UNIVERSIDAD NACIONAL DEL ALTIPLANO

FACULTAD DE INGENIERÍA MECÁNICA ELÉCTRICA ELECTRÓNICA Y SISTEMAS

ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMAS

HERRAMIENTA DE REUTILIZACIÓN DEL SOFTWARE PARA EL SOPORTE OPERATIVO EN LA ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMAS - U.N.A. PUNO 2013

TESIS

PRESENTADO POR:

Julio Enrique Rojas Ramos

PARA OPTAR EL TÍTULO PROFESIONAL DE:

INGENIERO DE SISTEMAS

PUNO – PERÚ
2013

UNIVERSIDAD NACIONAL DEL ALTIPLANO

FACULTAD DE INGENIERÍA MECÁNICA ELÉCTRICA ELECTRÓNICA Y SISTEMAS

ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMAS

HERRAMIENTA DE REUTILIZACIÓN DEL SOFTWARE PARA EL SOPORTE OPERATIVO EN LA ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMAS - U.N.A. PUNO 2013

TESIS PRESENTADA POR:

Julio Enrique Rojas Ramos

PARA OPTAR EL TÍTULO PROFESIONAL DE:

INGENIERO DE SISTEMAS

APROBADA POR EL JURADO REVISOR CONFORMADO POR:

PRESIDENTE	:	Mg. Elmer COYLA IDME
PRIMER MIEMBRO	:	M. Sc. Milder ZANABRIA ORTEGA
SEGUNDO MIEMBRO	:	Ing. Adolfo Carlos JIMENEZ CHURA
DIRECTOR DE TESIS	:	Mg. Robert Antonio ROMERO FLORES
ASESOR DE TESIS	:	Ing. Pedro Feder PONCE CORDERO

ÁREA: Informática
TEMA: Software de base de datos y de aplicación

DEDICATORIA

Con mucho amor a mi mamá Yhobanna,
por su gratitud, esfuerzo y sacrificio conmigo.
A mi abuelo Amador, mi gran ejemplo a seguir.
A toda mi familia por su cariño inmensurable.

AGRADECIMIENTOS

A mis maestros de Ingeniería de Sistemas
de la Universidad Nacional del Altiplano
por la calidad de sus saberes, su orientación,
su apoyo irrestricto y su sincera amistad.

A mis colegas de la Asociación de
Comunicadores Escolares del Sur Peruano (ACESP).

A quienes colaboraron
para que este trabajo se realizara con éxito.

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

RESUMEN

En este trabajo de investigación se describe la importancia de la reutilización de los procesos y proyectos para conseguir una mejora continua de los procesos de desarrollo y mantenimiento de software utilizadas por pequeñas y medianas empresas dedicadas a la construcción y desarrollo de software, facilitando de esta manera, la obtención por parte de las mismas, de altos niveles de madurez.

Aunque la reutilización de definiciones de procesos y proyectos software supone un gran beneficio para el desarrollo del software no se encuentra disponible en ninguna herramienta que facilite su reutilización efectiva. En este trabajo se han identificado los factores que caracterizan a los diferentes tipos de procesos y proyectos software, se han analizado, comparado y contrastado. Posteriormente, se han agrupado en grupos de factores siguiendo criterios de similitud. Además, en el presente trabajo se propone un método para la recuperación efectiva de procesos y proyectos software a partir de la especificación, a alto nivel, de sus características generales.

Este método de reutilización ha servido como base para la creación de una herramienta con soporte efectivo para la recuperación de componentes software, así como la gestión de los mismos, con el único afán de lograr una mejora continua de los procesos software.

ABSTRACT

In this research paper describes the importance of the reuse of processes and projects to achieve continuous improvement of the processes of development and maintenance of software used by small and medium companies engaged in the construction and development of software, facilitating in this way, obtaining by them, of high levels of maturity.

Although the reuse of definitions of processes and software projects is a great benefit for the development of the software it is not available in any tool that facilitates effective reuse. This work identified the factors that characterize different types of processes and software projects, have been analyzed, compared and contrasted. Subsequently, they have grouped in sets of factors according to criteria of similarity. Furthermore, this work is proposed a method for the effective recovery of processes and projects software from specification, high level of their general characteristics.

This method of reuse has served as basis for the creation of a tool with affective support for the recovery of software components, as well as the management of the same, with the only intention of achieving continuous improvement in the software process.

INTRODUCCIÓN

Sabemos que la reutilización es por defecto la estrategia de resolución de problemas en la mayoría de las actividades del ser humano. Los científicos cognitivos coinciden en afirmar que lo primero que hacemos cuando enfrentamos un problema es determinar si éste ha sido resuelto antes; en caso contrario, buscamos en nuestro espacio mental problemas análogos que ya se han resuelto y adaptamos su solución al problema actual; cuando ninguna de las anteriores situaciones se cumple, utilizamos entonces las habilidades y el conocimiento general analítico para resolución de problemas.

La reutilización de software es considerada una tecnología capaz de reducir drásticamente los costes de producción y los tiempos de desarrollo. La idea de base es común a otras ramas de la ingeniería y consiste simplemente en reutilizar el trabajo hecho en proyectos precedentes para evitar realizar varias veces el mismo o similar trabajo. Sin embargo, hasta el momento, pocos programas de reutilización se pueden considerar verdaderamente fructíferos. Los resultados conseguidos son sólo parciales y no producen las significativas mejoras prometidas.

En el campo de la ingeniería de software la reutilización ofrece un gran potencial en términos de productividad y calidad del software. La automatización del proceso software es el medio para asegurar la consistencia en la ejecución del proceso y para reducir costes. Esta automatización debe incluir las herramientas apropiadas para ejecutar las tareas específicas, la eliminación de errores y una guía para realizar las actividades críticas. Todo este soporte debe permitir que los desarrolladores se centren en el aspecto creativo de su trabajo y no estar perdiendo más tiempo en crear un software desde sus inicios, ahorrando con esto una muy valiosa cantidad de horas y días al desarrollar el software.

Por otra parte, la mayor dificultad en el éxito de la instauración de un programa de reutilización reside en los cambios de organización y cultura de la empresa. Por lo

tanto, la introducción de prácticas de reutilización de software debe abordarse principalmente desde la perspectiva del proceso de producción de software, considerando la reutilización como parte esencial del modo de operar de la organización. Es por estas razones que este trabajo centra su atención en temas de soporte operativo al proceso de reutilización de software.

Este método de reutilización ha servido como base para la creación de una herramienta con soporte efectivo para la recuperación de componentes software, así como la gestión de los mismos, con el único afán de lograr una mejora continua de los procesos software.

El presente trabajo de investigación se estructura en cinco capítulos:

El Capítulo I: Contiene el planteamiento de problema y su justificación, los antecedentes y los objetivos de la investigación.

El Capítulo II: Refiere a la fundamentación teórica y al marco teórico conceptual utilizado para realizar el presente trabajo de investigación.

El Capítulo III: Describe el método de investigación, la operacionalización de variables y las hipótesis de la investigación.

El Capítulo IV: Expone y analiza los resultados de la investigación.

El Capítulo V: Da a conocer las conclusiones y recomendaciones obtenidas al realizar la investigación.

Finalmente las referencias bibliográficas y los anexos.

CAPÍTULO I
EL PROBLEMA

1 PROBLEMA DE INVESTIGACIÓN.

1.1 PLANTEAMIENTO DEL PROBLEMA.

La reutilización de software es considerada una tecnología capaz de reducir drásticamente los costes de producción y los tiempos de desarrollo. La idea de base es común a otras ramas de la ingeniería y consiste simplemente en reutilizar el trabajo hecho en proyectos precedentes para evitar realizar varias veces el mismo o similar trabajo. Sin embargo, hasta el momento, pocos programas de reutilización se pueden considerar verdaderamente fructíferos. Los resultados conseguidos son sólo parciales y no producen las significativas mejoras prometidas.

La automatización del proceso software es el medio para asegurar la consistencia en la ejecución del proceso y para reducir costes. Esta automatización debe incluir las herramientas apropiadas para ejecutar las tareas específicas, la eliminación de errores y una guía para realizar las actividades críticas. Todo este soporte debe permitir que los desarrolladores se centren en el aspecto creativo de su trabajo.

Por otra parte, la mayor dificultad en el éxito de la instauración de un programa de reutilización reside en los cambios de organización y cultura de la empresa. Por lo tanto, la introducción de prácticas de reutilización de software debe abordarse principalmente desde la perspectiva del proceso de producción de software, considerando la reutilización como parte esencial del modo de operar de la organización. Es por estas razones que este trabajo centra su atención en temas de soporte operativo al proceso de reutilización de software.

Considerando las condiciones expuestas y la importancia de tratar el problema de la reutilización del software, se formula la siguiente interrogante:

¿Es posible desarrollar una herramienta de reutilización del software para el soporte operativo?

1.2 JUSTIFICACIÓN DEL PROBLEMA.

- La investigación aporta una contribución académica útil a la formulación de una metodología de desarrollo de software para y con reutilización.

- La reutilización juega un papel clave en varios temas como son: la productividad, la capacidad de mantenimiento, la portabilidad y la calidad. Por ello la reutilización debe aplicarse a cada etapa del ciclo de vida.

- La solución del problema servirá para que los desarrolladores de aplicaciones tengan una referencia del enfoque de reutilización que satisfaga sus expectativas, así como también una guía a seguir en el diseño de sus aplicaciones con y para reutilización.

- Asimismo, el problema a investigarse posee una originalidad específica, toda vez que aún no ha sido estudiado, cuando menos en la Escuela Profesional de Ingeniería de Sistemas de la Universidad Nacional del Altiplano.

1.3 ANTECEDENTES DE LA INVESTIGACIÓN.

Habiendo realizado una búsqueda en las bibliotecas de la Escuela Profesional de Ingeniería de Sistemas de la Universidad Nacional del Altiplano, no se ha encontrado ningún trabajo similar o en el área de investigación.

Más en internet se logró encontrar los siguientes trabajos de investigación:

Granell, (2006), realizó la investigación: *Reutilización de servicios web mediante componentes integrados*, en la Universidad Jaume I., Departamento de Lenguajes y Sistemas Informáticos. La investigación llego a las siguientes principales conclusiones:

- El objetivo y contribución más importante de esta tesis fue el aportar una aproximación para la composición de servicios web con el fin de facilitar el desarrollo de aplicaciones web basadas en servicios flexibles y reutilizables, que integra las características más relevantes sobre reutilización presente en los sistemas basados en componentes software; aportando también a la

creación, composición y reutilización de componentes integrados para transformarlos finalmente en procesos ejecutables.

López, (2010), realizó la investigación: *Reutilización del Software a partir de Requisitos Funcionales en el Modelo de Mecano: Comparación de Escenarios*, en el Instituto Tecnológico de Costa Rica, Miguel Ángel Laguna, José Manuel Marqués - Universidad de Valladolid. La investigación llego a las siguientes principales conclusiones:

- Este artículo propone una aproximación para incorporar el proceso de reutilización desde las etapas iniciales del ciclo de vida del software. La propuesta se basa en generalizar escenarios del problema y compararlos con escenarios genéricos. Los primeros se obtienen con una herramienta automatizada que está en fase de desarrollo. La generalización se realiza mediante reducción de redes de Petri. Los escenarios genéricos se almacenan en un repositorio asociados a estructuras complejas de reutilización denominadas mecanos que enlazan elementos de análisis, diseño e implementación. La comparación se realiza mediante analogía. De este modo, con una estrategia para reutilizar software desde los requisitos funcionales, se ofrece una alternativa para acelerar el desarrollo de soluciones software con reutilización a la vez que se eleva el nivel de abstracción de los elementos reutilizables. La motivación principal del artículo es que los requisitos representan el conocimiento más abstracto del dominio y un alto porcentaje de ellos son reutilizables.

1.4 OBJETIVOS.

1.4.1 OBJETIVO GENERAL.

- Desarrollar una herramienta de reutilización del software para el soporte operativo en la Escuela Profesional de Ingeniería de Sistemas de la U.N.A. - Puno.

1.4.2 OBJETIVOS ESPECÍFICOS.

- Motivar el estudio de los principios y conceptos de familia inherentes al enfoque de reutilización.
- Definir los pasos necesarios para la adopción de los procesos orientados a la reutilización.
- Implementar un modelo de una herramienta que permita la reutilización del software para el soporte operativo.

CAPÍTULO II
FUNDAMENTACIÓN TEÓRICA

2 MARCO TEÓRICO CONCEPTUAL.

2.1 MARCO TEÓRICO.

2.1.1 REUTILIZACIÓN DEL SOFTWARE.

La reutilización es por defecto la estrategia de resolución de problemas en la mayoría de las actividades del ser humano. Los científicos cognitivos coinciden en afirmar que lo primero que hacemos cuando enfrentamos un problema es determinar si éste ha sido resuelto antes; en caso contrario, buscamos en nuestro espacio mental problemas análogos que ya se han resuelto y adaptamos su solución al problema actual; cuando ninguna de las anteriores situaciones se cumple, utilizamos entonces las habilidades y el conocimiento general analítico para resolución de problemas (Mili, Mili, & Mili, 1995). Se puede hablar de reutilización en diferentes grados: reutilización informal e intuitiva que se basa en el conocimiento individual de las personas, reutilización esquematizada por medio de la utilización de procedimientos que guían el desarrollo de actividades y reutilización orientada a productos cuando se concreta en artefactos que representan el conocimiento y que facilitan la labor de reutilizar.

En el campo de la ingeniería de software la reutilización ofrece un gran potencial en términos de productividad y calidad del software. Productividad, porque amplifica la capacidad de programación, en el sentido de escribir menos código, reduce la cantidad de documentación y pruebas que se deben realizar y genera un efecto de sinergia sobre la funcionalidad del sistema completo a partir de la funcionalidad de sus componentes. Calidad, porque el diseño de componentes se realiza pensando en su posterior utilización, con una documentación precisa, con procesos certificados de prueba y validación y con una estructuración adecuada de las partes del componente para que sea entendible por el usuario.

La demanda de sistemas de información cada vez más complejos, la dinámica de cambio en las organizaciones y la globalización de las operaciones del negocio bajo unas restricciones de tiempo, costo y calidad, hacen de la reutilización, una necesidad imperiosa dentro de todo proceso de ingeniería de software. En este contexto la reutilización puede ser definida como la propiedad de utilizar conocimiento, procesos, metodologías o componentes de software ya existente para adaptarlo a una nueva necesidad, incrementando significativamente la calidad y productividad del desarrollo (Frakes & Terry, 1996).

2.1.2 LA REUTILIZACIÓN EN INGENIERÍA DEL SOFTWARE.

2.1.2.1 REUTILIZACIÓN Y COMPONENTES SOFTWARE.

La reutilización es el hecho de aprovechar los productos del proceso de desarrollo de una aplicación en otra -entendiéndose por producto cualquier porción de la definición o de la implementación de una aplicación-. Un producto reutilizable debe llenar dos requerimientos básicos. Primero, el producto no deberá depender del contexto en el cual se utilice. Segundo, el producto deberá ser útil también en otras aplicaciones. Un componente es cualquier producto que cumpla con estos requisitos. Por lo tanto un componente es un producto diferente y útil para el desarrollo.

Los componentes, por lo tanto, son las unidades fundamentales de reutilización. La capacidad de localizar y aprovechar componentes útiles determina su capacidad de reutilización en una aplicación. En la figura 2.1 se ilustran distintos tipos de componentes. El proceso de desarrollo genera productos útiles en todas sus fases. De ahí que se generen componentes tanto de la especificación como de la implementación de una aplicación.

Habitualmente los componentes son documentos en papel o archivos digitales.

