Universidad de las Ciencias Informáticas

Centro de Informatización Universitaria

Facultad 1

Trabajo de Diploma para optar por el Título de

Ingeniero en Ciencias Informáticas

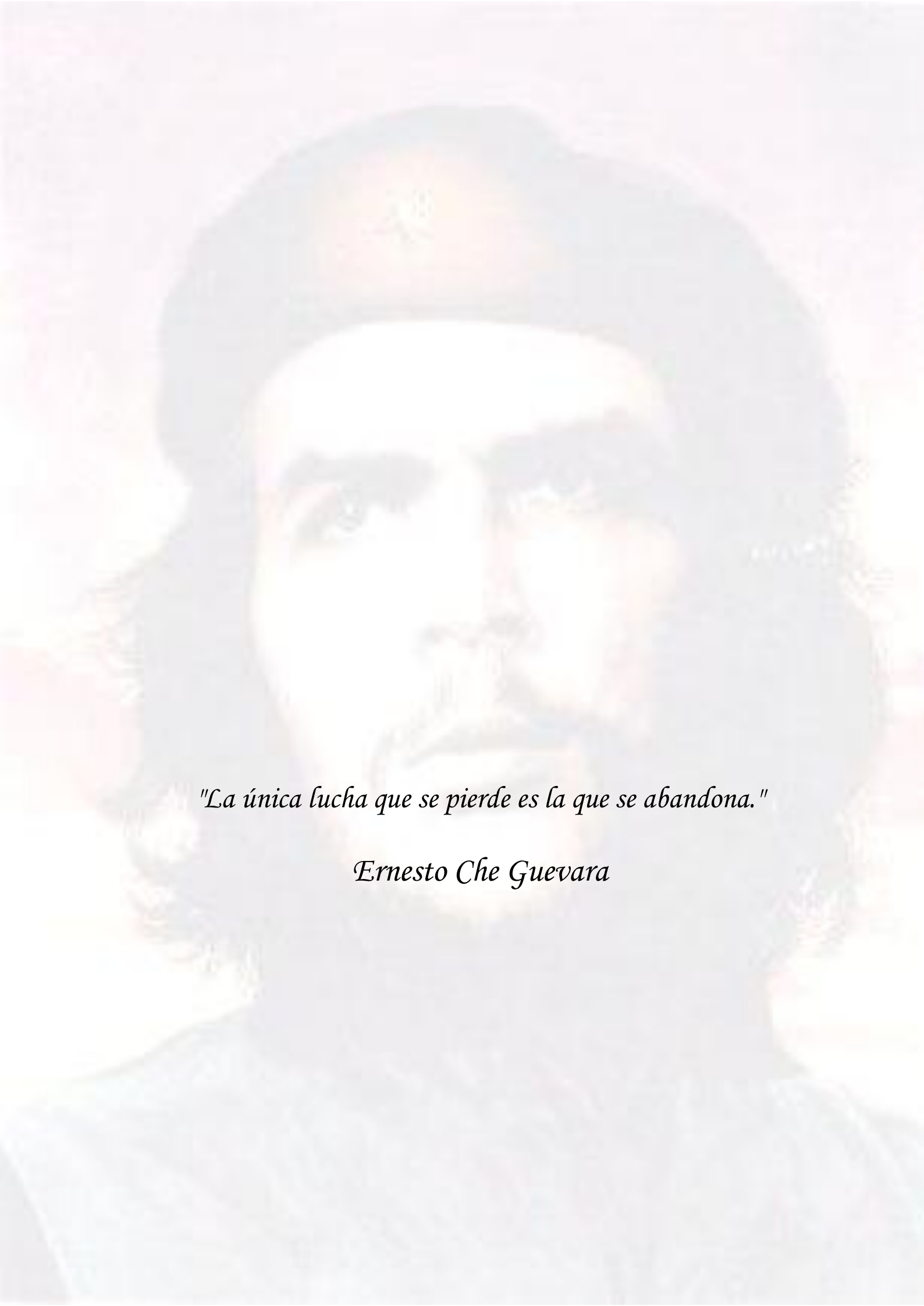
Título: Componentes para diagramas de organización en la herramienta de diagramación del paquete “Abad”.

Diplomante: Yadier Aviles Membribes

Tutores: Ing. Yanicet Aveleira Rodríguez
Ing. Malena Coneza Roig

Consultante: Ing. Yamilka Aviles Membribes

Ciudad de La Habana, Junio, 2012



"La única lucha que se pierde es la que se abandona."

Ernesto Che Guevara

Declaramos ser autor del trabajo de diploma “Componentes para diagramas de organización en la herramienta de diagramación del paquete “Abad” y confiero a la Universidad de las Ciencias Informáticas (UCI) los derechos patrimoniales del mismo, con carácter exclusivo.

Para que así conste firmamos el presente a los ____ días del mes de ____ del año **2012**.

_____
Yadier Aviles Membribes
Autor

_____
Ing. Yanicet Aveleira Rodríguez
Tutor

_____
Ing. Malena Coneza Roig
Tutor

Ing. Yanicet Aveleira Rodríguez

Ingeniera en Ciencias Informáticas, graduada en 2008. Perteneció al grupo de trabajo Arquitectura y Estándares de información de la Dirección Técnica de la Universidad de las Ciencias Informáticas. Actualmente es líder del Proyecto Gestión de Arquitectura de Información (Abad) del Centro de Informatización Universitaria (CENIA) de la facultad 1 de la propia universidad.

Correo: yaveleira@uci.cu

Ing. Malena Coneza Roig

Ingeniera en Ciencias Informáticas, graduada en 2010. Se desempeña como asesora de planificación y control del Centro de Informatización Universitaria (CENIA) de la facultad 1 de la propia universidad.

Correo: mconeza@uci.cu

A mis padres, por ser el mejor ejemplo que un hijo puede tener, por su sacrificio y dedicación, por guiarme a tomar el camino correcto en todo momento.

A mi hermana por ser mi guía tanto en lo profesional como en lo personal, por brindarme siempre su ayuda sin importar el sacrificio que tuviera que hacer.

A mi hermano, porque en eso se ha convertido mi cuñado, gracias por cuidar a mi hermana y mis padres en mi ausencia.

A mi familia en general por apoyarme en los momentos buenos y malos.

A Rosa por su cariño, apoyo y guía en todo momento, puedo decir que solo una madre puede dar tanto de forma desinteresada y eso es lo que ha sido para mí en estos años.

A mis tutoras por su paciencia, ayuda y dedicación durante el desarrollo de la investigación.

A todos aquellos que de una forma u otra han ayudado en mi formación como profesional y como persona.

A todos muchas gracias.

Yadier Aviles Membribes.

.

A mi mamá y papá por estar siempre atentos a mí, por sus consejos que han hecho de mí cada día mejor persona. A mi hermana por ser siempre mi guía.

El presente trabajo de diploma se centra en el diagrama de organización como parte de la técnica de diagramación que aplican los arquitectos de información en su trabajo. El objetivo está orientado a implementar los componentes[1] que se utilizan en este diagrama integrándolos a una herramienta que permita realizar todos (diagramas de presentación, funcionamiento y organización) los que se proponen en la técnica. Las herramientas existentes en la actualidad no lo hacen con el nivel de acabado necesario y en muchos de los casos son propietarias, obligando utilizar varias plataformas. El ciclo de desarrollo fue guiado por la metodología de desarrollo SXP 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 organización, diagramas de funcionamiento, diagrama de presentación

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

Introducción

Los sistemas de comunicación y trasmisión de información han evolucionado progresivamente a lo largo de la historia. Esta evolución ha sido influenciada principalmente por el desarrollo de Internet y otros adelantos científico-técnicos donde las nuevas tecnologías de la información desempeñan un papel importante que influye directamente en el orden económico y social del mundo. El crecimiento acelerado de la información a la que se puede acceder actualmente en Internet, ha provocado que entenderla sea un proceso relativamente complejo. Los contenidos en una gran parte de productos web, no tienen una correcta distribución espacial y atentando contra la buena asimilación y la comprensión de los usuarios finales, que son los más altos consumidores de los servicios de las redes.

En este marco surge una disciplina de estudio para intentar dar solución a estos problemas organizativos orientados a los clientes potenciales de los productos que se exhiben. Esta disciplina se acuñó como arquitectura de información, por la semejanza con la arquitectura tradicional donde se organiza en planos, la construcción de edificaciones con materiales como cemento, arena, piedra y algunas herramientas útiles para ello. En total analogía con esto, la mencionada disciplina se encarga de estudiar la organización de los espacios informacionales en los productos que se ofrecen a la masa consumidora.

La Arquitectura de Información (AI) surge como una disciplina para estudiar, analizar y organizar la disposición y estructuración de la información en cualquier medio, de tal forma que sea consultada por el usuario de manera intuitiva (Garrett, 2002). El término AI, fue utilizado por primera vez por Richard Saul Wurman en 1975 quien lo define como: “el estudio de la organización de la información con el objetivo de permitir al usuario encontrar su vía de navegación hacia el conocimiento y la comprensión de la información” (Fernández, 2003).

La AI en su ciclo de vida contempla las etapas de recolección, procesamiento, diseño y validación de contenidos. Dentro de estas etapas se encuentran un conjunto de técnicas, que responden a lo que se hace para lograr los procesos y las tareas. Entre ellas se puede mencionar el análisis documental, la arquitectura inversa, los cuestionarios, los mapas de contenidos, el ordenamiento de tarjetas, el análisis de homólogos, análisis de frecuencia, diseño participativo y diagramación, entre otras. La

técnica de diagramación consiste en la realización de diagramas que concreten las propuestas de diseño realizadas por los arquitectos de información. Estos diagramas ayudan tanto a las personas implicadas en la producción como a los usuarios. Se usan con el objetivo de que todas las personas conozcan y comprendan cómo será la estructura y funcionamiento del producto a realizar.

Para la realización de los diagramas existen diferentes aplicaciones, las que se identificarán y examinarán por el analista. Estos sistemas, orientados a apoyar la actividad de organización de los contenidos en diagramas representativos que servirán de base para la construcción del producto final, generalmente son productos con licencia propietaria, aunque también se pueden encontrar bajo licencias de software libre y de código abierto o dual. Los mismos posibilitan la interacción dinámica con los actores del proceso, presentando una interfaz intuitiva y accesible en la mayoría de los casos. Sin embargo, no permiten la integración en una misma herramienta de los diagramas que crea el arquitecto de información durante todo el proceso que para el efecto de este trabajo de diploma se orientarán en tres grupos: funcionamiento, organización y presentación de contenidos. Al no haber una aplicación libre que integre los tres diagramas, obliga al arquitecto de información a usar varias herramientas para este propósito, existiendo la dificultad de que ellas no siempre se ejecuten en el mismo sistema operativo, siendo más engorroso y llevando más tiempo realizar estos diagramas.

Con el objetivo de facilitar la integración de los diferentes tipos de diagramas mencionados, el proyecto Abad[2] que se dedica a desarrollar soluciones para soportar las técnicas de diseño de experiencia de usuario en apoyo al proceso de desarrollo software de la UCI, propone dentro de sus soluciones una herramienta de diagramación que resuelva la integración de los diagramas. La herramienta no cuenta con los componentes que permitan representar la organización de los contenidos a partir de los diagramas de organización.

De esta situación problemática se deriva el siguiente **problema de investigación** del presente trabajo: ¿Cómo facilitar la representación de la organización de los contenidos que propone el arquitecto de información para un producto?

[2] Superior de un monasterio que tiene categoría de abadía. Entre sus principales actividades estaba la custodia y organización de la información que guardaba el monasterio.

Para dar respuesta a esta interrogante el **objetivo general** que se define es: implementar los componentes para diagramas de organización en la herramienta de diagramación del paquete “Abad”, contribuyendo a mejorar la representación visual del contenido que hace el arquitecto de información en los distintos niveles de jerarquía y la relación existente entre ellos.

En atención a esto se ha determinado como **objeto de estudio**: el proceso de arquitectura de información **campo de acción:** la técnica de diagramación aplicada dentro del proceso de la arquitectura de información.

Para lograr determinar el cumplimiento del objetivo general del presente trabajo se definieron las siguientes tareas de la investigación:

- Análisis del estado del arte de la técnica de diagramación dentro de la AI.
- Estudio de herramientas homólogas existentes.
- Análisis de las herramientas, metodologías de desarrollo y estándares de codificación que van a ser utilizados durante el desarrollo de la propuesta.
- Análisis crítico de la propuesta de solución del analista.
- Análisis de los vocabularios visuales existentes.
- Descripción de la arquitectura del sistema.
- Definición del estándar de codificación a utilizar en la implementación.
- Implementación de las funcionalidades descritas en las historias de usuarios
- Aplicación de las pruebas para validar el funcionamiento de los componentes implementados.

Como **posible resultado** se espera la implementación de un módulo que permita presentar la organización los contenidos que propone el arquitecto de información en un producto determinado, mediante la incorporación de componentes para crear diagramas de organización en la herramienta de diagramación del paquete “Abad”.

Se utilizaron como **métodos de investigación científica**:

Analítico-sintético: a través de este método se pudo establecer conceptos, variantes y necesidades de la disciplina AI y en particular de la técnica de diagramación mediante la revisión bibliográfica lo que permitió el análisis de todos los elementos que utiliza el arquitecto de información en la modelación de los productos, esta acción posibilitó identificar y sintetizar las necesidades y características que presentan.

Histórico-lógico: se utilizó para el estudio de la trayectoria histórica de la disciplina de Arquitectura de Información y en especial la diagramación, lo que permitió elaborar una sucesión cronológica y argumentada que facilitó el estudio del objeto en el presente Trabajo de Diploma apoyándose en la lógica de todo el proceso que permite definir la línea de investigación según los antecedentes, características y necesidades del fenómeno estudiado, lo que posibilitó definir las carencias y dificultades de los productos analizados que motivan la realización de esta aplicación.

Observación: permitió conocer las características, detalles de funcionamiento y operaciones realizadas por el usuario en las aplicaciones existentes que sirven de apoyo a los Arquitectos de Información en la modelación de productos, en particular la técnica de diseño.

Modelación: a través de la creación de modelos, permitió representar de manera gráfica los contenidos que lo requiera, como los componentes, utilizando las herramientas necesarias para ello.

El siguiente documento consta de 3 capítulos, estructurados de la siguiente forma:

Capítulo 1 - Fundamentación teórica: abarca el estado del arte de las distintas aplicaciones que existen a en el mundo que se utilizan en la AI para la técnica de diagramación y específicamente en la creación de los diagramas de organización. Incluye además los principales conceptos utilizados en el trascurso de la investigación. Se define el ambiente de desarrollo. Finalmente, se exponen las conclusiones parciales del capítulo.

Capítulo 2 – Implementación de la solución propuesta: se realiza una valoración crítica del diseño propuesto por el analista y un refinamiento de la arquitectura. Se orienta a la descripción de la

implementación del módulo; mediante artefactos, implementación existente, componentes y estándar de códigos.

Capítulo 3 – Validación de la solución propuesta: se diseñan las pruebas que permiten validar la solución propuesta, se describen los valores utilizados para las pruebas y se evalúan los resultados obtenidos después de su aplicación.

Capítulo 1: Fundamentación teórica:

1.1 Introducción

En el siguiente capítulo se abordan los principales conceptos que se relacionan con el objeto de estudio del presente trabajo de diploma. Se hará un análisis de las necesidades existentes que dan al traste con el surgimiento de la disciplina de AI, con la técnica de diagramación y en particular los diagramas de organización; así como un análisis sobre las distintas herramientas y metodologías de desarrollo que se pueden utilizar en la implementación del módulo.

1.2 Arquitectura de la Información

La AI es “el estudio de la organización de la información con el objetivo de permitir al usuario encontrar su vía de navegación hacia el conocimiento y la comprensión de la información” (Fernández, 2003). Es la disciplina encargada del estudio, análisis, organización, disposición y estructuración de la información en espacios de información. Esta disciplina surge por la necesidad de distribuir correctamente los contenidos en los productos que se diseñen: de esta manera los usuarios del mismo lograrán asimilar y entender todo la información que se le ofrece. Actualmente el proceso de AI está estrechamente ligado al diseño de las interfaces para sitios web y aplicaciones de escritorio (Llano, 2010).

