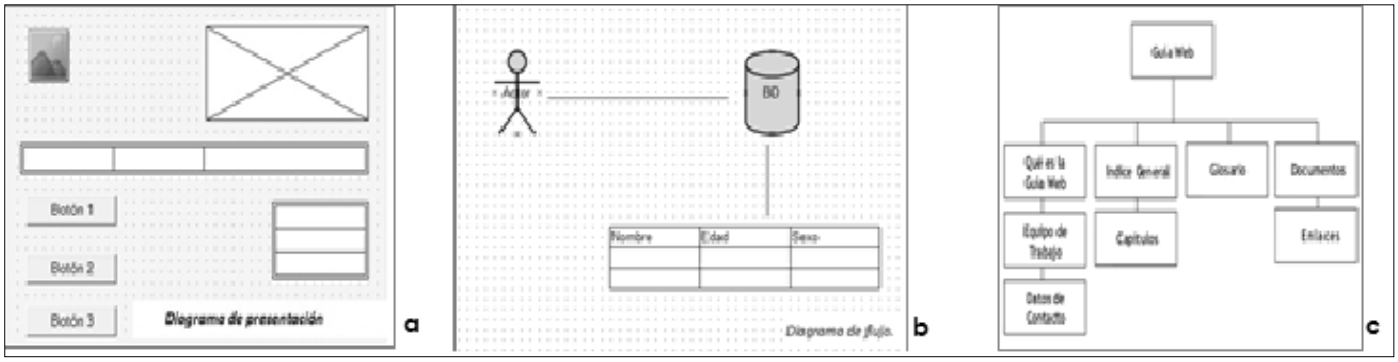


Fig. 1: Diagramas en la técnica de diagramación. a) Presentación, b) Flujo, c) Organización(León, 2007)

1.4 Diagramas de flujo:

Los diagramas de flujo son la representación de las estructuras con los flujos de navegación. Este diagrama tiene un nivel de acabado superior al anterior y complementa al mismo. Debe ser el que muestre los niveles de navegación así como los tipos de navegación en el producto (León, 2007).

Los diagramas de flujo son múltiples y diversos y pueden abordar muchos temas distintos de formas también muy diferentes. En cualquier caso, el aspecto en común entre ellos es la presencia de un vínculo entre los conceptos enunciados y una interrelación entre las ideas. Se vale de distintos símbolos para representar la trayectoria de operaciones precisas a través de flechas. Siempre que existe un diagrama de flujo existe un proceso o sistema que pretende ser graficado a través de símbolos visuales que, en vez de términos verbales, simplifican el funcionamiento de dicho proceso y lo hacen más claro y evidente al lector (Definicion ABC, 2007).

Los diagramas de flujo pueden clasificarse mediante formatos; en el formato horizontal, el proceso se desarrolla desde la izquierda hacia la derecha, mientras que en el diagrama de formato vertical, el esquema se dirige de arriba hacia abajo, formando una serie ordenada de las fases del proceso. Por otro lado, está el diagrama de flujo de formato panorámico, que como su nombre lo indica, brinda una visión general y clara del proceso; éste tiene la particularidad de desplazarse en forma vertical y horizontal (Sobre conceptos, 2009).

Ventajas de los diagramas de flujo

- Favorecen la comprensión del proceso a través de mostrarlo como un dibujo. El cerebro humano reconoce fácilmente los dibujos. Un buen diagrama de flujo reemplaza varias páginas de texto.

- Permiten identificar los problemas y las oportunidades de mejora del proceso. Se identifican los pasos redundantes, los flujos de los re-procesos, los conflictos de autoridad, las responsabilidades y los puntos de decisión.

Los diagramas de flujo son una herramienta valiosa para la mejora de los procesos, permiten detectar las actividades que agregan valor y aquellas que son redundantes o innecesarias. También son de gran utilidad durante el desarrollo de la documentación de los sistemas de gestión, pues proveen una descripción de los procesos y un detalle de las operaciones mucho más amigable que los procedimientos e instructivos basados en texto. Contribuyen a resolver uno de los principales problemas, que es la resistencia del personal a emplear los documentos como referentes para el desempeño de las tareas. Una copia ampliada del diagrama de flujo al alcance de los operadores del proceso facilita la consulta y promueve la creatividad. Es conveniente emplear programas específicos para la confección de los diagramas de flujo. En general, estos programas son de manejo sencillo y facilitan notablemente la tarea (Vazquez, 2009).
Este trabajo se centra en la definición de componentes a incorporar en los diagramas de flujo, por lo que se hace necesario investigar sistemas capaces de realizar diagramas de flujo.

1.5 Alfabetos visuales

Para cada tipo de diagrama Rodrigo Ronda León propone varios componentes (León, 2007).

- Diagrama de funcionamiento y presentación: iconos más trabajados visualmente, con el objetivo de representar el comportamiento interactivo del producto.
- Diagrama de organización: iconos simples, cajas, flechas y conectores.

Una notación muy usada por arquitectos de información y diseñadores de interacción para hacer la diagramación de sitios web es la propuesta por Jesse James Garrett y que consiste, según el propio autor, en un "vocabulario visual para describir arquitectura de información y diseño de interacción" (Garrett, enero 2001). El sistema de diagramación está compuesto de símbolos geométricos, flechas y líneas.

El vocabulario visual de Garrett es muy útil para representar tanto el diseño de interacción, como la estructura conceptual y organizativa del contenido. Esta notación gráfica está concebida para realizar un diseño de lo general a lo particular, pues sigue el principio de la simplificación de representación a partir de cajas y flechas. Este principio es el que facilita a cualquier diseñador comunicar arquitecturas de información de forma fácilmente comprensible (Garrett, enero 2001).

Otra propuesta bastante diversa y completa fue la de Nick Finck también. El vocabulario visual propuesto por Nick para los diagramas de flujo utilizado para explicar y definir un proceso dentro de un sitio web, una aplicación web, el software o sistema pues este es bastante amplio. Se construye de cajas que representan las pantallas o las acciones, los diamantes que representan los puntos de decisión en el proceso, los usuarios que representan a los actores individuales. En algunos casos también hay cilindros que representan las bases de datos o fuentes de información, las regiones que representan a las agrupaciones (Ver Anexo 1).

El vocabulario visual utilizado en esta solución es una unión de lo propuesto por Nick Finck y Garrett, y así tener una serie de componentes visuales diversos y los más útiles para realizar un completo y correcto diagrama de flujo dentro del proceso de AI.

1.6 Soluciones homólogas existentes

1.6.1 Axure RP Pro 5.5

Es una de las herramientas de prototipado más usadas por los arquitectos de información. Se trata de una solución de escritorio en idioma inglés y de pago. Axure RP permite a los diseñadores de aplicaciones crear diagramas de flujo, prototipos y especificaciones para las aplicaciones y sitios web más rápido. Utilizado por profesionales como analistas de negocios, expertos de usabilidad para recopilar y presentar la funcionalidad, navegación y contenido de un proyecto. Permite adjuntar anotaciones o notas a elementos o componentes en el modelo de alambres[4], eliminar las suposiciones y ayudar a esclarecer las funciones específicas.

[4] Incluyen líneas, arcos, curvas, puntos, texto, dimensiones etc. Estos pueden ser de 2D o 3D y son esenciales para el dibujo y además para crear la mayor parte de las superficies y sólidos.

Axure permite realizar diagramas de funcionamiento o flujo, para esto posee 15 componentes la mayoría figuras geométricas, cada una con propiedades específicas para su comportamiento en el diagrama de flujo. Las propiedades de cada componente pueden ser editadas dándoles doble clic al mismo después de ponerlas en el área de trabajo (Solutions, 2002-2011b).

Posee un componente que da la opción de cargar una imagen en la posición que se encuentre este en el diagrama. Esta herramienta utiliza componentes ya preparados, incluyendo formas y elementos dinámicos. A los cuales se pueden editar y darles formato en un ambiente familiar.

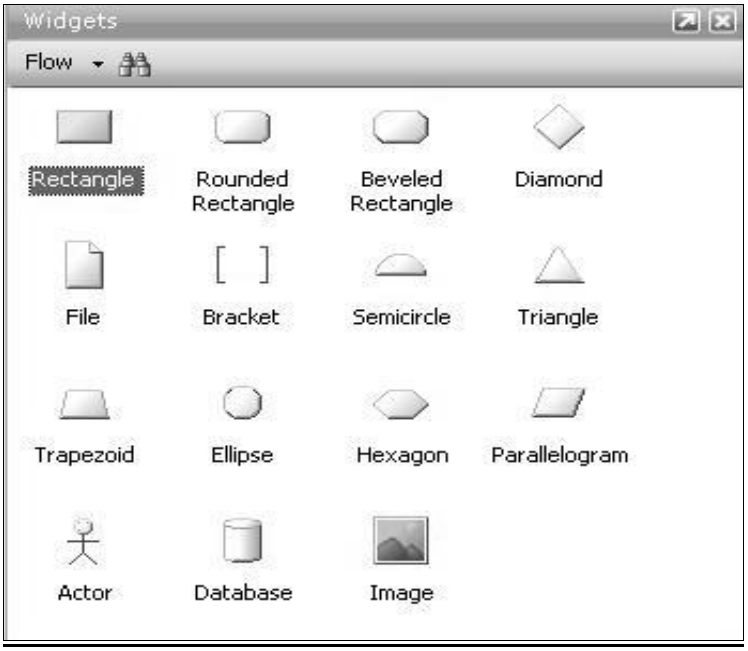


Fig. 2: Componentes visuales de flujo de la herramienta Axure RP (Solutions, 2002-2011a).

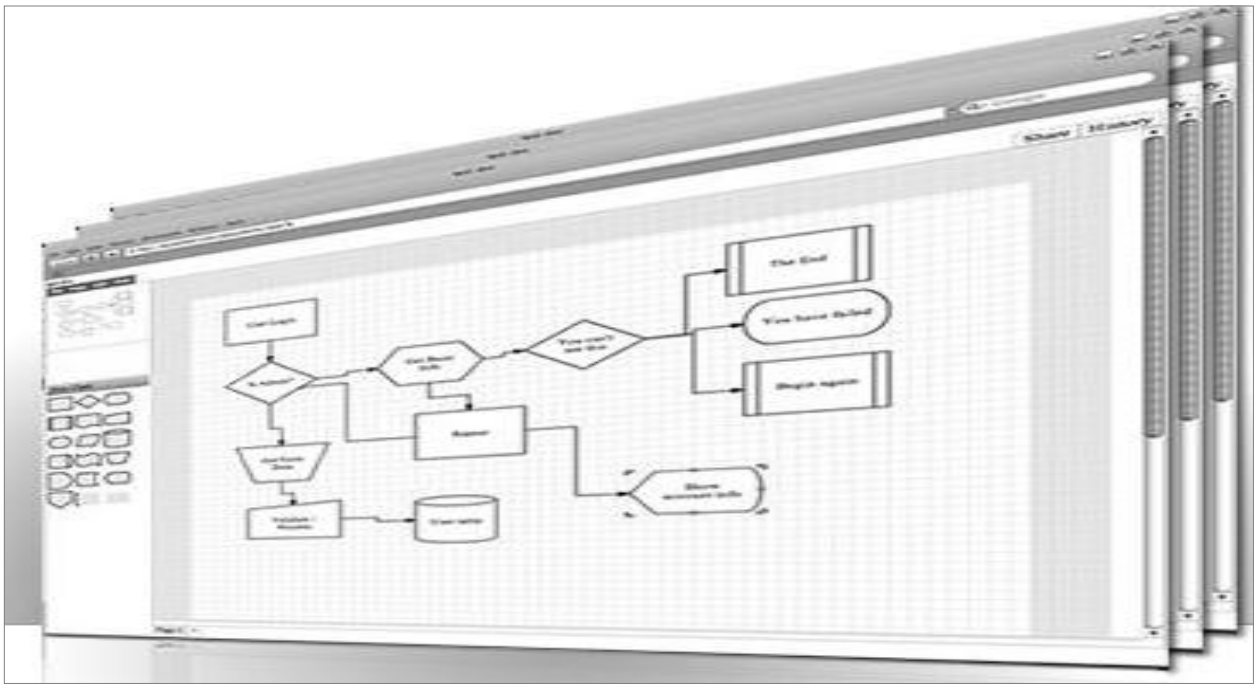


Fig. 3: Ejemplo de diagrama de flujo en Axure RP (Solutions, 2002-2011a).

1.6.2 Bizagi Process Modeler 1.1

Es una aplicación gratuita, que permite representar de forma esquemática todas las actividades y decisiones que se toman en el negocio. Con una interfaz que recuerda al Microsoft Office y cumple con el estándar de BPMN (*Bussiness Process Management Notation*). BPMN es una notación gráfica que describe la lógica de los pasos de un proceso de Negocio. Esta notación ha sido especialmente diseñada para coordinar la secuencia de los procesos y los mensajes que fluyen entre los participantes de las diferentes actividades. Este proporciona un lenguaje común para que las partes involucradas puedan comunicar los procesos de forma clara, completa y eficiente (Bizagi, 2011).

Objetos de Flujo:

Son los principales elementos gráficos que definen el comportamiento de los procesos de interacción y tratamiento de información de una aplicación. Dentro de los objetos de Flujo se encuentran:

• **Eventos**: Son algo que sucede durante el curso de un proceso de negocio, afectan el flujo del proceso y usualmente tienen una causa y un resultado. Dentro de estos ejemplos se utiliza inicio, fin y temporizador, estos elementos son eventos y a su vez se encuentran clasificados en 3 tipos.

Fig. 4: Componentes que representan eventos a) Eventos de Inicio b) Eventos Intermedios c) Eventos de Fin

Actividades: Estas representan el trabajo que es ejecutado dentro de un proceso de negocio. Las actividades pueden ser compuestas o no, por lo que dentro de los ejemplos que se utilizan están los dos tipos de actividades existentes:

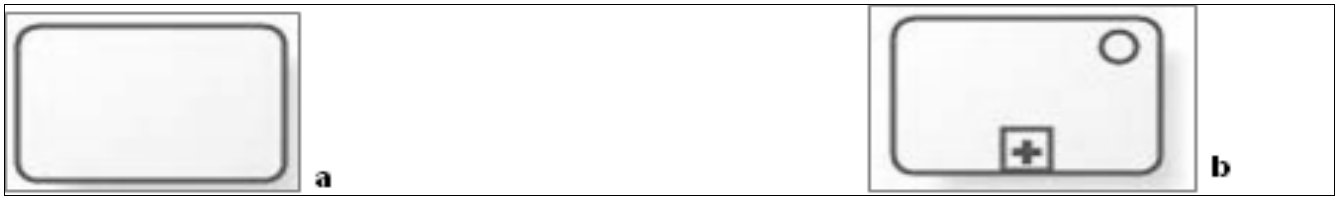


Fig. 5: Componentes representativos de a) Tareas b) Subprocesos

Existen diferentes tipos de tareas (Simple, automáticas, manuales, de usuario, entre otras) y de subprocesos (embebido, reusable, etc.) que permiten diagramar con más profundidad los procesos suministrando más información y claridad al lector (Rowman, 2009).

Compuertas:

Son elementos del modelado que se utilizan para controlar la divergencia y la convergencia del flujo. Existen varios tipos de compuertas.

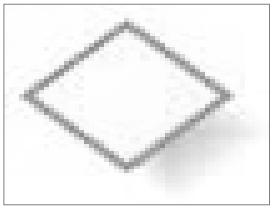


Fig. 6: Componente que representa una compuerta en Bizagi.

Los 5 tipos de compuertas son:

- Compuerta exclusiva,
- compuerta basada en eventos,
- compuerta paralela,
- compuerta inclusiva y
- compuerta compleja.

Objetos de Conexión:

Son los elementos usados para conectar dos objetos del flujo dentro de un proceso.

Existen 3 tipos de objetos de conexión:

- Líneas de secuencia,
- asociaciones y
- líneas de mensaje.