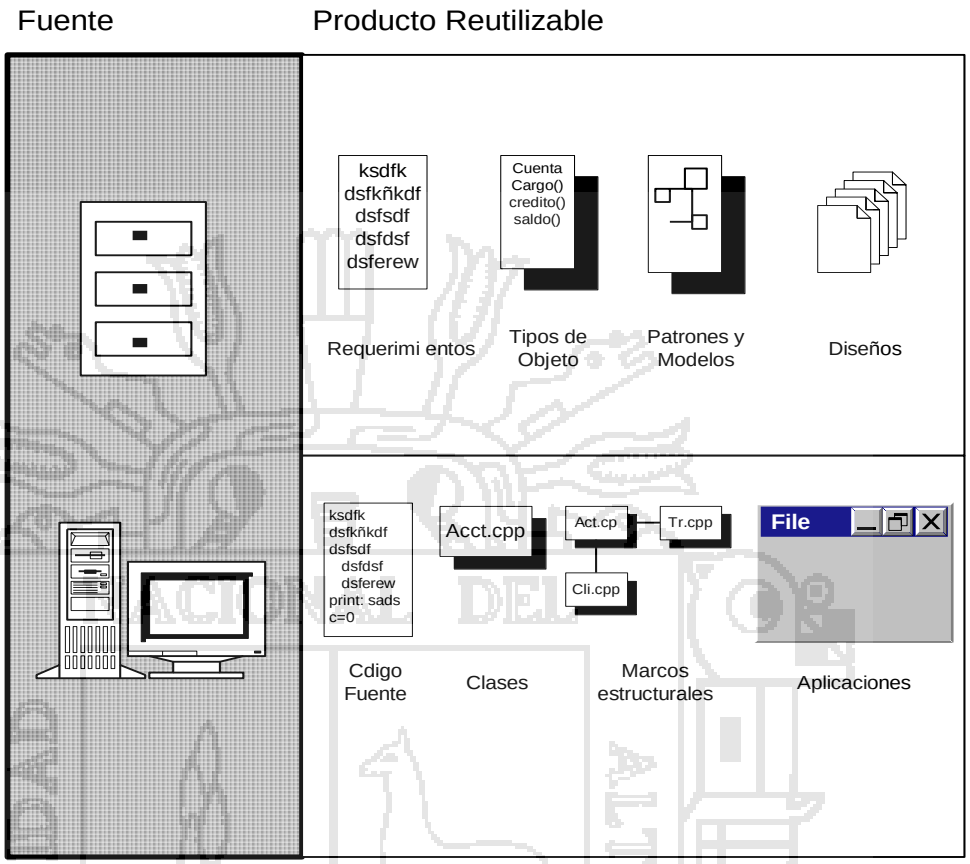


FIGURA 2.1: TIPOS DE COMPONENTES QUE FORMAN LAS UNIDADES FUNDAMENTALES DE REUTILIZACIÓN.

Los componentes de especificación describen algún aspecto del problema o de la estructura de la aplicación. Los componentes típicos de especificación incluyen documentos de requerimientos, tipos de objetos, patrones o modelos y diseños completos. Por definición todos estos componentes son elementos de especificación independientes. Un desarrollador debería ser capaz de comprender un componente independientemente de la aplicación en la que se utilice. Sin embargo, se puede construir componentes partiendo de otros componentes. Esto permite la elaboración de una jerarquía de subcomponentes y de componentes más complejos. Por ejemplo

se puede construir patrones y modelos a partir de tipos de objetos, o un diseño que comprenda varios modelos diferentes.

Los componentes de especificación pueden ser más o menos formales. Por ejemplo los requerimientos de aplicación se pueden construir, se pueden escribir como texto informal o expresarse en notación gráfica, utilizando una sintaxis y una semántica definida. Aunque esta variante en la formalidad del componente no impide su utilización, sí afectará su capacidad de reutilización. Un componente especificado con sintaxis formal resultará por lo regular más fácil de reutilizar y de mantener. Sin embargo, la formalización incrementa el costo de construcción del componente. Este equilibrio entre costo de desarrollo inicial y reutilización futura es una consideración clave en la especificación de componentes.

Los componentes de implementación deben ser ejecutables o traducidos directamente a una forma ejecutable. Estos incluyen unidades reutilizables, como código fuente, clases, marcos estructurales y aplicaciones. Además, un componente de implementación podría incluir otros componentes, por ejemplo, la construcción de un marco estructural de aplicación a partir de clases constitutivas.

Los componentes de especificación y los de implementación son importantes en la reutilización. Sin embargo, a la fecha, tanto los investigadores, como la industria han centrado su atención en los componentes de implementación. La reutilización de clases, bibliotecas de clases y marcos estructurales es ya un proceso relativamente bien comprendido. Aun así, en razón de que la reutilización destaca el poder de los productos existentes, los componentes de especificación deberían proporcionar mayores beneficios. Las especificaciones de análisis representan conocimiento sobre el problema y son independientes de la tecnología de implementación. Esta comprensión fundamental

de un dominio es habitualmente estable y solamente evoluciona con el tiempo.

2.1.2.2 ARTEFACTOS REUTILIZABLES.

Ya se ha indicado que la reutilización del software abarca algo más que el simple código fuente. **¿Pero cuánto más?** Capers Jones define diez artefactos del software que son candidatos para la reutilización:

- **Planes de proyecto.** La estructura básica y gran parte del contenido (por ejemplo: el plan SQA) de un plan de proyecto de software se podrán reutilizar entre proyectos sucesivos. Esto reduce el tiempo necesario para desarrollar el plan y la incertidumbre asociada al establecimiento de planes temporales, al análisis de riesgos, y otras características.

- **Estimaciones de coste.** Dado que es frecuente implementar funciones similares en distintos proyectos, quizá sea posible reutilizar las estimaciones para esas funciones con pocas modificaciones o quizá con ninguna.

- **Arquitecturas.** Existen relativamente pocos programas y pocas arquitecturas de datos distintas aun cuando se consideren distintos programas de aplicación. Es posible crear un conjunto de plantillas arquitectónicas genéricas (por ejemplo: una arquitectura de procesamiento de transacciones) y se pueden establecer esas plantillas como entornos reutilizables para el diseño.

- **Especificaciones y modelos de requisitos.** Los modelos y especificaciones de clases y objetos son candidatos evidentes para la reutilización. Además, los modelos de análisis (por ejemplo: los diagramas de flujo de datos) desarrollados empleando enfoques convencionales de ingeniería del software también se pueden reutilizar.

- **Diseños.** Los diseños arquitectónicos, de datos, de interfaz y de procedimientos desarrollados empleando métodos convencionales son candidatos para la reutilización. Con más frecuencia los diseños de sistemas y de objetos van a ser reutilizados.

- **Código fuente.** Los componentes verificados de programa escritos en lenguajes de programación compatibles son candidatos para la reutilización.

- **Documentación del usuario y técnica.** Suele ser posible reutilizar grandes porciones de la documentación de usuario y técnica, aun cuando las aplicaciones específicas sean distintas.

- **Interfaces humanas.** Siendo posiblemente el artefacto del software más ampliamente reutilizado, el software de IGU se reutiliza con frecuencia. Dado que puede dar cuenta de casi un 60 por ciento del volumen de código de las aplicaciones, los beneficios de su reutilización son significativos.

- **Datos.** Entre los artefactos más frecuentemente reutilizados, los datos abarcan las tablas internas, listas y estructuras de registros, así como los archivos y bases de datos completas.

- **Casos de prueba.** Siempre que es preciso reutilizar un diseño o un componente de código, el caso de prueba relevante deberá de estar "asociado" a él.

Es importante tener en cuenta que la reutilización va más allá de los artefactos suministrables que se describían más arriba, y que incluye también elementos del proceso de ingeniería del software. Los métodos específicos de modelado de análisis, las técnicas de inspección, las técnicas de diseño de casos de prueba, los procedimientos de control de calidad y muchas otras aplicaciones de la ingeniería del software se pueden "reutilizar".

2.1.3 ENFOQUES DE LA REUTILIZACIÓN.

Son variadas las aproximaciones, enfoques y consideraciones que diferentes autores dan al tema de la reutilización. La tabla 2.1 presenta una visión general del tratamiento de la reutilización considerando diferentes aspectos (Frakes & Terry, 1996).

La aproximación tecnológica se refiere a la técnica utilizada para desarrollar componentes. El enfoque metodológico representa la manera como se integra la reutilización dentro del proceso mismo de desarrollo. La modificación sugiere la forma como los componentes son accedidos y manipulados para adecuarlos a nuevas necesidades. El alcance de desarrollo indica si la reutilización sólo se aplica a componentes internos o permite integrar componentes de otras bibliotecas. El alcance del dominio delimita la reutilización a una familia de sistemas o permite su aplicación entre diferentes dominios de aplicación. El objeto de reutilización se refiere a los diferentes enfoques que diversos autores proponen al considerar la reutilización: desde el punto de vista del proceso de desarrollo, considerando el nivel de abstracción del componente o según el tipo de conocimiento que se reutiliza.

FACETA	TIPO	DESCRIPCION
Aproximación tecnológica	Por generación	Parte de una especificación del problema que va siendo refinada en iteraciones sucesivas.
	Por composición	Utiliza pequeñas partes de software como invariantes para proveer funcionalidad variante por medio de su integración.

Enfoque metodológico	Desarrollo con reutilización	Centrado en el desarrollo de componentes adecuados para ser utilizados en un problema específico - visión proveedor.
	Desarrollo para reutilizar	Centrado en el proceso mismo de construcción de soluciones software a partir de componentes almacenadas en una biblioteca-visión demanda.
Modificación	Caja blanca	Componentes que son reutilizados por modificación y adaptación.
	Caja negra	Componentes que son reutilizados sin ninguna modificación.
	Adaptativo	Utiliza estructuras grandes de software como invariantes y restringe la variabilidad a un conjunto de argumentos o parámetros.
Alcance del desarrollo	Interno	Mide el nivel de reutilización de componentes de un repositorio interno o de componentes del mismo proyecto.
	Externo	Mide el nivel de reutilización de componentes que provienen de repositorios externos o la proporción de productos que fueron adquiridos.
Alcance del dominio	Vertical	Reutilización dentro del mismo dominio de aplicación.
	Horizontal	Reutilización dentro de dominios de aplicación diferentes.
Objeto de reuso	Según el estado del proceso de desarrollo	Según la etapa en el ciclo de vida en la cual el conocimiento es producido o reutilizado. Tecnologías de reuso de amplio espectro y de limitado espectro (Biggerstaff).
	Según el nivel de abstracción	Según la propiedad del componente de representar conceptos abstractos, concretos o implementados. Código reciclado, componentes de código, esquemas (Frakes). Código fuente, diseño, especificación, objetos, texto, arquitecturas (Prieto-Díaz).
	Según la naturaleza del conocimiento	Según el tipo de conocimiento que se reutiliza concretado en artefactos o habilidades. Artefactos reutilizables: datos, arquitecturas, diseño, programas (Jones) Naturaleza del conocimiento: informal, esquematizado, herramientas y productos (Basili).

Tabla 2.1: Visión General de la Reutilización

2.1.4 OBJETO DE LA REUTILIZACIÓN.

En este apartado se hace énfasis en el objeto de la reutilización desde la perspectiva del nivel de abstracción del componente. Lo que diferencia un tipo de componente de otro es su nivel de abstracción. En términos generales cuanto más alto sea el nivel de abstracción, la potencialidad para entender, redefinir y adaptar será mayor. Es importante entonces considerar la efectividad de la abstracción en una técnica de reutilización de software. Krueger evalúa esta característica en términos del esfuerzo intelectual requerido para utilizar el componente (Krueger, 1992). Las

mejores abstracciones significan que el usuario requiere menos esfuerzo para su utilización. La distancia cognitiva es definida como la cantidad de esfuerzo intelectual realizado por los desarrolladores para tomar un componente de software en un estado y llevarlo a otro estado; aunque no es una medida formal que puede ser expresada en números y unidades da una noción intuitiva del esfuerzo de reutilizar.

Toda abstracción de software tiene dos niveles (figura 2.2). El nivel más alto se refiere a la especificación de la abstracción y el nivel más bajo se refiere a la realización de la abstracción, que no necesariamente se trata del código fuente. A su vez, una abstracción debe estar conformada por una parte oculta (h), una parte variable (v) y una parte fija (f). La parte oculta contiene detalles de realización de la abstracción que no son visibles en el nivel de especificación de la abstracción; la parte variable representa las características modificables en la realización de la abstracción y la parte fija representa las propiedades de la abstracción que determinan su comportamiento básico. En la figura 2.2, se tiene un artefacto X con dos abstracciones M y L, con su respectiva distancia cognitiva (dc1, dc2). Si la abstracción M es por ejemplo la especificación estructural de una venta escrita en un modelo entidad-relación, la realización de esta abstracción será el esquema lógico de la base de datos que será a la vez la especificación L del siguiente nivel; la realización de L será, por ejemplo, una representación en estructuras de registros del esquema lógico. A continuación se describen los principales objetos de reutilización y su evaluación frente al esfuerzo requerido para su reutilización.

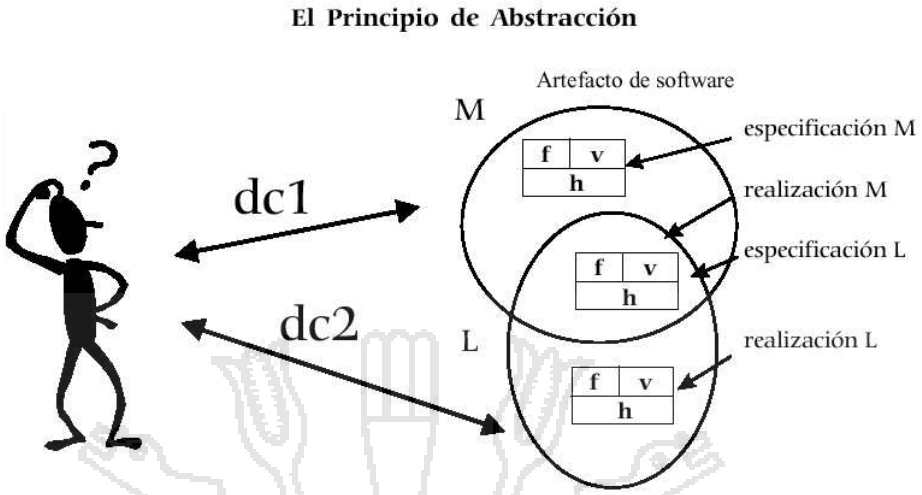


FIGURA 2.2: EL PRINCIPIO DE ABSTRACCIÓN.

Código Reciclado.

Es la aproximación primitiva de reutilización donde los programadores recuperan código fuente generalmente construido por ellos mismos en desarrollos anteriores con el propósito de reducir el esfuerzo de diseño, escritura y prueba del nuevo software. La reutilización está originada por la experiencia y capacidad del programador de recordar desarrollos anteriores que guardan analogía con el nuevo problema. En estos casos generalmente la persona que reutiliza es la misma que construye lo cual disminuye la distancia cognitiva.

Componentes de Código.

Esta fue la noción inicial de reutilización que introdujo McIlory: componentes de código adquiridos con el propósito de ensamblarlos para construir nuevo software. Estos componentes son escritos, evaluados y almacenados específicamente con el propósito de ser reutilizados. Los lenguajes de programación orientados a objetos impulsaron considerablemente este tipo de reutilización. Hoy en día, son innumerables las librerías que se distribuyen con una funcionalidad específica y modularizada por naturaleza a través de la definición de

clases. Por el principio de encapsulación, la definición de la interfaz de los métodos (nombre, argumentos de entrada y argumentos de salida) que ofrece cada clase, permite un acercamiento gradual al código sin tener que conocer los detalles de implantación de cada método. En otras palabras, la definición de clase es un mecanismo de abstracción que separa las propiedades de la clase (especificación) de sus instancias (realización). Sin embargo, en este tipo de reutilización, las clases y métodos representan unidades de reutilización a muy pequeña escala, originando un problema de composición para ensamblar una funcionalidad útil en el dominio del problema. La distancia cognitiva depende del grado de familiaridad que el programador tiene de las bibliotecas de clase, algunas de las cuales no ofrecen una documentación adecuada.

Esquemas.

Enfatiza la utilización de abstracciones para representar una solución software a un alto nivel de abstracción. El esquema describe un conjunto de especificaciones escritas bajo determinado formalismo (modelo entidad relación, diagramas de flujos de datos, diagramas de clases, especificaciones formales etc.) que recibe el nombre de modelo conceptual. La reutilización de estos modelos permite que el analista aproveche especificaciones ya existentes y validadas para la construcción de nuevas especificaciones. El alto nivel de abstracción de la especificación reduce considerablemente la distancia cognitiva entre el requerimiento y la implementación de algoritmos y estructuras de datos. La mayoría de productos y esfuerzos de investigación de reutilización a alto nivel se concentran en la reutilización de modelos conceptuales.

El trabajo de Altmeyer define modelos conceptuales en diferentes formalismos orientados a dominios específicos y un entorno de desarrollo que permite derivar y caracterizar nuevos modelos (Altmeyer & Otros, 1997). El trabajo de Ruggia utiliza modelos entidad–relación extendidos y provee un conjunto de herramientas que ofrecen servicios de reutilización (Ruggia & Ambrosio, 1997). Trabajos como el de

Ambrosio (Ambrosio, 1996) y Castaño (Castaño & De Antonellis, 1997) también utilizan el modelo entidad-relación como formalismo para representar una especificación e introducen además un modelo de métricas para medir la afinidad entre diferentes modelos conceptuales. Bajo el enfoque Orientado a Objetos uno de los trabajos más representativos es el proyecto Ithaca, que desarrolló un ambiente para reutilización de especificaciones a diferente nivel de abstracción, en el que utiliza un modelo de especificación OO con funcionalidad extendida llamado FORM (Funcionality Object with Roles Model). Este modelo enriquece el concepto de clase para manejar clases que representan comportamientos (clases de procesos) además de las clases que describen estructura y comportamiento (clases de recursos) (Bellinzona, Fugini, & Pernici, 1995).

Frameworks.