La AI es la habilidad de encontrar cosas fácilmente, no está limitada al contenido, tampoco está limitada a la Web. Esta habilidad es parte integral del diseño de sistemas que ayudan a las personas a encontrar lo que necesitan (Morville & Rosenfeld, 1999).

La AI trata de organizar la información, de manera que los usuarios puedan encontrar las respuestas correctas a sus interrogantes (Montes de Oca, 2004). Para esto se vale de varias técnicas entre las que se destacan los métodos de representación de la información y dentro de estos se encuentran el de diagramación o bocetado.

1.3 Diagramación

La diagramación dentro de la arquitectura de la información: es la organización de los contenidos del producto, su funcionamiento básico, y la ubicación que tendrán estos contenidos en la interfaz gráfica.

Un grupo de autores, como Dan Brown, Jesse James Garret, pioneros en los temas del diseño y representación del software, dividen estos diagramas en 2 tipos: (Brown, 2006), (Garret, 2001):

- Planos (Como se conoce en el idioma Inglés Blueprints).
- Maquetas (Denominada así en el idioma Inglés Wireframes).

El primer tipo de diagrama (planos) tiene como objetivo representar las principales áreas de organización y rotulado, están enfocados a los aspectos estructurales y de funcionamiento del producto. Generalmente se representan con textos, cajas y flechas. Su función es reflejar y facilitar la toma de decisiones en cuanto a diseño, manteniendo la comunicación constante de dichos cambios al resto del equipo de desarrollo.

Christina Wodtke conceptualiza los Blueprint como: "Un plano de diseño es justamente una buena idea llevada a la realidad a través de la escritura".

El segundo grupo (maqueta) tienen el objetivo de mostrar el contenido de las páginas, utilizando los elementos que se definieron en los planos y ubicándolos en las páginas o pantallas del producto final. Están comprendidos como prototipos de baja fidelidad, pues se realizan a escala de grises y no muestran el diseño gráfico del producto ni la funcionalidad de sus códigos de programación.

Los niveles asociados al prototipado son:

- Prototipos[3] de baja fidelidad o estáticos.
- Prototipos de fidelidad intermedia (diseño gráfico).
- Prototipos de alta fidelidad o dinámicos (Web, HTML).

[3] Son documentos, diseños o sistemas que simulan o tienen implementadas partes del sistema final, constituyen una herramienta muy útil para hacer participar al usuario en el desarrollo y poder evaluar el producto desde las primeras fases del desarrollo.

Un estudio del Lic. Rodrigo Ronda León, propone un sistema de diagramación donde son creados tres tipos de diagramas teniendo en cuenta las funciones que debe cumplir el arquitecto de información durante el diseño de un producto digital (León, 2007). Estos diagramas son:

- Diagramas de organización (planos - blueprints).
- Diagramas de funcionamiento (planos avanzados - blueprints).
- Diagramas de presentación (presentación - wireframes).

Está clasificación, no significa que un diagrama pueda sustituir a otro, sino todo lo contrario, deben estar relacionados entre ellos, para que cada diagrama complemente el anterior y sirva de referencia al siguiente. Esta categorización de los diagramas no significa que deben existir siempre los tres tipos diagramas.

Después de realizar un estudio sobre las diferentes formas de ejecutar la técnica de diagramación se ha decidido utilizar el sistema de diagramación del autor Rodrigo Ronda León que está basado en la creación de tres tipos de diagramas (organización, funcionamiento, presentación) ya que la mayoría de los arquitectos de información en la UCI para lo que en un principio se le orienta esta herramienta, utilizan un esquema estrechamente relacionado con los tres tipos expuesto anteriormente.

1.4 Diagrama de organización de contenidos

Los diagramas de organización: consisten en la representación de los grupos organizados y de los elementos básicos que contienen, siendo el diagrama básico para entender la estructura general del producto (León, 2007). Estos diagramas permiten analizar la estructura del producto representado y cumple con un rol informativo, al ofrecer datos sobre las características generales. Pueden incluir los nombres de los niveles más importantes del producto, para explicitar las relaciones jerárquicas y competencias vigentes.

De esta manera, los diagramas de organización deben representar de forma gráfica o esquemática los distintos niveles de jerarquía y la relación existente entre ellos. No tienen que abundar en detalles, sino que su misión es ofrecer información fácil de comprender y sencilla de utilizar.

Al confeccionar un diagrama de organización se tiene en cuenta los siguientes principios:

- Las unidades organizativas más importantes se presentan en la parte superior del gráfico y se añaden debajo en orden descendiente de importancia.
- Las unidades que dependen de un jefe común se colocan en un mismo nivel.
- Las unidades funcionales aparecen en un plano superior al de las ejecutivas cuando ambas pertenecen a un mismo jefe.
- En algunos casos se indica por debajo de cada unidad organizativa sus principales objetivos o tareas.

1.5 Aplicaciones que permiten realizar los diagramas de organización de contenidos

1.5.1 MindManager

Es una herramienta utilizada para la gestión de la información y el diseño de mapas mentales, flexible y fácil de usar. Los diagramas gráficos del MindManager comienzan con un tema central, y ofrece la posibilidad de agregar ramas con ideas, notas, imágenes, tareas e incluso hipervínculos y ficheros adjuntos. Esto genera resultados predecibles, de forma que cualquiera persona pueda comprender el panorama general. El MindManager mejora los procesos de trabajo y la documentación visual de ideas. Es muy utilizado por los arquitectos de información para la tormenta de ideas y los mapas de navegación ayudando en el proceso de toma de decisiones. Publica mapas en formatos PDF para una distribución independiente de plataforma.

Sin duda una aplicación interesante pero presenta inconvenientes como su precio, además no es multiplataforma. Tienen restringido el acceso a su código fuente, por tanto, no permite su integración a las otras herramientas que conforma el paquete de herramientas “ABAD”.

1.5.2 Axure

Se trata de una solución de escritorio en inglés y de pago. Axure es un software líder para la diagramación de aplicaciones y sitios web. Permite crear diagramas de organización (blueprint), diagramas de presentación de los contenidos (wireframes) y prototipos. Diseñado para especialistas en arquitectura de información, diseño de interacción y experiencia de usuario. El Axure, permite exportar los diseños realizados tanto a un documento como a un prototipo con niveles de interacción. Está disponible tanto para plataforma Windows como para Mac (A. S Solutions, 2002-2011).

Incorpora una biblioteca estándar formada para el prototipado web que permite la gestión y edición de sus elementos y la incorporación de nuevos componentes gráficos externos. Se destaca, además, por permitir anotaciones a pie de página, la edición colaborativa con un sistema de control de versiones y la creación de prototipos dinámicos y navegables.

Al ser una aplicación bajo licenciamiento propietario[4], que no es multiplataforma y tienen restringido el acceso a su código fuente no se pueden integrar a las otras herramientas que conforma la plataforma "ABAD".

1.5.3 FreeMind

Es una herramienta para la elaboración y manipulación de mapas conceptuales. Es decir, una herramienta para organizar y estructurar las ideas, los conceptos, su relación entre ellos y su evolución. Se utiliza como mecanismo o forma de plasmar tormentas de ideas de todo tipo para su posterior reutilización.

Está basado en Java y es de código abierto, bajo licencia GPL[5]. Es un software sencillo de instalar, configurar y utilizar. Al ser Java es multiplataforma y, por lo tanto, disponible en Windows y GNU/Linux. Su requerimiento más importante es tener instalado el entorno de ejecución Java JRE 1.4 o superior. Los proyectos generados los guarda como lenguaje de marcas extensibles (XML, Extensible Markup Language) garantizando así su reutilización (FreeMind, 2012).

Presenta la dificultad de que no integra los tres tipos de diagramas (organización, funcionamiento y presentación) que crea el arquitecto de información durante todo el proceso. Además, no puede integrarse fácilmente a arquitectura que propone la herramienta de diagramación que desarrolla el proyecto Abab.

1.5.4 CMap Tools

Es un programa libre que funciona correctamente en sistemas operativos Windows, Linux y Mac. Los productos hechos en este software pueden ser salvados en forma de imagen, como entorno Web y

[4] Es cualquier programa informático en el que el usuario tiene limitaciones para usarlo, modificarlo o redistribuirlo

[5] Licencia pública general.

documento portátil. Los mecanismos de elaboración del mapa conceptual son fáciles y cómodos; acompañados de una interfaz comunicativa (Cmap, 2011), presenta funcionalidades como:

1. Los mecanismos de elaboración del mapa conceptual (inserción y eliminación de conceptos y enlaces) son fáciles y cómodos.
2. Interfaz comunicativa en cuanto a las funcionalidades disponibles.
3. Los conceptos pueden hacerse acompañar de una imagen.
4. Al enlace es posible incorporarle flechas indicativas de la dirección del enlace.
5. Posibilidad de vincular recursos a los conceptos.

Presenta la dificultad de que no integra los tres tipos de diagramas (organización, funcionamiento y presentación) que crea el arquitecto de información durante todo el proceso. Además, no puede integrarse fácilmente a arquitectura que propone la herramienta de diagramación que desarrolla el proyecto Abab.

Existen muchas aplicaciones en el mundo que se utilizan para realizar la técnica de diagramación pero las mayorías son propietarias y no son multiplataforma impidiendo la integración con las herramientas que conforman la plataforma “ABAD”. Además, se necesita que integren los tres tipos de diagramas que crea el arquitecto de información durante todo el proceso. El estudio de estas herramientas permitió identificar:

- Como se aplica de la técnica de diagramación en el mundo.
- La necesidad de integrar los tres tipos de diagramas que crea el arquitecto durante todo el proceso de esta técnica.
- Sirvió como punto de partida para el levantamiento de los requisitos.
- Se identificaron elementos para el diseño de la herramienta.
- Se identificaron componentes necesarios para el diagrama de organización.

Por estos motivos se decidió crear una aplicación de escritorio y compatible con las otras herramientas de dicha plataforma. Esta herramienta debe permitir al arquitecto realizar los tres tipos de diagrama

que son creados durante todo el proceso de diagramación, presentando todas las funcionalidades para realizar un correcto diagrama de organización.

1.6 Metodologías de desarrollo de software.

Las metodologías de desarrollo de software (SW) indican cómo hay que obtener los distintos productos parciales y finales. Definen un conjunto de procesos y actividades que guían a los desarrolladores de software durante el ciclo de vida de un proyecto, especifican los artefactos a construir y organizan el personal involucrado en roles dentro del equipo de trabajo.

1.6.1 SXP

La metodología SXP está compuesta por las metodologías XP (Schenone, 2004) y SCRUM (Alaimo, 2010), donde utiliza XP para el desarrollo del software y a SCRUM para la gestión del proceso de desarrollo. Utiliza SCRUM como forma de gestionar un equipo, de manera que trabaje de forma eficiente y de tener siempre medidos los progresos, de forma que sepamos por dónde andamos. La metodología XP se utiliza para el desarrollo; consiste en una programación rápida o extrema, cuya particularidad es tener como parte del equipo, al usuario final, pues es uno de los requisitos para llegar el éxito del proyecto.

La metodología de desarrollo SXP la propone el centro para guiar todo el proceso teniendo las siguientes características:

Consta de 4 fases principales:

- **Planificación-Definición:** donde se establece la visión, se fijan las expectativas y se realiza el aseguramiento del financiamiento del proyecto;
- **Desarrollo:** es donde se realiza la implementación del sistema hasta que esté listo para ser entregado;
- **Entrega:** puesta en marcha;
- **Mantenimiento:** donde se realiza el soporte para el cliente.

Cada iteración se decidió que tendría una duración de 2 meses. El tiempo puede reducirse en la medida que se vaya refinando el producto. Estimula el trabajo en equipo, de manera que los

integrantes sigan una misma dirección, el mismo objetivo, conociendo con claridad el avance de las tareas a realizar, de forma que las personas que dirigen el proceso puedan supervisar el progreso eficiente del trabajo. Típicamente es un equipo de entre 5 y 10 personas cada una especializada en algún elemento que conforma los objetivos a cumplir.

La metodología SXP está orientada a los proyectos que le ocurran 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. Los miembros solo pueden cambiar entre sprint (no durante).

Consiste en una programación rápida o extrema, cuya particularidad es tener como parte del equipo, al usuario final, pues es uno de los requisitos para llegar al éxito del proyecto (Peñalver, Meneses, & S.García, 2010).

SXP propone los siguientes roles para el trabajo en equipo:

- Líder del proyecto.
- Gerente.
- Especialista.
- Cliente.
- Consultor.
- Miembros del equipo.

El equipo del proyecto será conformado por otros roles como:

- Programadores.
- Analista.
- Diseñadores.
- Encargado de prueba.
- Arquitecto.

Durante el proceso de desarrollo del módulo de diagramación el equipo de trabajo va a estar integrado por menos de 5 personas (1 analista y 4 desarrolladores) y se espera que no ocurran cambios en el equipo de desarrollo, por lo que no se requiere que el proceso esté sólidamente documentado. Pueden

ocurrir cambios frecuentes de los requisitos y se cuenta con la presencia constante del cliente, formando parte del grupo de trabajo, por lo que la metodología SXP sé adaptada perfectamente a las necesidades de proyecto ABAD para el desarrollo del módulo de diagramación.

1.7 Aplicaciones web y aplicaciones de escritorio.

En los inicios de la computación, solo existía aplicaciones del tipo consola, posteriormente aparecieron las aplicaciones de escritorio basadas en GUI, y como toda evolución, en los años 90 con el nacimiento de Internet fueron surgiendo lo que hoy conocemos como aplicaciones web. Las diferencias responden a la forma en la cual un software se distribuye, a la manera en que se produce el flujo de datos en el mismo, el modo de instalación, y de procesamiento de los datos.

Se denominan aplicaciones web a aquellos sistemas que los usuarios pueden utilizar accediendo a un servidor web a través de Internet o de una intranet mediante un navegador. En otras palabras, es una aplicación software que se codifica en un lenguaje soportado por los navegadores web en la que se confía la ejecución al navegador (Ailonwebs, 2009). Una aplicación de escritorio a diferencia de las aplicaciones Web consiste en un software almacenado y ejecutado directamente en las computadoras de sus usuarios.

Se definió crear una aplicación de escritorio porque no siempre los usuarios que deseen utilizar la herramienta de diagramación pueden contar con conexión en Internet.

1.8 Lenguaje de Programación.

Los lenguajes de programación son una herramienta que nos permite comunicarnos e instruir a la computadora para que realice una tarea específica. Cada lenguaje de programación posee una sintaxis y un léxico particular, es decir, forma de escribirse que es diferente en cada uno por la forma que fue creado y por la forma que trabaja su compilador para revisar, acomodar y reservar el mismo programa en memoria (Pérez García, 2008).

1.8.1 Java.

La solución propuesta será integrada con las herramientas que conforman la plataforma “ABAD” y están desarrollada en Java por lo cual se definió que la solución será implementada con dicho

lenguaje. Java es un lenguaje orientado a objeto. Permite crear sistemas informáticos en una plataforma y ejecutarlo en otra distinta de la inicial. Se pueden realizar aplicaciones como un procesador de palabras, una hoja que sirva para cálculos o una aplicación gráfica, etc. Es un lenguaje de programación que ofrece la potencia del diseño orientado a objetos con una sintaxis fácilmente accesible y un entorno robusto y agradable. Existen dentro de su librería clases gráficas como awt y swing, las cuales permiten crear objetos gráficos comunes altamente configurables y con una arquitectura independiente de la plataforma (Salamanca, 1999).

Se propone utilizar Java como lenguaje de programación para el desarrollo de los componentes para diagramas de organización en la herramienta de diagramación del paquete “Abad”. Presenta características tales como: que las soluciones informáticas creadas en él son multiplataforma, cuenta con librerías bien documentadas para la gestión de interfaces de usuario, facilita la escalabilidad hacia la Web y permite el diseño y construcción de aplicaciones de múltiples propósitos una de las grandes ventajas que presenta este lenguaje por la diversidad de aplicaciones que va a tener incorporada el paquete de herramientas “Abad”.