Artefactos:

Los artefactos son usados para proveer información adicional sobre el proceso. Existen 3 tipos de artefactos:

- Objetos de datos,
- grupos y
- anotaciones.

Con el Modelador Bizagi, podrás hacer diagramas y documentar tus procesos de la manera más eficiente, rápida, fácil y buscando fomentar la colaboración en tu organización.

- Soporta una gran cantidad de patrones de proceso complejos y eventos dinámicos, que garantizan que los procesos del cliente puedan ser modelados y automatizados.
- Bizagi promueve el uso de estándares tales como BPMN (*Business Process Modeling Notation) (Rowman, 2009).*

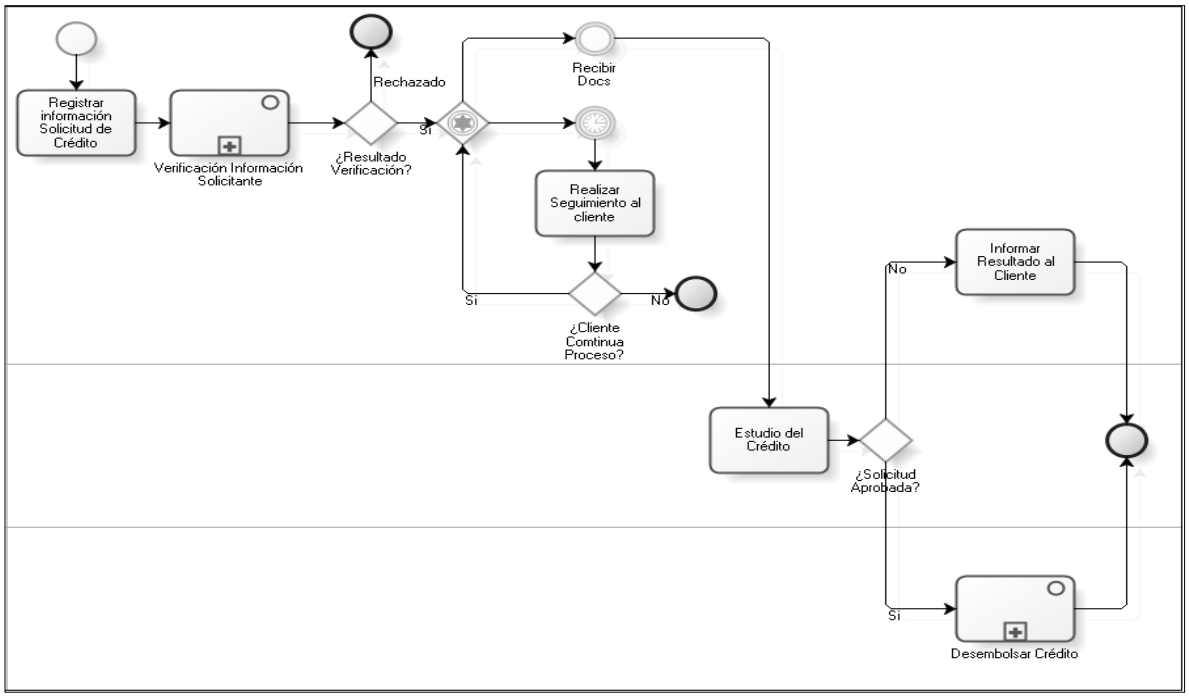


Fig. 7: Ejemplo de diagrama proceso en Bizagi (Bizagi, 2011)

1.6.3 Microsoft Office Visio 2007

Es un programa informático exclusivamente para Windows. Permite trabajar con una interfaz en inglés o en español. Ofrece una biblioteca estándar formada por más de cincuenta elementos gráficos para el diseño del prototipado web, que incluye varias plantillas para diagramas de flujo que puede utilizar con el fin de ilustrar procesos empresariales concretos. Existen varios tipos de diagramas de flujo (Microsoft, 2011).

1. Diagrama de flujo básico: Crea diagramas de flujo, descendentes, de seguimiento de información, de diseño de procesos y de predicción de estructuras.
2. Diagrama de flujo de funciones cruzadas: Crea diagramas que ilustran las relaciones existentes entre los procesos y los departamentos de una organización.
3. Diagrama de flujo de datos: Se utiliza para modelos orientados a procesos o datos, diagramas de flujos de datos, de proceso de datos, de análisis estructurado y de flujo de información.

Permite agregar datos de interés a las formas. Por defecto, los campos son coste, duración y recursos (número de personas). Pueden ser útiles para una toma de decisiones, aunque únicamente pueden verse en el documento a través del software, nunca en versión impresa, por lo que su aplicación no parece pertinente para la investigación. Entre otras funcionalidades, permite agrupar todas las formas del diagrama, de manera que se tiene un todo compacto. Esto puede ayudar, por ejemplo, para copiar el diagrama completo en otro documento. Indican cada fase o ítem del proceso. Estas son las formas más frecuentes en los diagramas de flujo, por ser las que más fácilmente pueden adaptarse a los procesos y procedimientos propios de un entorno administrativo. Además de estas formas precisas, pueden utilizarse las anotaciones, para aquellos elementos del proceso que no puedan expresarse con las anteriores (Microsoft, 2011).

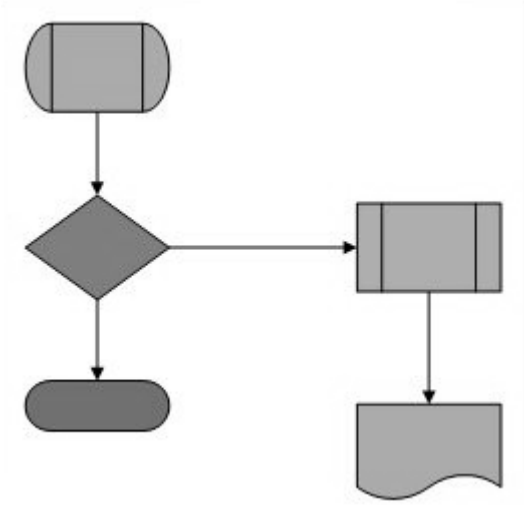


Fig. 8: Diagrama flujo en Visio (Microsoft, 2011)

1.6.4 PencilProject 1.3

Es una herramienta de código libre, con la que se puede desarrollar interfaces de usuarios de forma bastante completa. Esta solamente en Inglés. De por sí dispone de una colección de diseños y gráficos que se puede utilizar para crear interfaces, pero en caso de desearlo también hacer uso de las colecciones de Pencil. Para ventaja, las colecciones de Pencil se pueden descargar gratis, o mejor aún, se puede utilizar archivos SVG, ya que este formato es soportado como formato vectorial, es decir que las imágenes tienen una mejor calidad (Project, 2010).

Algunas de las colecciones para descargar incluyen: Diagramas de flujo

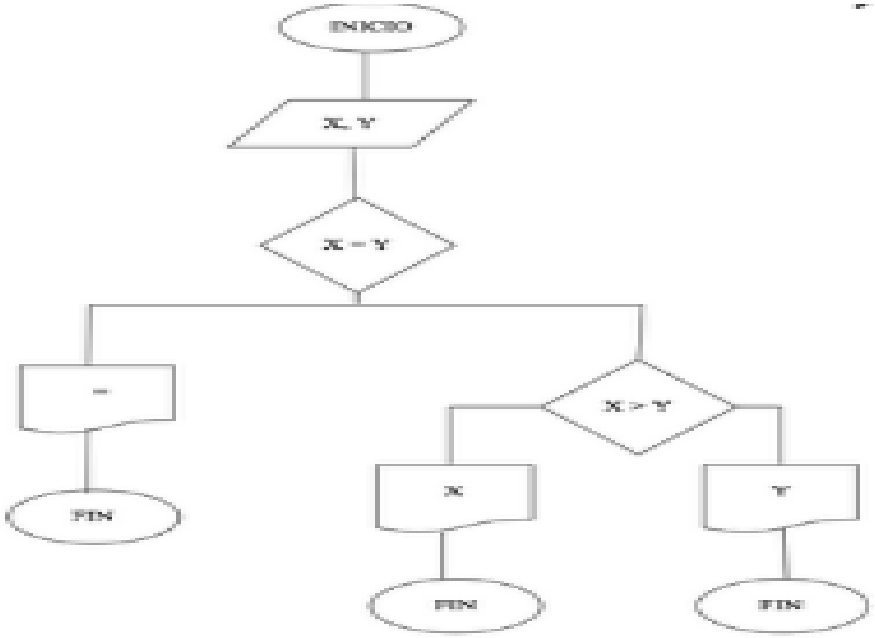


Fig. 9: Diagrama de flujo en Pencil

1.6.5 Visual Paradigm 5.0

Es una herramienta UML profesional que soporta el ciclo de vida completo del desarrollo de software: análisis y diseño orientados a objetos, construcción, pruebas y despliegue. Permite dibujar todos los tipos de diagramas de clases, código inverso, generar código desde diagramas y generar documentación. También proporciona abundantes tutoriales de UML, demostraciones interactivas y proyectos de este lenguaje de modelado. Presenta licencia gratuita y comercial. Es fácil de instalar y actualizar y compatible entre ediciones. Plataforma Java (Windows/Linux/Mac OS X). (Paradigm, 2011a)

Diagrama de flujo de datos

Diagrama de flujo de datos (DFD) presenta el flujo de datos. DFD es un diagrama de ingeniería de software clásico para el diseño estructural. Hay varios niveles de DFD. DFD de nivel 0 es el diagrama de contexto que muestra cómo el sistema interactúa con entidades externas. DFD de nivel 1 se muestran algunos detalles del sistema. DFD no es una notación popular después de que la industria del software fue cambiando de enfoque la estructura de orientación a objetos. Son más y más los analistas de negocio que

hacen uso del DFD para modelar los datos y flujo de documentos de la organización de modelado. Un diagrama de flujo de datos típico se muestra en la siguiente captura de pantalla. (Paradigm, 2011a)

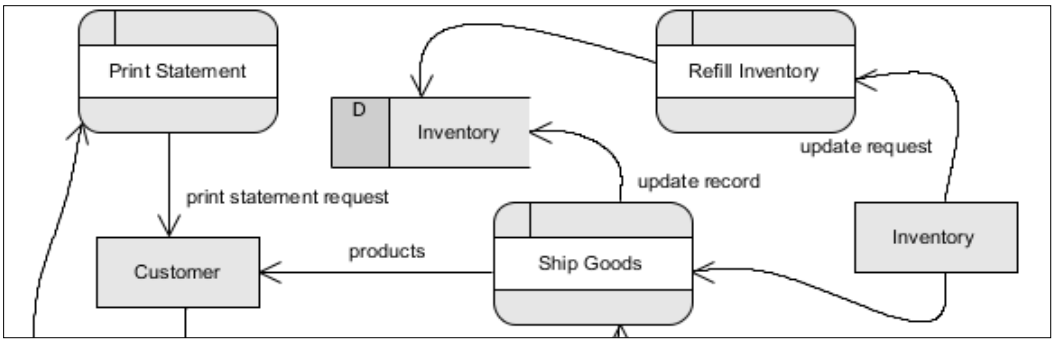


Fig. 10: Ejemplo de diagrama de flujo en Visual Paradigm (Paradigm, 2011b)

1.6.6 Dia v0.96

Dia es una aplicación para la creación de diagramas técnicos. Su interfaz y las funciones son más o menos semejantes al modelo del programa de Windows Visio. Dentro de sus características se incluyen varias páginas de impresión, exportación a muchos formatos (EPS, SVG, CGM y PNG), y la posibilidad de usar formas personalizadas creadas por el usuario como simples descripciones XML. Dia es útil para dibujar diagramas UML, mapas de la red, y diagramas de flujo (Dia, 2010).

Un diagrama se compone de objetos. Los objetos son formas o líneas, que pueden ser de diferentes colores y tamaños. Los objetos pueden ser simples dibujos de línea, texto o imágenes a todo color. Algunos de los objetos permiten escribir texto dentro de la forma.

Dia incluye un conjunto de forma estándar y objetos de línea. Incluye un gran número de objetos predefinidos para distintos usos. Estos incluyen diagramas de flujo, diagramas UML, diagramas de red, y muchos otros. Muchas tareas comunes en Dia requieren la selección de uno o más objetos. Dia incluye una serie de formas de seleccionar objetos con rapidez (Dia, 2010).

Fig. 11: Componentes de flujo de la herramienta Dia

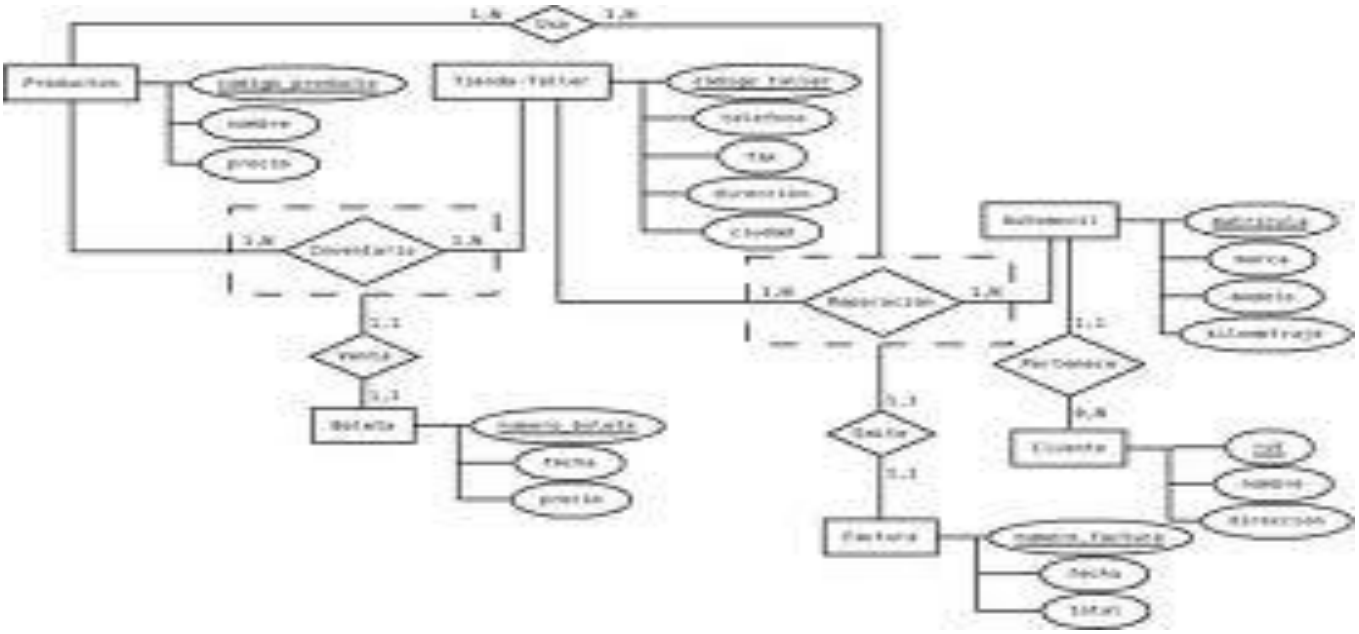
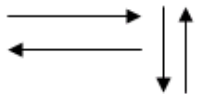
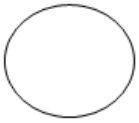
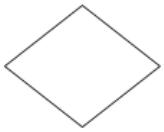
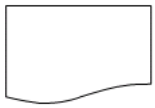


Fig. 12: Ejemplo de diagrama de flujo en Dia (Dia, 2010)

Capítulo 1: Base teórica de la investigación y estudio de homólogos

De las diferentes herramientas estudiadas se determina un grupo de componentes visuales definidos como los más comunes y utilizados, los que se incluyen en la tabla siguiente:

Estereotipos de componentes visuales	**Nombre**
	Línea de flujos (Conexiones de pasos o flechas)
	Terminador (Comienzo o final del proceso)
	Proceso
	Conector
	Datos. Entrada/Salida
	Decisión (Decisión o bifurcación)
	Documento
	Actor

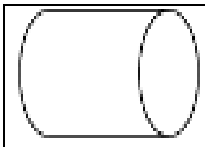
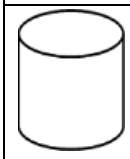
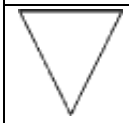
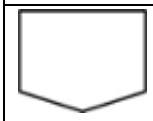
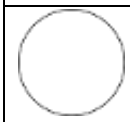
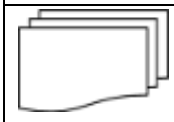
	Almacenamiento de acceso directo
	Almacenamiento interno
	Base de datos
	Tarjeta
	Combinar
	Conector fuera de página
	Conector
	Datos almacenados
	Entrada manual
	Multidocumentos

Tabla 1: Componentes más comunes de los diagramas de flujo

Conclusiones del capítulo 1

En este capítulo se arriban a las siguientes conclusiones:

- El marco conceptual ayuda a desglosar los conceptos principales establecidos para la investigación, determinando los elementos que caracterizan los diagramas de flujo dentro de la AI.
- Luego del análisis del estado del arte se determina la no posibilidad de utilizar para el desarrollo las herramientas estudiadas, pues ninguna gestiona los tres tipos de diagramas definidos en la concepción de la herramienta de diagramación en general.
- Las herramientas homólogas estudiadas ayudan a determinar un grupo de componentes visuales definidos como los más comunes y utilizados, los que se deben incluirse en la implementación de la propuesta de este trabajo de diploma.