Una nueva tendencia para facilitar la reutilización de arquitecturas de software es el concepto de framework. Un framework es una solución integrada ejecutable de comunicación entre un conjunto de clases abstractas que representan un comportamiento útil en el espacio del problema. Los framework representan soluciones en un contexto particular y proveen mecanismos eficientes para la adecuación del comportamiento a una nueva necesidad. El trabajo de Baumer, por ejemplo, presenta un framework para desarrollo de sistemas a grande escala en proyectos financieros (Baumer & Otros, 1997). Otros trabajos proponen el desarrollo de framework a partir de patrones de diseño. Los patrones de diseño ofrecen una terminología y comportamiento básico que permite capturar las especificaciones de un diseño reutilizable orientado a objetos (Pree, 1994) (Meijler & Otros, 1997).

En resumen, un alto nivel de abstracción en las técnicas de reutilización, reduce el esfuerzo requerido para ir del concepto inicial (requerimiento) a su representación en una especificación y automáticamente reduce el esfuerzo para ir de la abstracción a una implementación ejecutable.

2.1.5 NIVELES DE REUTILIZACIÓN.

La adopción de reutilización en una organización ha de seguir una evolución incremental donde se parte de soluciones tecnológicas más simples como las bibliotecas de clases a soluciones de mayor complejidad como son las arquitecturas de referencia.

Griss propone un modelo de evolución incremental en la adopción de la reutilización por una organización que conlleva cinco etapas. Este modelo es independiente de la utilización de orientación a objetos aunque se puede adaptar bien a esta tecnología. Las cinco etapas en la evolución de la reutilización son:

2.1.5.1 CORTA-PEGA.

En esta primera etapa o fase de la evolución, la organización trata de disminuir los tiempos de desarrollo mediante la clonación de trozos de código. Aunque es una solución a primera vista, los problemas aparecen en el mantenimiento, pues a la hora de realizar cambios hay que ser consciente de todos los sitios en el código donde estos se aplican. Es un problema típico de la gestión de configuración.

2.1.5.2 CAJA NEGRA.

En la reutilización de caja negra se identifican componentes software de uso común en la organización. Se garantiza que existe un original único de ellas. La utilización de bibliotecas y taxonomías de clasificación puede ayudar en esta fase de la evolución de la organización.

2.1.5.3 ACTIVOS REUTILIZABLES.

En esta fase la organización considera otros activos reutilizables distintos del código. Entre estos destacan principalmente los procedimientos de prueba, las especificaciones, ficheros de ayuda, herramientas y otros. Con esta aproximación se

incrementa la reutilización en las diversas fases del ciclo de vida del software, no sólo en la fase de implementación como ocurría especialmente en la fase de Caja Negra.

2.1.5.4 ARQUITECTURAS.

En esta fase se generan componentes reutilizables, que podrían ser los anteriores, y una arquitectura software que los aglutina, permitiendo desarrollar aplicaciones en un dominio específico de negocio, sea este financiero, control de tráfico aéreo, gestión de red u otros. Las guías para la creación de arquitecturas de referencia o específicas de dominio de este proyecto pueden ser utilizadas en dominios de aplicación diferentes.

2.1.5.5 REUTILIZACIÓN SISTEMÁTICA.

La reutilización sistemática en una organización se basa en la estandarización de los activos reutilizables y los procesos para producirlos, la creación de una infraestructura para la producción de estos activos y los mecanismos organizativos adecuados para facilitar la reutilización de estos. La separación del personal en grupo de ingeniería de dominio, o responsable de crear y mantener los activos reutilizables y el grupo de ingeniería de aplicación, responsable de utilizarlos, es un aspecto fundamental en esta fase.

2.1.6 EL PROCESO DE REUTILIZACIÓN.

En la figura 2.3 se muestran las actividades de reutilización tanto para el productor como para el consumidor de un componente. El productor es responsable de la creación del componente. Si el objetivo fuera simplemente crear una aplicación independiente, la responsabilidad del productor terminaría ahí. Sin embargo, para hacer posible un componente reutilizable, el productor también deberá preparar al componente para su reutilización. Esta tarea implica la responsabilidad adicional del diseño y de la construcción para la reutilización. La

generalización, la selección de la interfaz del componente y la definición de los parámetros de implementación son algunas de las técnicas específicas que se aplicarán aquí. Las pruebas iniciales del componente también serán efectuadas por el productor.

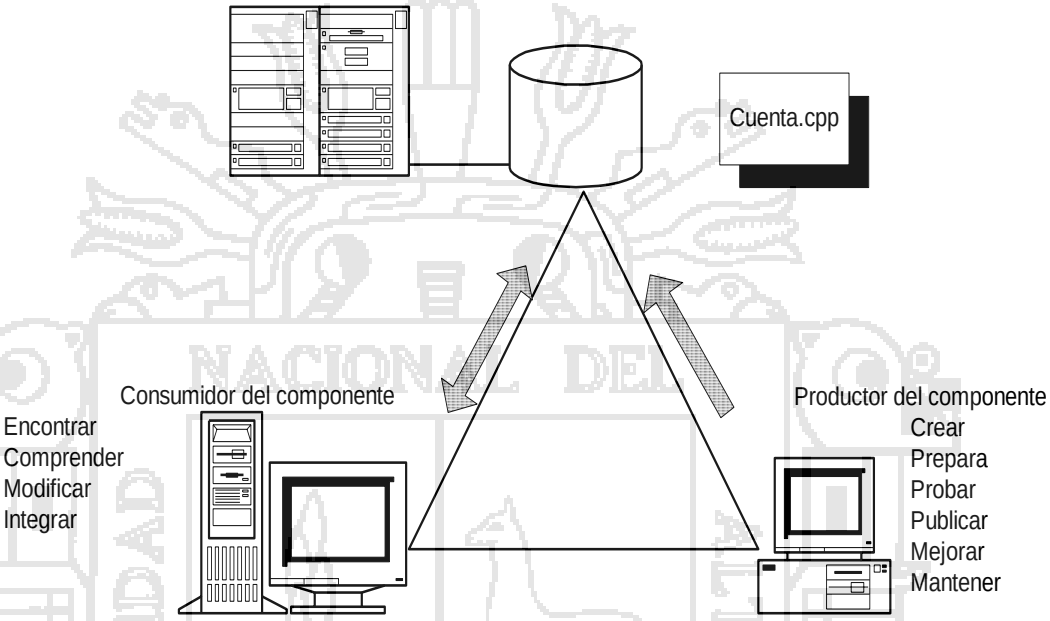


FIGURA 2.3: ACTIVIDADES DE REUTILIZACIÓN PARA EL PRODUCTOR Y EL CONSUMIDOR DE UN COMPONENTE.

El productor deberá luego publicar de alguna forma dicho componente. El procedimiento más sencillo será colocar el componente en un directorio público, es decir, en un depósito. Alternativamente, se le podría solicitar al productor que someta al componente a un grupo dedicado a administrar componentes reutilizables o bien que lo introduzca en un depósito (en la figura 2 se sugiere este último procedimiento). En este caso, también se le requeriría al productor que catalogue el componente o que proporcione documentación adicional. Después de su publicación inicial, los componentes pueden requerir trabajos posteriores, como mejora, mantenimiento y pruebas subsecuentes. Estas tareas podrían quedar asignadas al productor, al consumidor, o a un grupo especializado.

Llegado a este punto, el componente estará disponible para su reutilización. Sin embargo, para que esto ocurra, el consumidor tendrá que estar consciente tanto de los componentes disponibles como del mecanismo de recuperación. El consumidor también deberá reconocer que algún aspecto de la nueva aplicación puede parecerse a alguna especificación o implementación existente. El primer paso será encontrar un componente apropiado. El consumidor podría automatizar dicha tarea utilizando un catálogo o herramienta de búsqueda. Sin embargo, la mayoría de los desarrolladores busca simplemente componentes candidatos en directorios compartidos o en otras localizaciones. La búsqueda puede implicar la lectura de la especificación del componente y quizá su prueba. Los desarrolladores también leen el código fuente, aunque este paso realmente es indicación de que la especificación no es adecuada para juzgarlo. Si el componente es adecuado, se hace una copia local.

El desarrollador puede entonces empezar a adquirir una comprensión detallada del componente. Este proceso a menudo requiere entender la implementación. Quizá finalmente esta necesidad se elimine mediante técnicas de especificación mejoradas o una validación más completa de los componentes. Una vez comprendido el componente por parte del consumidor, será integrado en la aplicación inmediata. Esta tarea puede requerir modificaciones tanto de la nueva aplicación como del componente. Sin embargo, un desarrollador cuidadoso introduciría modificaciones locales sólo mediante una especialización o alguna otra extensión útil del componente.

2.1.6.1 ETAPAS EN EL PROCESO DE REUTILIZACIÓN.

- **Etapa 1:** Adquisición del requerimiento.

  Es el paso inicial donde el analista especifica al sistema una necesidad de información. Idealmente esta necesidad debe estar formulada en el contexto del problema y no en

contexto de la solución. Es decir, el analista no debe estar obligado a conocer el dominio de la solución. En estos casos el modelo del dominio se convierte en un modelo de requerimientos que facilita la labor de precisar un requerimiento.

- **Etapa 2:** Búsqueda y Recuperación.

  Es el mecanismo de búsqueda que permite evaluar los objetos de la biblioteca para recuperar el subconjunto de objetos candidatos a reutilizar. El problema principal radica en cómo organizar y recuperar los diversos componentes de software que son potencialmente reutilizables de tal manera que cualquier desarrollador pueda accederlos de manera consistente y flexible. La mayoría de las técnicas de recuperación tienen su origen en las técnicas utilizadas para recuperación de elementos en una base de datos documental.

- **Etapa 3:** Identificación.

  En esta etapa se examinan los objetos recuperados para seleccionar aquellos que posean la funcionalidad deseada. Aquí es deseable contar con un sistema de métricas que mida la afinidad entre los componentes del esquema. Es decir, la métrica de similaridad se convierte en una herramienta que ayuda al diseñador para identificar los componentes adecuados a un requerimiento. La identificación es un paso posterior que complementa la recuperación de la etapa anterior. Mientras que la recuperación (retrieve) permite obtener de la biblioteca un conjunto de componentes utilizando un criterio general de selección, la identificación (browsing) es un proceso más detallado que explora los componentes recuperados teniendo un componente como referencia (current object) y

generando una vista del vecindario de interés, generalmente con alguna clase de medida de distancia (Constantopoulos & Pataki, 1992) (Motro & Goullioud, 1995).

- **Etapa 4:** Adecuación.

  En esta etapa se debe disponer de herramientas para abstraer componentes afines y adecuarlos a sus nuevas necesidades. Las operaciones de adecuación pueden darse al interior de un modelo (intra-esquemas) o entre diferentes modelos (inter-esquema). El trabajo de Ruggia, por ejemplo, define operaciones de unión, intersección, substracción y extracción entre los componentes de esquemas diferentes (Ruggia & Ambrosio, 1997).

2.1.7 EL ENTORNO DE REUTILIZACIÓN.

La reutilización de componentes de software debe de estar apoyada por un entorno que abarque los elementos siguientes:

- Una base de datos de componentes capaz de almacenar componentes de software, así como la información de clasificación necesaria para recuperarlos.
- Un sistema de gestión de bibliotecas que proporcione acceso a la base de datos.
- Un sistema de recuperación de componentes de software (por ejemplo: un distribuidor de solicitudes de objetos) que permita que la aplicación cliente recupere los componentes y servicios del servidor de la biblioteca.
- Unas herramientas CASE que presten su apoyo a la integración de componentes reutilizados en un nuevo diseño o implementación.

Cada una de estas funciones interactúa con una biblioteca de reutilización o bien se encuentra incorporada a esta última.

La biblioteca de reutilización es un elemento de un depósito CASE más extenso y proporciona posibilidades adecuadas para el almacenamiento de una amplia gama de artefactos reutilizables (por ejemplo: especificación, diseño, casos de prueba, guías para el usuario). La biblioteca abarca una base de datos y las herramientas necesarias para consultar esa base de datos y recuperar de ella componentes. Un esquema de clasificación de componentes servirá como base para las consultas efectuadas a la biblioteca.

2.1.8 SOPORTE OPERATIVO.

El proceso de producción de un determinado sistema software mediante reutilización sólo tiene sentido si existe un enlace entre el desarrollo para reutilización, donde los componentes son producidos, y el desarrollo con reutilización, donde éstos son utilizados. Esto lleva a la necesidad de contar con un almacén de componentes software que enlace los dos procesos.

Estos almacenes se conocen como repositorios de reutilización, constituyéndose en elementos centrales para el soporte operativo de la reutilización.

FIGURA 2.4: LA REUTILIZACIÓN Y EL PROCESO DE PRODUCCIÓN DE SOFTWARE.

2.1.9 REPOSITORIOS Y/O BIBLIOTECAS DE REUTILIZACIÓN.

2.1.9.1 CONCEPTO.

No obstante, el concepto de repositorio es un concepto amplio que va desde sencillos sistemas de almacenamiento hasta complejos entornos que incorporan, además de los sistemas de almacenamiento, conjuntos de herramientas de ayuda al proceso de reutilización.

Debe tenerse presente que un repositorio no es un fin en sí mismo, sino un soporte al proceso de reutilización, de forma que el esquema del repositorio ha de responder al modelo de elemento software reutilizable y al tipo de reutilización adoptado por la organización que lo pone en funcionamiento.

En (Bernstein & Dayal, 1994) se define repositorio como una base de datos de información compartida sobre los elementos que se producen o se usan en un desarrollo software.

Inicialmente el concepto de repositorio se corresponde con una simple base de datos para el almacenamiento de componentes software. Sin embargo, el concepto de repositorio evoluciona hacia entornos más sofisticados, con complejos métodos de almacenamiento, búsqueda, navegación, examen detallado de los componentes almacenados y recuperación (Kara, 1997) (Viasoft Inc., 1997).

Esta evolución del concepto de repositorio casa con el concepto de biblioteca de reutilización propugnado por el DoD

(Department of Defense) de EEUU y los estándares de la OTAN para reutilización. Así, una biblioteca de reutilización se puede definir como una colección de elementos software reutilizables, junto a los procedimientos y funciones de soporte requeridas para ofrecer los componentes a los usuarios (NATO, 1992).

De esta forma en la literatura especializada se alternan los términos repositorio y biblioteca, aunque con idéntico significado. Para una mayor información sobre los repositorios se recomienda la consulta de (Marqués Corral, 1998).

2.1.9.2 TIPOS DE REPOSITORIOS.

2.1.9.2.1 REPOSITORIOS COMO SOPORTE A LAS FASES DE DESARROLLO.

Repositorios de Diseño.

Estos repositorios almacenan componentes referidos al diseño de procesos, diseño de base de datos, así como información de atributos.

Repositorios de Aplicación.

Este tipo de repositorios normalmente se encuentran enlazados con herramientas de desarrollo de aplicaciones. Además, almacenan y distribuyen objetos y métodos definidos dentro de la herramienta.

Repositorios Universales.

Tienen por función almacenar y gestionar información de diseño así como objetos y metadatos.

2.1.9.2.2 REPOSITORIOS COMO ENTORNOS DE DESARROLLO.

Repositorios Propietarios.

Forman parte de grandes sistemas de desarrollo. El soporte que ofrecen puede ser exclusivo para los elementos del sistema. C++ Builder, Delphi, Object Manager.

Repositorios Dependientes.

Diseñados para trabajar con otro producto específico. OIS Repository (Ontos Inc). Entre las funciones principales tenemos:

- Enlaces entre bases de datos relacionales y programación or.obj
- Almacena esquemas relacionales; modelos objeto y las correspondencias objeto/relacional.
- Soporte el acceso a la base de datos en tiempo de ejecución utilizando la correspondencia.

Repositorios Independientes.

Soportan software que puede trabajar con una gran variedad de productos. Un ejemplo de este es el Universal Repository (Unisys Corp) que tiene un modelo de información propio (metamodelo) y se pueden definir modelos propios.

2.1.10 DESCRIPCIÓN DE COMPONENTES REUTILIZABLES.

Un componente de software reutilizable se puede describir de muchas formas, pero la descripción ideal abarca lo que Tracz ha llamado el modelo 3C –concepto, contenido y contexto.

El concepto de un componente de software es “una descripción de lo que hace el componente”. La interfaz con el componente se describe por completo, y su semántica –representada dentro del contexto de pre y post

condiciones- se identifica también. El concepto debe de comunicar la intención del componente.

El contenido de un componente describe la forma en que se construye el concepto. En esencia, el contenido es una información que queda oculta a los ojos del usuario habitual y que solamente necesita conocer quién intente modificar ese componente.

El contexto sitúa el componente de software reutilizable en el seno de su dominio de aplicabilidad, esto es, mediante la especificación de características conceptuales, operaciones y de implementación, el contexto capacita al ingeniero del software para hallar el componente adecuado para satisfacer los requisitos de la aplicación.