1.9 Marcos de trabajos.

Es una estructura de soporte definida mediante la cual otro proyecto de software puede ser desarrollado y organizado. Puede incluir soporte de programas, bibliotecas y un lenguaje interpretado entre otros programas, para ayudar a desarrollar y unir los diferentes componentes de un proyecto. Los marcos de trabajo son construidos en base a lenguajes orientados a objetos, esto permite una mejor modularización de los componentes y óptima reutilización de código. Además, en la mayoría de los casos un marco de trabajo implementará uno o más patrones de diseño de software que aseguren la escalabilidad del producto (Celis, 2005).

Marcos de trabajos para la gestión de interfaces de usuario en Java

1.9.1 XForms

Es un formato XML para construir interfaces de usuario Web, principalmente formularios Web. Fue diseñado para ser la generación siguiente de HTML HyperText Markup Language (Lenguaje de Marcado de Hipertexto) y XHTM eXtensible HyperText Markup Language (Lenguaje Extensible de

Marcado de Hipertexto). Propuesta es la creación de una aplicación de escritorio por lo que este marco de trabajo queda descartado.

1.9.2 Standard Widget Toolkit (SWT)

Es un framework para crear interfaces gráficas en Java, que crea a través de JNI (Java Native Interface) interfaces gráficas nativas del Sistema Operativo en donde ejecutemos nuestra aplicación SWT. Aunque está implementado en Java se basa en componentes nativos de cada sistema operativo, lo que conlleva a que cada implementación sea única para cada plataforma. SWT fue creado por la Fundación Eclipse y pensada para el desarrollo de Eclipse IDE, hay controles o widgets que no consideraron, ya que Eclipse no los ocupa. SWT varía su rendimiento según sea la plataforma utilizada y demanda de una gestión explícita de la destrucción de objetos, por lo que no será utilizado para el desarrollo de esta solución.

1.9.3 AWT/Swing

Para la creación de interfaces visuales Java utiliza los paquetes AWT/Swing. Swing abarca componentes como botones, tablas, marcos, etc. La biblioteca AWT brinda componentes de entrada y salida de datos. Los componentes Swing no están asociados al SO nativo, esto quiere decir que las propias librerías Java pueden emular cualquier entorno gráfico de cualquier Sistema Operativo. Esto le hace ser independiente de cualquier plataforma y AWT se encarga de la gestión de eventos y Swing brinda nuevos componentes. Considerando lo antes mencionado se utilizará el marco de trabajo AWT/Swing por estar orientado a la construcción de aplicaciones de escritorio que requieran de una considerable gestión de interfaces de usuario. Es nativo de Java por lo que para su uso no es necesario adquirir licencia alguna.

1.10 Ambiente de Desarrollo Integrado (IDE).

Un IDE es un entorno de programación que ha sido empaquetado como un programa de aplicación, consiste en un editor de código, un compilador, un depurador y un constructor de interfaz gráfica. Proveen un marco de trabajo amigable para los lenguajes de programación.

1.10.1 Netbeans.

Es un IDE Integrated Development Environment (Entorno de Desarrollo Integrado) escrito en Java de código abierto, gratuito para desarrolladores de software. Ofrece todas las herramientas necesarias para crear aplicaciones profesionales, empresariales, web y móviles con el lenguaje Java. La plataforma NetBeans es una base modular y extensible usada como una estructura de integración para crear aplicaciones de escritorio de gran tamaño. Ofrece servicios comunes permitiéndole al desarrollador enfocarse en la lógica específica de su aplicación.

La implementación del módulo se realizará sobre el IDE NetBeans basado en las funcionalidades que ofrece para escribir, compilar, depurar y ejecutar programas escritos en Java. Es distribuido de manera libre, sin restricciones de uso. Se ejecuta sobre varias plataformas (Windows, Mac OS X, GNU/Linux y Solaris). Facilita la gestión de las interfaces de usuario, la administración de las configuraciones del usuario y gestión de almacenamiento.

1.11 Patrones

En la tecnología de objetos un patrón es una descripción de un problema y la solución, a la que se le da nombre, y que se puede aplicar a nuevos contextos. En el caso de los patrones arquitectónicos proponen un esquema organizativo estructural para los sistemas informáticos (KRUCHTEN, 1995).

1.11.1 Patrón Modelo-Vista-Controlador

El patrón arquitectónico Modelo Vista Controlador (en lo adelante MVC) separa los datos de una aplicación, la lógica de la misma y las interfaces de usuario, en tres estructuras distintas las cuales tienen definidas reglas de comunicación. El Modelo representa la estructura de datos. Típicamente sus clases de modelo contendrán funciones que lo ayudarán a recuperar, insertar y actualizar información en su base de datos. La Vista se encarga de visualizar de una manera entendible los datos obtenidos desde el modelo. El Controlador sirve como un intermediario entre el Modelo y la Vista. Adoptar MVC como patrón de arquitectura proporciona una buena organización del código generado, su reutilización y flexibilidad, ofrece un punto de entrada único para toda la aplicación y reduce el tiempo necesario para modificar y añadir mejoras a una aplicación. Es muy apropiado para aplicaciones Web (Valls, 2009).

1.11.2 Patrón arquitectónico en 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. La programación por capas es un estilo de programación en el que el objetivo primordial es la separación de la lógica de negocios de la lógica de diseño; un ejemplo básico de esto consiste en separar la capa de datos de la capa de presentación al usuario. Una variante de este patrón muy utilizada es la de 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, 1999).

Se adoptó la utilización del patrón arquitectónico de tres capas, debido que permite estructurar mucho más las aplicaciones que se desarrolla, además de depurar más fácilmente los errores que van surgiendo a medida que la aplicación va evolucionando. Es factible y cómodo para desarrollar varias personas al mismo tiempo dentro de un equipo, proporcionando una distribución clara del trabajo entre los miembros de un equipo de desarrollo. Proporciona una buena organización y reutilización del código generado. Además, es la arquitectura que se adoptó para el desarrollo de la solución general.

1.12 Lenguaje Unificado de Modelado (UML)

Lenguaje unificado de modelado en el campo de la ingeniería de software que constituye un estándar de uso general. Es importante resaltar que UML no es una guía para realizar el análisis y diseño orientado a objetos, es decir, no es un proceso. UML es un lenguaje que permite la modelación de sistemas con tecnología orientada a objetos. También hay que tener en cuenta que UML es independiente del lenguaje de implementación en que se desarrolle el producto. Permite verificar y validar lo que ha sido modelado para automatizar procesos. La finalidad de los diagramas es presentar diversas perspectivas de un sistema, a las cuales se les conoce como modelo. Es importante destacar que un modelo UML describe lo que supuestamente hará un sistema, pero no dice cómo implementar dicho sistema (Jacobson, Boch, & Rumbaugh, 2000).

Funciones del UML:

- Visualizar: permite expresar de una forma gráfica un sistema de forma que otro lo puede entender.
- Especificar: permite especificar cuáles son las características de un sistema antes de su construcción.
- Construir: a partir de los modelos especificados se pueden construir los sistemas requeridos.
- Documentar: los propios elementos gráficos sirven como documentación del sistema desarrollado.

Se propone utilizar este lenguaje pues permite modelar sucesos de la vida real con mucha facilidad y además brinda la posibilidad de adaptar el sistema a cualquier lenguaje de programación, factor que lo hace reutilizable.

1.13 Herramienta de modelado.

Cuando se habla de herramientas de modelado en la disciplina de ingeniería del software, es importante mencionar las herramientas CASE (Computer Aided Software Engineering en español Ingeniería de Software Asistida por ordenador). Estas están tomando cada vez más relevancia en la planeación y ejecución de proyectos que involucren sistemas de información, pues suelen inducir a sus usuarios a la correcta utilización de metodologías que le ayuden a llegar con facilidad a los productos de software construidos (Menéndez, 2011).

1.13.1 Visual Paradigm

Es una herramienta muy completa y fácil de usar. Utiliza el lenguaje de modelado estándar UML, diseñada para realizar el modelado visual de software que utiliza la programación orientada a objetos. Fue creada para el ciclo vida completo del desarrollo del software que lo automatiza y acelera, permitiendo la captura de requisitos, análisis, diseño e implementación. Es una herramienta colaborativa, es decir, soporta múltiples usuarios trabajando sobre el mismo proyecto; permite control de versiones; facilita la organización, visualización, diseño, integración y despliegue mediante diagramas. La mayor ventaja que tiene el Visual Paradigm es que es multiplataforma y por lo tanto puede correr tanto sobre plataformas libres como sobre plataformas privativas. También presenta como una ventaja fundamental que a través de ella se pueden realizar prototipos de interfaz de usuarios que permiten tener una visión más cercana de cómo quedarían las interfaces del sistema (Paradigm, 2010).

1.14 Conclusiones del capítulo

Con la realización de este capítulo se concluye que:

- El estudio del estado del arte permite conocer como se aplica la técnica de diagramación en el mundo.
- La definición de los conceptos asociados al problema y los conceptos de AI, de la técnica de diagramación y del diagrama de organización permite un mejor entendimiento del negocio.
- El estudio de las herramientas homólogas permite demostrar la importancia del trabajo al integrar los tres diagramas que crea el arquitecto durante todo el proceso de la técnica de diagramación en una sola herramienta.
- La fundamentación de la metodología de desarrollo, los lenguajes y las herramientas propuestas permiten un mejor desarrollo de la solución propuesta.

Capítulo 2 – Implementación de la solución propuesta.

2.1 Introducción

En el presente capítulo se aborda: la valoración crítica del diseño propuesto por el analista y los patrones de diseños utilizados. Se realiza un refinamiento de la arquitectura. Se orienta a la descripción de la implementación del módulo; mediante artefactos, clases rehusadas, componentes y estándar de códigos.

2.2 Valoración crítica del análisis

El analista y diseñador del sistema ha realizado una propuesta de acuerdo con el desarrollo que se quiere llevar a cabo. Con esta propuesta comienza la fase de implementación de la versión 1.0 de la aplicación, estudiando el diseño entregado por el analista (Izquierdo, 2011). Se realiza un examen crítico de las funcionalidades necesarias y las clases persistentes relacionada con estas para el correcto funcionamiento de la aplicación, verificando que la descripción esté bien clara y comprobar que se puedan desarrollar, para la hora de implementarlas se pueda cumplir con lo pensado por el analista. Se determinó que el diseño es correcto pudiendo así continuar con la fase de elaboración de la solución informática.

El analista entregó 28 historias de usuarios, de las cuales se definió la prioridad para implementarlas en 13 de alta, 8 de media y 7 de baja prioridad. Esta prioridad fue dada a medida que se identificaba cuales eran las funcionalidades más importantes a la hora del desarrollo de la solución informática. 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 adicionaron y modificaron varias funcionalidades tales como:

- Tener más de un proyecto abierto a la vez, permitiendo al usuario trabajar en varios proyectos.
- La funcionalidad de guardar el proyecto fue modificada, donde se le incorporó la opción que permita guardar un diagrama por separado sin necesidad de tener que guardar el proyecto completo.

- Guardar todos los proyectos que ha creado el usuario.
- Salvar y cargar los proyectos en formato XML permitiendo en este formato utilizarlo para realizar otras nuevas funcionalidades.
- Salvar los diagramas en formato JPEG para que el usuario pueda ver el resultado del diagrama.
- Mover los proyectos hacia arriba o abajo permitiendo al usuario tener el proyecto que más utilice en la parte superior del árbol de proyecto.

2.3 Patrones de diseño

Un patrón de diseño es: una solución estándar para un problema común de programación, siendo más práctico a la hora de describir ciertos aspectos de la organización de un programa y conexiones entre componentes de programas.

2.3.1 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 diseño de objetos para la asignación de responsabilidades. Constituyen un apoyo para la enseñanza que ayuda a entender el diseño de objeto esencial y aplica el razonamiento para el diseño de una forma sistemática, racional y explicable (Kuchana, 2004).

En el desarrollo de esta solución se aplicaron los patrones GRASP: Experto, Controlador, No Hables con Extraños, Bajo Acoplamiento, Alta Cohesión y Polimorfismo.

- 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 nos presenta la oportunidad de reutilizar los componentes en futuras aplicaciones.
- El patrón Controlador: asigna la responsabilidad de recibir o manejar los mensajes de evento del sistema.

- 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.
- El patrón Alta Cohesión aconseja mantener la clase lo más cohesionada posible para controlar la complejidad de la misma.
- El patrón No Hables con Extraños ayuda a asignar las responsabilidades, de modo que se desconozca la estructura de los objetos indirectos.
- El patrón Polimorfismo recomienda la definición de un método homónimo en la clase generalización y en cada subclase con un comportamiento específico (Larman, 1999).

2.3.2 Patrones GoF

Son patrones de diseño creado por “La Pandilla de los Cuatro” (Gang of Four, GoF). Se utilizarán los patrones Fábrica y Fachada.

- Fábrica: su propósito es crear objetos, permitiendo al sistema identificar que clase se debe instanciar en tiempo de ejecución.

- 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, 1999)

2.4 Estándares de codificación

Es prudente establecer estándares de codificación para todos los programadores. Estos estándares consisten en estilos de codificación a la hora de escribir el código. Mantener un estándar en el código que escribimos es muy recomendable, nos ayuda a mantener proyectos ordenados y de fácil lectura para el resto de las personas. Luego, es una buena práctica que sigas siempre los consejos de un buen estándar.

La legibilidad del código fuente repercute directamente en el entendimiento que pueda tener otro programador del mismo, aspecto crucial ya que todo software tiene que someterse constantemente a mantenimiento y mejora de sus funcionalidades. El mejor método para lograr que un grupo de desarrolladores mantenga un código de calidad es establecer un estándar de codificación sobre el cual se realizarán revisiones rutinarias (Estándar_de_Código, 2010).

Existen diferentes estándares de código, uno de ellos es la notación Camel, esta consiste en escribir los identificadores con la primera letra de cada palabra en mayúsculas y el resto en minúscula. Se llama notación Camel porque los identificadores recuerdan las jorobas de un camello. Existen dos variantes:

- UpperCamelCase, CamelCase o PascalCase: en esta variante la primera letra también es mayúscula.
- LowerCamelCase, camelCase o dromedaryCase: la primera letra es minúscula.

Al conformar un estándar de codificación deben tenerse en cuenta varios criterios entre los cuales se pueden mencionar:

- Nombres representativos para clases, variables y procedimientos: se definen nombres de variables, controles y procedimientos de forma representativa con el propósito de facilitar el entendimiento.

- Indentación (sangrías) y espacios apropiados en el código: visualizar el código fuente puede resultar complicado pero haciendo uso de la indentación se obtiene una mejor visibilidad pues se muestran las líneas que están sujetas a otras.

- Documentación del código (poner comentarios para aclarar): es una buena práctica documentar aquellas secciones de código más complicadas e inusuales para que pueda ser comprendida posteriormente.

- Procedimientos coherentes: no debe añadírsele demasiada complejidad a los procedimientos para que sean más fácil de entender y no tengan una alta probabilidad de ocurrencia de errores.

Se especifican a continuación los criterios de estandarización de código utilizados para el desarrollo del módulo de Diagramación.

2.4.1 Nomenclatura de las clases, variables y procedimientos

Para las clases y procedimientos se establece UpperCamelCase que define que la primera letra de cada una de las palabras es mayúscula.

```
 * @author Yadier
 */
public class ComponenteDibujablePanel extends ComponenteDibujable
{
```

Figura 1: Ejemplo de nomenclatura UpperCamelCase

Para las variables se establece lowerCamelCase donde es igual que la anterior con la excepción de que la primera letra es minúscula y su nombre debe guardar relación con el valor que almacena.

```
    private BarraComponentes barraComponente;
    private BarraHerramienta herramienta;
```

Figura 2: Ejemplo de nomenclatura lowerCamelCase

2.4.2 Indentación

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. La razón es que los tabuladores se muestran con distinta anchura en función del editor de texto utilizado, y porque el código que mezcla tabuladores con espacios en blanco es bastante difícil de leer.

```
    public DefaultMutableTreeNode AdicionarProyecto ()
    {
        int c = getCantProy() + 1;
        String pagina = "Proyecto" + c;
        return AdicionarProyecto(pagina);
    }
```

Figura 3: Código Indentado

2.4.2.1 Saltos de línea

- Añadir un salto de línea después del cierre de los paréntesis de los parámetros.
- Añadir un salto de línea después de un punto y coma, cuando termina la sentencia.