Capítulo 2: Entorno de desarrollo y valoración del análisis

2.1 Introducción al capítulo: Entorno de desarrollo y valoración del análisis

A lo largo de este capítulo se realiza un análisis crítico y valorativo del análisis y diseño realizado por el analista del sistema, revisando cada uno de los requisitos entregado y la selección de componentes visuales realizados por este. Se hace un examen de las tecnologías y herramientas a usar, así como la descripción de los macos de trabajo usados para el desarrollo de la solución. Se lleva a cabo una valoración crítica de la arquitectura definida por el analista para llevar a cabo la implementación del sistema.

2.2 Valoración del análisis entregado

Al obtener los resultados del análisis y diseño, se procede con la fase de implementación de la aplicación, para ello se estudia primeramente el diseño elaborado por el analista. Se verifica que lo expuesto en estas descripciones de las funcionalidades sea posible de implementar. Mediante la descripción detallada de las historias de usuarios, se puede establecer una estrategia de trabajo para guiar la implementación de la aplicación, teniendo paso por paso lo que esta debe hacer. Durante este proceso fueron encontradas incongruencias que se corrigieron para obtener un sistema mejor elaborado y de esta forma aumentar el nivel de aceptación de los usuarios.

Luego de este análisis se concluye que:

- ✓ Agregar 10 componentes visuales más.
- ✓ Agregar más propiedades a los componentes visuales. Pues esto ayuda a que el usuario pueda usarlos con mayor facilidad.
- ✓ Redefinir el proceso de guardado de un fichero, haciéndolo en un proyecto por separado (guardar página a página cada proyecto y luego entonces como un proyecto general)

En base a los cambios propuestos, se modificaron e incluyeron un grupo de requisitos. La diferencia entre los RF iniciales y los propuestos luego del análisis crítico se describen en la tabla 2, y estos últimos se detallan en la tabla 3.

	Muy alta	Alta	Media	Baja	Total
RF actuales	**7**	**26**	**33**	**4**	**70**
RF propuestos	5	18	30	4	57

Tabla 2: Resultado de la valoración de la propuesta del analista.

RFAD1	Crear árbol de proyecto.
RFAD2	Presentar el árbol de proyecto.
RFAD3	Adicionar página al árbol de proyecto.
RFAD4	Adicionar páginas hijas a cada página del árbol de proyecto.
RFAD5	Adicionar página después de una existente.
RFAD6	Mover página arriba y abajo.
RFAD7	Mover página dentro y afuera.
RFAD8	Eliminar página del árbol.
RFAD9	Renombrar página del árbol.
RFAD10	Permitir desde una página del árbol de proyecto ir al área de trabajo.
RFAD11	Crear un área de trabajo.
RFAD12	Cerrar área de trabajo.
RFAD13	Insertar componentes al área de trabajo.
RFAD14	Eliminar componentes del área de trabajo.
RFAD15	Modificar componentes del área de trabajo.
RFAD16	Modificar el alto de los componentes del área de trabajo.
RFAD17	Modificar el ancho de los componentes del área de trabajo.
RFAD18	Insertar conector entre los componentes en el área de trabajo.
RFAD19	Eliminar conector entre los componentes en el área de trabajo.

RFAD20	Agrupar componentes en el área de trabajo.
RFAD21	Desagrupar componentes en el área de trabajo.
RFAD22	Agregar barra de propiedades de cada componente.
RFAD23	Presentar todos los componentes de los diagramas.
RFAD24	Presentar las propiedades de componentes con la opción de clic derecho.
RFAD25	Representar componentes para el diagrama de funcionamiento.
RFAD26	Representar un componente que representa un Actor.
RFAD27	Representar un componente que representa un almacenamiento de acceso directo.
RFAD28	Representar un componente que representa un almacenamiento interno.
RFAD29	Representar un componente que representa una base de datos.
RFAD30	Representar un componente que representa una cinta perforada.
RFAD31	Representar un componente que representa un combinar.
RFAD32	Representar un componente que representa un círculo.
RFAD33	Representar un componente que representa un conector fuera de página.
RFAD34	Representar un componente que representa los datos almacenados.
RFAD35	Representar un componente que representa los datos.
RFAD36	Representar un componente que representa una decisión.
RFAD37	Representar un componente que representa un documento.
RFAD38	Representar un componente que representa la entrada manual.
RFAD39	Representar un componente que representa el evento de inicio.
RFAD40	Representar un componente que representa el evento intermedio.
RFAD41	Representar un componente que representa el evento de fin.
RFAD42	Representar un componente que representa un evento intercalar.
RFAD43	Representar un componente que representa multidocumentos.
RFAD44	Representar un componente que representa un disco magnético.

RFAD45	Representar un componente que representa las notas.
RFAD46	Representar un componente que representa un O lógico.
RFAD47	Representar un componente que representa un Y lógico.
RFAD48	Representar un componente que representa una operación manual.
RFAD49	Representar un componente que representa un ordenar.
RFAD50	Representar un componente que representa una pantalla.
RFAD51	Representar un componente que representa una preparación.
RFAD52	Representar un componente que representa un proceso alternativo.
RFAD53	Representar un componente que representa un proceso definido.
RFAD54	Representar un componente que representa un proceso.
RFAD55	Representar un componente que representa un retraso.
RFAD56	Representar un componente que representa una tarjeta.
RFAD57	Representar un componente que representa un fin/inicio.
RFAD58	Representar un componente que representa un conector con texto.
RFAD59	Permitir sobre los componentes con la opción de clic derecho Cortar, Copiar y Pegar.
RFAD60	Permitir sobre los componentes con la opción de clic derecho el agrupamiento, agrupar y desagrupar.
RFAD61	Permitir sobre los componentes la opción de ordenar delante y detrás.
RFAD62	Permitir sobre los componentes la opción alineación, alinear al centro, derecha y a la izquierda desde la barra de propiedades.
RFAD63	Permitir por cada página del árbol de proyecto tener una pestaña en el área de trabajo.
RFAD64	Mostrar el tipo de componentes pasando el ratón por el mismo en el área de trabajo.
RFAD65	Permitir desde la barra de herramienta crear proyecto, abrir proyecto y guardar proyecto.
RFAD66	Permitir desde la barra de herramienta copiar elemento, pegar elemento y cortar elemento.
RFAD67	Permitir desde la barra de propiedades modificar el color de relleno de los componentes.

RFAD68	Permitir desde la barra de propiedades modificar el color de la fuente de los componentes.
RFAD69	Permitir desde la barra de propiedades establecer la visibilidad del componente.
RFAD70	Permitir desde la barra de propiedades cambiar el nombre del componente.
RFAD50	Representar un componente que representa una pantalla.
RFAD51	Representar un componente que representa una preparación.
RFAD52	Representar un componente que representa un proceso alternativo.
RFAD53	Representar un componente que representa un proceso definido.
RFAD54	Representar un componente que representa un proceso.
RFAD55	Representar un componente que representa un retraso.
RFAD56	Representar un componente que representa una tarjeta.
RFAD57	Representar un componente que representa un fin/inicio.
RFAD58	Representar un componente que representa un conector con texto.
RFAD59	Permitir sobre los componentes con la opción de clic derecho Cortar, Copiar y Pegar.
RFAD60	Permitir sobre los componentes con la opción de clic derecho el agrupamiento, agrupar y desagrupar.
RFAD61	Permitir sobre los componentes la opción de ordenar delante y detrás.
RFAD62	Permitir sobre los componentes la opción alineación, alinear al centro, derecha y a la izquierda desde la barra de propiedades.
RFAD63	Permitir por cada página del árbol de proyecto tener una pestaña en el área de trabajo.
RFAD64	Mostrar el tipo de componentes pasando el ratón por el mismo en el área de trabajo.
RFAD65	Permitir desde la barra de herramienta crear proyecto, abrir proyecto y guardar proyecto.
RFAD66	Permitir desde la barra de herramienta copiar elemento, pegar elemento y cortar elemento.
RFAD67	Permitir desde la barra de propiedades modificar el color de relleno de los componentes.
RFAD68	Permitir desde la barra de propiedades modificar el color de la fuente de los componentes.
RFAD69	Permitir desde la barra de propiedades establecer la visibilidad del componente.

RFAD70	Permitir desde la barra de propiedades cambiar el nombre del componente.

Tabla 3: Requisitos a desarrollar

Aunque se concuerda con los componentes visuales determinados por el analista (Ver anexo 2), se incorporan los componentes visuales identificados en el estudio de homólogos (Ver tabla 1), que facilitan la interacción del usuario con la aplicación.

2.3 Valoración del entorno de desarrollo

2.3.1 Metodologías de desarrollo de software

El proceso de desarrollo de software es el conjunto de actividades necesarias para transformar los requisitos del usuario en un sistema de software. Las metodologías imponen un proceso disciplinado sobre el desarrollo de software en aras de lograr una mayor eficiencia y predictibilidad.

Estas metodologías desarrollan un proceso detallado con un fuerte énfasis en la planificación inspirado por otras disciplinas de la ingeniería. Las mismas abarcan todo el ciclo de vida del software, y se definen como un conjunto de procedimientos, técnicas, herramientas y un soporte documental que ayuda a los desarrolladores a realizar un nuevo software. Los procedimientos detallan consejos para elaborar una actividad; las técnicas serían la forma de ejecutar un procedimiento para obtener un resultado determinado; las herramientas de software son las que hacen posible automatizar el proceso de desarrollo del software y la documentación es la que identifica el software que se está desarrollando (Barzanallana, 2008).

2.3.2 Metodologías ágiles

Las metodologías ágiles están principalmente orientadas para proyectos pequeños, brindando una elevada simplificación que a pesar de ello no renuncia a las prácticas esenciales para asegurar la calidad del producto. Las metodologías ágiles surgen como una extensión a las tradicionales, para mejorar el desarrollo de sistemas, según el tipo de proyecto, optimizando las prácticas de desarrollo de software.

Principales características:

- un desarrollo iterativo, ágil, dinámico y muy flexible,
- ubica el programa que funciona por encima de la documentación exhaustiva,
- prioriza la colaboración con el cliente, por encima de la negociación contractual,
- sitúa la respuesta al cambio, por encima del seguimiento de un plan y
- propone a los individuos y su interacción, por encima de los procesos y las herramientas.

Están dirigidas a equipos pequeños o medianos, el entorno físico debe ser un espacio que permita la comunicación y colaboración entre todos los miembros del equipo durante todo el tiempo, cualquier resistencia del cliente o del equipo de desarrollo hacia las prácticas y principios puede llevar el proceso al fracaso (el clima de trabajo, la colaboración y la relación contractual son claves) (Amaro Calderón and Carlos, 2007).

2.3.2.1 SXP

SXP es un híbrido de las metodologías SCRUM y XP que ofrece una estrategia tecnológica, a partir de la introducción de procedimientos ágiles que permitan actualizar los procesos de software para el mejoramiento de la actividad productiva fomentando el desarrollo de la creatividad, aumentando el nivel de preocupación y responsabilidad de los miembros del equipo, ayudando al líder del proyecto a tener un mejor control del mismo (Romero, 2008).

Consta de 4 fases principales:

- Planificación-Definición,
- desarrollo,
- entrega y
- mantenimiento.

Está especialmente indicada para proyectos de pequeños equipos de trabajo, rápidos cambios de requisitos o requisitos imprecisos, muy cambiantes, donde existe un alto riesgo técnico y se orienta a una entrega rápida de resultados y una alta flexibilidad. Ayuda a que trabajen todos juntos, en la misma dirección, con un objetivo claro, permitiendo además seguir de forma clara el avance de las tareas a

realizar, de forma que los que dirigen el proceso puedan ver cada día el progreso del trabajo (Romero, 2008).

Se escoge como metodología ágil SXP para el modelado de la propuesta del analista, porque está condicionada para cambios frecuentes en los requerimientos, genera poca documentación y el cliente es parte del equipo de desarrollo. Principalmente orientada a satisfacer las necesidades de los clientes y cuenta con un grupo reducido de roles.

2.3.3 Lenguaje de modelado: UML

UML (en inglés *Unified Modeling Language*) es un estándar ampliamente utilizado en la industria del software para el modelado de software. Ayuda a los profesionales a visualizar, comunicar y aplicar sus diseños para proporcionar un entorno de modelado visual que se reúne hoy el software de la tecnología y las necesidades de comunicación.

Como UML es un lenguaje, cuenta con pautas para mezclar los elementos gráficos a través de un grupo de herramientas CASE donde se obtiene como resultado final los diferentes diagramas. Es de suma importancia acentuar que este lenguaje no es una guía para el análisis y el diseño, sino que permite modelar sistemas orientados a objetos. Por lo que se hace imprescindible hacer referencia a la herramienta CASE que utiliza a UML para su modelado, dicha herramienta es Visual Paradigm, se definió que esta es la más adecuada para alcanzar los propósitos que se persiguen (Mora, 2003).

UML permite:

- visualizar: UML posibilita expresar de una forma gráfica un sistema de forma que otro lo puede entender,
- especificar: UML brinda la oportunidad de especificar cuáles son las características de un sistema antes de su construcción,
- construir: A partir de los modelos especificados se pueden construir los sistemas diseñados y
- documentar: Los propios elementos gráficos sirven como documentación del sistema desarrollado que puedes servir para su futura revisión.

2.3.4 Herramienta de modelado: Visual Paradigm (VP)

Es una solución informática profesional para el modelado de software. Permite visualizar, diseñar y documentar diagramas de UML 2.0 en un entorno de diseño intuitivo (Paradigm, 2011a). Está compuesto por una serie de elementos gráficos que le permite al usuario lograr realizar un diseño del software.

A continuación se describen algunas de sus características:

- Entorno de creación de diagramas para UML 2.0.
- Uso de un lenguaje estándar común a todo el equipo de desarrollo que facilita la comunicación.
- Modelo y código que permanece sincronizado en todo el ciclo de desarrollo.
- Disponibilidad de múltiples versiones, para cada necesidad.
- Disponibilidad en múltiples plataformas.
- Esta herramienta permite realizar ingeniería tanto directa como inversa.

Se propone utilizar Visual Paradigm porque es una herramienta multiplataforma que permite dibujar todos los tipos de diagramas de clases, generar código desde diagramas, documentar con mayor exactitud todo el trabajo efectuado, generando la información en varios formatos y soporta múltiples usuarios trabajando sobre el mismo proyecto.

2.3.5 Entorno integrado de desarrollo: NetBeans 6.9

Es un entorno de desarrollo visual de código abierto para aplicaciones programadas mediante Java. Su aprendizaje es ahora fundamental para quienes están interesados en el desarrollo de aplicaciones multiplataforma.