2.1.11 CLASIFICACIÓN / RECUPERACIÓN DE COMPONENTES.

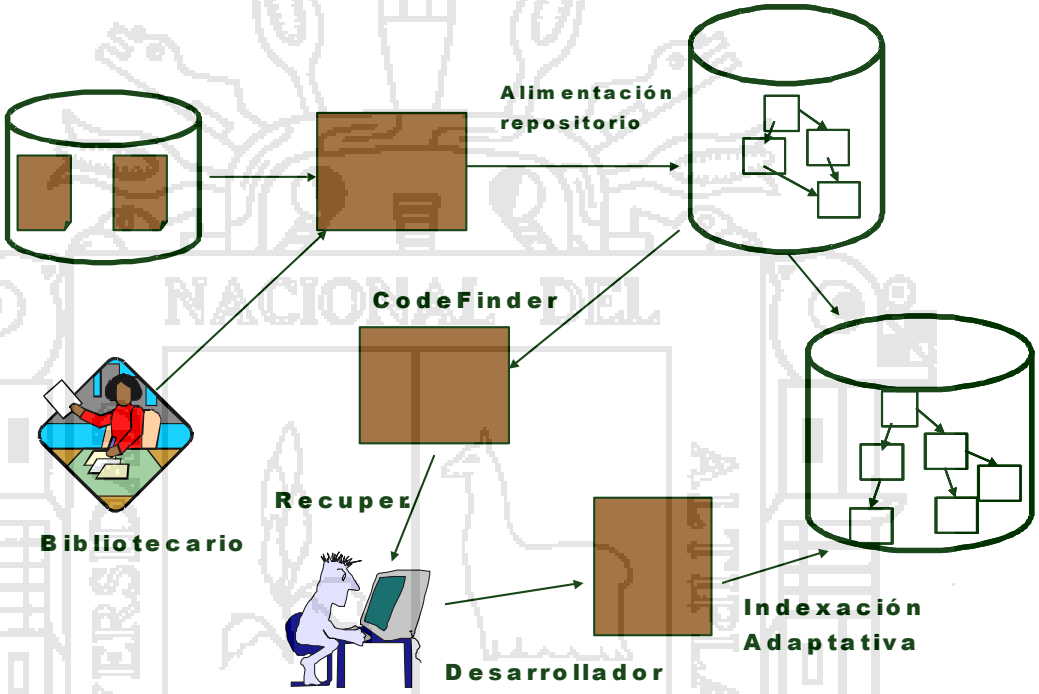
Considere una gran biblioteca universitaria. Están disponibles para su utilización decenas de millares de libros, periódicos y otras fuentes de información. Ahora bien, para acceder a estos recursos, es preciso desarrollar algún sistema de clasificación. Para recorrer este gran volumen de información, los bibliotecarios han definido un esquema de clasificación que incluye un código de clasificación de la biblioteca, palabras reservadas, nombres de autor, y otras entradas de índice. Todas ellas capacitan al usuario para encontrar el recurso necesario de forma rápida y sencilla.

Considere ahora un gran depósito de componentes. Residen en él decenas de millares de componentes de software reutilizables. ¿Cómo puede hallar el ingeniero del software el componente que necesite? Para responder a esta pregunta, surge otra más. ¿Cómo se describen los componentes de software en términos no ambiguos y fácilmente clasificables? Se trata de cuestiones difíciles y todavía no se ha desarrollado una respuesta definitiva. En esta sección, se exploran las

tendencias actuales que capacitaran a los futuros ingenieros del software para navegar las bibliotecas reutilizables.

FIGURA 2.5: ESQUEMA SIMPLIFICADO DE CLASIFICACIÓN/RECUPERACIÓN DE COMPONENTES.

Para que resulte útil en un sentido práctico, el concepto, el contenido y el contexto tienen que traducirse a un esquema de especificación concreto.



Los métodos de clasificación/recuperación propuestos se pueden descomponer en tres zonas fundamentales:

- Métodos de las ciencias de la documentación y de biblioteconomía.
- Métodos de inteligencia artificial (basado en datos y modelos de conocimiento)
- Métodos basados en Sistemas de hipertexto.

2.1.11.1 MÉTODO DE LAS CIENCIAS DE LA DOCUMENTACIÓN Y DE BIBLIOTECONOMIA.

La figura 2.6 presenta una taxonomía de los métodos de indización en biblioteconomía. Los vocabularios controlados de indización limitan los términos y sintaxis que se pueden utilizar para clasificar los objetos (componentes). Los vocabularios de indización no controlados no imponen restricciones sobre la naturaleza de la descripción. La gran mayoría de los esquemas de clasificación para los componentes software caen dentro de tres categorías:

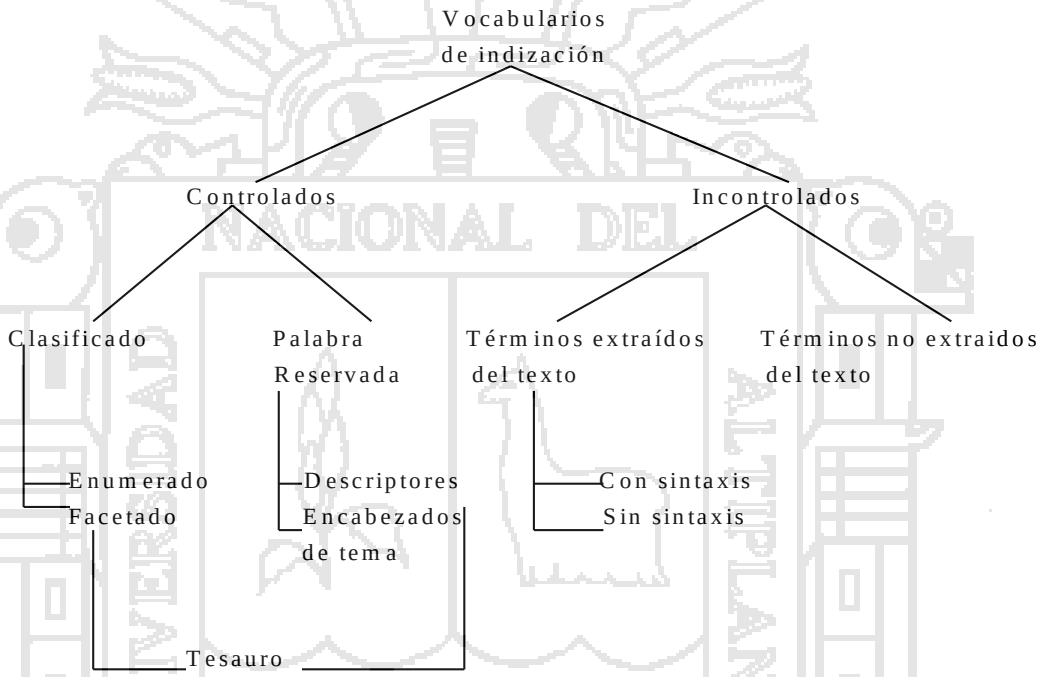


FIGURA 2.6: UNA TAXONOMÍA DE MÉTODOS DE INDIZACIÓN PROCEDENTE DE LA BIBLIOTECONOMÍA.

2.1.11.1.1 CLASIFICACIÓN ENUMERADA O JERÁRQUICA.

Se describen los componentes mediante la definición de una estructura jerárquica en la cual se definen clases y diferentes niveles de subclases de componentes de software. Los componentes en si se enumeran en el nivel más bajo de cualquier ruta de la jerarquía enumerada.

La estructura jerárquica de un esquema de clasificación enumerado hace que sea sencillo comprenderlo y utilizarlo. Sin embargo, antes de que sea posible construir una jerarquía, es preciso llevar a cabo una ingeniería del dominio para que esté disponible una cantidad suficiente de conocimientos acerca de las entradas correctas de esa jerarquía.

D.- SOFTWARE
- **D.1.- TÉCNICAS DE PROGRAMCIÓN**
  - **D.1.5.- PROGRAMACIÓN ORIENTADA AL OBJETO**
- **D.2.- INGENIERIA DEL SOFTWARE**
  - **D.2.1.- REQUISITOS/ESPECIFICACIONES**
    - **LENGUAJES**
    - **METODOLOGIAS**
    - **HERRAMIENTAS**
  - **D.2.2.- HERRAMIENTAS Y TÉCNICAS**
    - **CASE**
    - **MODULOS E INTERFACES**
    - **SOFTWARE LIBRARIES**
    - **USER INTERFACES**
  - **D.2.5.- TESTING**
  - **D.2.6.- ENTORNOS DE PROGRAMACIÓN**
  - **D.2.10.- DISEÑO**
    - **METODOLOGIAS**
    - **REPRESENTACION**
  - **D.2.11.- MISCELANEA**
    - **PROTOTIPADO RÁPIDO**
    - **SOFTWARE REUTILIZABLE**
- **D.3.- LENGUAJES DE PROGRAMACION**
  - **D.3.2.- CLASIFICACION DEL LENGUAJE**
    - **LENGUAJES DE DISEÑO**
    - **LENGUAJES ORIENTADOS A OBJETOS**
  - **D.3.3.- CONSTRUCTORES Y CARACTERÍSTICAS DE LOS LENGUAJES**
    - **ADT**
    - **TIPOS Y CONSTRUCTORES**

FIGURA 2.7: EJEMPLO DE ESQUEMA DE CLASIFICACIÓN JERÁRQUICA.

2.1.11.1.2 CLASIFICACIÓN POR FACETAS.

Se analiza una cierta área de dominio y se identifica un conjunto de características descriptivas básicas. Estas características, que se denominan facetas, reciben entonces diferentes prioridades según su importancia, y se relacionan con un componente. La faceta puede describir la función que lleva a cabo el componente, los datos que se manipulan, el contexto en que se aplican, o cualquier otra característica. El conjunto de facetas que describen un componente se denomina descriptor de facetas. Generalmente, la descripción por facetas se limita a un máximo de siete u ocho facetas.

Como ilustración sencilla del uso de facetas en la clasificación de componentes, considérese un esquema que hace uso del siguiente descriptor de facetas:

{función, tipo de objeto, tipo de sistema}

Cada una de las facetas del descriptor de facetas adopta uno o más valores que en general serán palabras reservadas descriptivas. Por ejemplo, si función es una faceta de un componente, entonces entre los valores típicos que se asignen a esta faceta podríamos contar:

función = (copiar, desde) o bien (copiar, sustituir todo)

La utilización de múltiples valores de faceta hace posible que la función primitiva copiar se refine más exactamente.

Se asignan las palabras reservadas (valores) al conjunto de facetas de cada componente de una

cierta biblioteca de reutilización. Cuando un ingeniero del software desea consultar la biblioteca en busca de posibles componentes para un diseño, se especifica una lista de valores y se consulta la biblioteca en busca de posibles candidatos. Se puede emplear herramientas automatizadas, esto hace posible que la búsqueda abarque no sólo las palabras reservadas especificadas por el ingeniero del software, sino también otros sinónimos técnicos de esas mismas palabras reservadas.

Un esquema de clasificación por facetas proporciona al ingeniero del dominio una mayor flexibilidad al especificar descriptores complejos de los componentes. Dado que es posible añadir nuevos valores de facetas con facilidad, el esquema de clasificación por facetas es más fácil de extender y de adaptar que el enfoque de enumeración por lo siguiente:

- Esquema multi-dimensional (n descriptores).
- Faceta = perspectiva, punto de vista o dimensiones de un dominio particular.
- Los términos se agrupan en un número fijo de facetas mutuamente excluyentes.
- Esquema más flexible que los enumerados. La modificación en una faceta no afecta al resto.
- Se mantiene el problema de encontrar la combinación "correcta" de términos que describan con precisión la información necesaria.
- Requiere que el usuario conozca la estructuración de los términos y el significado de cada faceta.

2.1.11.1.3 CLASIFICACIÓN DE ATRIBUTOS Y VALORES.

Se define un conjunto de atributos para todos los componentes de una cierta zona de dominio. A continuación se asignan valores a estos atributos de forma muy parecida a una clasificación por facetas. De hecho, la clasificación de atributos y valores es parecida a la clasificación por facetas con las excepciones siguientes: (1) no hay límite para el número de atributos que se pueden utilizar; (2) no se asignan prioridades a los atributos; y (3) no se utiliza la función de tesauro.

Parece entonces que es preciso realizar un trabajo más extenso en el desarrollo de unos esquemas de clasificación efectivos para las bibliotecas de reutilización.

2.1.11.2 MÉTODOS DE INTELIGENCIA ARTIFICIAL (BASADOS EN DATOS Y MODELOS DE CONOCIMIENTO).

2.1.11.2.1 MARCOS DE CONOCIMIENTO.

Estas técnicas se caracterizan por hacer un tipo de análisis lexicográfico, sintáctico y semántico de las especificaciones en lenguaje natural de los componentes software, sin pretender una comprensión completa de los documentos. Se apoyan en una Base de Conocimientos que almacena la información semántica sobre un dominio de aplicación específico, y sobre el propio lenguaje natural. Estos sistemas son más poderosos que los que usan palabras clave, por contra, requieren enormes recursos humanos para construir las bases de conocimiento -dependientes del dominio- y poblarlas.

2.1.11.2.2 ESPECIFICACIONES FORMALES.

Existen varios sistemas de recuperación de información sobre la base de especificaciones formales. En todos estos acercamientos las consultas son especificaciones formales de requerimientos, y el sistema devuelve los componentes relevantes desde el repositorio, componentes que, a su vez, están especificados formalmente. La recuperación se hace mediante un demostrador de teoremas, para comprobar si las especificaciones del componente satisfacen los requerimientos formales de la consulta. Entre las ventajas destacadas figuran:

- Las especificaciones formales se centran en el comportamiento del componente, en lugar de su descripción.
- No presentan ambigüedad y proporcionan mayor precisión que los métodos informales.

Por otra parte, tiene desventajas, entre las que cabe citar:

- Solo una parte muy pequeña del software disponible para reutilización se encuentra especificado formalmente.
- El tiempo de proceso para los algoritmos de búsqueda puede ser excesivo.
- Es más fácil -y barato- usar las descripciones textuales que las especificaciones formales.
- La concordancia parcial, sobre la base de similitud, es muy difícil de controlar, dado que estos

acercamientos se basan más en concordancia exacta.

- Finalmente, el actual estado de la tecnología de demostración de teoremas hace imposible la implementación de muchos de estos acercamientos.

Es probable que si el desarrollo formal de software se hace lo suficientemente popular como para justificar la inversión en la construcción de bases de especificaciones formales, estas especificaciones serán la principal fuente de información para las actividades de indización (clasificación). No solo por el hecho de que esta información proviene directamente del componente software (y no de la documentación), sino también porque parece ser la forma más natural de determinar las diferencias entre el componente buscado y el encontrado en el repositorio, y saber así el esfuerzo necesario para adaptarlo (reutilizarlo). El principal problema es que los actuales acercamientos para la recuperación mediante especificaciones formales usan formalismos que tal vez no sean adecuados para lograr una indización efectiva, y por tanto, son insuficientes para permitir una recuperación automática.

2.1.11.2.3 REDES NEURONALES.

En los sistemas de recuperación sobre la base de un esquema de clasificación, el cálculo de la similitud entre componentes suele hacerse por medio de un grafo de distancia conceptual, el cual establece la similitud entre los términos que describen al

componente en su representación interna, pertenecientes -los términos- al vocabulario controlado. El ajuste manual del grafo, a medida que aumenta el repositorio, es imposible, al crecer el número de conexiones (arcos) en forma combinatoria Sin embargo, los métodos sobre la base de redes neuronales, atacan el problema de forma distinta. En lugar de grafos de distancia conceptual, se usan redes neuronales artificiales para estructurar el repositorio de acuerdo a la similitud funcional de los componentes almacenados, es decir, los componentes que presentan un comportamiento similar son almacenados cerca, el uno del otro. Por tanto, la similitud entre componentes se hace explícita (distancia geográfica). Las funciones de vecindad usadas por el algoritmo se basan en las medidas tradicionales de similitud en los modelos de espacio vectorial.

2.1.11.3 MÉTODOS BASADOS EN SISTEMAS DE HIPERTEXTO.

2.1.11.3.1 BROWSING.

La mayoría de las técnicas anteriores utilizan la recuperación lineal como su principal mecanismo, es decir, recuperar un conjunto de componentes potencialmente reutilizables a partir de una especificación del usuario sobre lo que desea. Sin embargo, cada vez más aplicaciones de clasificación/recuperación incorporan -como mecanismo secundario- la visualización rápida (hojear, browsing en inglés). Por lo general, el proceso de hojear comienza con el conjunto de candidatos recuperado a partir de la consulta del

usuario. También han sido propuestos mecanismos basados únicamente en visualización rápida, la mayor parte sobre la base de niveles de jerarquía. La principal desventaja de estas técnicas es que requieren que sea el propio usuario el encargado de manejar el proceso de navegación, lo cual puede resultar confuso y difícil en grandes repositorios.

2.1.11.3.2 HIPERTEXTO.

Son varias las propuestas para recuperación sobre la base de la tecnología hipertexto. En tales sistemas la información se organiza como una red de nodos -unidades de información- que se interconectan por medio de enlaces y relaciones. El usuario puede navegar por la red siguiendo los enlaces preestablecidos, y guiándose en este proceso por la semántica de cada enlace.