2.4.2.2 Espacios y líneas en blanco

- Usar espacios en blanco para mejorar la legibilidad del código.

- Usar espacios en blanco en ambos lados del operador de símbolos, después de comas y después de las declaraciones.

- Usar líneas en blanco para separar trozos de código.

2.4.3 Normas de comentarios

Se debe comentar todo lo que se haga dentro del desarrollo, establecer las pautas que conlleven a lograr un código más legible y reutilizable y así se pueda aumentar la posibilidad de mantenimiento a lo largo plazo. Deben ser claros y precisos de forma tal que se entienda el propósito de los que se está desarrollando.

```
    /*Este método permite Focalizar un componente, pasado por parámetro
     * si desea Focalizarlo o no.
     */
    public void Focalizarse(boolean foco) {
        if (foco)
        {
            getComponenteVisual().setBorder(getColorDelBorde());
            getComponenteVisual().repaint();
            if (getComponenteVisual().getParent() != null)
                getComponenteVisual().getParent().repaint();
        }
        else
        {
            getComponenteVisual().setBorder(null);
        }
        seleccionado = foco;
    }
```

Figura 4: Ejemplo de comentario

2.4.4 Estructura de control

Entre la estructura de control (if, for, foreach, while, switch), y los paréntesis debe de existir siempre un espacio. Se recomienda utilizar siempre llaves de apertura y cierre. Esto aumenta la legibilidad y disminuye la probabilidad de errores lógicos.

```
private void EliminarPestanna()
{
    if (getListaProyectos().get(indiceProyActivo).getListaPestanna(
    {
        JOptionPane.showMessageDialog(null, "La página seleccionads
    }
    else
    {
        int opcion = JOptionPane.showOptionDialog(null, "¿Está segu
        if (opcion == 0)
        {
            String nombrePag = barraComponente.getDirectorio().getSe
            barraComponente.getDirectorio().EliminarNodo(null);
            area.getTabPestannas().EliminarPagina(nombrePag);
            getListaProyectos().get(indiceProyActivo).EliminarLaPest
        }
    }
}
```

Figura 5: Estructura de control

2.4.5 XML (Lenguaje de Etiquetado Extensible)

Es un lenguaje muy simple, pero estricto que desempeña un papel fundamental en el intercambio de una gran variedad de datos. Es un lenguaje muy similar a HTML, pero su función principal es describir datos y no mostrarlos como es el caso de HTML. XML es un formato que permite la lectura de datos a través de diferentes aplicaciones.

Los esquemas XML definen los elementos y atributos presentes en un documento y son la base de la organización de la información del mismo (Mercer, 2001). La solución debe recuperar información de ficheros XML, por lo que es necesario un esquema para garantizar la organización de la información en el archivo. A continuación se expone un ejemplo de código XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<Diagrama>
    <Organizacion>
        <nombre>Organización</nombre>
        <comentario>Este diagrama consisten en la representación de los grupos organizados
        <Componente>
            <nombreClase>Presentacion.Componentes.ComponenteDibujablePanel</nombreClase>
            <nombreComponente>ComponentePanelB</nombreComponente>
            <comentario>La ComponenteB es la encargada de realizar .....</comentario>
            <colorDelFondo>Blanco</colorDelFondo>
            <colorDelBorde>Negro</colorDelBorde>
            <textoMostrar>Texto</textoMostrar>
        </Componente>