NetBeans funciona en sistemas operativos compatibles con la máquina virtual Java (Windows XP, Vista, Windows 7, Ubuntu 9.10, Solaris, Mac OS X 10.5 o superior).

NetBeans posee todos los beneficios del software disponible en forma gratuita, el cual ha sido examinado por una comunidad de desarrolladores. Este enfoque de bienes comunes creativos ha permitido una mayor capacidad de uso, con cada nueva versión y ha proporcionado al desarrollador mayor flexibilidad, al modificar el ambiente de desarrollo integrado por sus siglas en inglés (IDE), si así lo desean (Netbeans, 2007).

2.3.6 Lenguaje de programación: Java

Java es un lenguaje de programación que permite realizar cualquier tipo de programa. En la actualidad es un lenguaje muy extendido y cada vez cobra más importancia tanto en el ámbito de internet como en la informática en general. Java es un lenguaje independiente de la plataforma. Lo que significa que si se realiza un programa en Java podrá funcionar en cualquier sistema operativo.

Java fue diseñado para crear software altamente fiable. Para ello proporciona numerosas comprobaciones en compilación y en tiempo de ejecución. Se escoge Java como lenguaje buscando que la solución final sea multiplataforma.

Java permite el diseño y construcción de aplicaciones de múltiples propósitos, cuenta con librerías bien documentadas para la gestión de interfaces de usuario. Sus características de memoria liberan a los programadores de una gran cantidad de errores, ya que ha complementado en trabajo con los punteros. Además de lo antes expuesto se considera este como lenguaje de programación debido a que la plataforma a la cual va integrado el producto esta desarrolla en Java (Zukowski, 2003).

2.4 Descripción de la propuesta:

El sistema propuesto como solución al problema planteado en esta investigación, consiste en una aplicación de escritorio que permitirá a los arquitectos de información realizar prototipos de interfaz, diagramas de flujo y maquetas. En este trabajo se realizan componentes visuales específicamente de flujo, para luego, como parte del proyecto lograr una integración de los tres tipos de diagramas en el proceso de la arquitectura de información en una sola herramienta. Se implementarán los componentes tales como: Actor, documento, bases de datos, multidocumentos, evento de inicio, intermedio y fin, almacenamiento de acceso directo, almacenamiento interno, conector, flechas con texto y sin texto, datos, decisión, discos magnéticos, retrasos, entradas manuales, intercalar, imágenes, procesos definidos, procesos alternativos, preparación, procesos, tarjetas, variables lógicas como O e Y.

2.5 Marcos de trabajo para la gestión de interfaces de usuario en Java: 2.5.1 AWT/ Swing

La *Abstract Window Toolkit* (AWT, en español Kit de Herramientas de Ventana Abstracta) es un kit de herramientas de gráficos, interfaz de usuario y sistema de ventanas independiente de la plataforma original de Java. AWT es ahora parte de las Java *Foundation Classes* (JFC) - la API estándar para suministrar una interfaz gráfica de usuario (GUI) para un programa Java.

El paquete Swing hereda todo el manejo de eventos de AWT, pero además presenta la independencia de plataforma que este necesitaba. Los componentes de Swing utilizan la infraestructura de AWT, incluyendo su modelo de eventos, el cual se rige como un componente y reacciona a eventos tales como, eventos de teclado, ratón, y otros.

Swing además de proveer un conjunto más rico de componentes UI, Swing dibuja sus propios componentes (usando Java 2D para llamar a las subrutinas de bajo nivel en el subsistema de gráficos local) en lugar de confiar en el módulo de interfaz de usuario de alto nivel del sistema operativo. Swing suministra la opción de usar un aspecto nativo o de plataforma cruzada para la aplicación.

AWT continúa suministrando el núcleo del subsistema de eventos GUI y la interfaz entre el sistema de ventanas nativo y la aplicación Java, suministrando la estructura que necesita Swing. También suministra gestores de disposición básicos, un paquete de transferencia de datos para uso con el Bloc de notas y arrastrar y soltar y la interface para los dispositivos de entrada tales como el ratón y el teclado.

De esta forma, la unión de estos dos marcos de trabajo es muy provechosa para desarrolladores que buscan una mejor interfaz gráfica. Al estar AWT al mando de la gestión de eventos y el Swing encargado de los componentes agregados, que son las principales cualidades de estos marcos de trabajo, se logran excelentes resultados en el área del diseño. Debido a la necesidad de crear una interfaz agradable para el usuario, además de la necesaria interacción entre ellos, es que se utiliza este marco de trabajo en la implementación de la solución propuesta.

2.6 Valoración de la arquitectura: Patrones arquitectónicos

Un patrón es una solución a un problema en un contexto, codifica conocimiento específico acumulado por la experiencia en un dominio. Describe un problema que se instancia varias veces en un dominio determinado y propone su solución. En el caso de los patrones arquitectónicos proponen un esquema organizativo estructural para los sistemas informáticos.

2.6.1 Patrón arquitectónico de tres capas

Una arquitectura de software diseñada en capas consiste en la definición de niveles de abstracción, los cuales tienen una función específica permitiendo un diseño modular. Ello permite que sean creados sistemas con un bajo acoplamiento entre sus módulos o componentes. Una variante de este patrón muy

utilizada es la de tres capas, mediante la cual se fracciona el sistema informático en la capa de presentación, la capa de lógica del negocio y la capa de acceso a datos (Larman, 1999a).

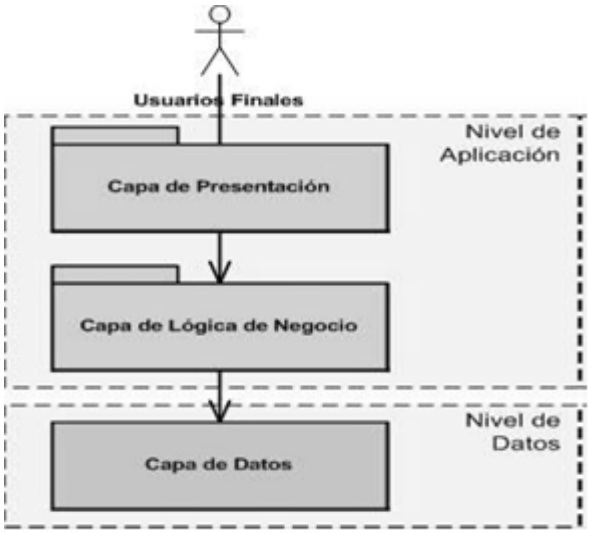


Fig. 13: Patrón arquitectónico tres capas

Permite dividir la aplicación informática en tres capas, capa de presentación, capa de lógica de negocio y la capa de acceso a datos.

- Capa de presentación: es la que observa, presenta el sistema, le comunica la información y captura la información del usuario. Esta capa se comunica únicamente con la capa de negocio.
- Capa de negocio: es donde residen los programas que se ejecutan, se reciben las peticiones del usuario. Es donde se establece todas las reglas que se deben cumplir. Esta capa se comunica con la capa de presentación, para recibir las solicitudes y presentar los resultados, y con la capa de datos, para solicitar al gestor de base de datos almacenar o recuperar datos de él.
- Capa de acceso a datos: es la que gestiona el almacenamiento de los datos, ya sea en una base de datos o en un fichero, así como la consulta a los mismos (Flower, 2003).

Una restricción de este patrón consiste en que las capas inferiores no deben de conocer ni hacer llamadas a procedimientos implementados en capas superiores, sino que las funcionalidades que ellas ofrecen son accedidas desde niveles mayores (Larman, 1999a).

Conclusiones capítulo 2

Según lo antes expuesto se arriban a las siguientes conclusiones:

- El resultado del estudio de herramientas homólogas y la valoración crítica del análisis propuesto por el analista determinan la inserción de funcionalidades que aportan mayor valor agregado al sistema en desarrollo.
- La valoración crítica del entorno de desarrollo propuesto por el analista facilita la evaluación y entendimiento de las herramientas y la metodología definidas.
- La definición del lenguaje de programación y el IDE ayudaron a seleccionar los marcos de trabajo de interfaz gráfica AWT/Swing.
- La evaluación de la arquitectura ayuda a entender cómo se organizan los componentes de software a implementarse.

Capítulo 3: Descripción de la implementación y estrategia de pruebas

3.1 Introducción*:* Descripción de la implementación y estrategia de pruebas

En este capítulo se presentan los patrones de diseño usados en el desarrollo del sistema y estándares de código para guiar la implementación de forma correcta, además de la etapa de pruebas que propone la metodología, donde se muestran los casos de pruebas de aceptación realizadas al sistema. Estos casos de prueba deben se realizan en cada una de las iteraciones para poder continuar el desarrollo y avanzar hacia la siguiente iteración.

3.2 Modelo de diseño

De acuerdo con la arquitectura en tres capas que organiza las clases en tres paquetes como se plantea en este trabajo de diploma. Cada uno de los paquetes se ha dividido a la vez en sub-paquetes de acuerdo con las funcionalidades de las clases contenidas. Cada paquete representa una capa de la arquitectura y está construida sobre su predecesor. Las clases pertenecientes a una capa están relacionadas con las clases de la capa inmediata inferior y desconocen a las clases de las capas superiores (Ver Figura 14).

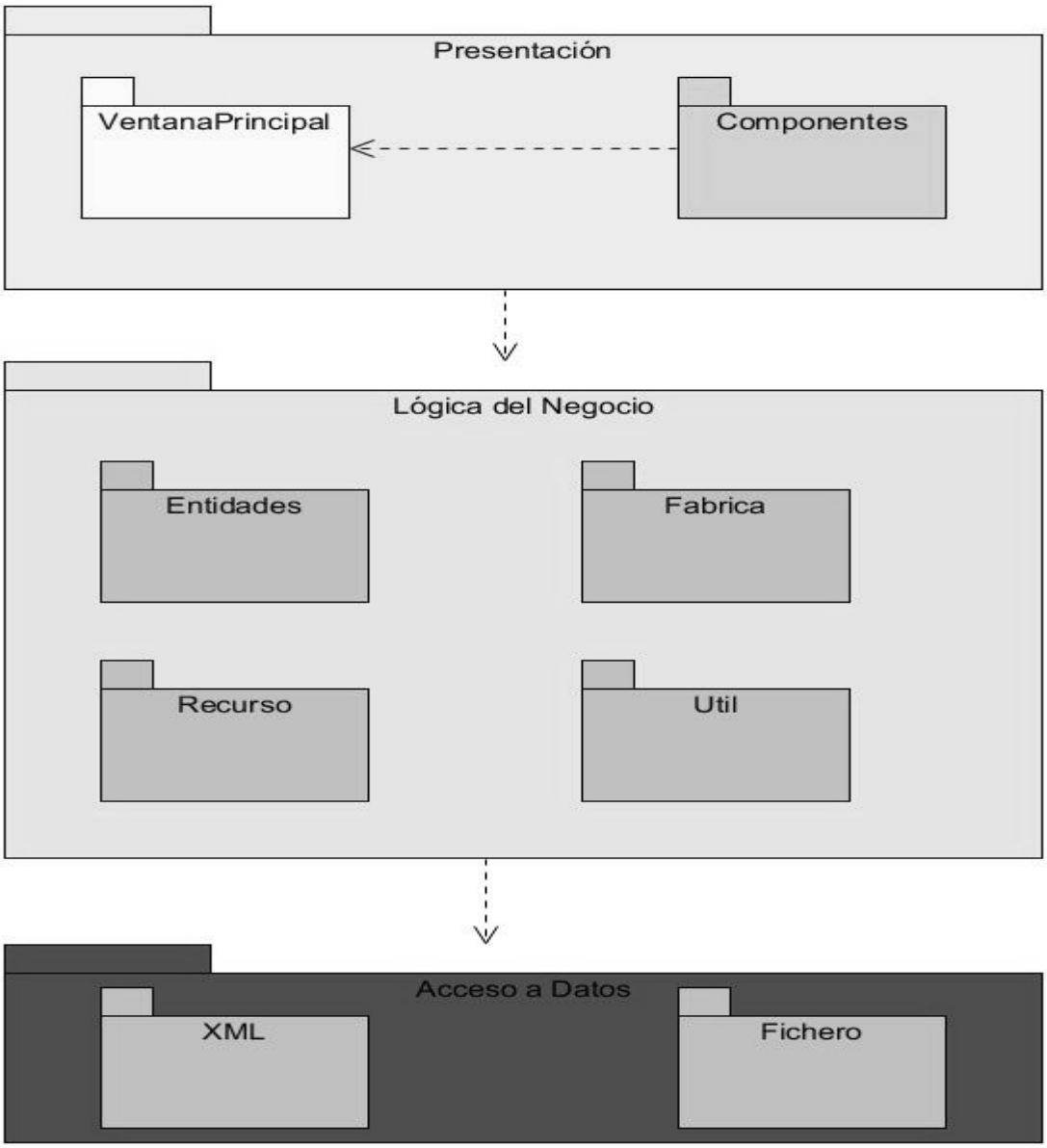


Fig. 14: Diagrama de paquetes

3.2.1 Acceso a datos

La capa de acceso a datos es la encargada de la persistencia y recuperación de objetos, específicamente de la interacción de la aplicación con los ficheros de almacenamiento. Estos ficheros pueden contener las páginas o proyectos en forma de objeto serializado. El paquete de acceso a datos mantiene un bajo acoplamiento con las entidades del negocio. Las clases de este paquete desconocen a las entidades del negocio y solo se centran en la entrada y salida de datos. Este paquete contiene la clase Fichero, esta representa el fichero físico en el que se encuentra almacenada la información a recuperar o modificar y en XML se encuentra la clase encargada de la persistencia de los datos en XML (Ver Figura 15).

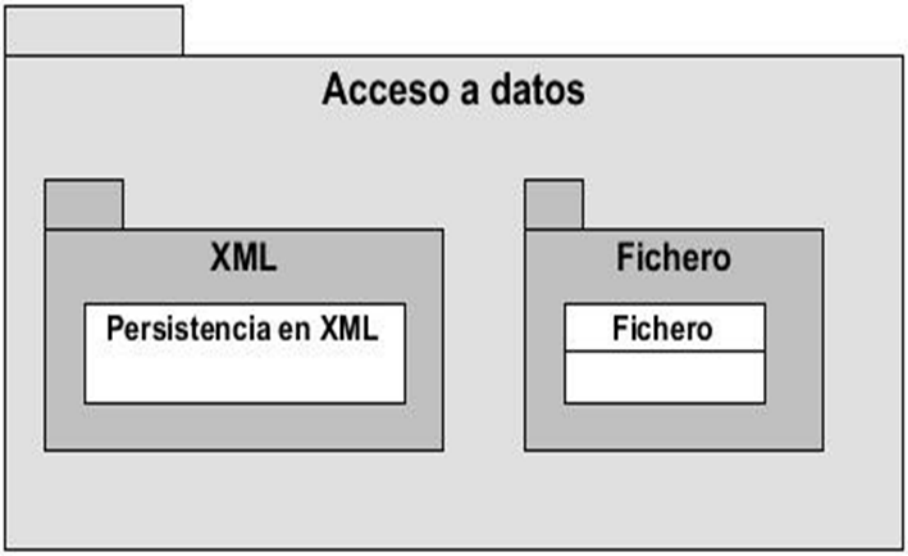


Fig. 15: Capa acceso a datos

3.2.2 Capa lógica del negocio

La capa lógica del negocio engloba a las clases encargadas de ejecutar las tareas y reglas que rigen el proceso. En esta solución el paquete de lógica del negocio está dividida en dos paquetes: *Entidades* y las correspondientes al negocio. Por la magnitud de la capa de lógica de negocio se describen los paquetes por separados para un mejor entendimiento (Ver Figura 16).