El principal problema de esta técnica es que el desarrollo y mantenimiento de un repositorio en un ambiente hipertexto requiere de una inversión muy grande en recursos humanos. Otras desventajas apuntan a:

- No existen buenas sugerencias en los enlaces para permitir decidir cuál seguir.
- No siempre la red formada por los enlaces es la más adecuada, más aún, es imposible decidir si existe una red óptima
- El proceso de añadir un nuevo componente a la red es tremendamente complicado, pues deben considerarse todos los posibles enlaces que se relacionen con el nuevo componente.

- En el proceso de eliminar un componente de la red debe asegurarse la destrucción de todos los enlaces que llevaban a él.

No existe garantía de encontrar lo que se busca, aunque exista un camino que lleve a él, el usuario puede viajar en "círculos" puesto que no es posible explorar todos los caminos (explosión geométrica).

2.1.12 BÚSQUEDA DE COMPONENTES.

2.1.12.1 REGLAS PARA CONSTRUIR EXPRESIONES DE BUSQUEDA.

La expresión de búsqueda puede contener uno o varios de los siguientes elementos:

- El término a buscar.
- Operador.

Un término puede ser:

- Una palabra o.
- Una frase.

Reglas.

- Para representar una **palabra o frase** se permiten solamente caracteres alfanuméricos. Al realizar la búsqueda se ignora si la palabra se escribió con letras mayúsculas o minúsculas.
- Los términos pueden ser unidos para formar una expresión con los operadores: "Y", "O", "NO" y paréntesis.
- Los operadores también se pueden representar en inglés o con el carácter especial que le corresponde a algunos de ellos.

- Los operadores se pueden teclear en mayúsculas o en minúsculas.

- Si un operador se encuentra al principio o al final de una expresión de búsqueda se toma como una palabra.

- Si se teclean términos separadas con espacios, se unirán con el operador "Y". Es decir los espacios en blanco entre términos hacen lo mismo que si se pusiera el operador AND entre ellas.

- El operador "NO" se puede usar solamente para limitar una búsqueda y solamente al final de la expresión, es decir, en una expresión debe haber una parte que no esté afectada por el operador "NO". Al hacer la búsqueda se localizarán los documentos que cumplan con la primera parte de la expresión y que no contengan el término afectado por el operador "No".

- Las frases deben de estar delimitadas por comillas dobles. Un ejemplo correcto de una frase es: "más vale tarde que nunca"

- Las frases son términos booleanos, por lo que se pueden hacer operaciones booleanas con ellas como si se tratase de palabras. Por ejemplo, una expresión válida sería: "más vale tarde que nunca" OR "sabiduría popular"

En caso de que una expresión cuenta con más de un operador de igual prioridad, entonces se evaluará la expresión de izquierda a derecha.

Prioridad	Operador	Tipo	Equivalentes	Descripción
1	**( )**	Precedencia		Sirve para delimitar expresiones que se desea que se realicen primero
2	**NO**	Lógico	no not	Los documentos encontrados deberán contener el objeto de búsqueda que se encuentra a la izquierda del operador

			NOT !	excluyendo los documentos que contengan el objeto de búsqueda que se encuentra a la derecha del mismo
3	**Y**	Lógico	y and AND & (espacio)	Los documentos encontrados deberán contener por fuerza los objetos de búsqueda unidos por dicho operador
4	**O**	Lógico	o or OR |	Los documentos encontrados deberán contener alguno de los objetos de búsqueda unidos por dicho operador

Tabla 2.2: Reglas para construir expresiones de búsqueda

2.1.12.2 OBJETO DE BUSQUEDA.

Las expresiones válidas son:

- Término
- (Objeto de búsqueda)
- Objeto de búsqueda operador lógico Objeto de búsqueda

Es importante mencionar que si la expresión de búsqueda no se teclea correctamente no se encontrarán los resultados esperados.

Ejemplos de expresiones de búsqueda.

Computadora; esta expresión indica que se quieren localizar documentos que contengan la palabra Computadora.

Algoritmo o Estructura; esta expresión indica que se quieren localizar documentos que contengan la palabra Algoritmo, documentos que contengan la palabra Estructura y documentos que contengan ambas palabras.

Base y Datos y Objetos; con esta expresión se quieren localizar documentos que contengan las 3 palabras Base, Datos y Objetos.

Base Datos Objetos; esta expresión es equivalente a la expresión anterior, al poner palabras separadas por espacios se utiliza automáticamente el operador "y".

Base y (Datos o Conocimientos); esta expresión indica que se quiere localizar documentos que contengan la palabra Base y cualquiera de las dos palabras Datos o Conocimientos.

Base y (Datos O Conocimientos) No Distribuida; esta expresión especifica la búsqueda de documentos que contienen la palabra Base, cualquiera de las dos palabras Datos o Conocimientos y además que no contenga la palabra Distribuida.

"Programación Orientada a Objetos"; esta expresión indica que se quiere localizar documentos que contengan esa frase.

CAPÍTULO III
MÉTODO DE INVESTIGACIÓN

3 METODOLOGÍA DE LA INVESTIGACIÓN.

3.1 HIPÓTESIS.

3.1.1 HIPÓTESIS GENERAL.

- Se ha logrado desarrollar un modelo de una herramienta que permita la reutilización del software para el soporte operativo.

3.1.2 HIPÓTESIS ESPECÍFICAS.

- Se ha logrado motivar el estudio de los principios y conceptos de familia inherentes al enfoque de reutilización.
- Se ha logrado definir los pasos necesarios para la adopción de los procesos orientados a la reutilización.
- Se ha logrado implementar un modelo de una herramienta que permita la reutilización del software para el soporte operativo.

3.2 OPERACIONALIZACIÓN DE VARIABLES.

3.2.1 VARIABLES INDEPENDIENTES.

- Componentes software.
- Documentación asociada al componente.

3.2.2 VARIABLE DEPENDIENTE.

- Componentes candidato.

3.2.3 INDICADORES E ÍNDICES DE RESPUESTA.

- Número de componentes candidato.

3.3 METODOLOGÍA.

Las metodologías de análisis y diseño con orientación a objetos constituyen, hoy en día, la mejor opción para diseñar y construir sistemas robustos, flexibles y confiables, en corto tiempo, aún para las aplicaciones más complejas. En este apartado, se pretende mostrar, muy en pequeño, el estado actual de la tecnología de objetos respecto de su aplicación en proyectos reales de desarrollo de software. Es muy difícil resumir y condensar en tan pocas páginas lo que es la Tecnología de Objetos (OT), pero al menos, se tratará de exponer mediante cuadros, algunos de los puntos prácticos acerca de técnicas y métodos orientados a objetos desde 1987.

METODOS	SIGLAS	AUTORES
Object Oriented Design	OOD	Grady Booch
Object Behaviour Analysis	OBA	Rubin & Goldberg
Methodology for Object Oriented S.E. of	MOSES	Henderson-Sellers & Edwards
Systems	GOOD	Seidewitz & Stark
General Object Oriented Design	OOSE	Ivar Jacobson
Object Oriented Software Engineering	VMT	IBM
Visual Modeling Technique		Texel
Texel	OMT	Rumbaugh y otros
Object Modeling Technique	BON	Nerson
Better Object Notation	OOSA	Shlaer & Mellor
Object Oriented System Analysis	OOSD	Wasserman et al.
Object Oriented Structured Design	SEOO	LBMS
Systems Engineering OO		Cook y otros
Syntropy	OOJSD	Jackson
Object Oriented Jackson Structured Design	HOOD	ESA
Hierarchical Object Oriented Design	OOA	Coad & Yourdon
Object Oriented Analysis	OOD	Coad & Yourdon
Object Oriented Design	OSA	Embley y otros
Object Oriented System Analysis		E. Colbert
Colbert	FOA	Andleigh/Gretzingr
Frame Object Analysis	SOMA	Ian Graham
Semantic Object Modelling Approach		Berard
Berard		Donald Firesmith

ADM3		Martin & Odell
Ptech	OOA&D	Reenskaug et al.
Obj Oriented Role Analysis Synthesis	OORASS	Coleman y otros
Structuring		Softeam
Fusion		Wirfs-Brock et al.
Desfray		Booch, Jacobson & Rumbaugh
Responsability Driven Design	UML	
Unified Modeling Language		

Tabla 4.1: Principales métodos de análisis y de diseño orientado a objetos

Caract / Método	OOA/OOD	OMT	OOADA	OL	OOAD	Fusion	OOSE	OOSD	MOSES	UML
Objetos	Nivel Objeto OOA model	Modelo Objeto	Diagrama Objeto	-	Diagramas Fern	-	-	Modelo estático	Modelo O/C	Diagramas de objeto
Objetos dinámicos	Tablas de Estado de Objetos	Diagramas de Estado	Diagramas de transición de estados	Diagramas de transición de estados	Diagramas de transición de estados	-	Diagrama de Transición de Estados	Transición de trabajo en red	Diagrama Objeto	Clases activas
Clases /Estructuras de clases	Nivel de Clases OOA model	Modelo Objeto	Diagrama de clases	Información de modelos; Diagrama de clases; diagramas de herencia	Esquema de objetos; composición de diagramas; diagramas Fern; Generación de Diagramas de especificación	Modelo de objetos; descripción de clases; grafica de herencia	Modelo del objeto dominio problema; Modelo de análisis	Modelo Estático	Modelo O/C; Modelo de Clases Especificación de servicio Especificación de herencia	Las clases estan comprendidas en Cosas que son estructuras. Hay Diagrama de clases, Clases Activas, Interfaces
Particionado de clases y Estructuras de clases	OOA Model	Modulos	Diagramas de Categoría de Clases	-	Diagrama de flujo de objetos	-	-	Subsistema-Ensamble	Técnicas de administración de complejidad	Diagrama de clase, Diagrama de casos de uso, Estructuras: Interfaces, Clases, Clases activas
Comunicación entre clases / objetos	OOA Modelo de conexión de mensajes	-	Sincronización sobre Diagrama Objeto, Interacción de diagramas	Diagramas de dependencia	Diagrama de comunicación de clases	Graficas de visibilidad; grafica de interacción de objetos	Diagramas de Interacción	Diagramas de interacción	Modelo Evento	Diagrama de colaboración, Diagrama de secuencia
Arquitectura Funcional	Servicio de diagramas	Digramas de flujo de datos	-	Diagramas de flujo de acción de datos	Diagrama de flujo de objetos	-	Model de casos de uso	-	Servicio de Especificación	Diagrama de actividades, Diagrama de componentes
Diagrama Sistemas Dinámicos	-	Diagrama de flujo de eventos	Diagramas de interacción	Modelo de comunicación de objetos; Diagrama de secuencia de control; modelo de acceso a objetos	Diagrama de eventos	Modelo de interfaz	Modelo de casos de uso; diagramas de interacción	-	Modelo Evento	Diagrama de colaboración, Diagrama de casos de uso
Implementación de propiedades	-	Subsistemas	Diagramas Modulo/Proceso	Diagrama de estructura de clases	-	-	Modelo de Implementación Modelo de examen	-	-	Estructuras: Clases, Interfaces, Colaboraciones, Casos de uso

Tabla 4.2: Comparación de las técnicas de las diferentes metodologías orientadas a objetos

Para la elección de la metodología adecuada se han advertido además, características tales como: permitirnos reducir la complejidad del sistema a evaluar; permitirnos representar las relaciones existentes entre los elementos constitutivos del sistema y; permitirnos modelar con el mismo énfasis los datos y los procesos. Es por estas razones y por las ventajas evidentes en la tabla 4.2, que hemos elegido la Unified Modeling Language – UML- como la metodología más apropiada para el modelamiento de la aplicación.

3.4 TIPO DE INVESTIGACIÓN.

Descriptiva, porque describe las etapas de análisis, diseño e implementación del modelo.

3.5 DISEÑO DE INVESTIGACIÓN.

El diseño es de tipo experimental, ya que se ha realizado un test de prueba del modelo de la herramienta implementada, donde se muestran sus resultados de la funcionalidad, apariencia y facilidad de uso, que indicaron las personas después de probar la herramienta al ser encuestados.

3.6 POBLACIÓN.

La población está conformado por todos los egresados de la Carrera Profesional de Ingeniería de Sistemas de la Universidad Nacional del Altiplano.

3.7 MUESTRA.

Dado que nuestra población es bastante grande, es que se ha optado por entrevistar a 42 egresados de las diferentes promociones a los que se pudo contactar y tuvieron la predisposición de participar en la presente investigación, quienes presentan las condiciones apropiadas para lograr este cometido.

Por lo tanto el tipo de muestreo utilizado es no probabilístico, es decir es intencionado o de manera directa.

3.8 MÉTODO DE INVESTIGACIÓN.

- Método científico

3.9 TÉCNICAS E INSTRUMENTOS.

Se ha utilizado la técnica de la encuesta con su respectivo instrumento que es la guía de la encuesta

3.10 MÉTODOS DE RECOPILACIÓN DE DATOS.

Será realizado mediante un Análisis Documental.

3.11 TÉCNICAS DE ANÁLISIS DE DATOS.

El método se realizara con una Estadística Descriptiva.

Lo primero es brindarle una charla de aproximadamente 20 minutos sobre la forma como se usa el modelo y su objetivo, después del cual se dejó un espacio de 20 minutos adicionales, para que la persona termine de familiarizarse con el modelo. Posteriormente se le aplicó la encuesta correspondiente.

CAPÍTULO IV
RESULTADOS DEL ESTUDIO

4 RESULTADOS Y DISCUSIÓN.

4.1 LENGUAJE DE PROGRAMACIÓN.

Es probablemente imposible ofrecer una definición completa y carente de fallos de lo que es realmente un lenguaje orientado a objetos. Sin embargo sabemos que la orientación a objetos implica, como mínimo estar basado en clases, mas herencia y referencia a sí mismo. La tabla 4.3 describe las características de algunos de los lenguajes que satisfacen estos requisitos, y que la mayoría estaría de acuerdo en denominar orientados a objetos.

L. O.O / Características OO	Visual Basic	C++	Delphi
Comprobación de tipos	Tardia	Tardia	Tardia
Polimorfismo	Si	Si	Si
Ocultamiento de Información	Si	Si	Si
Concurrencia	No	Escasa	No
Herencia	No	Si	Basada en clases
Herencia múltiple	No	Si	Si
Persistencia	No	No	No
Genericidad	No	Si	Si
Bibliotecas de objetos	Unas pocas	Pocas	Pocas

Tabla 4.3: Principales Características de los Lenguajes Orientados a Objetos

Tomando en cuenta las características de cada uno de los lenguajes de programación así como la experiencia del desarrollador del modelo, se escogerá como lenguaje de programación el MS Visual Basic.

4.2 MODELAMIENTO.

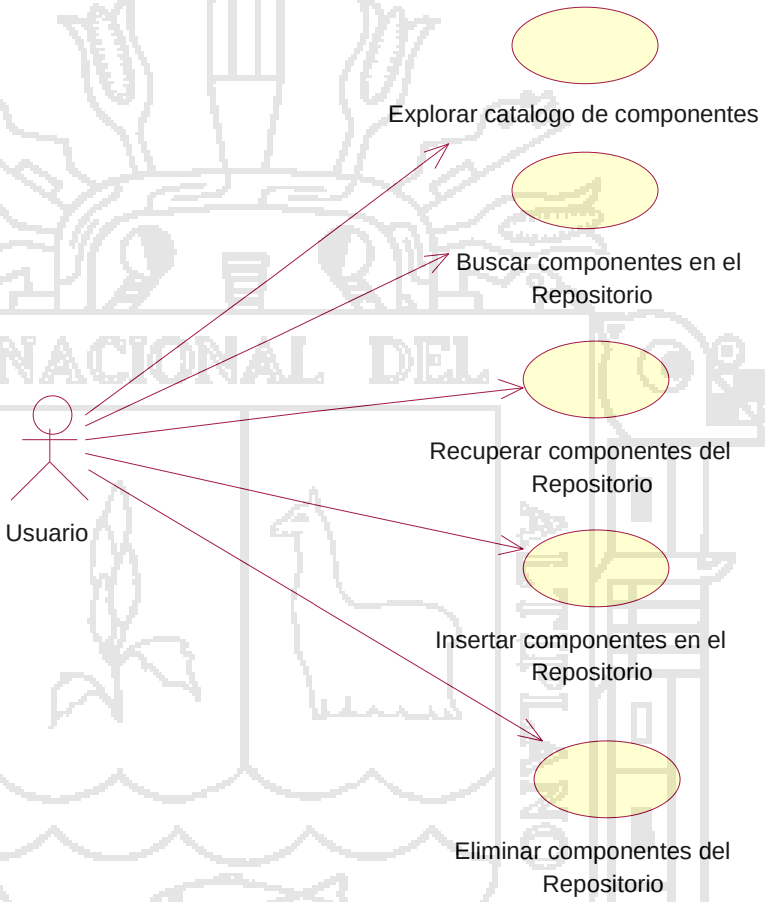
4.2.1 REQUERIMIENTOS FUNCIONALES.