        <Componente>
            <nombreClase>Presentacion.Componentes.ComponenteDibujableNodo</nombreClase>
            <nombreComponente>Nodo</nombreComponente>
            <comentario>Nodo del diagrama</comentario>
            <colorDelFondo>Blanco</colorDelFondo>
            <colorDelBorde>Negro</colorDelBorde>
            <textoMostrar>Texto</textoMostrar>
        </Componente>
</Diagrama>
```

Figura 6: Ejemplo de código XML

2.5 Modelo de diseño

Para una mejor comprensión y organización de la solución teniendo en cuenta la utilización del patrón arquitectónico de tres capas se han organizado las clases en tres paquetes principales[6]: Presentación, Lógica del negocio y Acceso a datos. Cada paquete 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 construido sobre la capa que la precede. 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.

[6] En los diagramas que se presentan en este epígrafe los nombres de los paquetes y las clases no están descritos como se hacen en la especificación técnica (código de la herramienta), respetando los estándares propuestos. Fueron escritos respetando las normas del español para lograr un mayor entendimiento de los diagramas

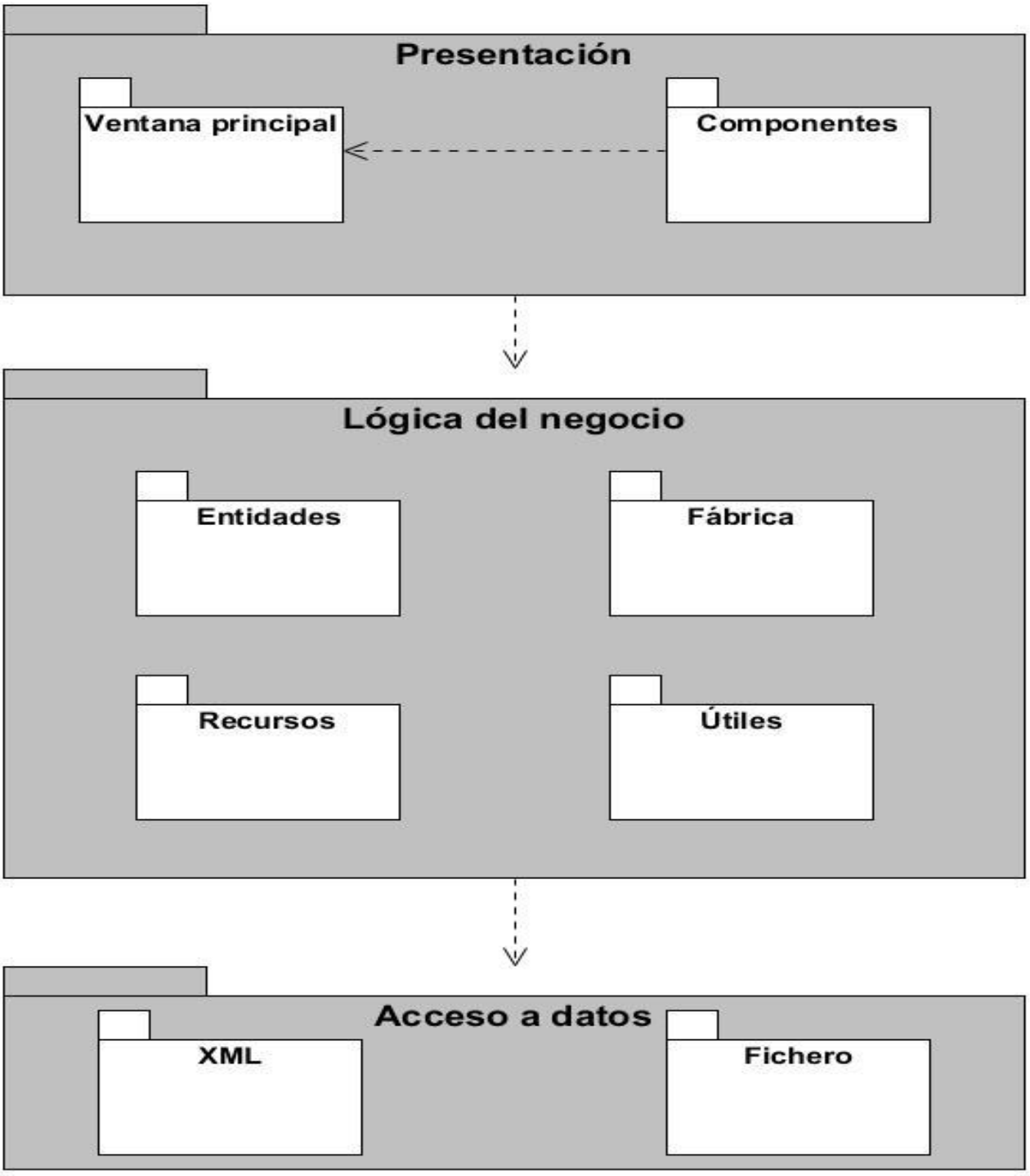


Figura 7: Diagrama de paquetes del sistema

2.5.1 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. La interfaz gráfica de usuario consiste en un conjunto de componentes empleados por los 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 (figura 7) queda dividida en paquetes según las funcionalidades que brinde cada uno. Los paquetes empleados son Ventana principal (figura 8) y Componentes (figura 9).

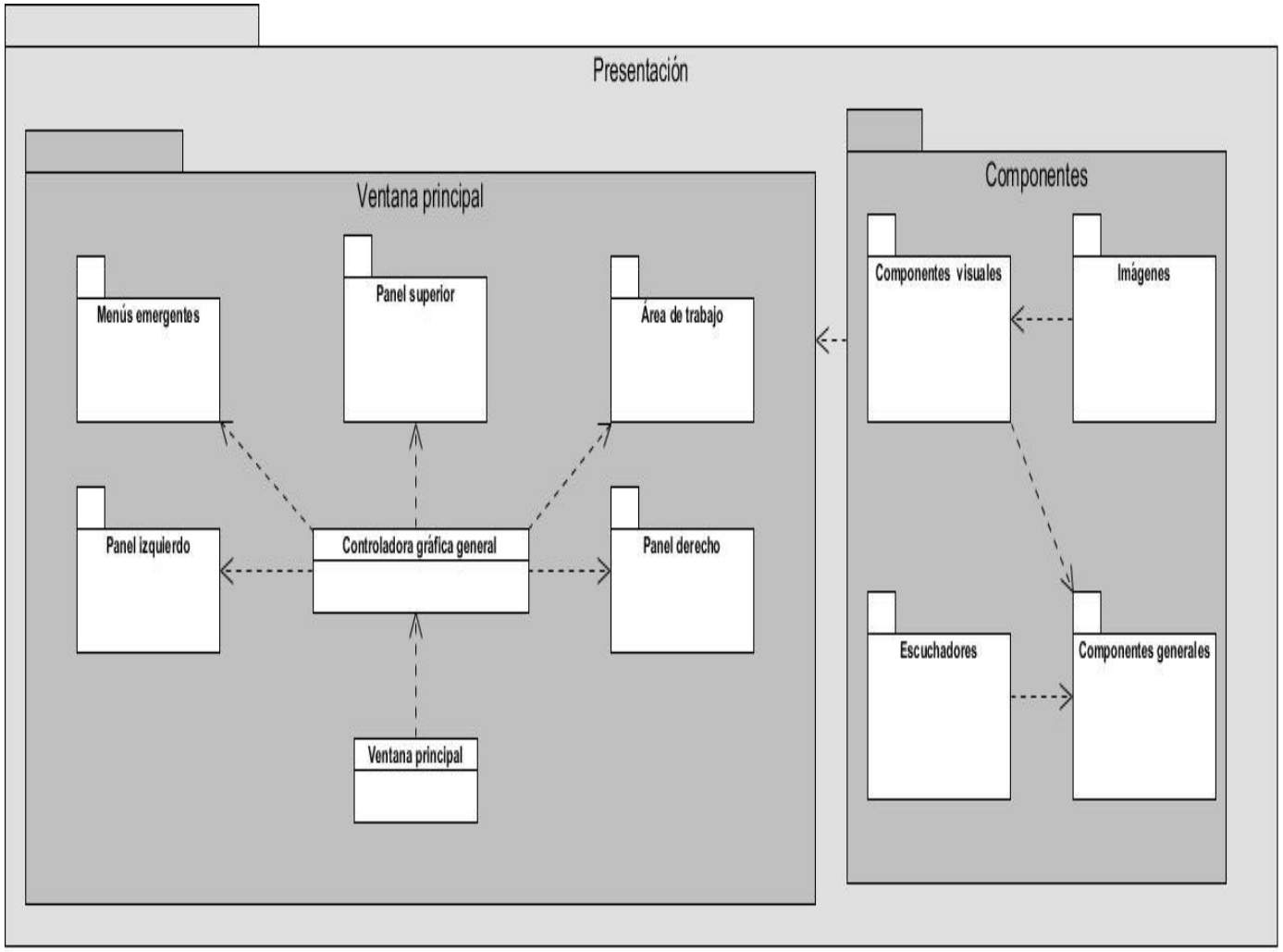


Figura 8: Presentación

El paquete Ventana Principal contiene la interfaz gráfica principal donde el usuario interactúa con el sistema mediante acciones y eventos realizados.

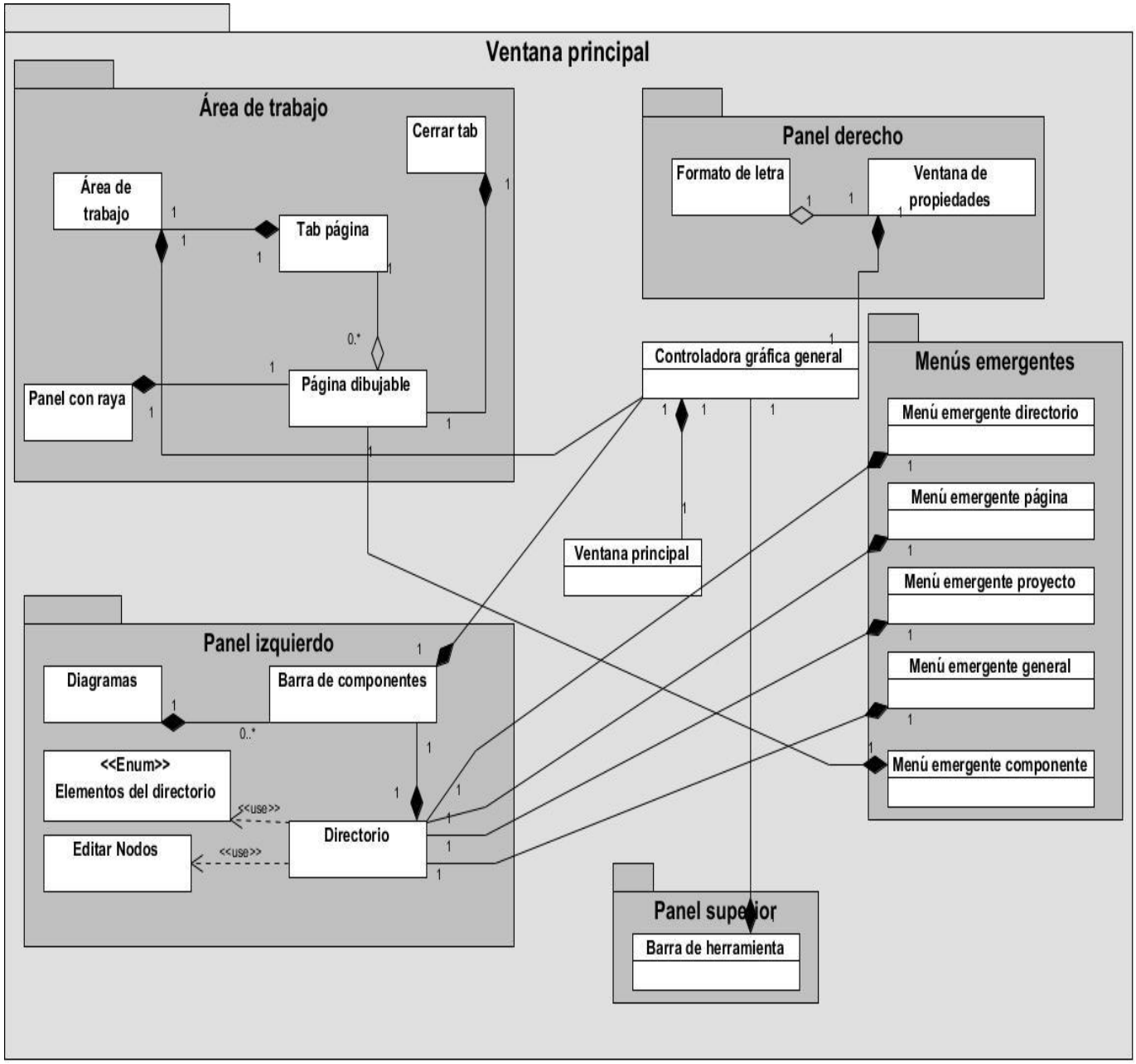


Figura 9: Presentación. Ventana principal

El propósito del paquete Componentes consiste en agrupar todos los funcionamientos de los componentes creados, por ejemplo moverse, redimensionarse, etc.

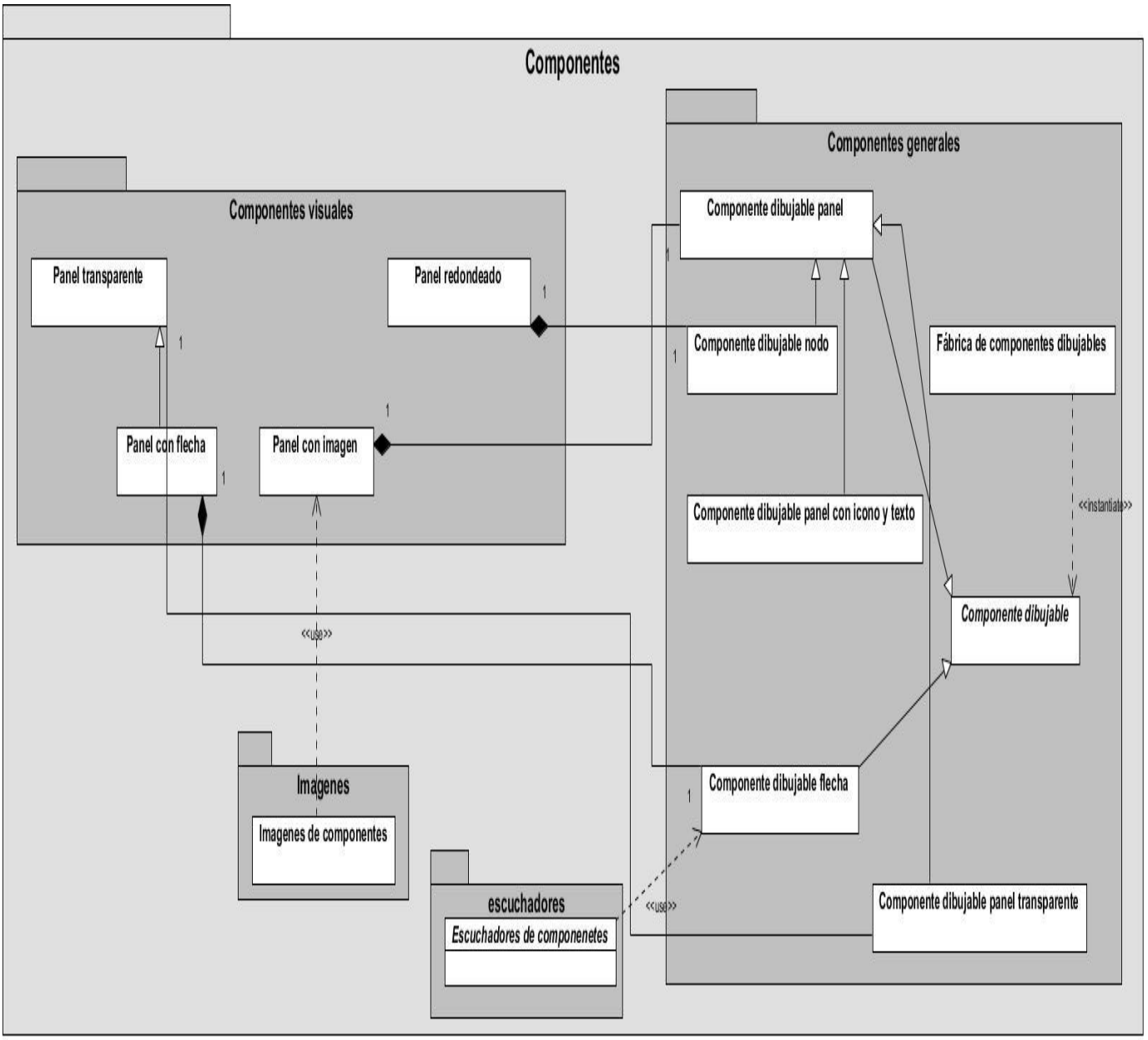


Figura 10: Presentación. Componentes

2.5.2 Lógica del negocio

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 con la capa de datos para solicitar almacenar los datos en ficheros o en archivo XML o recuperar datos de él. En el diseño orientado a objetos la capa Lógica del

negocio se divide en otras menos densas. En esta solución el paquete de Lógica del negocio (figura 10) está dividida en cuatro paquetes: Entidades, Fábrica, Recursos y Útiles.

El paquete Entidades es la base de la capa, En este paquete está la clase Controladora general que gestiona toda la información de Proyecto, Página y Componentes donde se comunica con la capa de Acceso a datos. El paquete Fábrica contiene la clase Fábrica de objeto que a través del nombre de una clase crea una instancia de ella, su propósito es crear objetos, permitiendo al sistema identificar que clase se debe instanciar en tiempo de ejecución. El paquete Recursos permite cargar la configuración de los componentes que se utilizarán en la barra de componente para la diagramación, dicha configuración será descrita en un fichero XML, esto permite que a medida que se implementen los componentes de cada diagrama se integre con a la aplicación, solo es necesario llenar el archivo de Configuración XML donde se especifica el diagrama a que pertenece el componente y el nombre, el componente debe estar en la capa Presentación y en el paquete Componentes. El paquete Útiles contiene un conjunto de clases que agrupan funcionalidad utilizadas en varias clases.

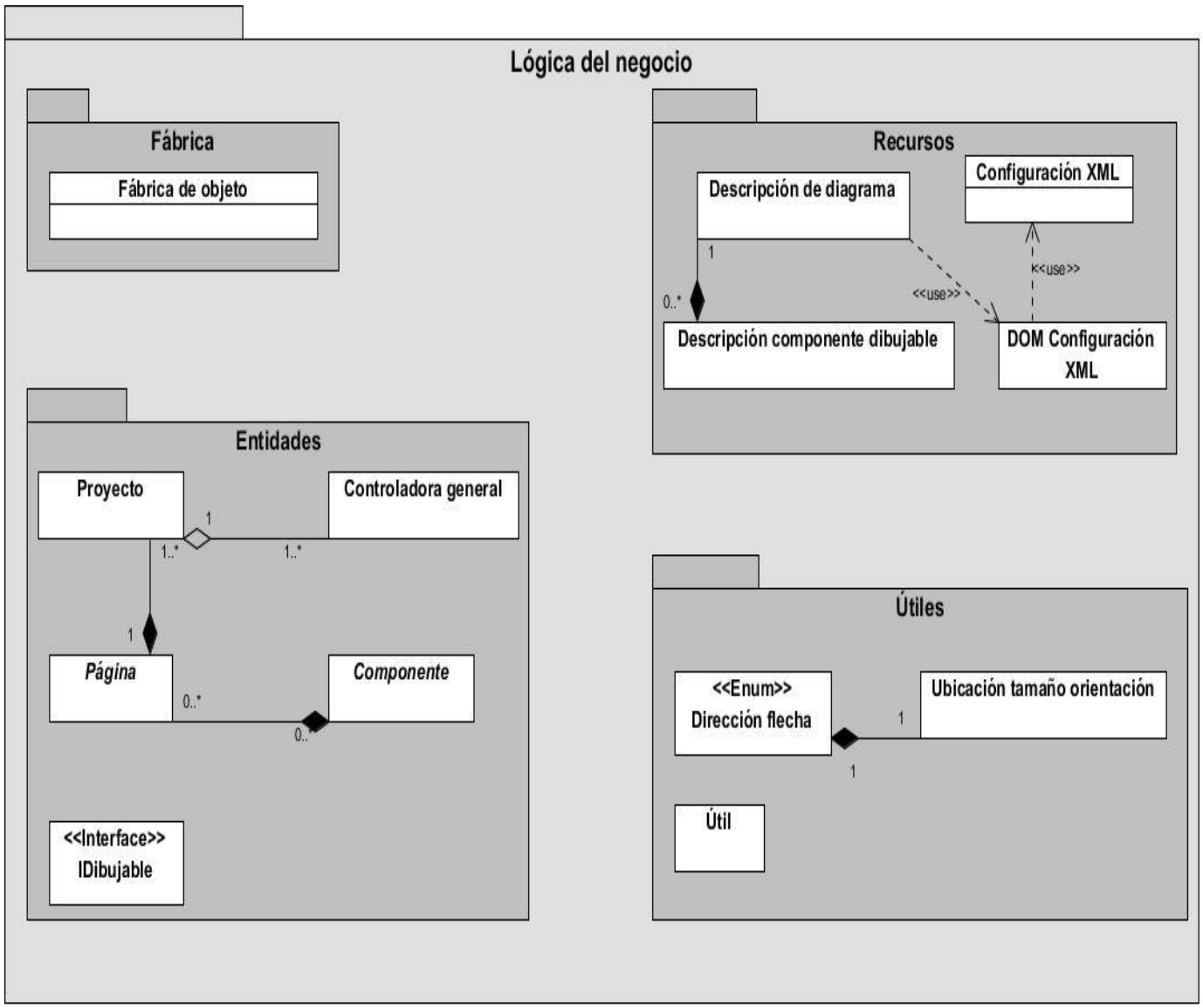


Figura 11: Lógica del negocio

2.5.3 Capa de 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 de los diagramas realizados. Estos ficheros pueden contener los diagramas en forma de objeto serializado o de documento XML. El paquete de Acceso a datos (figura 11) 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.

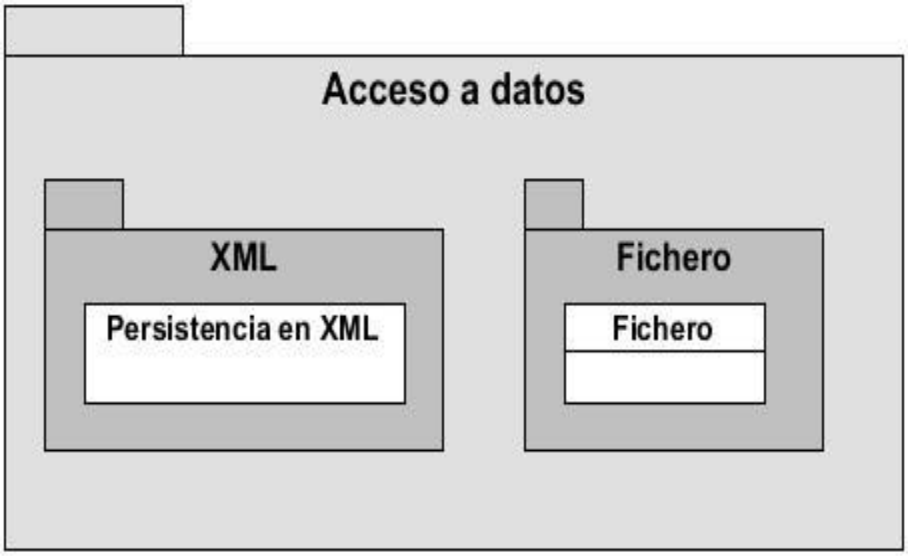


Figura 12: Acceso a datos