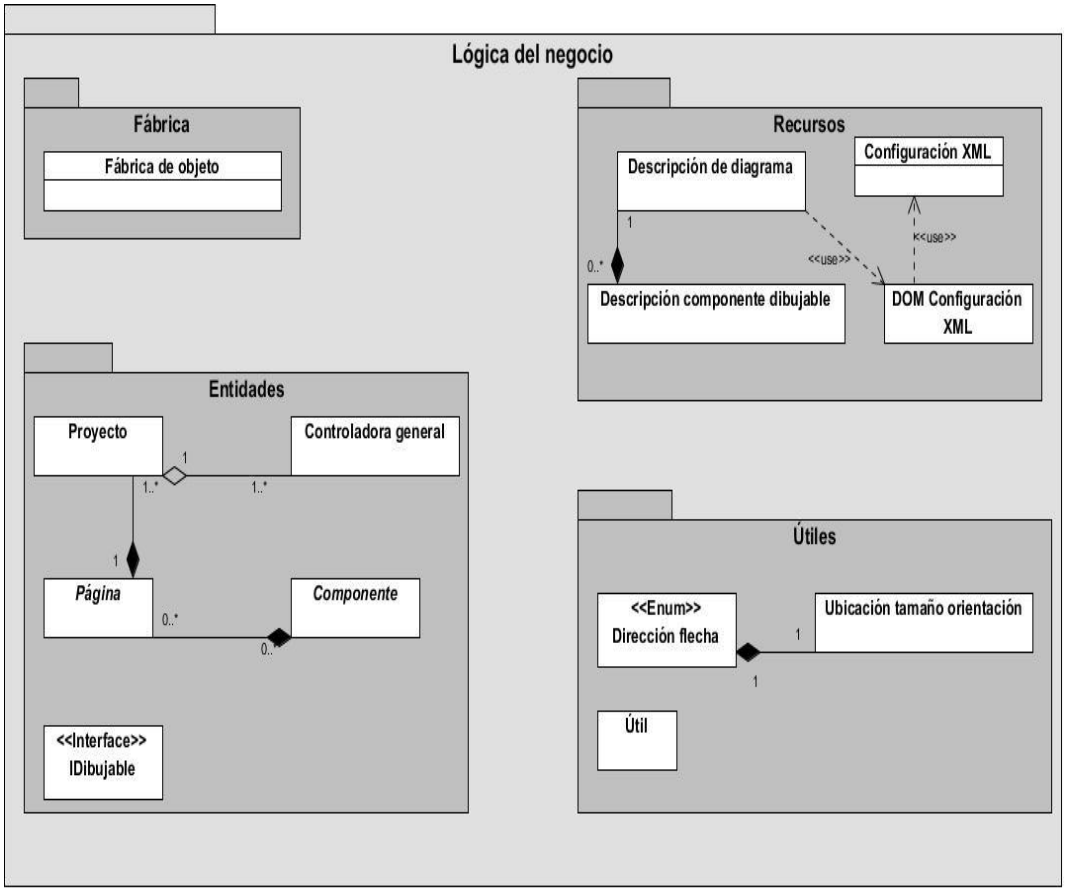


Fig. 16: Capa de lógica del negocio.

En esta capa se encuentra el paquete *Fabrica*, que contiene la clase *FabricaDeObjeto* encargada de crear los componentes visuales que se dibujan en la aplicación. Se encuentra también en esta capa el paquete *Recursos* que posee las clases encargadas de montar los componentes en la aplicación. El paquete de *Util* tiene las clases que orientan el sentido de la flecha a dibujar para enlazar los componentes visuales. En el paquete *Entidades* se encuentran las clases padres encargadas de implementar las propiedades de

los componentes visuales, páginas de proyecto, pestañas y de las opciones de guardar los proyectos y diagramas realizado.

3.2.3 Capa presentación

La capa de presentación es la encargada de lograr la comunicación directa entre el sistema y el usuario final a través de una interfaz gráfica. Una interfaz gráfica de usuario consiste en un conjunto de componentes empleados por usuarios para comunicarse con los sistemas informáticos. Los usuarios dirigen el funcionamiento del sistema mediante la generación de eventos. Para una mayor organización de la capa presentación queda dividida en paquetes según las funcionalidades de cada uno. Los paquetes empleados son: Componentes y VentanaPrincipal (Ver Figura 17).

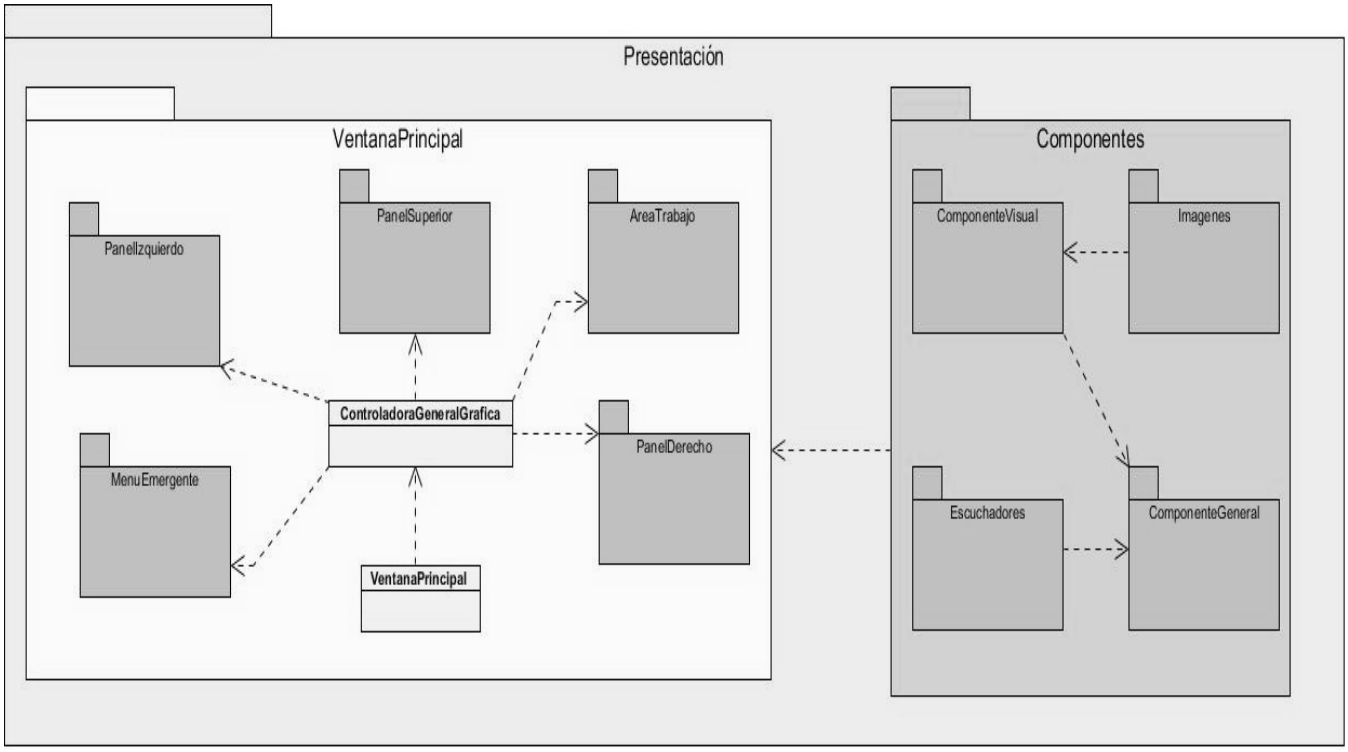


Fig. 17: Capa de presentación

El paquete *VentanaPrincipal* contiene toda la interfaz gráfica principal donde el usuario interactúa con el sistema mediante acciones y eventos realizados. Contiene el paquete de *AreaTrabajo,* esta posee los

componentes de la aplicación a usar, por ejemplo el área de trabajo, las páginas, las pestañas y los paneles transparentes donde se dibujan las líneas. *PanelIzquierdo* y *PanelDerecho* tienen las barras decomponentes visuales árbol de proyecto y la barra de propiedades de los componentes visuales. El paquete de *MenuEmergente* posee las clases encargadas de las propiedades del clic derecho, pues en todos los componentes de la herramienta no funciona igual (Ver Figura 18).

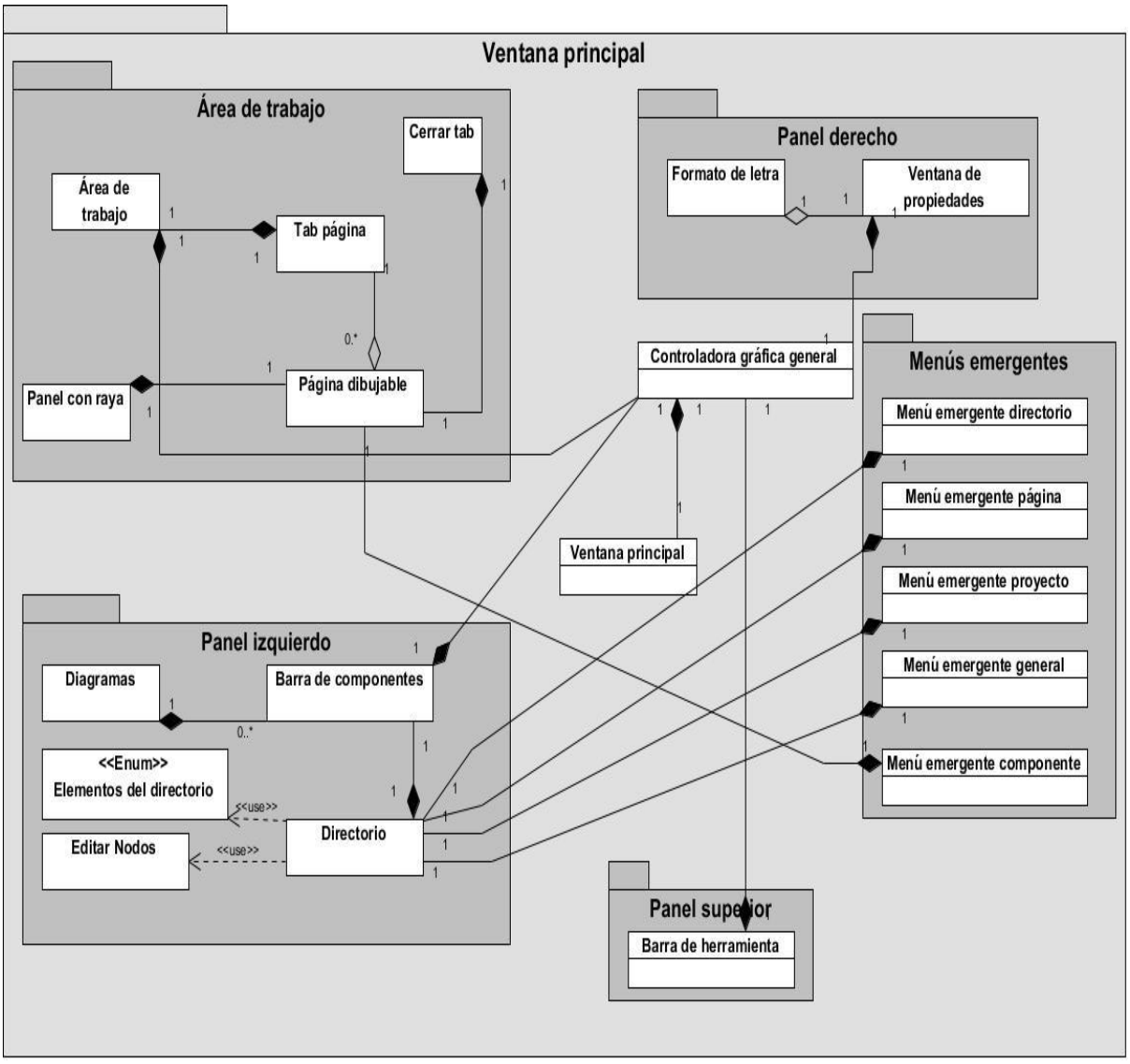


Fig. 18: Presentación.VentanaPrincipal

Capítulo 3: Descripción de la implementación y estrategia de pruebas

El propósito del paquete Componentes consiste en agrupar todos los funcionamientos de los componentes creados, este paquete por ejemplo el paquete *ComponenteVisual* contiene junto a la clase padre *ComponenteDibujable* todas las clases hijas que son los componentes visuales implementados en el sistema, con cada una de las funcionalidades a usar para el trabajo del usuario con estos componentes visuales por ejemplo las propiedades de cambiar color del fondo, color del borde, alinear texto, cambiar ancho y largo, además en el paquete de Imágenes están todos los estereotipos correspondiente a cada uno de los componentes visuales, junto a los eventos y los paneles usados en algunos de los componentes visuales (Ver Figura 19).

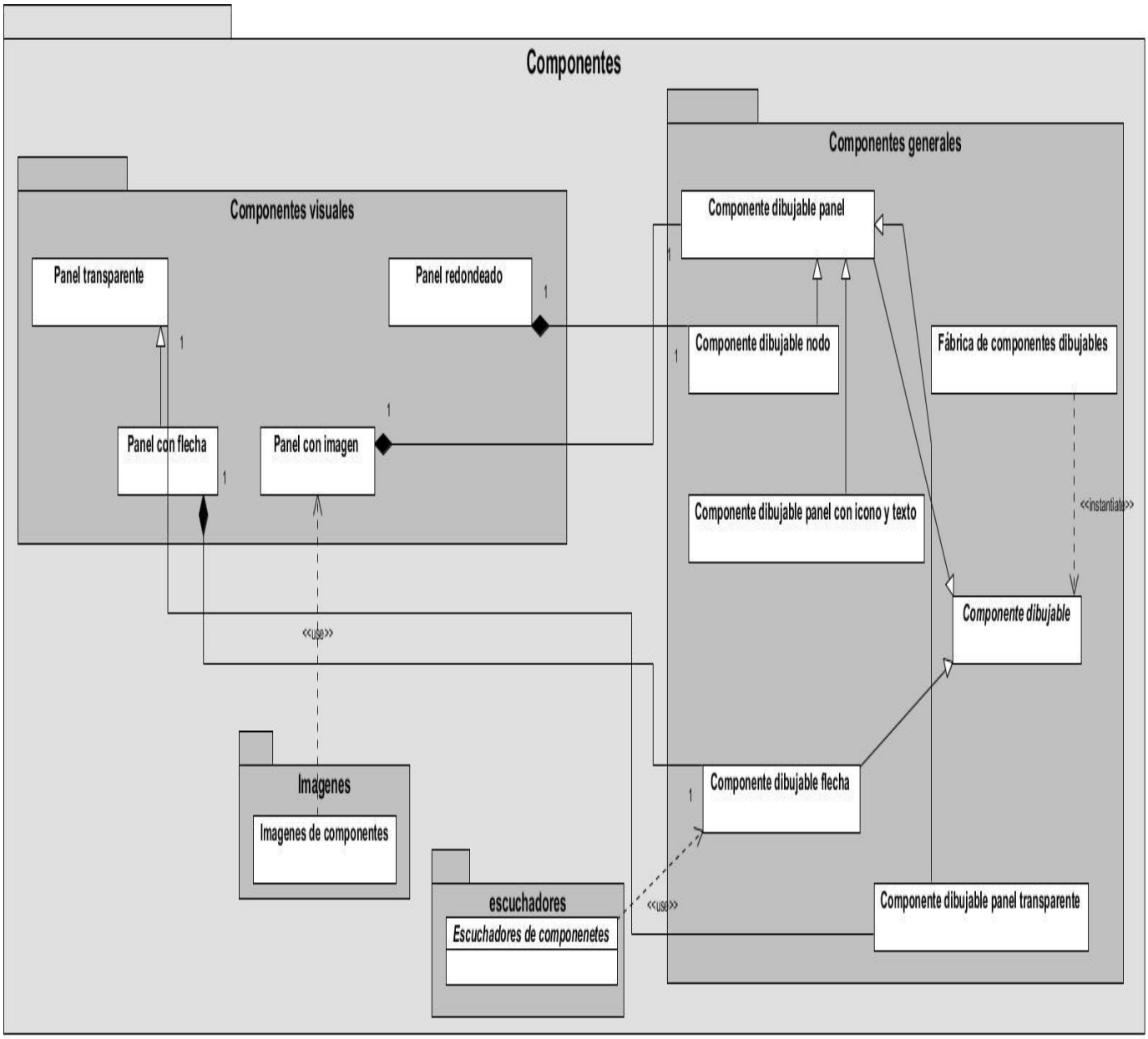


Fig. 19: Presentacion.Componentes

3.3 Estándares de codificación

Un estándar de codificación comprende los aspectos de la generación de código y repercute directamente en la legibilidad y la extensibilidad en el código de un proyecto de software, permitiendo que nuevos desarrolladores se acoplen rápidamente al proceso de desarrollo.