- Explorar componentes en el repositorio.
- Buscar componentes en el repositorio.
- Recuperar componentes del repositorio.

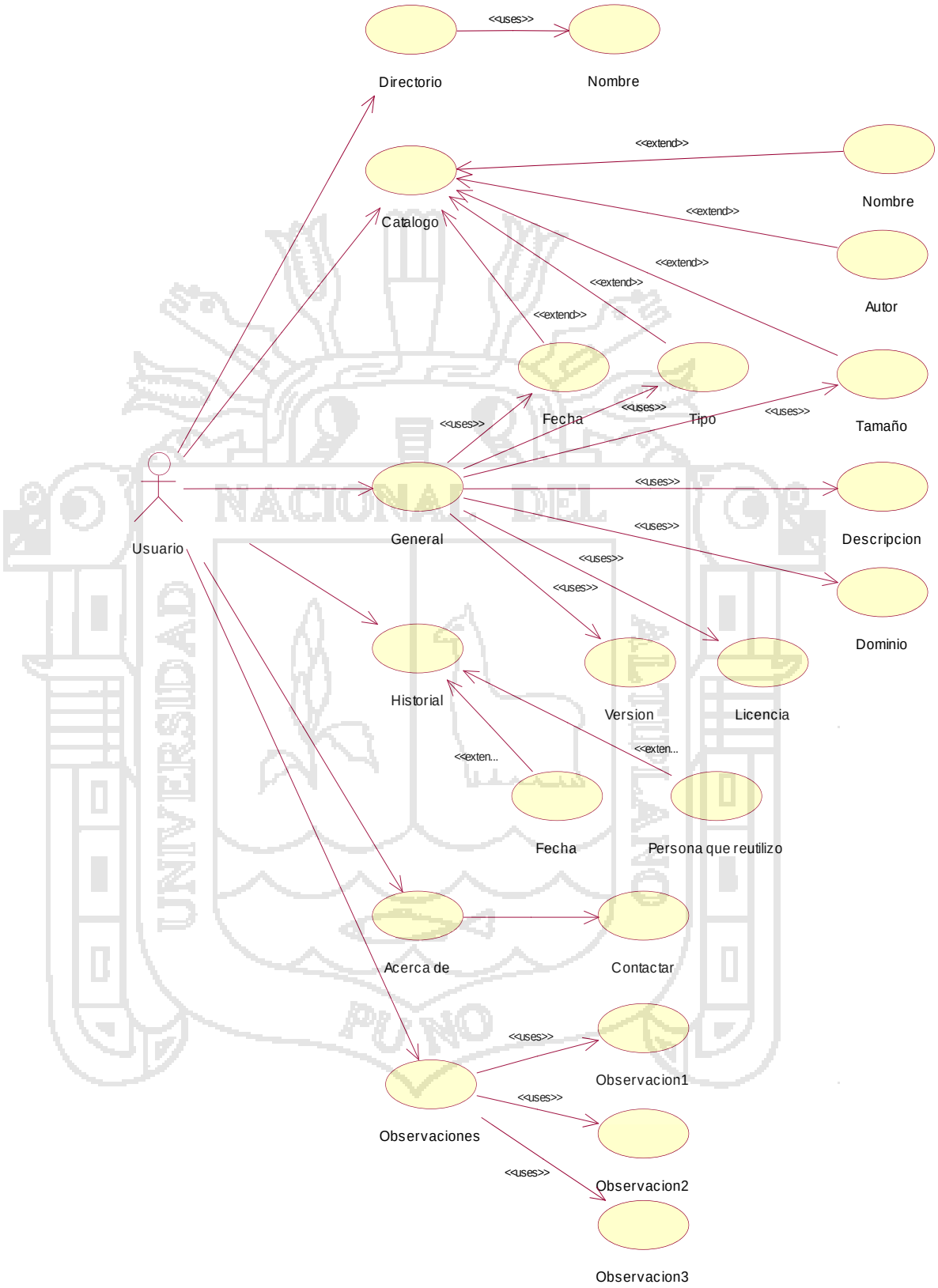
- Insertar componentes en el repositorio.
- Eliminar componentes del repositorio.

4.2.2 DIAGRAMAS DE CASOS DE USO.

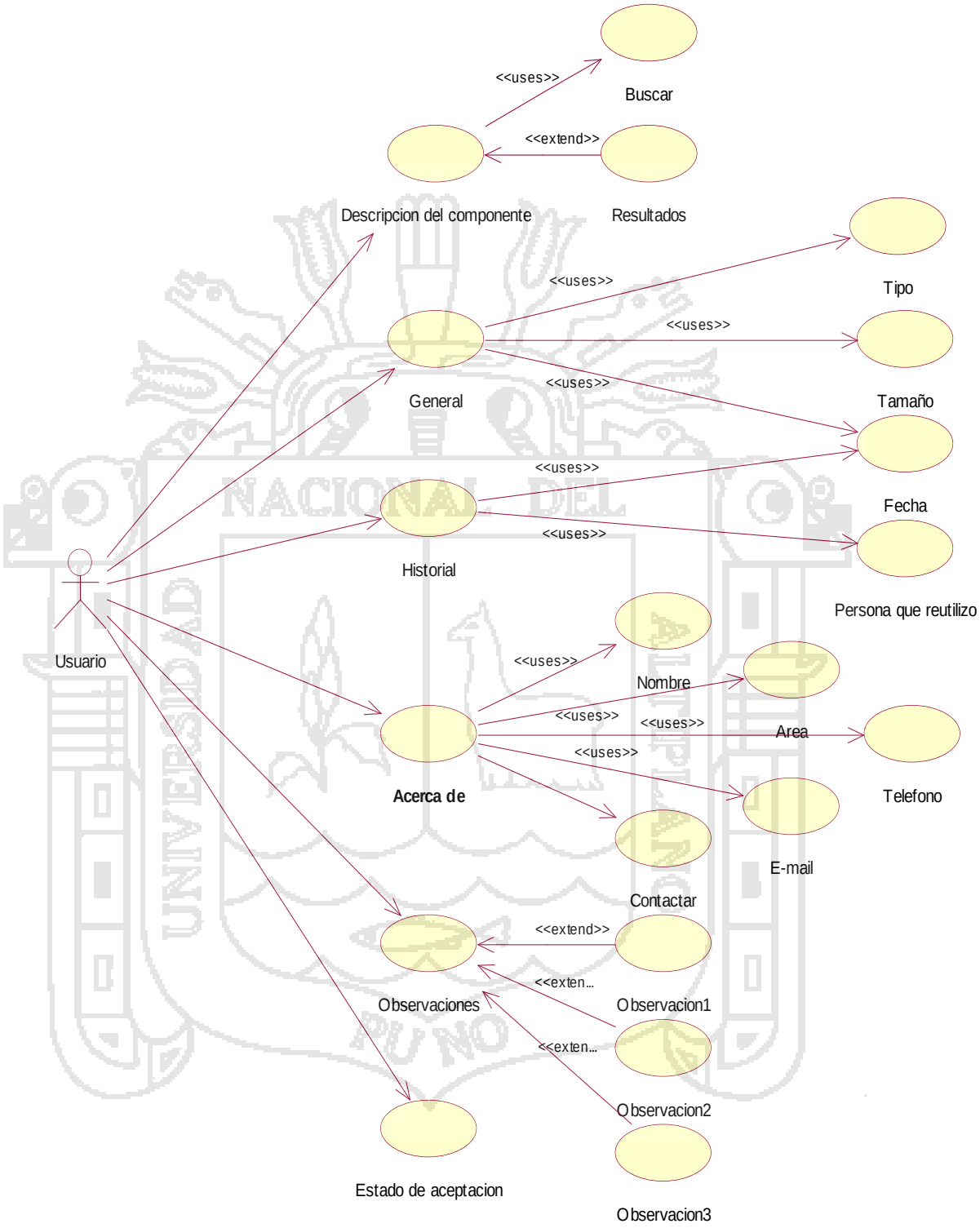
4.2.2.1 CASO DE USO: PROCESO GENERAL.



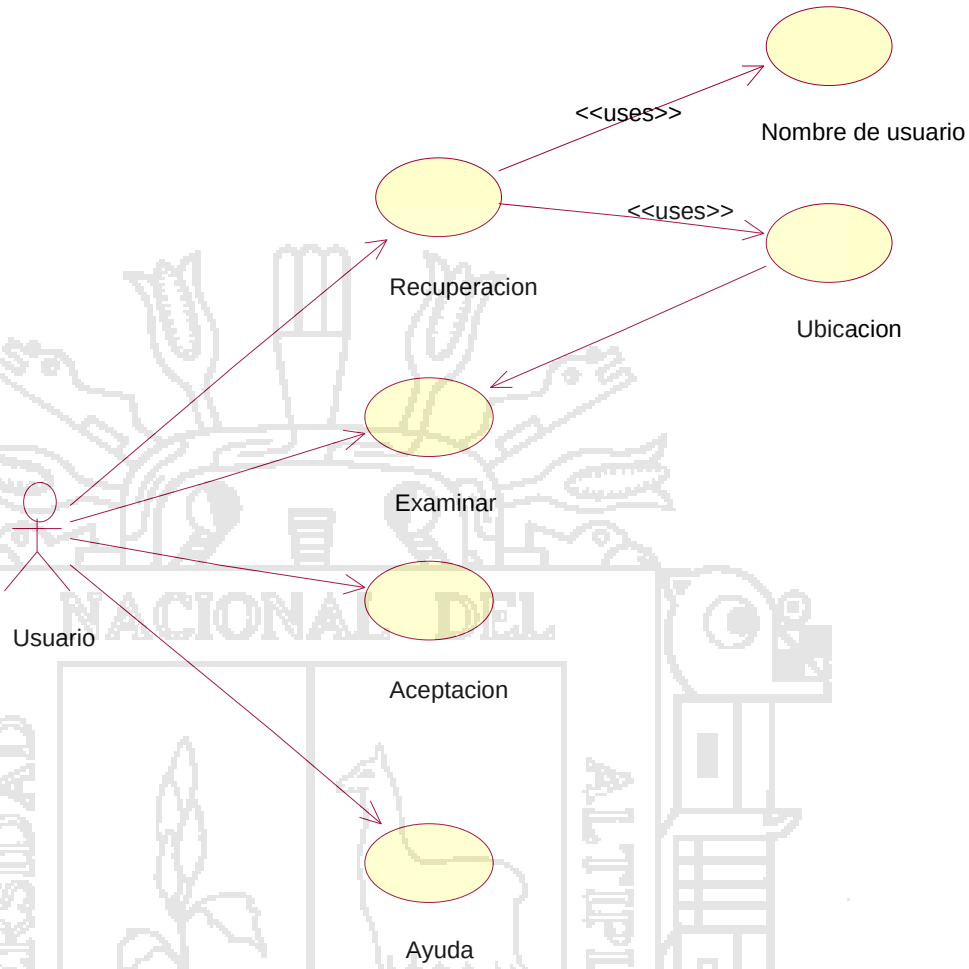
4.2.2.2 CASO DE USO: EXPLORAR COMPONENTES EN EL REPOSITORIO.



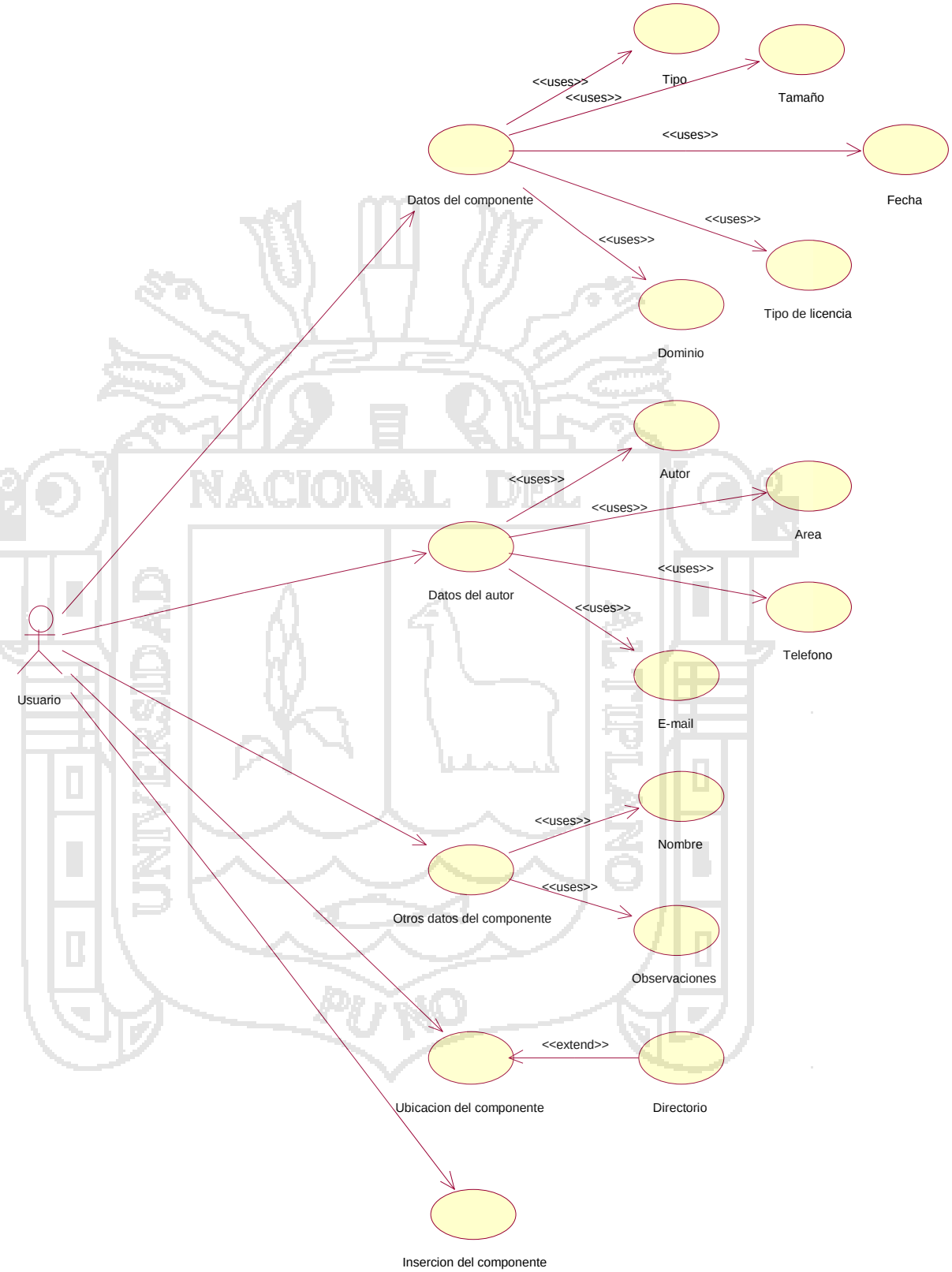
4.2.2.3 CASO DE USO: BUSCAR COMPONENTES EN EL REPOSITORIO.



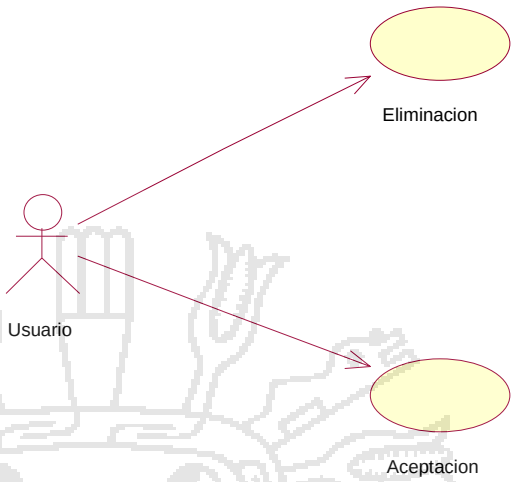
4.2.2.4 CASO DE USO: RECUPERAR COMPONENTES DEL REPOSITORIO.



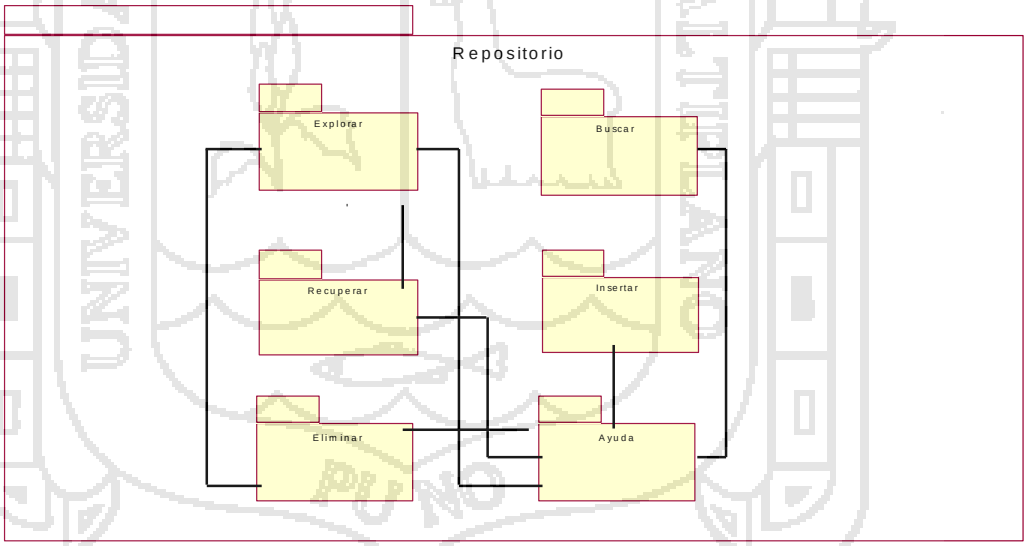
4.2.2.5 CASO DE USO: INSERTAR COMPONENTES EN EL REPOSITORIO.