2.6 Modelo de despliegue

Es un modelo que muestra la disposición de las particiones físicas del sistema de información y la asignación de los componentes software a estas particiones. Es decir, las relaciones físicas entre los componentes software y hardware en el sistema a entregar (Jacobson, Boch, & Rumbaugh, 2000). La presente solución es una aplicación para entornos de escritorio por lo que será ejecutada en una estación de trabajo, el ordenador debe contar con 100 MB libres en el disco duro como mínimo, 256 MB de memoria RAM y tener instalado la máquina virtual de Java. A continuación, la figura 12 muestra el modelo de despliegue de la solución.

Figura 13: Modelo de despliegue

2.7 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" (Jacobson, Boch, & Rumbaugh, 2000).

A continuación se muestra el modelo de implementación de la solución.

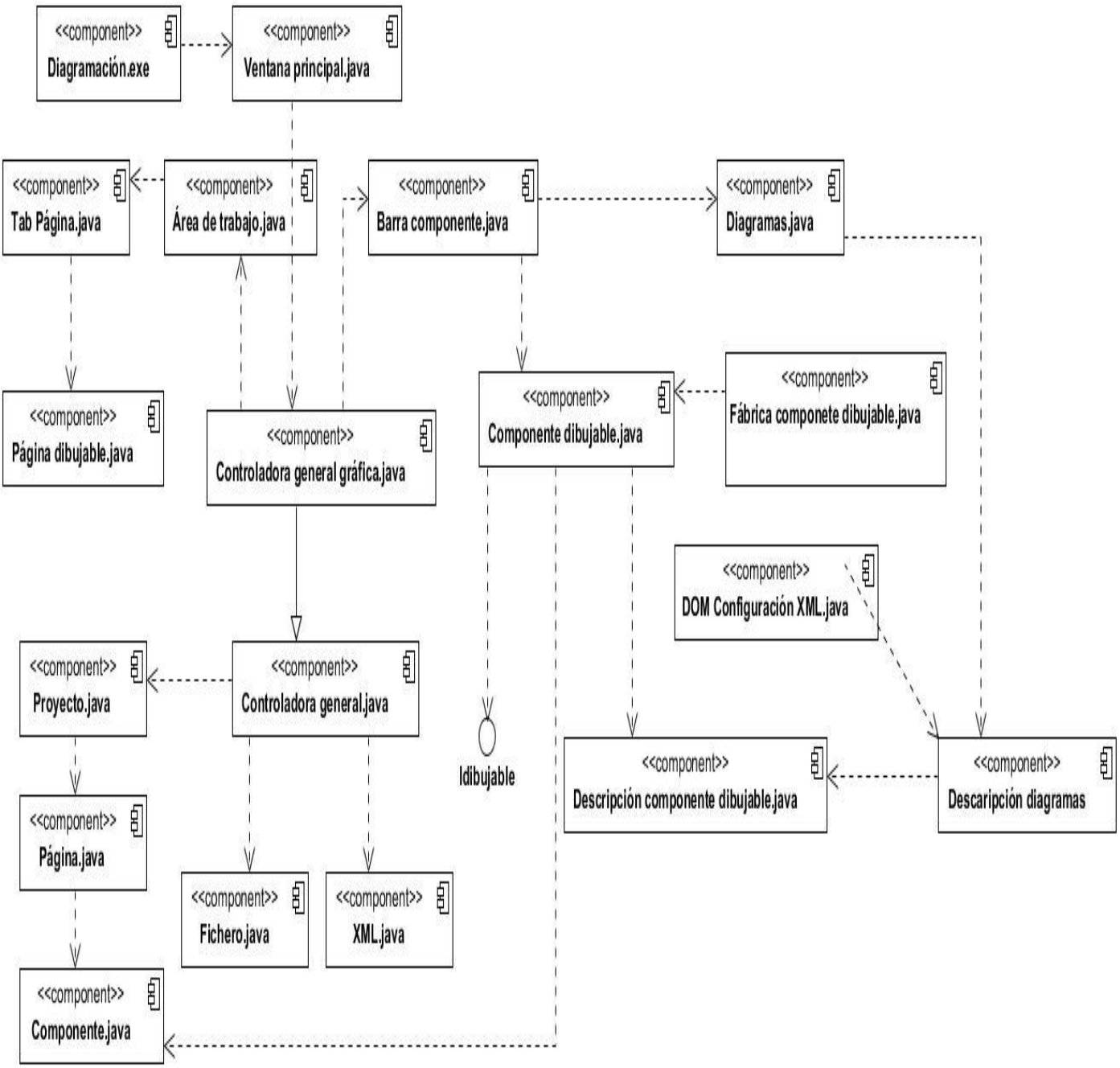


Figura 14: Diagrama de componentes

2.8 Conclusiones del capítulo.

Con la realización de este capítulo se concluye que:

- La modificación realizada al análisis y diseño permite obtener el resultado deseado, contribuyendo a mejorar el proceso de implementación.
- La definición de de los patrones de diseños y de arquitectura permite la obtención de un diseño exitoso.
- El esbozo de los módulos utilizados en la solución con las funcionalidades principales facilitó el entendimiento de la arquitectura del sistema.

Capítulo 3 – Validación de la solución propuesta.

3.1 Introducción al capítulo

Durante el desarrollo del software se puede cometer muchos errores por lo que este proceso debe estar acompañado de alguna actividad que garantice el rendimiento y funcionalidad del producto que se le brinda al usuario quedando así lo más libre posible de fallas. En este capítulo se aborda el tema referente a la realización de las pruebas; se realiza un análisis del tipo de prueba a utilizar para validar el software y el diseño de casos de prueba. En este capítulo también se crean los casos de prueba para las historias de usuario que se han implementado donde se detallan las mismas con sus descripciones generales, condiciones de ejecución, y secciones a probar.

3.2 Pruebas de software

La etapa de pruebas es una de las fases del ciclo de vida de los proyectos. Se la podría ubicar después del análisis, el diseño y la programación, pero esto depende del proyecto en cuestión y del modelo de proceso elegido. Las pruebas de software no tienen el objeto de prevenir errores sino de detectarlos. Se efectúan sobre el trabajo realizado y se deben encarar con la intención de descubrir la mayor cantidad de errores posible. Clasificarlas es difícil, pues no son mutuamente disjuntas, sino muy entrelazadas. En lo que sigue intentaremos la siguiente estructura (Pablo Suárez, 2003):

Niveles de Prueba:

- Unidades
- Aceptación

3.2.1 Pruebas de unidad

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. En ciertos sistemas también se verifican o se prueban los drivers y el diseño de la arquitectura (Pressman, 1998). Normalmente cabe distinguir una fase informal antes de entrar en la fase de pruebas propiamente dicha. La fase informal la lleva a cabo el propio codificador en su despacho, y consiste en ir ejecutando el código para convencerse de que "básicamente, funciona". Esta fase suele consistir en pequeños ejemplos que se intentan ejecutar. Si el módulo falla, se suele utilizar un depurador para observar la evolución dinámica

del sistema, localizar el fallo, y repararlo. 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.
- Fallo de salida cuando se encuentra una iteración divergente.

Se tomó 10 funciones de las tres capas con el objetivo de buscar diversidad en el código a la hora de realizar la prueba. Luego de probadas las unidades, se llegó a la conclusión que todas funcionaban correctamente, lo que no quiere decir que todas los métodos del sistema lo hagan, para asegurarse de ello es necesario realizar otras pruebas que abarquen más código. A continuación se muestra uno de los fragmentos de código probados mediante el método de camino básico.