Capítulo 3: Descripción de la implementación y estrategia de pruebas

Usar estándares de codificación sólidos y realizar buenas prácticas de programación con vistas a generar un código de alta calidad es de gran importancia para la calidad del software y para obtener un buen rendimiento del mismo (Estándar _de _Código CENIA, 2010).

En el Centro de Informatización Universitaria de la Universidad de las Ciencias Informáticas se definió un estándar de codificación para aplicaciones web de gestión de información, fundamentalmente desarrolladas con lenguaje *PHP*. Como no se han definido estándares de codificación para las aplicaciones de escritorio desarrolladas en el centro, el autor de este trabajo tomó elementos de la propuesta del CENIA para aplicaciones web, adaptándolas a las particularidades de una aplicación de escritorio desarrolladas en java, solamente se explican los elementos que se tomaron del estándar del CENIA, con las adaptaciones realizadas.

3.3.1 Identación llaves de apertura y cierre y tamaño de las líneas

Identación consiste en mover un bloque de texto hacia la derecha insertando espacios o tabuladores para separarlo del texto adyacente. Usar una identación sin tabulaciones, con un equivalente a 4 espacios, para mantener integridad en las revisiones svn. El uso de las llaves “{” será en una nueva línea. La longitud de las líneas de código es aproximadamente de 75-80 caracteres para mantener la legibilidad del código.

Ejemplo:

1....TipoDato a = TipoDato b;

2

3....public void ejemplo ()

4.... {

5....//BI

6....}

3.3.2 Estructuras de control

Se incluye if, for, foreach, while y switch. Entre la estructura de control y los paréntesis debe existir un espacio. Se recomienda utilizar siempre llaves de apertura y cierre, incluso en situaciones en las que técnicamente son opcionales, esto aumenta la legibilidad y disminuye la probabilidad de errores lógicos.

Ejemplos:

```
1....if (condición)
2.... {
3 ....//Bloque de instrucciones
4....}
5....elseif (condición)
6.... {
7 ....//Bloque de instrucciones
8....}
```

3.3.3 Convención de nomenclatura

Variables: se rigen por la nomenclatura camelCase. Siempre comienzan con minúscula y en caso de nombres compuestos la primera letra de cada palabra comienza con mayúscula.

Ejemplo:

```
1.... private TipoDato variable
2.... private TipoDato variableNombreCompuesto
```

Clases: siempre comienzan con mayúscula, en caso de nombre compuesto las palabras se separan con el carácter subrayado "_" y el resto en minúscula.

Ejemplo:

```
1....public class Clase
2....{
3....//BI
4....}
5
6....public class Clase_nombre_compuesto
7.... {
8....//BI
```

9....}

Métodos: se rigen por la nomenclatura camelCase. Siempre comienzan con minúscula y en caso de nombres compuestos la primera letra de cada palabra comienza con mayúscula. Los parámetros son separados por espacio luego de la coma que los separa.

Ejemplo:

1....public void Metodo (TipoDato parametro1, TipoDato parametro2)
2.... {
3....//BI
4....}
5
6....public void MetodoNombreCompuesto (TipoDato parametro1, TipoDatoparametro2)
7.... {
8....//BI
9....}

Ficheros: todo siempre en minúscula y en caso de nombres compuestos se usa el carácter subrayado"_".

3.4 Estándar de codificación para XML

El Lenguaje de Marcas Extensible (XML por sus siglas en ingles), "es un lenguaje de marcas que ofrece un formato para la descripción e intercambio de datos estructurados" (Herrera, 2010). Este metalenguaje (lenguaje o código utilizado para describir otro lenguaje o código) constituido en formato de texto plano, está conformado por un conjunto de etiquetas que describen y organizan el contenido del documento en diferentes partes. Dichas etiquetas son creadas por el propio diseñador dependiendo el contenido del documento, es decir, no son prefijadas con anterioridad y admite un conjunto ilimitado de estas.

En el centro no está definido un estándar de código para documentos XML, siendo así, el autor de este trabajo adaptó el estándar existente para aplicaciones web generalmente realizadas en *PHP* a las necesidades de esta solución. En la solución se utiliza este lenguaje para salvar los proyectos guardados en formato XML y para insertar componentes visuales a la aplicación. Ejemplo de algunas de las particularidades modificadas se exponen a continuación:

3.4.1 Un archivo XML válido comienza con una DTD o Declaración de Tipo de Documento:

<? xml version="1.0" encoding= "UTF-8"?>

<etiqueta 1>
//Bloque Instrucciones
</etiqueta 1>

Etiquetas definidas:

<Tipo de diagrama>
<nombre> Tipo de diagrama </ nombre >
<comentario> Diagrama encargado de representar flujo </ comentario >
<Componente>
<nombreClase> Direccion de la clase en el proyecto </ nombreClase >
<nombreComponente> ComponenteA </ nombreComponente >
<comentario> Componente que representa una BD </ comentario >
<colorDelBorde> </ colorDelBorde >
<colorDelFondo> </ colorDelFondo >
<nombreImagen> </ nombreImagen >
</Componente>
</ Tipo de diagrama >

3.4.2 Convención de nomenclatura: *Etiquetas*

Se rigen por la nomenclatura camelCase. Siempre comienzan con minúscula y en caso de nombres compuestos la primera letra de cada palabra comienza con mayúscula.

Ejemplo:

1.... <comentario> Componente que representa una BD </ comentario>
2.... <nombreComponente> ComponenteA </ nombreComponente>

3.5 Patrones de diseño

Un patrón de diseño es un conjunto de reglas que describen como afrontar tareas y solucionar problemas que surgen durante el desarrollo de software (Gamma, 2002) .

3.5.1 Patrones GRASP

Los patrones GRASP (*General Responsibility Assignment Software Patterns*, en español Patrones Generales de Software para la Asignación de Responsabilidades) describen los principios fundamentales de la asignación de responsabilidades a objetos (Larman, 1999a).

- ✓ El patrón Experto: soluciona el problema de asignar una responsabilidad de forma general, tomamos decisiones sobre la asignación de responsabilidades a las clases. Si se hacen en forma adecuada, los sistemas tienden a ser más fáciles de entender, mantener y ampliar y se presenta la oportunidad de reutilizar los componentes en futuras aplicaciones. Por ejemplo en la clase *ControladoraGraficaGeneral* que es la encargada de manejar todos los datos que tienen que ver con la creación de proyectos y que están dentro de la lógica del negocio.
- ✓ El patrón Controlador: asigna la responsabilidad de recibir o manejar los mensajes de evento del sistema. Por ejemplo se centralizan las actividades fundamentales en las clases controladoras del sistema: *ControladoraGeneral*, *ControladoraGraficaGeneral.*
- ✓ El patrón Bajo Acoplamiento: da soporte a una dependencia escasa y a un aumento de la reutilización, manteniendo las clases que no dependan de muchas otras clases. Por ejemplo se utilizan las clases *ControladoraGeneral*, *ControladoraGraficaGeneral* para reducir la dependencia entre las clases.
- ✓ El patrón Alta Cohesión: aconseja mantener la clase lo más relacionada posible para controlar la complejidad de la misma. La clase *FabricaDeObjeto* realiza una labor única dentro del sistema, y ninguna otra clase realiza sus funciones

3.5.2 Patrones GoF

El uso de patrones de diseño facilitará el diseño de aplicaciones (mediante su modelado), así como la gestión de cambios de requerimientos o posterior mantenimiento (Gamma, 2002).

Estos fueron los patrones utilizados:

- ✓ Fábrica: su propósito es crear objetos, permitiendo al sistema identificar que clase se debe instanciar en tiempo de ejecución. Por ejemplo en la solución se utilizó para crear una instancia de múltiples objetos, como es el caso de la clase *ComponenteDibujable* que en este caso es la clase padre, pero al instanciarla en realidad el objeto lo que va a contener es una instancia de una de

- ✓ sus clases hijas, las cuales pueden ser *ComponenteDibujableFlecha*, *ComponenteDibujableTexto* y todas las clases de componentes visuales.
- ✓ Fachada: Proporciona una interfaz unificada para un conjunto de interfaces de un subsistema. Define una interfaz de alto nivel que hace que el subsistema sea más fácil de usar (Larman, 1999b). Un ejemplo de este patrón en la aplicación es en el uso de las clases *java.awt.Graphics* y *java.awt.Font*, estas lo utilizan. En la solución se maneja la clase *Graphics* a la hora de dibujar la línea o relación entre componentes, usando el método *paint (Graphics obj)* que recibe un objeto de tipo esta clase. Y la clase *Font* se utiliza para el trabajo con las fuentes en la aplicación.

3.6 Modelo de despliegue

"El modelo de despliegue es un modelo de objetos que describe la organización física del sistema en términos de cómo se distribuye la funcionalidad entre los nodos de cómputo" (I. Jacobson, 2000). La presente solución es una aplicación para entornos de escritorio por lo que será ejecutada en una estación de trabajo, independientemente de que la misma disponga de conexión de red o no, dentro de la misma se encuentra toda la distribución que se hará de todo sus componentes.

Fig. 20: Diagrama de despliegue

3.7 Modelo de implementación

"El modelo de implementación describe cómo los elementos del modelo de diseño y las clases, se implementan en términos de componentes, como ficheros de código fuente, ejecutables, etc. El modelo de implementación describe también cómo se organizan los componentes de acuerdo con los mecanismos de estructuración y modularización disponibles en el entorno de implementación y en el lenguaje o lenguajes de programación utilizados y cómo dependen unos de otros" (I. Jacobson, 2000).

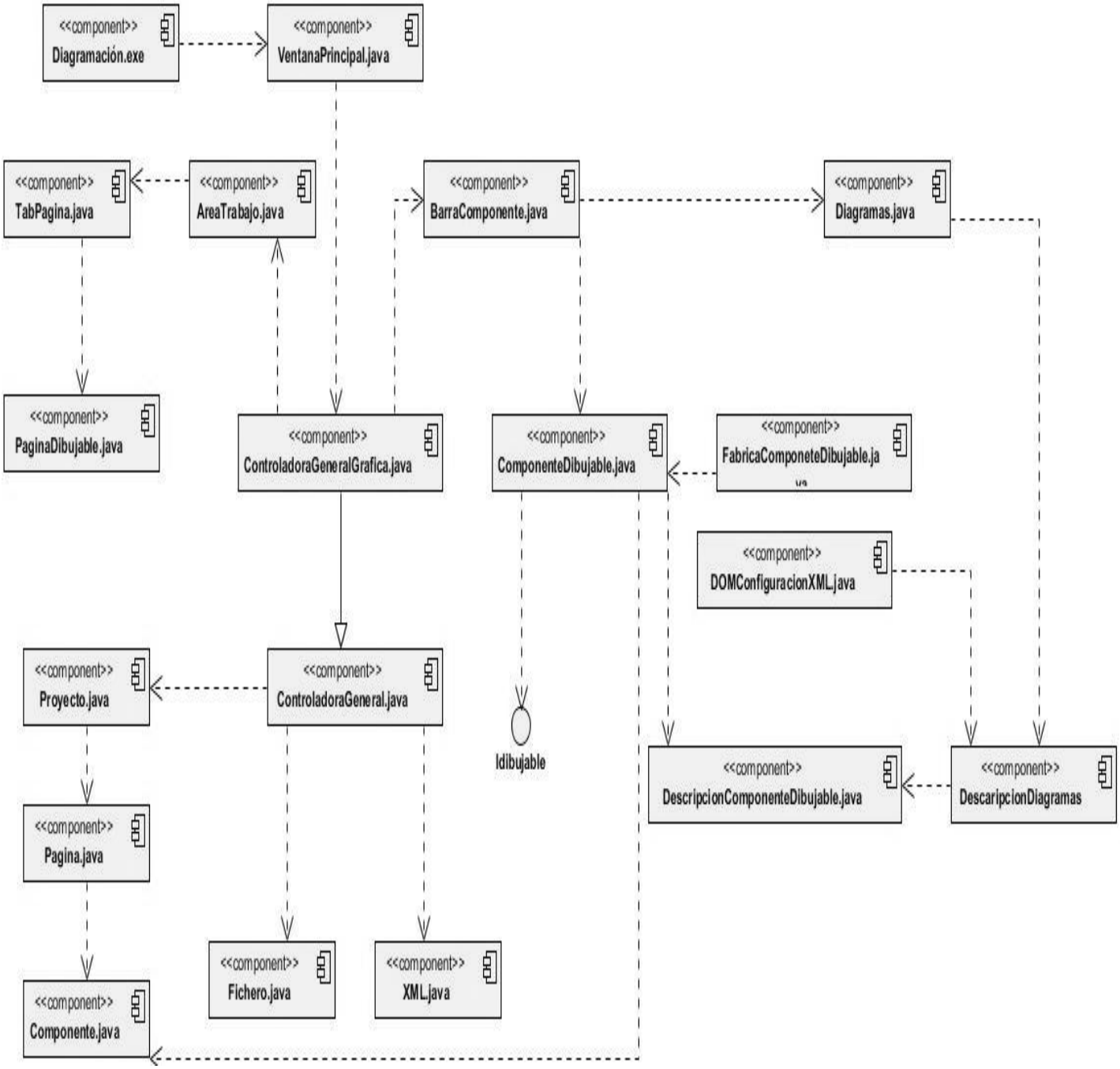


Fig. 21: Diagrama de componentes

Capítulo 3: Descripción de la implementación y estrategia de pruebas

3.8 Pruebas

Las pruebas de software son un conjunto de herramientas y técnicas que evalúan el desempeño de un programa. En todo proceso de desarrollo de aplicaciones es indispensable la presencia de un proceso de pruebas de software que garantice el buen funcionamiento y la calidad del producto final.

Las pruebas de software permiten conocer hasta qué punto las funciones del software operan de acuerdo con las especificaciones y requisitos del cliente. Estas involucran las operaciones del sistema evaluando los resultados bajo condiciones específicas, lo que hace que la realización de pruebas a los programas constituya un factor de vital importancia. Un proceso de prueba completo debe garantizar además que los defectos encontrados se han corregido antes de entregar el software al cliente.

3.8 Estrategias de prueba

3.8.1 Pruebas de unidad

Las pruebas de unidad son la primera fase de las pruebas que se le aplican a cada módulo de un software de manera independiente. Su objetivo es verificar que el módulo, entendido como una unidad funcional de un programa independiente, está correctamente codificado.

Es la escala más pequeña de las pruebas, está basada en la funcionalidad de los módulos del programa, como funciones, procedimientos y módulos de clases (Pressman, 2005a). Es aplicable a componentes representados en el modelo de implementación para verificar que los flujos de control y de datos están cubiertos y que ellos funcionen como se espera. Los casos de pruebas que se diseñen para este nivel de pruebas, deben descubrir errores como:

- comparaciones entre tipos de datos distintos,
- operadores lógicos o procedencia incorrecta,
- igualdad esperada cuando los errores de precisión la hacen poco probable,
- variables o comparaciones incorrectas y
- fallo de salida cuando se encuentra una iteración divergente.

La pruebas fueron realizadas bajo un enfoque ágil, donde a la aplicación se le realizan pruebas durante todo el desarrollo sin documentarse, se acudió al trabajo en equipo, donde el cliente forma parte del mismo y con el IDE NetBeans que brinda funcionalidades como *debug*, comparar variables, ver errores en la asignación de las variables. Se realizan también, pruebas para mejorar la estructura del código,

previendo la integración del sistema a la Plataforma Abad con un alto grado se seguridad en cuanto al funcionamiento correcto del código. Con la aplicación de estas pruebas los errores van a estar más acotados y son más fáciles de desenmascarar (Pressman, 2005a).