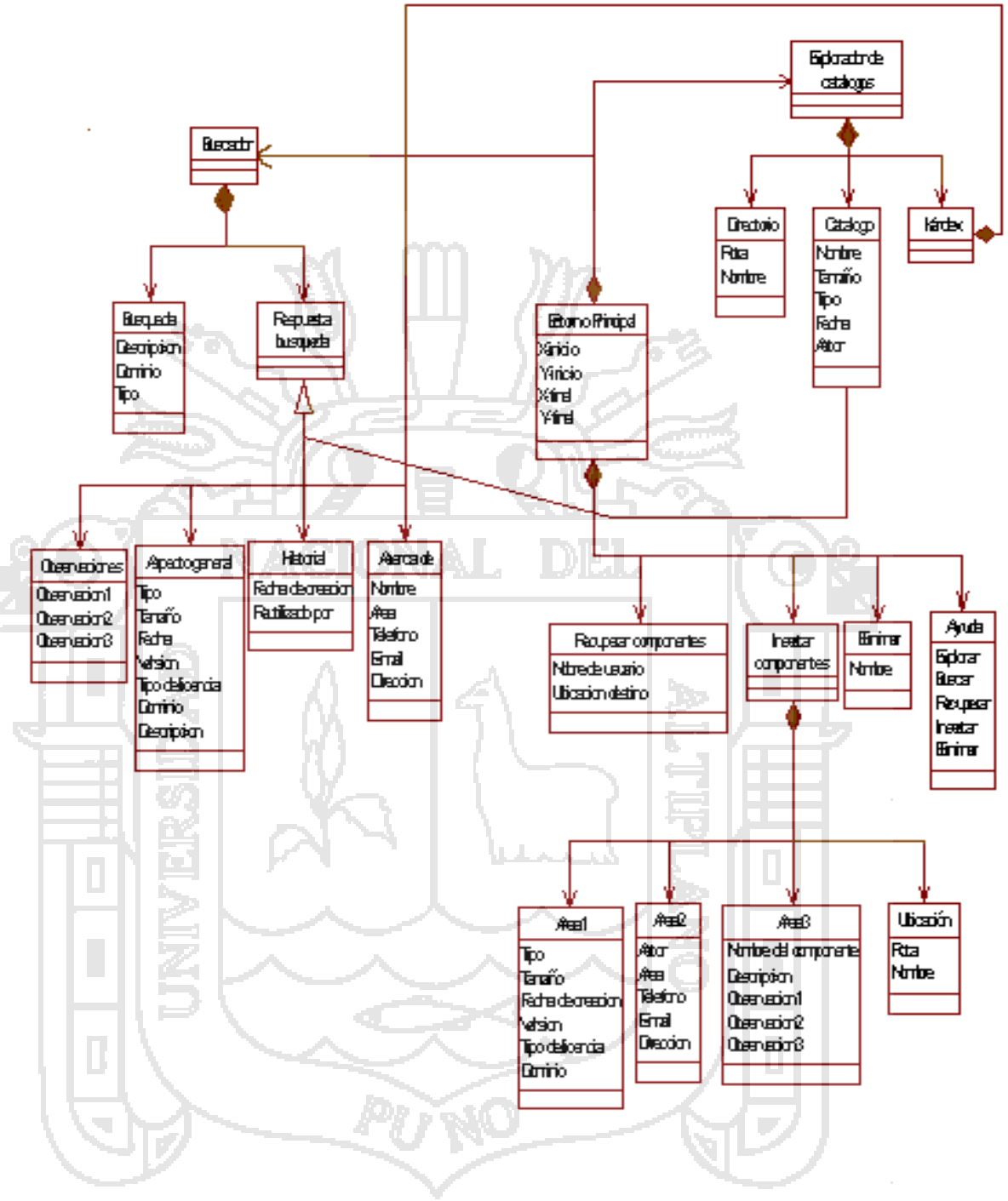
4.2.2.6 CASO DE USO: ELIMINAR COMPONENTES DEL REPOSITORIO.



4.2.3 DIAGRAMA DE PAQUETES.

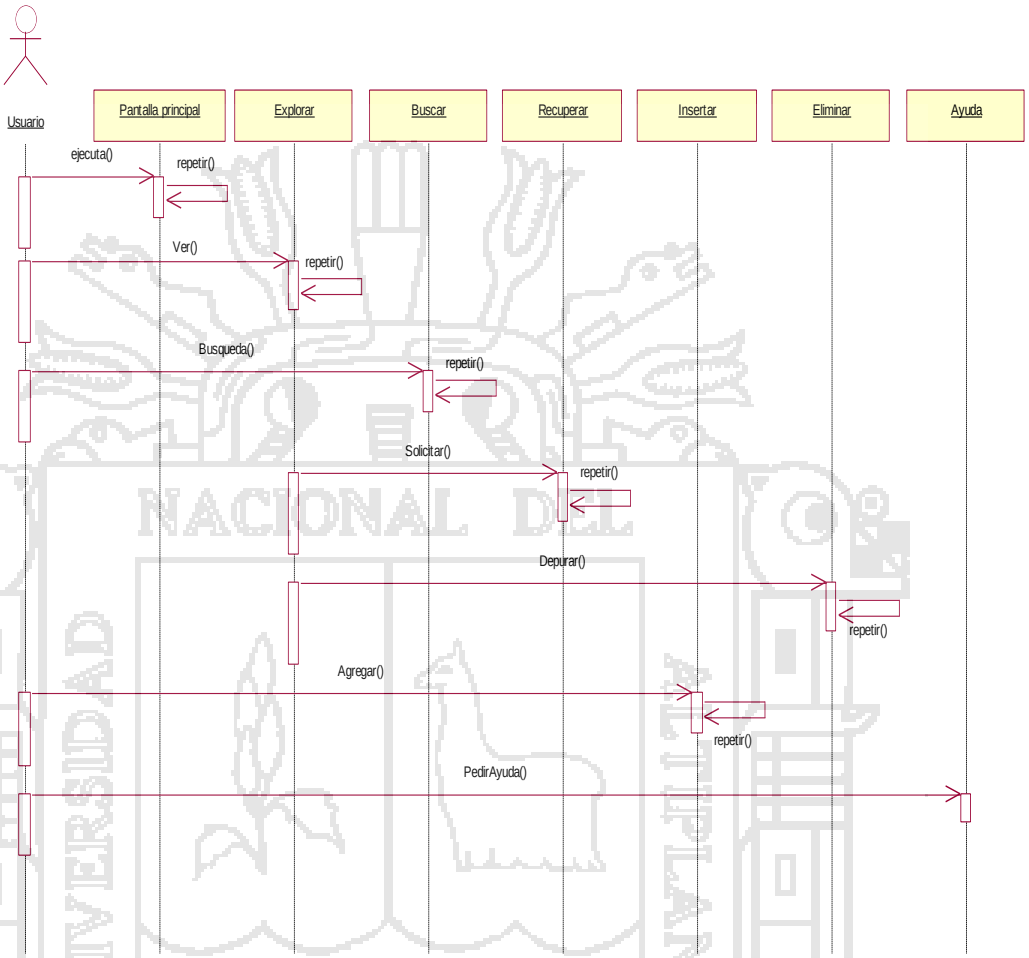


4.2.4 DIAGRAMA DE CLASES.

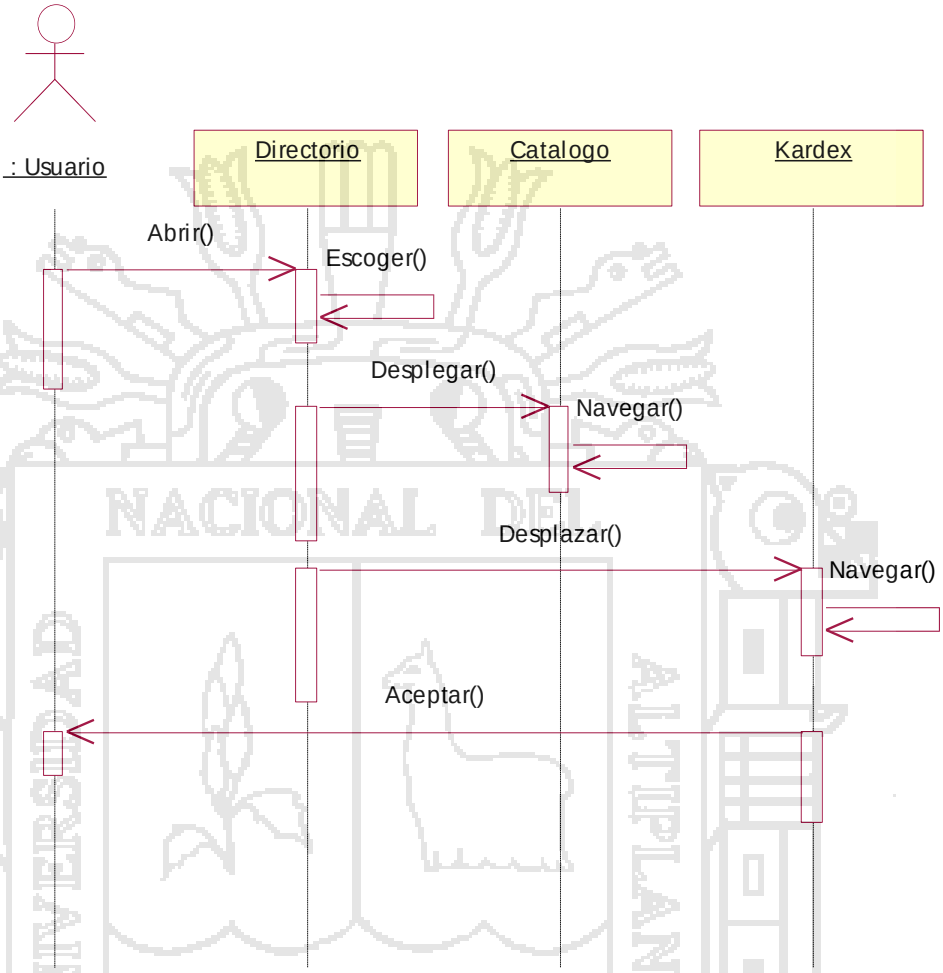


4.2.5 DIAGRAMAS DE SECUENCIA.

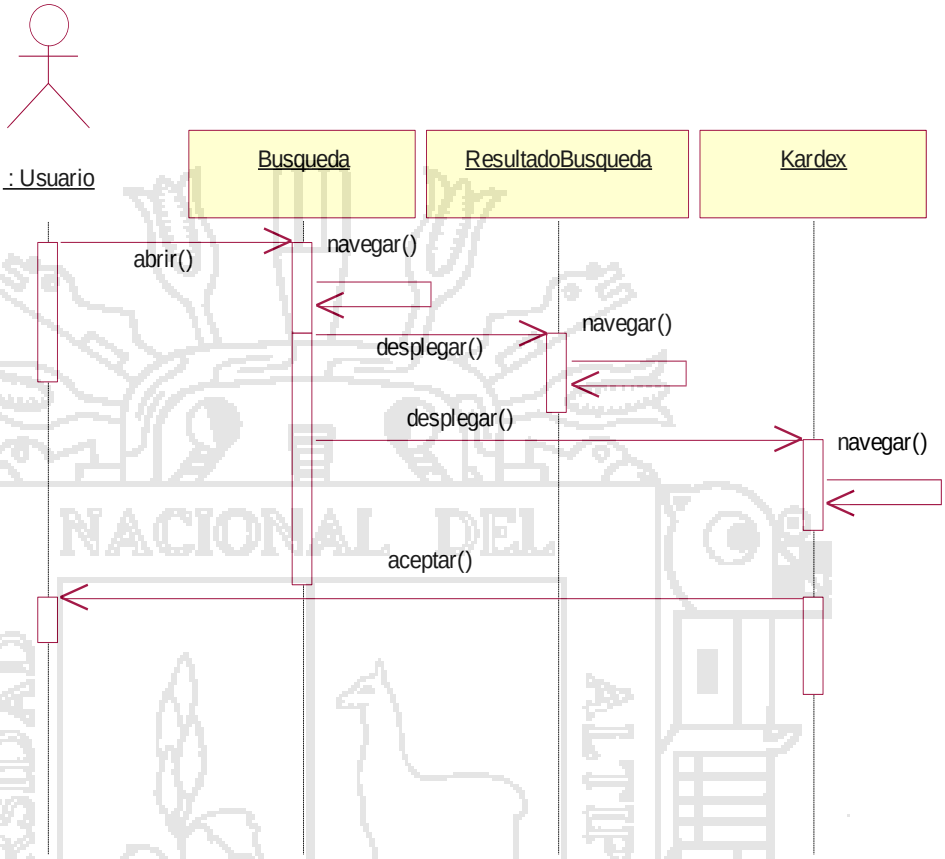
4.2.5.1 DIAGRAMA DE SECUENCIA GENERAL.



4.2.5.2 DIAGRAMA DE SECUENCIA PARA EL COMPONENTE "EXPLORAR".



4.2.5.3 DIAGRAMA DE SECUENCIA PARA EL COMPONENTE "BUSCAR".



4.3 DESCRIPCIÓN DEL MODELO.

Después de haber realizado un análisis de los conceptos más importantes para el proceso de reutilización de componentes software, se llegó a la conclusión de que se debía diseñar un instrumento que apoyase de manera específica a este proceso, dicha herramienta, tendrá la tarea de controlar y gestionar los componentes software en un repositorio, vale decir, se ocupará de asistir al proceso de reutilización de componentes software de manera sencilla, fácil de usar y con una interfaz bastante amigable y didáctica. En este capítulo se dan a conocer los módulos que contienen el sistema y la explicación individual de cada una de las opciones que presenta esta herramienta.

4.3.1 LA INTERFAZ DEL SISTEMA.

La interfaz de la herramienta que proponemos es una ventana donde se encuentran las principales tareas de soporte al enfoque de reutilización de software como son: Clasificar, Examinar, Buscar, Recuperar, Insertar y Eliminar componentes. La interfaz principal que ve el usuario se muestra en la figura siguiente:

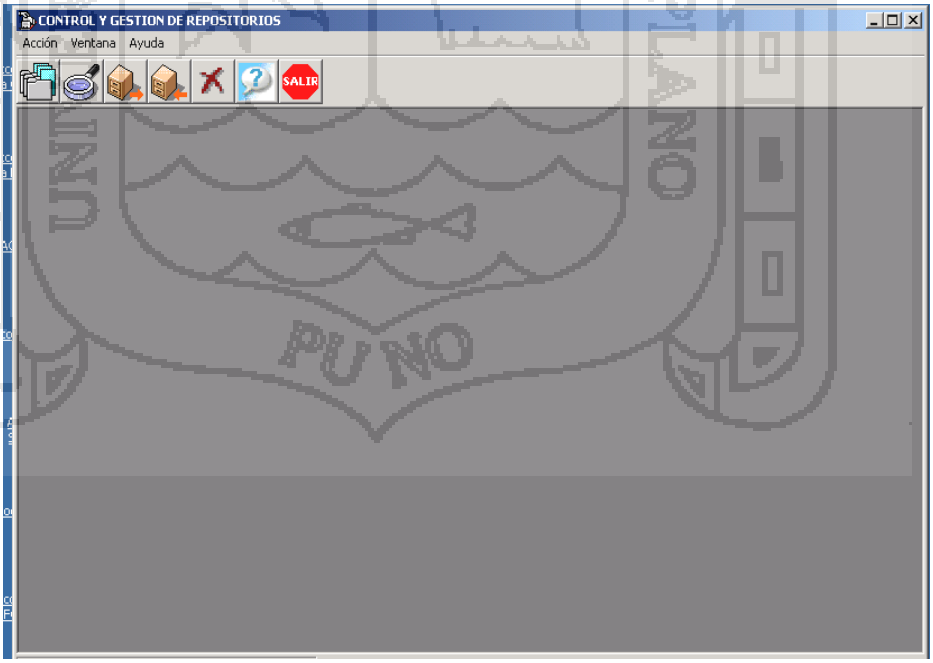


FIGURA 4.1: INTERFAZ DEL MODELO.