```
public void AccionMostrarDisennoPagina()
{
    if (barraComponente.getDirectorio().ElementoSeleccionado() == EElementosDelDirectorio.Proyecto
       || barraComponente.getDirectorio().ElementoSeleccionado() == EElementosDelDirectorio.Página)   1
    {
        NodoProyActivo();                                                                              2
        if (barraComponente.getDirectorio().ElementoSeleccionado() == EElementosDelDirectorio.Página   3
            && indicePagActiva != -1)
        {
            area.getTabPestannas().MostrarPagina(getListaProyectos().get(indiceProyActivo).
                getNombreProyecto(),(PaginaDibujable) getListaProyectos().
                get(indiceProyActivo).getListaPaginas().get(indicePagActiva));
            barraComponente.getDirectorio().getMenuEmergente().getItemMoverArribaPagina().
                setEnabled(indicePagActiva != 0);                                                      4
            barraComponente.getDirectorio().getMenuEmergente().getItemMoverAbajoPagina().
                setEnabled(indicePagActiva != getListaProyectos().get(indiceProyActivo).
                getListaPaginas().size() - 1);
        }
        else
        {
            barraComponente.getDirectorio().getMenuEmergente().getItemMoverArribaProyecto().
                setEnabled(indiceProyActivo != 0);
            barraComponente.getDirectorio().getMenuEmergente().getItemMoverAbajoProyecto().            5
                setEnabled(indiceProyActivo != getListaProyectos().size() - 1);
        }
    }                                                                                                  6
}                                                                                                      7
```

Figura 15: Procedimiento para mostrar el diseño de la página selecciona en el directorio del proyecto

Se confecciono un grafo de flujo con la lógica del código a probar. De esta manera se podrán determinar todos los caminos por los que el hilo de ejecución pueda llegar a pasar, y por consecuente elaborar los juegos de valores de pruebas para aplicar al módulo, con mayor facilidad y seguridad.

Un grafo de flujo se compone de:

- Nodos (círculos), que representan una o más acciones del módulo.
- Aristas (flechas), que representan el flujo de control entre los distintos nodos.

En este grafo están presentes todos los caminos que pueden guiar la trayectoria del módulo o unidad, de manera que todos sus nodos, excepto el primero y el último, deben tener al menos una entrada y una salida. A continuación se presenta el grafo perteneciente a esta prueba.

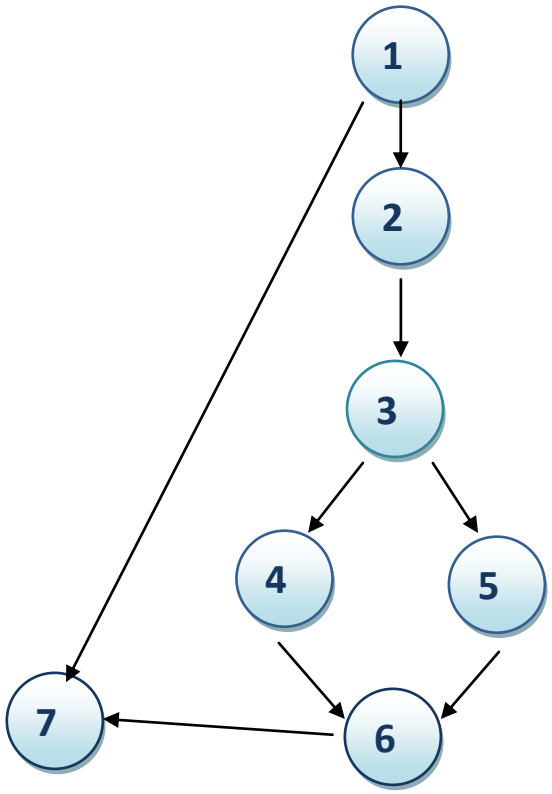


Figura 16: Grafo del camino del módulo a probar.

Este grafo se usa para calcular la complejidad ciclomática del módulo a probar. Para ello son usadas tres vías distintas, cada una de ellas se define por una fórmula, que independientemente de cuál de ellas se use, el resultado siempre debe ser el mismo.

Primera fórmula:

V (G) = (A - N) + 2 (donde A es el número aristas del grafo y N el número de nodos.)

V (G) = (8-7) + 2

V (G) = 3

Segunda fórmula:

V (G) = P + 1 (donde P es la cantidad de nodos desde los cuales se pueden tomar dos caminos.)

V (G) = 2 + 1

V (G) = 3

Tercera fórmula:

V (G) = R (donde R es la cantidad de regiones)

V (G) = 3

Luego de calcular el número de caminos independientes mediante una de las fórmulas, se deberán identificar todos los caminos independientes. Un camino independiente es un camino que introduce un nuevo grupo de acciones o nueva condición.

Caminos que pueden ser tomados: 1,7; 1,2,3,4,6,7; 1,2,3,5,6,7.

El siguiente paso es preparar los casos de prueba que obliguen a la ejecución de cada camino del conjunto básico, asegurando que los datos de entrada necesarios para ejecutar el módulo son correctos, para luego obtener, al terminar cada prueba, una respuesta como resultado a la correcta evaluación de la función.

Camino: 1,7

Caso de prueba: Mostrar el diseño de la página al seleccionarla en el directorio.

Entrada: No se da clic en una página o proyecto dentro del directorio de proyecto.

Resultado: No se visualiza el diseño de la página en el área de trabajo.

Camino: 1,2,3,4,6,7

Caso de prueba: Mostrar el diseño de la página al seleccionarla en el directorio.

Entrada: Se dio clic en una página del directorio de proyecto.

Resultado: Se visualiza el diseño de la página en el área de trabajo y se activa la opción mover hacia arriba y abajo la página.

Camino: 1,2,3,5,6,7

Caso de prueba: Mostrar el diseño de la página al seleccionarla en el directorio.

Entrada: Se dio clic en encima de un proyecto dentro del directorio de proyecto.

Resultado: No se visualiza el diseño de la página en el área de trabajo y se activa la opción mover hacia arriba y abajo el proyecto.

3.2.3 Pruebas de aceptación

Las pruebas de aceptación se realizan sobre el producto terminado e integrado, están concebidas para que sea un usuario final quien detecte los posibles errores. Se clasifican en dos tipos: pruebas Alfa y pruebas Beta. Las pruebas Alfa se realizan por un cliente en un entorno controlado por el equipo de desarrollo. Para que tengan validez, se debe primero crear un ambiente con las mismas condiciones que se encontrarán en las instalaciones del cliente. Una vez logrado esto, se procede a realizar las pruebas y a documentar los resultados. Cuando el software sea la adaptación de una versión previa, deberán probarse también los procesos de transformación de datos y actualización de archivos de todo tipo.

Por otro lado, las pruebas Beta 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 (Pressman, 1998).

Con la metodología SXP estas pruebas se realizan entre iteraciones y son las que definen el paso a la próxima iteración. Pueden aplicarse durante semanas o meses y van desde un informal paso de prueba hasta la ejecución continua de una serie de pruebas bien planificadas (Peñalver, Meneses, & S.García, 2010). Se realizó estas pruebas en su modalidad Alfa donde se escogieron usuarios con un conocimiento básico de los procesos de la AI.

3.3 Casos de pruebas

3.3.1 Caso de prueba "Cortar un componente en la barra de herramienta".

Escenario	**Descripción**	**Respuesta del sistema**	**Flujo Central**
Cortar componente	La herramienta permite cortar un componente.	La herramienta permite cortar un componente de la página de trabajo.	1. Clic en el componente. 2. Seleccionar la opción cortar en la barra de herramienta.

3.3.2 Caso de prueba "Modificar el ancho y largo del componente mediante la barra de propiedades".

Escenario	**Descripción**	**Ancho**	**Alto**	**Respuesta del sistema**	**Flujo central**

Darle el ancho y largo al componente correctamente.	La herramienta permite darle el ancho y alto a los componentes que uno desee.	V(50)	V(60)	Se modifica el componente en el área de trabajo.	1.Se selecciona el componente 2. Se inserta los valores del ancho y alto del componente correctamente. 3. Se oprime el botón Ok.
Darle ancho y largo al componente incorrectamente	La herramienta permite darle el ancho y alto a los componentes que uno desee.	V(80)	I(vacío)	No se modifica el componente y se muestra un mensaje ¨Los campos no pueden estar vacío ¨.	1.Se selecciona el componente 2. Se inserta letras en el valor del ancho y alto del componente. 3. Se oprime el botón Ok. 4. Se muestra un mensaje 5. Se oprime el botón Aceptar.
		I(vacío)	V(95)		
		I (vacío)	I(vacío)		
Darle ancho y largo a un componente con datos incorrectos.		I(pepe)	I(Pepe)	No se modifica el largo y el ancho y se muestra un	1.Se selecciona el componente 2. Se inserta letras en el valor del

Con letras				mensaje ¨Introduzca los datos correctamente¨	ancho y alto del componente. 3. Se oprime el botón Ok. 4. Se muestra un mensaje ¨Introduzca los datos correctamente.¨ 5. Se oprime el botón aceptar.

3.3.3 Caso de prueba "Mover hacia la derecha el componente"

Escenario	**Descripción**	**Respuesta del sistema**	**Flujo Central**
Mover componente hacia la derecha	La herramienta permite mover hacia la derecha un componente.	La herramienta permite mover hacia la derecha un componente de la página de trabajo.	1. Seleccionar el componente. 2. En la ventana de las propiedades de los componentes 3. Seleccionar la opción de Mover a la derecha.

3.3.4 Caso de prueba “Cambiar nombre al proyecto”

Escenario	Descripción	Nombre	Respuesta del sistema	Flujo central
Cambiar el nombre del proyecto correctamente	La herramienta permite cambiar el nombre del proyecto	V(Página nueva)	Se modifica nombre del proyecto.	1. Clic derecho encima del proyecto en el directorio. 2. Escoger la opción cambiar nombre al proyecto. 3. Introducir el nombre nuevo de la página. 4. Se oprime el botón Ok.
Cambiar el nombre del proyecto incorrecta	La herramienta permite cambiar el nombre del proyecto	V()	No se modifica el nombre del proyecto.	1. Clic derecho encima del proyecto en el directorio. 2. Escoger la opción cambiar nombre al proyecto. 3. Introducir el nombre nuevo de la página. 4. Se oprime el botón Ok. 5. Se muestra un mensaje “El nombre del proyecto no puede ser vacio”.

3.4 Resultados obtenidos al finalizar las iteraciones de pruebas

En el siguiente gráfico se encuentran representados los resultados obtenidos al finalizar cada iteración (figura 16):

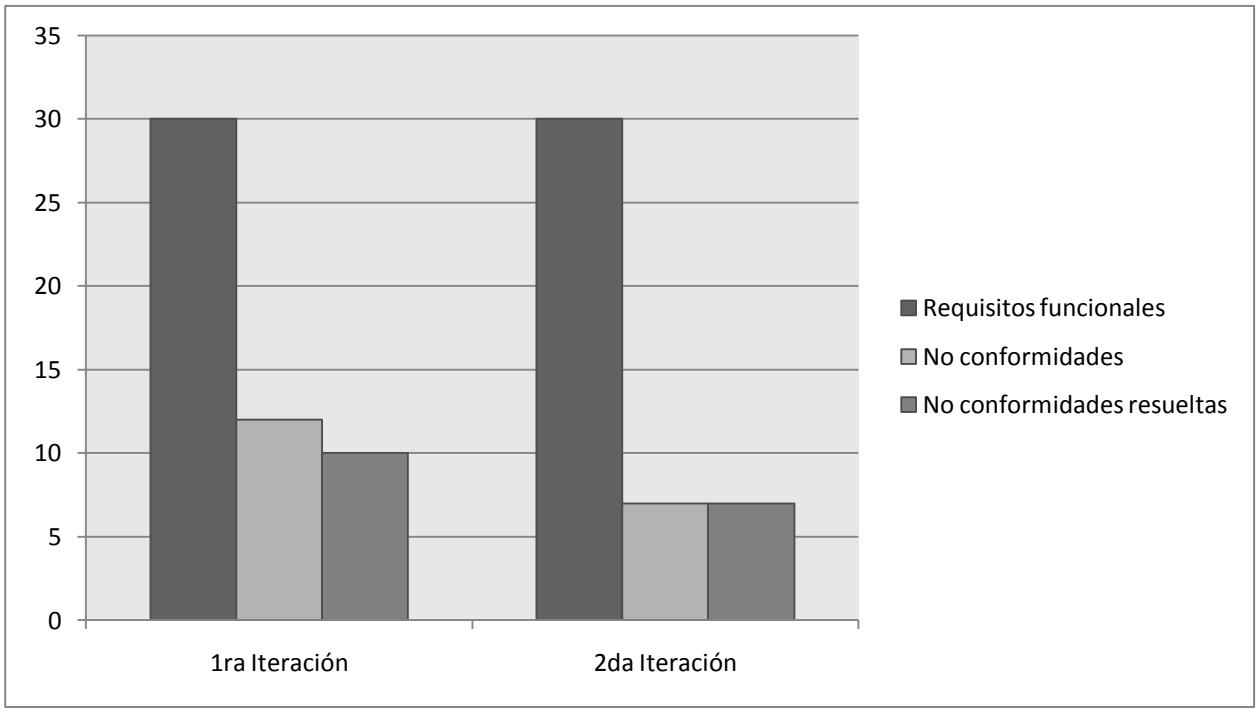


Figura 16: Pruebas del sistema

En la primera iteración de prueba de 30 requisitos funcionales se encontraron 13 no conformidades, que se clasificaron 8 significativas y 5 no significativas, de ellas se dio solución a 10. En la segunda iteración se encontraron 8 no conformidades, 7 de ellas significativas y 1 no significativa de las cuales se pudo resolver en su totalidad.

3.5 Conclusiones del capítulo

Con la realización de este capítulo se concluye que:

- El estudio de los distintos tipos de pruebas permitió identificar las pruebas a realizar para evaluar el funcionamiento del sistema.
- El diseño de casos de casos de pruebas permitirá corregir errores que puedan existir en el sistema.

Conclusiones generales

La presente investigación ha mostrado los sistemas existentes que se utilizan para realizar la técnica de diagramación y en qué tipos de diagramas realizan demostrando la necesidad de una aplicación que integre los tres tipos de diagramas que realiza el arquitecto de información durante todo el proceso para facilitar y agilizar el trabajo de este. En este marco y al finalizar el desarrollo de este trabajo de diploma, se concluye que:

- El estudio de soluciones homólogas y la valoración crítica que se hace de la propuesta del analista permiten refinar las características y funcionalidades del sistema acorde a las expectativas del usuario.

- El módulo de componentes para confeccionar los diagrama de organización, apoya la integración de los tres diagramas en una misma herramienta, lo que facilitará y agilizará el trabajo del arquitecto de información.

- El sistema desarrollado contribuye a mejorar el flujo de los procesos de arquitectura de información en la UCI.

Recomendaciones

- Desarrollar funcionalidades que permitan exportar el diagrama de organización de contenidos a otros formatos, como PDF, HTML.
- Incrementar los componentes con que cuenta el módulo para realizar diagramas de organización de contenidos.
- Exportar el diagrama a texto de forma tal que se respete la estructura taxonómica del mismo.
- Agregar a la herramienta la funcionalidad plantillas por defecto donde agilizar el trabajo al usuario.

Referencias bibliográficas

A. S Solutions. (2002-2011). Axure. Obtenido de http://www.axure.com/axurerpenvironment

Ailonwebs. (2009). Aplicaciones Web. Recuperado el 21 de 4 de 2011, de http://www.ailonwebs.com/aplicaciones-web.php

Alaimo, M. (2010). Simulación de SCRUM. Recuperado el 21 de 5 de 2011, de http://www.martinalaimo.com/es/2010/11/simulacion-extendida-de-scrum

Barzanalla, M. G. (2007). Metodologías de desarrollo de software. Obtenido de http://www.wikilearning.com/curso_gratis/metodologias_de_desarrollo_de_software/3617

Brown, D. (2006). Representing data in Wireframes. Recuperado el 17 de 2 de 2011, de http://www.greenonions.com/portfolio/dbrown_ia2005_wireframes.pdf

Celis, I. (2005). El ataque de los Frameworks. Recuperado el 21 de 3 de 2011, de http://www.webtaller.com/maletin/articulos/el-ataque-de-los-frameworks.php

Cmap. (2011). this concept map was created with IHMC Cmap Tools. Obtenido de http://cmap.ihmc.us/

Díaz, M. B. (2010). LA MODELACIÓN COMO MÉTODO TEÓRICO DE LA INVESTIGACIÓN.