3.3.2 Pruebas de Aceptación

Las pruebas de aceptación constituyen pruebas de caja negra que se centran en el correcto cumplimiento de los requisitos definidos para el sistema una vez concluido el mismo. Por lo tanto, estas pruebas constituyen la validación de la aplicación por parte del usuario final y según Pressman plantea lo siguiente: "La validación del software se consigue mediante una serie de pruebas de caja negra que demuestran la conformidad con los requisitos. Un plan de prueba traza la clase de pruebas que se han de llevar a cabo, y un procedimiento de prueba define los casos de prueba específicos en un intento por descubrir errores de acuerdo con los requisitos. Tanto el plan como el procedimiento estarán diseñados para asegurar que se satisfacen todos los requisitos funcionales, que se alcanzan todos los de rendimiento, que la documentación es correcta e inteligible y que se alcanzan otros requisitos (por ejemplo, portabilidad, compatibilidad, recuperación de errores, facilidad de mantenimiento)"(Pressman, 2005a).

Dentro de las pruebas de aceptación no formales, realizadas en software a la medida del cliente se encuentran las pruebas alfa y beta. Estas pruebas son muy convenientes cuando se desarrollan aplicaciones que van a ser utilizadas por muchos clientes, con el objetivo de descubrir errores que parezca que sólo el usuario final puede descubrir.

A la solución se le aplicaron las pruebas Beta, estas se realizan en las instalaciones propias de los clientes. Para que tengan lugar, en primer término se deben distribuir copias del sistema para que cada cliente lo instale en sus oficinas, dependencias y/o sucursales, según sea el caso. En el caso de las pruebas Beta, cada usuario realizará sus propias pruebas y documentará los errores que encuentre, así como las sugerencias que crea conveniente realizar, para que el equipo de desarrollo tenga en cuenta al momento de analizar las posibles modificaciones.

Con la metodología SXP estas pruebas se realizan entre iteraciones y son las que definen el paso a la próxima iteración. En cada iteración se usaron las pruebas de aceptación de manera informal, específicamente de tipo alfa, junto al cliente que es el propio proyecto, para comprobar y validar las funcionalidades del sistema, y de esta forma saber si está apto para ser liberado. A continuación como fueron divididos los requisitos en cada una de las iteraciones y en el resultado de las pruebas se explica en detalles todo lo realizado y analizado en las pruebas realizadas.

Las iteraciones se organizan de la siguiente forma:

Iteración 1	Iteración 2	Iteración 3
33 requisitos	33 requisitos	4 requisitos

Tabla 4: Requisitos por iteraciones

3.3.3 Casos de Pruebas

Los casos de prueba pretenden demostrar que las funciones del software son operativas, que la entrada se acepta de forma adecuada y que se produce un resultado correcto. Los casos de prueba escritos, incluyen una descripción de la funcionalidad que se probará, la cual es tomada ya sea de los requisitos o de los casos de uso, y la preparación requerida para asegurarse de que la prueba pueda ser dirigida.

Clases Válidas	**Resultado Esperado**	**Resultado de la Prueba**	**Observaciones**
El usuario selecciona en la barra de herramienta el icono para crear nuevo proyecto.	Se muestra del lado izquierdo de la aplicación el árbol de proyecto creado.	Positivo	
Clases Inválidas	**Resultado Esperado**	**Resultado de la Prueba**	**Observaciones**
El usuario se equivoca en la selección del icono en la barra de herramienta para crear un árbol de proyecto.	No se crea el árbol de proyecto.	Positivo	
Evaluación de la Prueba:		Satisfactoria	

Tabla 5: Crear un nuevo árbol de proyecto

Clases Válidas	**Resultado Esperado**	**Resultado de la Prueba**	**Observaciones**
El usuario crea el área de trabajo .	Si el usuario crea un área de trabajo, debe mostrar la acción realizada.	Positivo.	
Clases Inválidas	**Resultado Esperado**	**Resultado de la Prueba**	**Observaciones**
El usuario se equivoca en la selección de crear.	No se muestran los cambios.	Positivo.	
Evaluación de la Prueba:	Satisfactoria.		

Tabla 6: Crear el área de trabajo

Clases Válidas	**Resultado Esperado**	**Resultado de la Prueba**	**Observaciones**
El usuario selecciona mediante un clic la página que desea eliminar, con el clic derecho selecciona la opción del menú desplegable eliminar.	Se elimina del árbol de proyecto la página seleccionada.	Positivo	
Clases Inválidas	**Resultado Esperado**	**Resultado de la Prueba**	**Observaciones**
El usuario no seleccionó la página que desea eliminar.	No se elimina la página del árbol de proyecto.	Positivo	
Evaluación de la Prueba:	Satisfactoria		

Tabla 7: Eliminar página del árbol

Clases Válidas	Resultado Esperado	Resultado de la Prueba	Observaciones
El usuario selecciona la opción común.	La aplicación muestra los componentes visuales para los tres tipos de diagramas.	Positivo	
Clases Inválidas	**Resultado Esperado**	**Resultado de la Prueba**	**Observaciones**
El usuario no selecciona la opción común.	La aplicación no muestra los componentes visuales de la opción común.	Positivo	
Evaluación de la Prueba:	Satisfactoria		

Tabla 8: Presentar todos los componentes de los diagramas

Escenario	Descripción	Respuesta del sistema	Flujo central
Crear proyecto correctamente	El sistema permite crear un nuevo proyecto	Se muestra la ventana que posee la vista del proyecto.	1.Clic en el menú Archivo 2. Seleccionar la opción “Nuevo proyecto”.
			1-Clic derecho en el área de trabajo 2-clic en la opción Adicionar proyecto.

Tabla 9: Crear proyecto

3.5 Resultado de las pruebas

A la aplicación se le realizaron 3 iteraciones de prueba, encontrándose no 14 conformidades en la primera iteración de las cuales todas fueron resueltas, estas no conformidades que fueron encontrada

generalmente de validación, 10 no conformidades en la segunda y fueron resueltas 8 estas fueron de validación y errores de ortografía, las 2 no conformidades que quedaron sin resolver eran de asignación, estos pasaron a la próxima iteración, en la tercera se detectaron 2 no conformidades de tipo validación haciendo un total de 4 no conformidades en esa iteración, quedando resueltas en su totalidad. En la siguiente figura se evidencian los resultados obtenidos, se puede concluir que a medida de las pruebas el número de defectos encontrados fue menor, logrando un mayor ajuste del *software* con los requisitos del cliente.

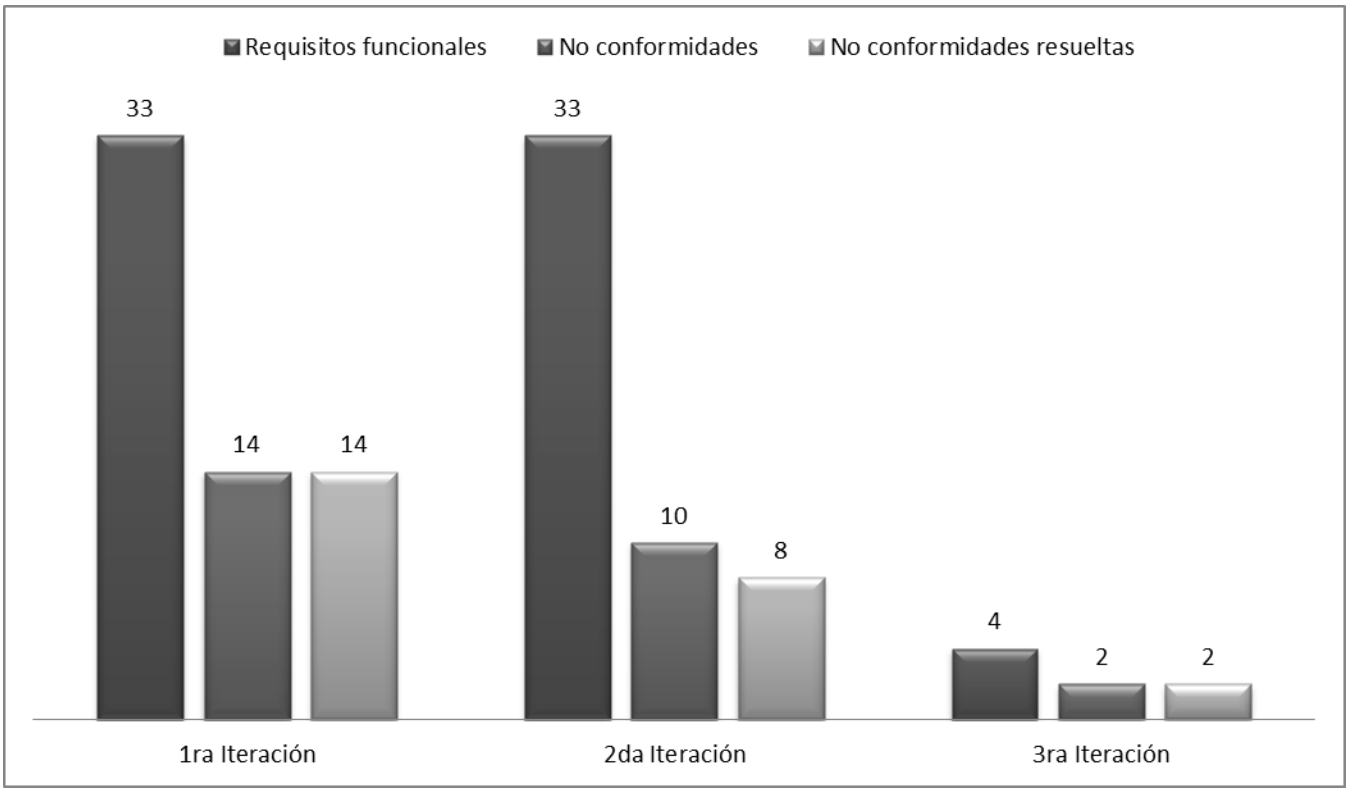


Fig. 22: Resultados de las pruebas aplicadas

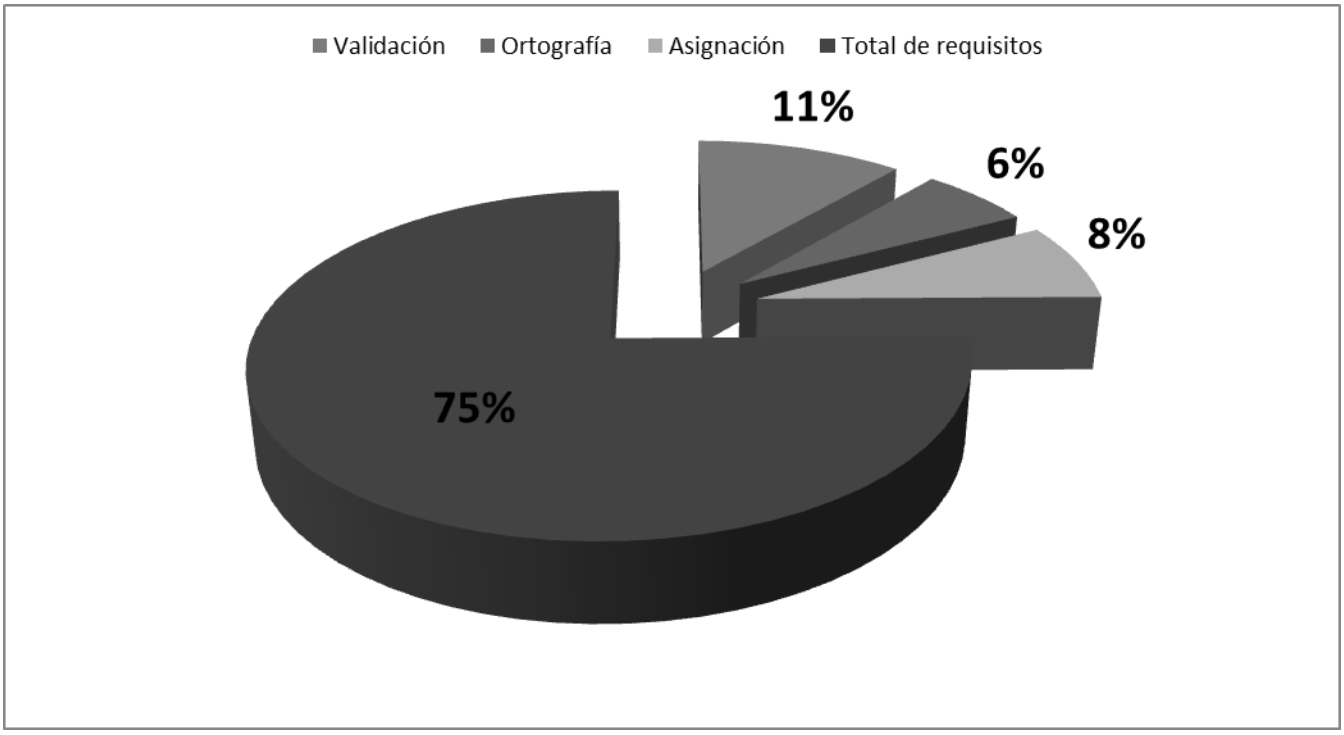


Fig. 23: Clasificación de las no conformidades

Conclusiones del capítulo 3

En este capítulo se arriban a las siguientes conclusiones:

- El estándar de código propuesto para el trabajo con el lenguaje Java, basado en el establecido en el centro CENIA, ayuda a obtener código con mayor organización, legibilidad y facilidad de mantenimiento.
- La confección del modelo de diseño facilita una mejor comprensión de la organización de los paquetes de la solución en las diferentes capas de la arquitectura.
- El desarrollo de las pruebas determina que los errores más comunes incurridos en el desarrollo son de tipo validación, los que fueron erradicados en las iteraciones de revisión.

Conclusiones

La investigación desarrollada y los resultados obtenidos permiten al autor plantear las siguientes conclusiones:

- Luego del análisis de la propuesta del analista y el estudio de herramientas homólogas, se realizaron cambios fundamentales a los requisitos a implementar.
- Se establecieron patrones y estándares de código para organizar la implementación, dándole mayor calidad al código.
- Se realizaron pruebas al sistema, verificando el desarrollo de una aplicación que cumple con las necesidades del proyecto.

Por lo antes expuesto, se considera que la valoración crítica y los aportes al análisis y la implementación de la propuesta descrita, son los principales aportes de este trabajo de diploma. La propuesta realizada sirve de base para la implementación de un sistema que apoye la toma de decisiones en el proceso de desarrollo de interfaces de usuario contribuyendo a una mayor usabilidad de las mismas.

Recomendaciones

El constante avance que experimenta el mundo del software es la premisa fundamental para continuar la perfección del sistema elaborado. Con este objetivo se recomienda para próximas versiones:

- mejorar la interfaz visual del producto y
- realizar un manual de usuario que brinde mayores facilidades de uso del sistema.

Bibliografía

Dia. Dia; [modified 2010]. Available from: http://live.gnome.org/Dia/Examples

Bizagi. Conceptos básicos; [modified 2011]. Available from: http://wiki.bizagi.com/es/index.php?title=Basic_concepts

Solutions, A.S. Diagramas de flujo; [modified 2002-2011a]. Available from: http://www.axure.com/flowdiagrams

León, R.R. La diagramación en la arquitectura de información. 2007.

Yusef, H.M. La Experiencia del Usuario. 2005.

Bustamante, A.M.d.O.S.d. Arquitectura de información y usabilidad: nociones básicas para los profesionales de la información. 2004.

Definicion ABC. Diagramas de flujo; [modified 2007]. Available from: http://www.definicionabc.com/comunicacion/diagrama-de-flujo.php

Sobre conceptos. Diagramas de flujos; [modified 2009]. Available from: http://sobreconceptos.com/diagrama-de-flujo

Vazquez, A.M. Diagramas de flujo; [modified 2009]. Available from: http://www.elprisma.com/apuntes/administracion_de_empresas/quesonlosdiagramasdeflujo/

Garrett, J.J. A visual vocabulary for describing information architecture and interaction design; [modified enero 2001]. Available from: http://www.jjg.net/ia/visvocab/

Solutions, A.S. Axure; [modified 2002-2011b]. Available from: http://www.axure.com/axurerpenvironment

Rowman, D. Manual de Bizagi. 2009. p. 12.

Microsoft, C. Microsoft Visio; [modified 2011]. Available from: http://office.microsoft.com/es-es/visio-help/

Project, P. pencil; [modified 2010]. Available from: http://pencil.evolus.vn/en-US/Home.aspx

Paradigm, V. vpuml; [modified 2011a]. Available from: http://www.visual-paradigm.com/product/vpuml/

Paradigm, V. Drawing Data Flow Diagram (DFD); [modified 2011b]. Available from: http://www.visual-paradigm.com/solution/drawdfd/

Dia. Dia; [modified 2010]. Available from: http://live.gnome.org/Dia

Barzanallana, R. Metodologías de desarrollo de software. 2008.

Amaro Calderón, S.D.; Carlos, V.R.J. Metodologías Ágiles [Investigación]. [Trujillo, Perú]: Universidad Nacional de Trujillo, Escuela de Informática; 2007.

Romero, G.M.P. “Trabajo de diploma: Metodología ágil para proyectos de software libre” [Habana]: Universidad de las Ciencias Informáticas.; 2008.

Mora, F. Lenguaje UML; [modified 2003]. Available from: http://www.dccia.ua.es/dccia/inf/asignaturas/GPS/archivos/Uml.PDF

Netbeans. Bienvenido a NetBeans; [modified 2007]. Available from: http://netbeans.org/index_es.html

Bibliografía

Zukowski, J. Java 2 J2SE 1.4.; 2003.

Larman, C. UML y Patrones. 970-17-0261-1 ed. Patience Hall, 1999.; 1999a.

Flower, M. Patterns of Enterprise Application Architecture: Addison Wesley; 2003.

Estándar _de _Código CENIA. Estándar de Código. . 2010.

Herrera, L.A.C., editor. La sindicación de contenidos: oportunidades y desventajas.: ACIMED; 2010.

Gamma, E., Booch, Grady. Patrones de diseño : elementos de software orientado a objetos reutilizable. 2002.

Larman, C. UML y Patrones: Patience Hall; 1999b.

I. Jacobson, G.B.y.J.R. El Proceso unificado de desarrollo de Software. Madrid: Addison Wesley 2000.

Pressman, R., editor. Ingeniería de Software Un Enfoque Práctico; 2005a. 958 p.

Bustamante, L.A.M.d.O.S.d. Arquitectura de información y usabilidad: nociones básicas para los profesionales de la información Ciudad de La Habana; nov.-dic. 2004.

Diseño, I.D.L.W.Y.P.D., editor. Ingenieria De La Web Y Patrones De Diseño. España: Pearson Educación; 2005.

Douglas Schmidt, M.S., Hans Rohnert, Frank Buschmann. Pattern-Oriented Software Architecture: Patterns for Concurrent and Networked Objects. Vol. 2; 2000. p. 1-482.

Falgueras, B.C. Ingenieria del software. In: UOC, editor. Barcelona, España; abril del 2033.

Gladys, M.P.R. “Trabajo de diploma: Metodología ágil para proyectos de software libre” [Habana]: Universidad de las Ciencias Informáticas.; 2008.

Martin, R.B.Y.C.R.L.y.J.V. Arquitectura de la informacion y usabilidad en la web. 2004:9.

Pressman, R. Estrategias De Pruebas De Software Convencionales; [modified 1998]. Available from: http://www.buenastareas.com/ensayos/Estrategias-De-Pruebas-De-Software-Convencionales/1045971

Pressman, R.S., editor. Ingeniería de Software Un Enfoque Práctico; 2005b. 958 p.

W3C. Extensible Markup Language (XML) 1.0. 10 February 1998.

Yusef Hassan Montero, F.J.M.F. LA experiencia del usuario. no solo usabilidad; 7 de septiembre de 2005.

Bibliografía

Bustamante, L.A.M.d.O.S.d. Arquitectura de información y usabilidad: nociones básicas para los profesionales de la información Ciudad de La Habana; nov.-dic. 2004.

Diseño, I.D.L.W.Y.P.D., editor. Ingenieria De La Web Y Patrones De Diseño. España: Pearson Educación; 2005.

Douglas Schmidt, M.S., Hans Rohnert, Frank Buschmann. Pattern-Oriented Software Architecture: Patterns for Concurrent and Networked Objects. Vol. 2; 2000. p. 1-482.

Falgueras, B.C. Ingenieria del software. In: UOC, editor. Barcelona, España; abril del 2033.

Gladys, M.P.R. "Trabajo de diploma: Metodología ágil para proyectos de software libre" [Habana]: Universidad de las Ciencias Informáticas.; 2008.

Martin, R.B.Y.C.R.L.y.J.V. Arquitectura de la informacion y usabilidad en la web. 2004:9.

Pressman, R.S., editor. Ingeniería de Software Un Enfoque Práctico; 2005b. 958 p.

W3C. Extensible Markup Language (XML) 1.0. 10 February 1998.

Yusef Hassan Montero, F.J.M.F. LA experiencia del usuario. no solo usabilidad; 7 de septiembre de 2005.

Perez, Meritxell Ramon. Evaluación de herramientas para prototipado de sistemas interactivos. Lleida : s.n., 2010.

Axure Software Solutions, Inc,2011. [En línea] 5 de 1 de 2002. Disponible en: http://www.axure.com. http://www.axure.com/features.

Valls, Ignasi Pérez. Orígenes de los patrones. 5 de 1 de 2009. Disponible en: http://www.javadabbadoo.org/cursos/infosintesis.net/javase/paqawt/selectorcolores/paso03patrones.html.

León, Rodrigo Ronda. [En línea] 2007. http://www.nosolousabilidad.com/articulos/tecnicas_ai.htm.

Sanchez de Bustamante, Antonio Montes de Oca. A.Arquitectura de información y usabilidad: nociones básicas para los profesionales de la información. 2004. Ciudad Habana : Acimed, 2004. Disponible en: http://bvs.sld.cu/revistas/aci/vol12_6_04/aci04604.htm.

Instituto Tecnológico de Hermosillo. El lenguaje Java. [En línea] 2009. http://eddi.ith.mx/Curso/Contenido/java.htm.

Celis, Ismael. [En línea] 2005. http://www.webtaller.com/maletin/articulos/el-ataque-de-los-frameworks.php.

Morville, Louis Rosenfeld Peter. Information Architecture for the World Wide Web, Third Edition. s.l : O´Reilly Media, 2006.

Sun Microsystems. Sun Developer Network. [En línea] 2010. http.//java.sun.com/javase/.

Grupo de Ingeniería del Software y Sistemas de Información. Ingeniería del Software y Sistemas de Información. [En línea] 2003. http://issi.dsci.upv.es/publications/archives/f-1069167248521/actas.pdf.

Lago, Ramiro. Patrones de diseño de software. 2007. Disponible en: http://www.proactiva-calidad.com/java/patrones/index.html.

Bibliografía

Marinilli, Mauro. Swing and SWT: A Tale of Two Java GUI Libraries. [En línea] 2006. http://www.developer.com/java/other/article.php/2179061/Swing-and-SWT-A-Tale-of-Java-GUI-Libraries.htm..

Mercer, Dave. Biblioteca.uci.cu. Fundamentos de Programación en XML. [En línea] 2001. http://bibliodoc.uci.cu/pdf/reg01311.pdf.958-41-0297-4.

Wells, Don. Extreme Programming: A gentle introduction. [En línea] 2009. http://www.extremeprogramming.org/rules.html..

Teo, Javier Moldes. Java 2 J2se 1.4 Guías Prácticas. s.l : Anaya Multimedia, 2003.

Wurman, Richard Saul. Information Architecture. Los Ángeles : Watson-Guptill Pubis, 1997.

Ágil. Metodología ágil. 2010. Disponible en: http://www.seccperu.org/files/Metodologias%20Agiles.pdf.

UML y Patrones. Introducción al análisis y diseño orientado a objetos. 2011.

García, Felipe U. Pérez. [En línea] 2008. http://www.larevistainformatica.com/tipo-lenguaje-programación.htm.

Brown, Dan. Communicating Desing: Developing Web Site Documentation and Planning. Berkeley : New Riders, 2007.

Montoro, Mario Pérez y Codina, Lluis. Software de prototipado para la arquitectura de la información: funcionalidad y evaluación. 2010.

Gutiérrez, Mario Pérez-Montoro. Arquitectura de la información en entornos web. Gijón : Trea, 2010. (En prensa).

Notación y estilo en programación. britiding.com. [En línea] http://www.bitriding.com/articulos/notacion-estilo-programacion.html.

Axure Software Solutions, Inc,2011. [En línea] 5 de 1 de 2002. Disponible en: http://www.axure.com. http://www.axure.com/features.

Valls, Ignasi Pérez. *Orígenes de los patrones.* 5 de 1 de 2009. Disponible en: http://www.javadabbadoo.org/cursos/infosintesis.net/javase/paqawt/selectorcolores/paso03patrones.html.

León, Rodrigo Ronda. [En línea] 2007. http://www.nosolousabilidad.com/articulos/tecnicas_ai.htm.

Sanchez de Bustamante, Antonio Montes de Oca. *A.Arquitectura de información y usabilidad: nociones básicas para los profesionales de la información. 2004.* Ciudad Habana : Acimed, 2004. Disponible en: http://bvs.sld.cu/revistas/aci/vol12_6_04/aci04604.htm.

Instituto Tecnológico de Hermosillo. *El lenguaje Java.* [En línea] 2009. http://eddi.ith.mx/Curso/Contenido/java.htm.

Celis, Ismael. [En línea] 2005. http://www.webtaller.com/maletin/articulos/el-ataque-de-los-frameworks.php.

Morville, Louis Rosenfeld Peter. *Information Architecture for the World Wide Web, Third Edition.* s.l : O´Reilly Media, 2006.

Sun Microsystems. Sun Developer Network. [En línea] 2010. http.//java.sun.com/javase/.

Bibliografía

Grupo de Ingeniería del Software y Sistemas de Información. Ingeniería del Software y Sistemas de Información. [En línea] 2003. http://issi.dsci.upv.es/publications/archives/f-1069167248521/actas.pdf.

Lago, Ramiro. *Patrones de diseño de software.* 2007. Disponible en: http://www.proactiva-calidad.com/java/patrones/index.html.

Marinilli, Mauro. Swing and SWT: A Tale of Two Java GUI Libraries. [En línea] 2006. http://www.developer.com/java/other/article.php/2179061/Swing-and-SWT-A-Tale-of-Java-GUI-Libraries.htm..

Mercer, Dave. Biblioteca.uci.cu. *Fundamentos de Programación en XML.* [En línea] 2001. http://bibliodoc.uci.cu/pdf/reg01311.pdf.958-41-0297-4.

Wells, Don. Extreme Programming: A gentle introduction. [En línea] 2009. http://www.extremeprogramming.org/rules.html.

Teo, Javier Moldes. *Java 2 J2se 1.4 Guías Prácticas.* s.l : Anaya Multimedia, 2003.

Wurman, Richard Saul. *Information Architecture.* Los Ángeles : Watson-Guptill Pubis, 1997.

Ágil. *Metodología ágil.* 2010. Disponible en: http://www.seccperu.org/files/Metodologias%20Agiles.pdf.

UML y Patrones. *Introducción al análisis y diseño orientado a objetos.* 2011.

García, Felipe U. Pérez. [En línea] 2008. http://www.larevistainformatica.com/tipo-lenguaje-programación.htm.

Brown, Dan. *Communicating Desing: Developing Web Site Documentation and Planning.* Berkeley : New Riders, 2007.

Montoro, Mario Pérez y Codina, Lluis. *Software de prototipado para la arquitectura de la información: funcionalidad y evaluación.* 2010.

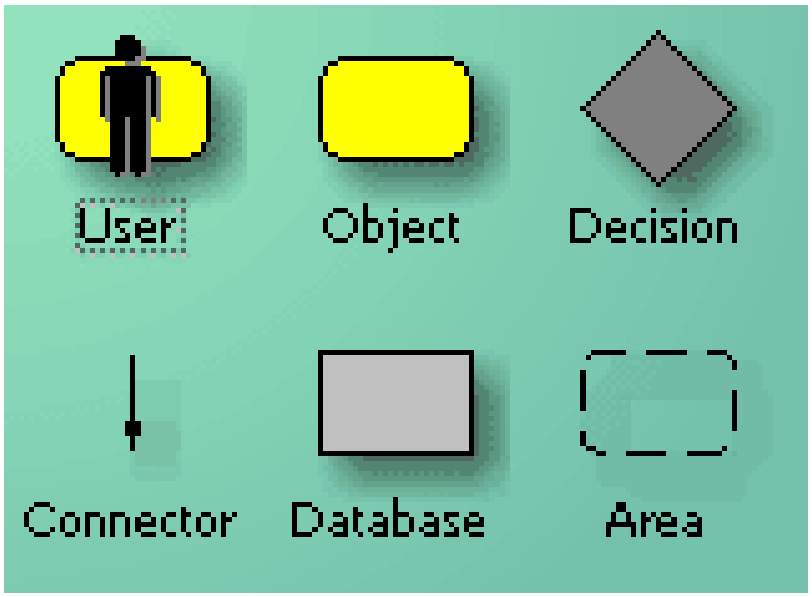
Gutiérrez, Mario Pérez-Montoro. *Arquitectura de la información en entornos web.* Gijón : Trea, 2010. (En prensa).

Notación y estilo en programación. britiding.com. [En línea] http://www.bitriding.com/articulos/notacion-estilo-programacion.html.

EcuRed: Enciclopedia cubana. [En línea] [Citado el: 5 de abril de 2012.] www.ecured.cu.

Anexos

Anexo 1: Alfabeto visual de Nick Finck.



Anexo 2: Componentes visuales propuestos por el analista.

Anexos

Elemento	Descripción
 *Rectángulo*	Este es rectángulo que se le puede modificar su tamaño y permite la inserción de texto en su interior. Además el elemento se puede mover por la escena.
 *Rectángulo redondeado*	Este es rectángulo redondeado que se le puede modificar su tamaño y permite la inserción de texto en su interior. Además el elemento se puede mover por la escena.
 *Rectángulo biselados*	Este es rectángulo biselado que se le puede modificar su tamaño y permite la inserción de texto en su interior. Además el elemento se puede mover por la escena.
 *Semicírculo*	Este es semicírculo que se le puede modificar su tamaño y permite la inserción de texto en su interior. Además el elemento se puede mover por la escena.
 *Triángulo*	Este es triángulo que se le puede modificar su tamaño y permite la inserción de texto en su interior. Además el elemento se puede mover por la escena.
 *Trapezoide*	Este es trapezoide que se le puede modificar su tamaño y permite la inserción de texto en su interior. Además el elemento se puede mover por la escena.
 *Elipse*	Este es elipse que se le que se le puede modificar su tamaño y permite la inserción de texto en su interior. Además el elemento se puede mover por la escena.
 *Hexágono*	Este es hexágono que se le puede modificar su tamaño y permite la inserción de texto en su interior. Además el elemento se puede mover por la escena.
 *Paralelogramo*	Este es paralelogramo que se le puede modificar su tamaño y permite la inserción de texto en su interior. Además el elemento se puede mover por la escena.

Anexos

 Actor	Elemento que sirve para la representación de una persona, se le puede modificar su tamaño, ancho y largo y permite la inserción de texto en su interior. Además el elemento se puede mover por la escena.
 *Base dato*	Elemento que sirve para la representación de una base de dato, se le puede modificar su tamaño, ancho y largo y permite la inserción de texto en su interior. Además el elemento se puede mover por la escena.
 *Imagen*	Permite cargar una imagen a la escena y modificarle su tamaño.
	Permite realizar nota