Estándar_de_Código. (2010). Estándar de Código. Obtenido de https://repositorio.cenia.prod.uci.cu/svn/documentacion/Dpto_UD/ABAD/EP_ABAD/4.general/CENIA_GAI_OT_Estándar _de _Código.pdf

Fernández, F. J. (16 de 2 de 2003). No solo usabilidad. Obtenido de http://www.nosolousabilidad.com/articulos/ai.htm.

Finck, N. (2005). Visio Stencils for Information Architects. Obtenido de http://www.nickfinck.com/blog/entry/visio_stencils_for_information_architects

Flower, M. (2003). Patterns of Enterprise Application Architecture. Addison Wesley.

FreeMind. (2012). FreeMind free mind mapping software. Recuperado el 12 de 4 de 2012, de http://freemind.sourceforge.net/wiki/index.php/Main_Page

Garret, J. J. (2001). A visual vocabulary for describing information architecture and interaction design. [En línea]. Recuperado el 16 de 2 de 2011, de http://www.jjg.net/ia/visvocab

Garrett, J. J. (2002). The Elements of User Experience. New York: New Riders Publishing.

Izquierdo, Y. M. (2011). Herramienta de diagramación para la plataforma de arquitectura de información ABAD. Obtenido de http://catalogoenlinea.uci.cu/cgi-bin/koha/opac-detail.pl?biblionumber=11213

Jacobson, I., Boch, G., & Rumbaugh, J. (2000). El Proceso Unificado de Desarrollo de Software. Madrid: Addison Wesley.

KRUCHTEN, P. (1995). Architectural Blueprints--The 4+1 View Model of Software Architecture.

Kuchana, P. (2004). software Arquitecture Design Patterns in Java. United Sates of America: Auerbach publications.

Larman, C. (1999). UML y Patrones. Patience Hall.

León, R. R. (25 de 12 de 2007). La diagramación en la arquitectura de información. [En línea] 25 de 12 de 2007. Recuperado el 12 de 2 de 2011, de http://www.nosolousabilidad.com/articulos/diagramacion.htm

Llano, M. C. (2010). Métodos de arquitectura de información.

Mercer, D. (2001). Fundamentos de Programación en XML. Recuperado el 3 de 4 de 2011, de http://bibliodoc.uci.cu/pdf/reg01311.pdf.

Montalvo, C. (2010). Aplicaciones web, ventajas y desventajas. Obtenido de http://www.carlosmontalvo.com/2010/12/aplicaciones-web-ventajas-y-desventajas

Montes de Oca, A. (2004). Arquitectura de información y usabilidad. La Habana: Acimed.

Mora, L. V. (9 de 5 de 2011). UML: Unified Modeling Language. Lenguaje Unificado de Modelado. Obtenido de http://lilianavargasmora.blogspot.com/2011/05/uml-unified-modeling-language-lenguaje.html

Morville, P., & Rosenfeld, L. (1999). Information Architecture for the world wide Web.

Pablo Suárez, C. F. (2003). Documentación y Pruebas.

Paradigm, V. (2 de 12 de 2010). Visual Paradigm for UML 7.4. Recuperado el 23 de 3 de 2011, de http://images.visual-paradigm.com/datasheets/vpuml72_datasheet.pdf

Peñalver, G., Meneses, A., & S.García. (2010). SXP, Metodología ágil para el desarrollo de software. La Habana: Universidad de las Ciencias Informáticas, 2010.

Pérez García, F. U. (2008). Tipo de Lenguaje de Programación. Obtenido de http://www.larevistainformatica.com/tipo-lenguaje-programacion.htm

Pressman. (1998). Estrategias De Pruebas De Software Convencionales. Obtenido de http://www.buenastareas.com/ensayos/Estrategias-De-Pruebas-De-Software-Convencionales/1045971

Salamanca, U. d. (1999). Guía de Iniciación al Lenguaje JAVA. Recuperado el 10 de 3 de 2011, de http://zarza.usal.es/~fgarcia/doc/tuto2/Index.htm.

Schenone, H. (2004). Diseño de una Metodología Ágil de Desarrollo de Software. Buenos Aires: Fiuba.

Scout, B. (17 de 11 de 2005). Storyboarding Rich Internet Applications with Visio. Obtenido de http://www.boxesandarrows.com/view/storyboarding_rich_internet_applications_with_visio

Sierra, D. (2007). VVisual Paradigm For Uml. Obtenido de http://www.slideshare.net/vanquishdarkenigma/visual-paradigm-for-uml

Valls, I. P. (2009). Orígenes de los patrones. Recuperado el 9 de 3 de 2011, de http://www.javadabbadoo.org/cursos/infosintesis.net/javase/paqawt/selectorcolores/paso03patrones.html

Bibliografía consultada

A. S Solutions. (2002-2011). Axure. Obtenido de http://www.axure.com/axurerpenvironment

Ailonwebs. (2009). Aplicaciones Web. Recuperado el 21 de 4 de 2011, de http://www.ailonwebs.com/aplicaciones-web.php

Alaimo, M. (2010). Simulación de SCRUM. Recuperado el 21 de 5 de 2011, de http://www.martinalaimo.com/es/2010/11/simulacion-extendida-de-scrum

Babylon. (21 de 5 de 2012). Obtenido de http://www.babylon.com/definition/plataforma_de_desarrollo/Spanish.

Barzanalla, M. G. (2007). Metodologías de desarrollo de software. Recuperado el 3 de 2 de 2011, de http://www.wikilearning.com/curso_gratis/metodologias_de_desarrollo_de_software/3617

Brown, D. (2006). Representing data in Wireframes. Recuperado el 17 de 2 de 2011, de http://www.greenonions.com/portfolio/dbrown_ia2005_wireframes.pdf

Celis, I. (2005). El ataque de los Frameworks. Recuperado el 21 de 3 de 2011, de http://www.webtaller.com/maletin/articulos/el-ataque-de-los-frameworks.php

Cmap. (2011). this concept map was created with IHMC Cmap Tools. Obtenido de http://cmap.ihmc.us/

desarrollo Web. (s.f.). Obtenido de http://www.desarrolloweb.com/articulos/sistemas-gestores-bases-datos.html.

Díaz, M. B. (2010). LA MODELACIÓN COMO MÉTODO TEÓRICO DE LA INVESTIGACIÓN.

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

Española, R. A. (s.f.). Diccionario de la lengua española - Vigésima segunda edición. Real Academia Española. Obtenido de http://www.rae.es

Estándar_de_Código. (2010). Estándar de Código. Obtenido de https://repositorio.cenia.prod.uci.cu/svn/documentacion/Dpto_UD/ABAD/EP_ABAD/4.general/CENIA_GAI_OT_Estándar _de _Código.pdf

Fernández, F. J. (16 de 2 de 2003). No solo usabilidad. Obtenido de http://www.nosolousabilidad.com/articulos/ai.htm.

Finck, N. (8 de 4 de 2004). Visio Stencils for Information Architects. Obtenido de http://www.nickfinck.com/blog/entry/visio_stencils_for_information_architects

Finck, N. (2005). Visio Stencils for Information Architects. Obtenido de http://www.nickfinck.com/blog/entry/visio_stencils_for_information_architects

Flower, M. (2003). Patterns of Enterprise Application Architecture. Addison Wesley.

ForeUI. (2009). UI Prototyping Tool. Obtenido de http://www.foreui.com

FreeMind. (2012). FreeMind free mind mapping software. Recuperado el 12 de 4 de 2012, de http://freemind.sourceforge.net/wiki/index.php/Main_Page

Garret, J. J. (2001). A visual vocabulary for describing information architecture and interaction design. [En línea]. Recuperado el 16 de 2 de 2011, de http://www.jjg.net/ia/visvocab

Garrett, J. J. (2002). The Elements of User Experience. New York: New Riders Publishing.

Hermosillo, I. T. (2009). El lenguaje Java. Obtenido de http://eddi.ith.mx/Curso/Contenido/java.htm

Información., G. d. (2003). Ingeniería del Software y Sistemas de Información. Obtenido de http://issi.dsic.upv.es/publications/archives/f-1069167248521/actas.pdf

Izquierdo, Y. M. (2011). Herramienta de diagramación para la plataforma de arquitectura de información ABAD. Obtenido de http://catalogoenlinea.uci.cu/cgi-bin/koha/opac-detail.pl?biblionumber=11213

Jacobson, I., Boch, G., & Rumbaugh, J. (2000). El Proceso Unificado de Desarrollo de Software. Madrid: Addison Wesley.

KRUCHTEN, P. (1995). Architectural Blueprints--The 4+1 View Model of Software Architecture.

Kuchana, P. (2004). software Arquitecture Design Patterns in Java. United Sates of America: Auerbach publications.

Larman, C. (1999). UML y Patrones. Patience Hall.

León, R. R. (25 de 12 de 2007). La diagramación en la arquitectura de información. [En línea] 25 de 12 de 2007. Recuperado el 12 de 2 de 2011, de http://www.nosolousabilidad.com/articulos/diagramacion.htm

Llano, M. C. (2010). Métodos de arquitectura de información.

Mercer, D. (2001). Fundamentos de Programación en XML. Recuperado el 3 de 4 de 2011, de http://bibliodoc.uci.cu/pdf/reg01311.pdf.

Montalvo, C. (2010). Aplicaciones web, ventajas y desventajas. Obtenido de http://www.carlosmontalvo.com/2010/12/aplicaciones-web-ventajas-y-desventajas

Montes de Oca, A. (2004). Arquitectura de información y usabilidad. La Habana: Acimed.

Mora, L. V. (9 de 5 de 2011). UML: Unified Modeling Language. Lenguaje Unificado de Modelado. Obtenido de http://lilianavargasmora.blogspot.com/2011/05/uml-unified-modeling-language-lenguaje.html

Morville, P., & Rosenfeld, L. (1999). Information Architecture for the world wide Web.

Pablo Suárez, C. F. (2003). Documentación y Pruebas.

Paradigm, V. (2 de 12 de 2010). Visual Paradigm for UML 7.4. Recuperado el 23 de 3 de 2011, de http://images.visual-paradigm.com/datasheets/vpuml72_datasheet.pdf

Pavón, J. (2008-2009). El patrón Modelo-Vista-Controlador (MVC). Madrid: Artificial Universidad Complutense.

Peñalver, G., Meneses, A., & S.García. (2010). SXP, Metodología ágil para el desarrollo de software. La Habana: Universidad de las Ciencias Informáticas, 2010.

Pérez García, F. U. (2008). Tipo de Lenguaje de Programación. Obtenido de http://www.larevistainformatica.com/tipo-lenguaje-programacion.htm

Pressman. (1998). Estrategias De Pruebas De Software Convencionales. Obtenido de http://www.buenastareas.com/ensayos/Estrategias-De-Pruebas-De-Software-Convencionales/1045971

Prieto, F. (2008-2009). Patrones de diseño. Universidad de Valladolid: Departamento de Informática.

Salamanca, U. d. (1999). Guía de Iniciación al Lenguaje JAVA. Recuperado el 10 de 3 de 2011, de http://zarza.usal.es/~fgarcia/doc/tuto2/Index.htm.

Schenone, H. (2004). Diseño de una Metodología Ágil de Desarrollo de Software. Buenos Aires: Fiuba.

Scout, B. (17 de 11 de 2005). Storyboarding Rich Internet Applications with Visio. Obtenido de http://www.boxesandarrows.com/view/storyboarding_rich_internet_applications_with_visio

Sierra, D. (2007). VVisual Paradigm For Uml. Obtenido de http://www.slideshare.net/vanquishdarkenigma/visual-paradigm-for-uml

UCI. (s.f.). Entorno Virtual de Aprendizaje. Práctica Profesional. Recuperado el 2012 de 4 de 3, de http://eva.uci.cu/mod/resource/view.php?id=29419

Valls, I. P. (2009). Orígenes de los patrones. Recuperado el 9 de 3 de 2011, de http://www.javadabbadoo.org/cursos/infosintesis.net/javase/paqawt/selectorcolores/paso03patrones.html

Wells, D. (2009). Extreme Programming: A gentle introduction. Obtenido de http://www.extremeprogramming.org/rules.html.

Glosario de términos

- Arquitecto de información: persona que crea el mapa o la estructura de información que permite a otros encontrar su camino personal al conocimiento.

- CASE: (Computer Aided Software Engineering en español Ingeniería de Software Asistida por ordenador) son aplicaciones informáticas para aumentar la productividad en el desarrollo de software reduciendo el coste de las mismas en términos de tiempo y de dinero.

- GNU/Linux: es un sistema operativo tipo Unix (también conocido como Linux que se distribuye bajo la licencia pública general de GNU (GNU GPL), es decir que es software libre.

- GPL (General Public License): Licencia Pública General que permite el uso y modificación del código para desarrollar software libre, pero no propietario.

- HTML (HyperText Markup Language): en español, Lenguaje de Marcado de Hipertexto, es el lenguaje de marcado predominante para la construcción de páginas Web. Se usa para describir la estructura y el contenido en forma de texto, así como para complementar el texto con objetos tales como imágenes.

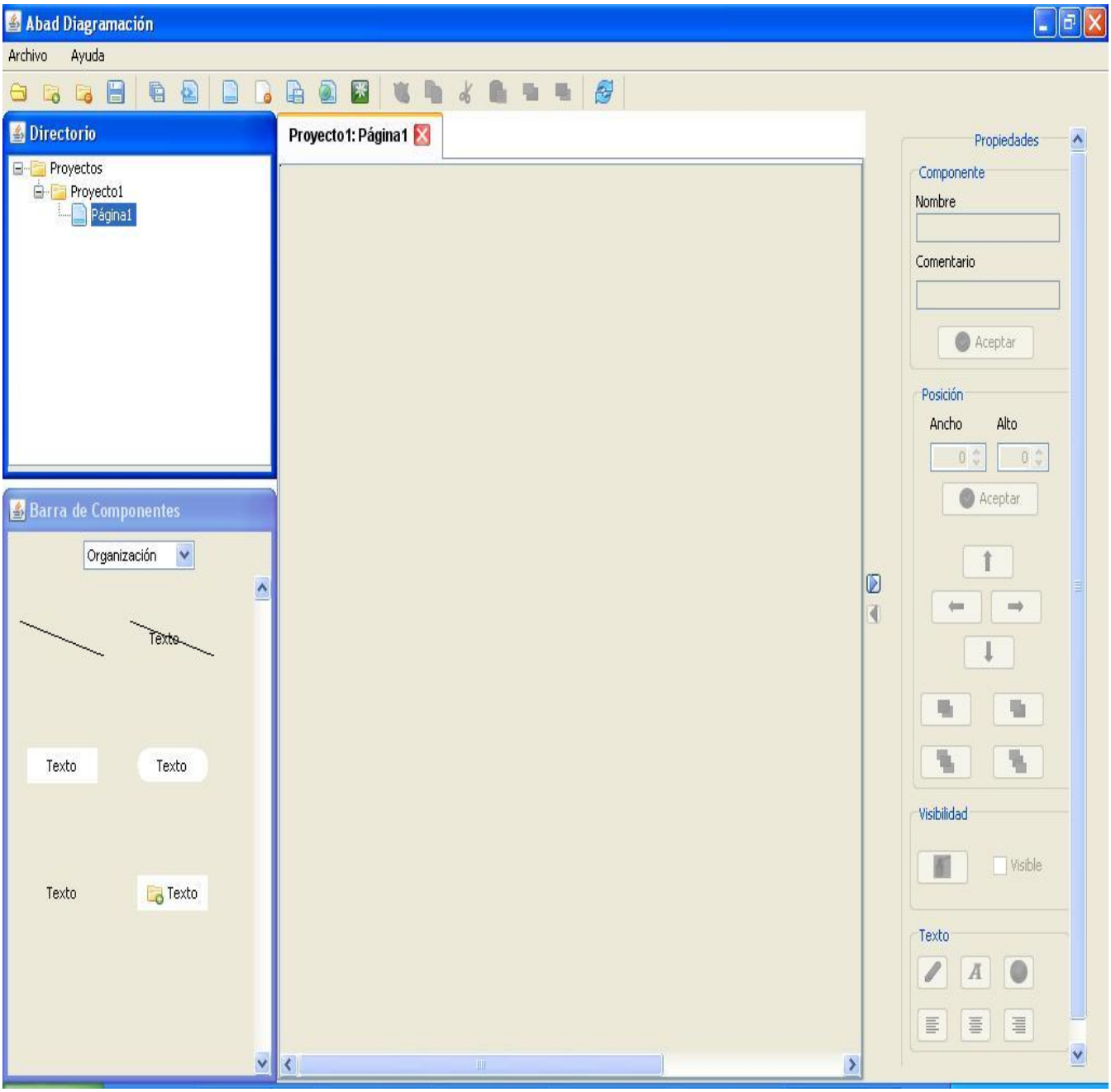
- Java: lenguaje de programación orientado a objetos.

- Web: Sistema para presentar información en internet basado en hipertexto.

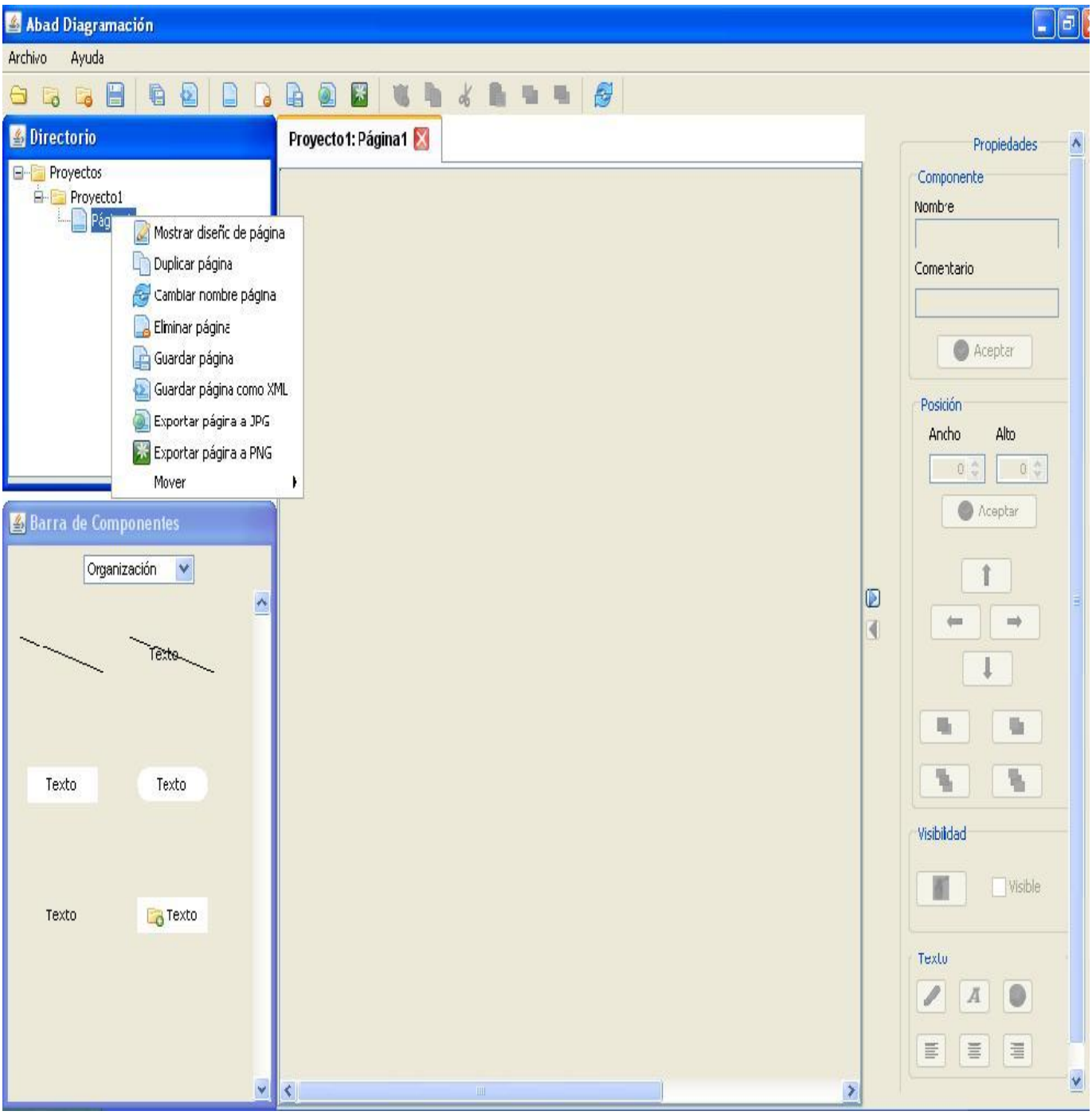
- XML (Extensible Markup Language): siglas en español, Lenguaje de Marcas Extensible. Es un formato estándar para el intercambio de datos.

Anexos

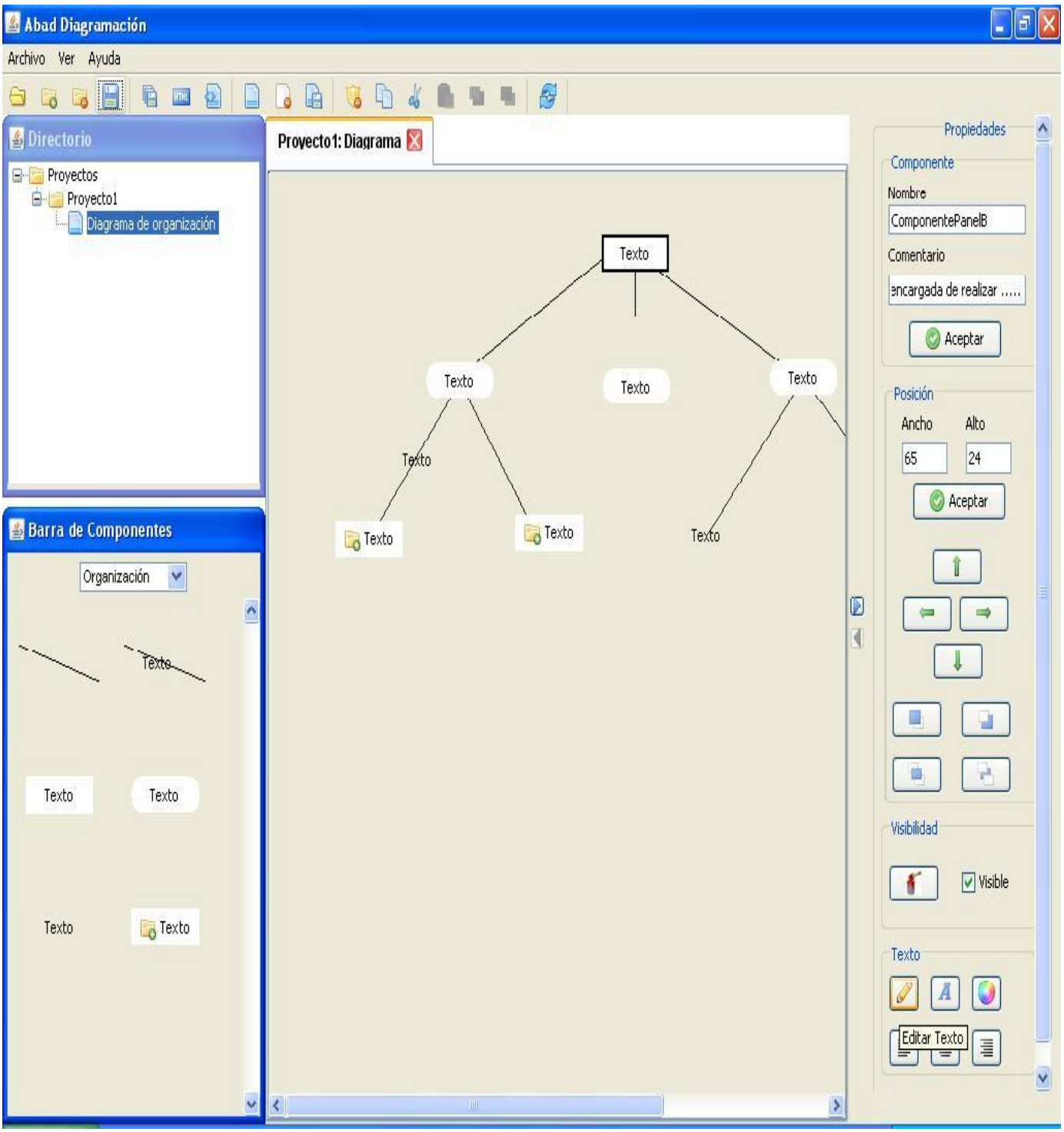
Anexo # 1: Ventana principal, está dividida en el menú, la barra de componentes, directorio de proyecto, barra de componentes, barra de propiedades y el área de trabajo.



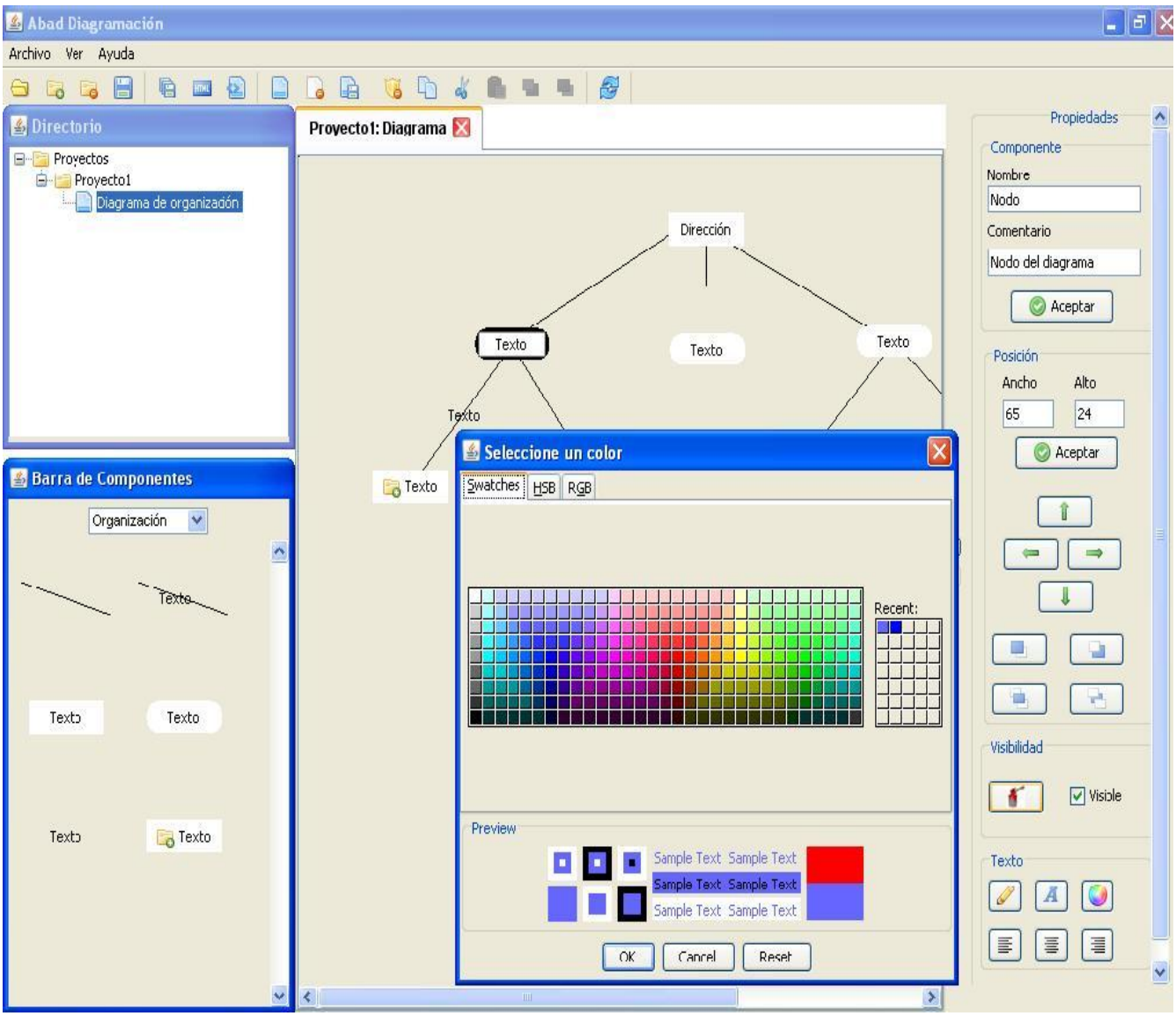
Anexo #2: Acciones que se pueden realizar a un diagrama, como cambiar nombre, eliminar diagrama, guardarlo en distintos formatos como archivo, XML o JPG etc.



Anexo #3: Confeccionando un diagrama de organización.



Anexo #4: Modificando la propiedad color de fondo de un componente.


Abad Diagramación
Archivo Ver Ayuda
Directorio
Proyectos
Proyecto1
Diagrama de organización
Barra de Componentes
Organización
Texto
Proyecto1: Diagrama
Dirección
Texto
Texto
Texto
Texto
Texto
Seleccione un color
Swatches HSB RGB
Recent:
Preview
Sample Text Sample Text
Sample Text Sample Text
Sample Text Sample Text
OK
Cancel
Reset
Propiedades
Componente
Nombre
Nodo
Comentario
Nodo del diagrama
Aceptar
Posición
Ancho
Alto
65
24
Aceptar
Visibilidad
Visible
Texto