En la barra de menú de la interfaz se visualizan tres opciones. Acción, Ventana y Ayuda. La primera opción del menú, -Acción-, es el que contiene básicamente todas las tareas que se dan en un proceso de reutilización, las mismas que por ser las principales detallaremos una por una en las siguientes secciones. La segunda opción del menú, -Ventana-, es pues simplemente una característica bastante conocida de los entornos windows, que proporciona al usuario la facilidad de personalizar las ventanas existentes en el sistema. Finalmente, la tercera opción del menú, –Ayuda–, proporciona al usuario un tutorial de manejo y consulta de las opciones que presenta la herramienta, así como la respectiva documentación anexa al sistema.

4.3.2 ACCIÓN-EXPLORAR.

Esta acción tiene por objetivo la revisión detallada por parte del usuario de los componentes software que se encuentren en el repositorio, estos componentes están clasificados en catálogos y subcatálogos mostrados en un árbol de directorio de manera tal, que el usuario pueda acceder a los componentes de una forma ya conocida y establecida, pues es de esta manera que se logra que el usuario no pueda perderse jamás en su exploración y que la interfaz del repositorio no pierda su estructura y se vea fácil de usar.

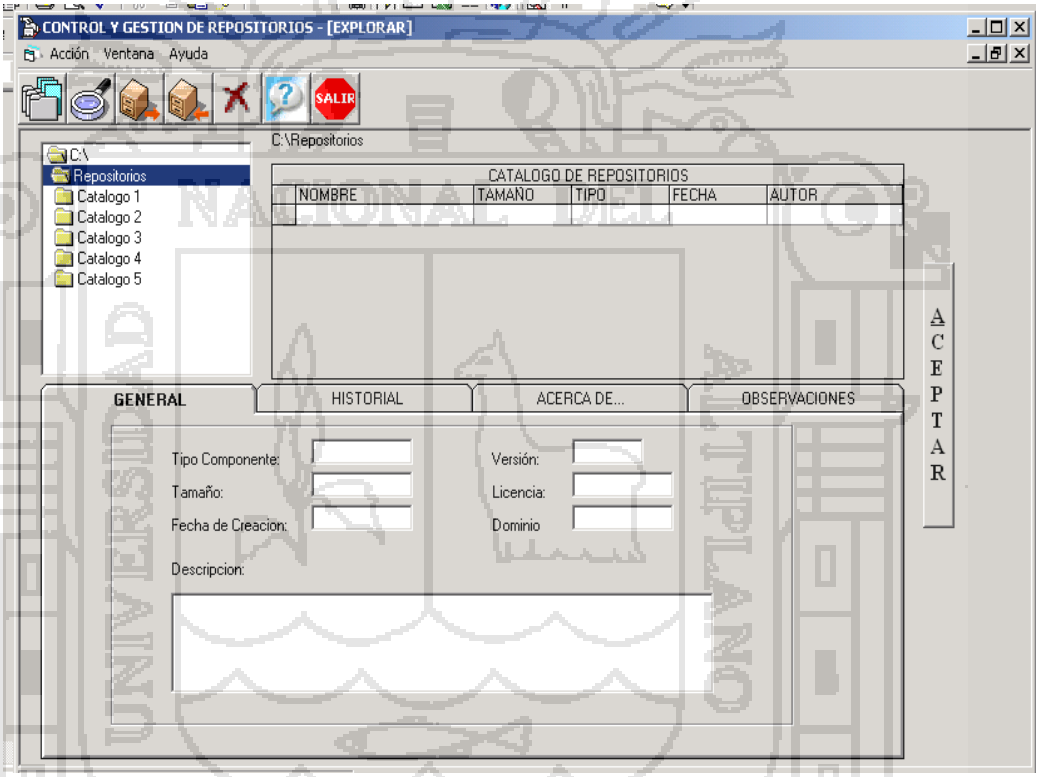


FIGURA 4.2: ACCIÓN-EXPLORAR.

4.3.3 ACCIÓN-BUSCAR.

Esta acción, incluida en la herramienta, es considerada la más importante ya que, permite al usuario realizar la búsqueda de componentes candidato de dos maneras, vale decir el usuario tendrá la posibilidad de realizar búsquedas tanto simples como avanzadas. Para el primer caso el usuario simplemente tendrá que ingresar una

descripción tentativa del componente a buscar y como resultado obtendrá los posibles componentes candidatos, los mismos que podrán ser revisados inmediatamente y en la misma ventana con un simple desplazamiento del cursor para poder ver la ficha múltiple asociada a cada uno de los candidatos. Para el segundo caso (búsqueda avanzada) el usuario debe escoger el tópico(s) y/o la descripción tentativa del componente que desea encontrar y/o consultar, después de haberlos escogido hay que presionar el botón de "Buscar". El resultado de esta búsqueda, al igual que en el caso anterior generará una lista de componentes candidatos listos para ser consultados y/o examinados mediante sus fichas múltiples asociadas a cada uno de los componentes resultado. La ventaja de este tipo de búsqueda avanzada radica en que el usuario obtendrá como resultado una lista de componentes más específica o con mayor éxito de aproximación. En la figura 4.3 se muestran ambos tipos de búsqueda y su acción asociada.

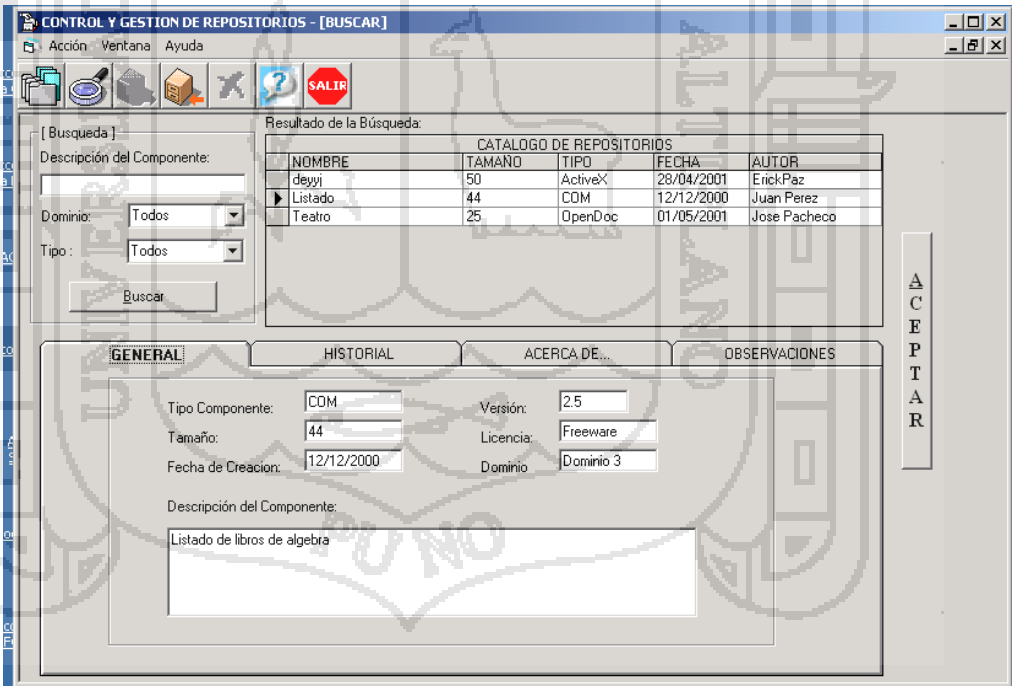


FIGURA 4 1: ACCIÓN-BUSCAR.

4.3.4 ACCIÓN-INSERTAR.

El objetivo de esta acción es la de introducir al repositorio nuevos componentes, para lo cual el usuario -que en este caso tendrá que ser el autor de dicho componente- ingresará toda la información asociada al componente como son: tipo, tamaño, versión, tipo de licencia que otorga, así como el dominio del componente y la descripción del mismo, además de su información personal y las observaciones que pudiera advertir. Todos estos datos son de carácter obligatorio, con excepción de las observaciones que son opcionales. Una vez terminado el ingreso, el usuario simplemente presionará el botón insertar, con lo cual el componente introducido pasará a formar parte del repositorio de componentes software de la organización, listo para ser consultado y/o examinado por otros usuarios.

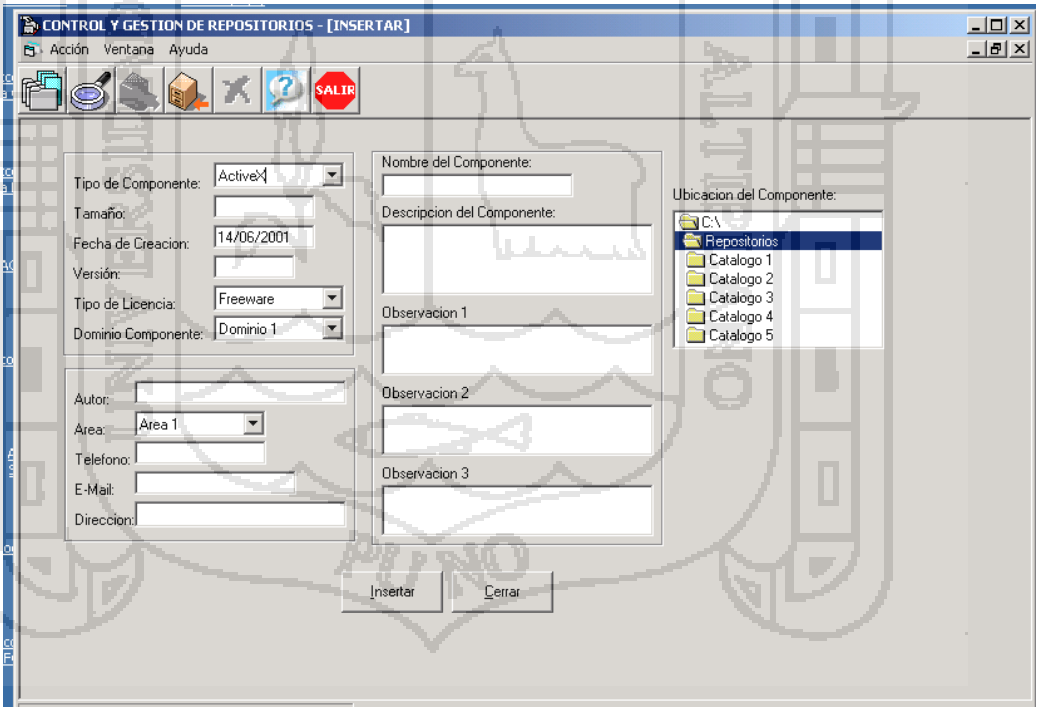


FIGURA 4.4: ACCIÓN-INSERTAR.

4.3.5 ACCIÓN-RECUPERAR.

Esta acción se activa sólo cuando hay un componente que este siendo señalado por el cursor, vale decir que el usuario tendrá que escoger el componente más adecuado de la lista de componentes candidato. Una vez realizado esto podrá acceder a él (recuperarlo) presionando el botón Recuperar en la barra de herramientas o en todo caso desde el menú Acción y señalar la opción Recuperar. Como resultado de dicha acción aparecerá una ventana pequeña sobrepuesta en la que el usuario -que en este caso será el reutilizador del componente- tendrá que realizar dos ingresos obligatorios como son, su nombre y la ubicación donde será copiado el componente para los fines que él crea conveniente.

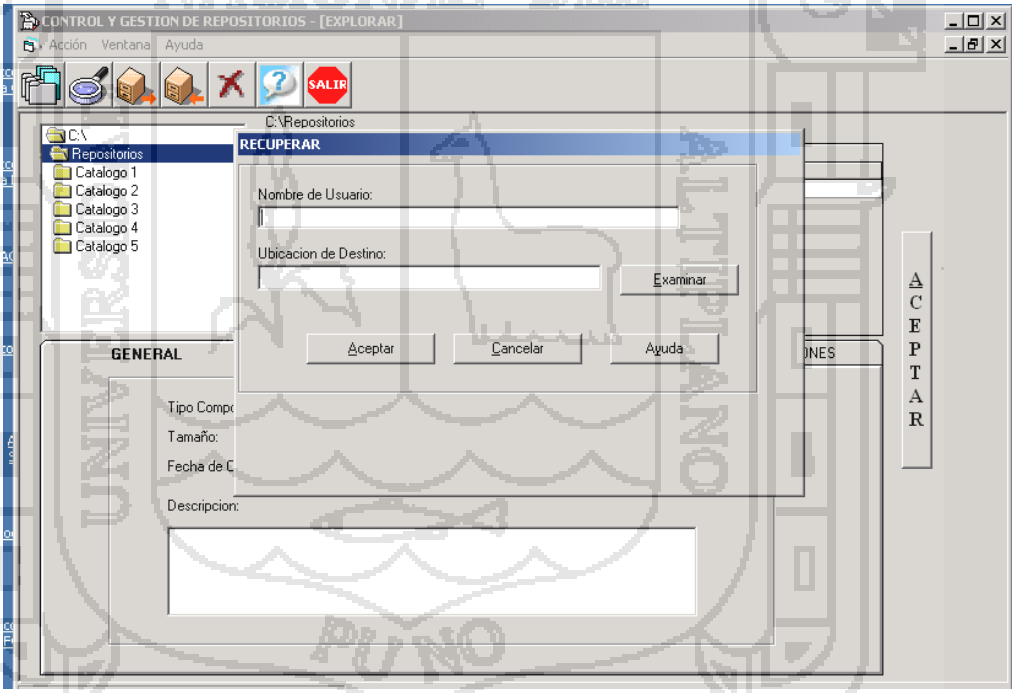


FIGURA 4.5: ACCIÓN-RECUPERAR.

4.3.6 ACCIÓN-ELIMINAR.

Al igual que en el caso anterior esta acción se activa sólo cuando hay un componente seleccionado y, como es obvio pensar, el usuario -que en este caso tendrá que ser un usuario autorizado- tendrá la posibilidad de eliminar físicamente dicho componente del repositorio. En respuesta, la herramienta pedirá únicamente la confirmación de esta acción, por lo mismo el usuario deberá estar seguro de lo que está haciendo pues no se ha considerado ningún procedimiento de recuperación de componentes eliminados.

En este capítulo el autor de la presente tesis da a conocer de manera descriptiva y gráfica la interfaz del instrumento denominado Control y Gestión de Repositorio de Componentes Software, el mismo que a juzgar por los elementos que presenta y con las limitaciones y alcances que se han presentado, ha sido construida a manera de prueba o ensayo que, en el campo de la Ingeniería de Sistemas es conocido como modelo, sin que ello signifique restarle validez a dicho instrumento, ya que pensamos, servirá como base para la construcción de herramientas mucho más complejas y robustas encargadas de brindar soporte operativo al proceso de Reutilización de Software.

4.4 COMPROBACIÓN DE LA APLICACIÓN.

En este capítulo validaremos nuestro modelo con respecto a dos puntos principales y de suma importancia como son: (1). La interfaz del modelo y (2) La funcionalidad del modelo; ambos parámetros serán sometidos al Modelo de McCall y a la Encuesta como instrumentos de validación.

4.4.1 VALIDACIÓN DE LA APLICACIÓN MEDIANTE EL MODELO DE McCALL.

4.4.1.1 FACTORES QUE DETERMINAN LA CALIDAD DE LA APLICACIÓN.

- Factores directos como por ejemplo la medición de errores, etc.
- Factores indirectos como por ejemplo la facilidad de empleo del modelo o el mantenimiento del mismo.

McCall propuso una clasificación de los factores que afectan a la calidad del software. Los factores de la calidad del software se centran en los aspectos importantes de un producto de software, características operacionales, capacidad de soportar cambios y su adaptabilidad a nuevos entornos. McCall proporciona las siguientes descripciones:

- Corrección. Grado en que un programa satisface sus especificaciones y consigue los objetivos de la misión encomendada por el cliente.
- Fiabilidad. Grado en que un programa lleve a cabo sus funciones y esperar las respuestas correctas.
- Eficiencia. Cantidad de recursos empleados.
- Integridad. Grado en que puede controlarse el acceso al software o a los datos por personal no autorizado.
- Facilidad de uso. Esfuerzo requerido para aprender.

- Facilidad de mantenimiento. Esfuerzo requerido para localizar y arreglar un error.
- Flexibilidad. Esfuerzo requerido para modificar un programa operativo.
- Facilidad de prueba. Esfuerzo requerido para probar un programa de forma que asegure que realiza su función.
- Portabilidad. Esfuerzo requerido para transferir el programa desde un hardware y/o un entorno de sistemas de software a otro.
- Reusabilidad. Grado en que un programa se puede reusar en otras aplicaciones.
- Facilidad de interoperación. Esfuerzo requerido para acoplar un sistema a otro.

Existe dificultad en desarrollar medidas directas para comprobar la calidad de un software. Para resolver esto, se define un conjunto de métricas usadas para desarrollar expresiones de cada uno de los factores. La escala de McCall emplea una escala de 0 (bajo) a 10 (alto). Las métricas a emplearse son las siguientes:

- Facilidad de auditoría para comprobar los estándares.
- Exactitud en los cálculos y el control.
- Completitud en la implementación de las funciones requeridas.
- Concisión en las líneas de código.
- Consistencia en la documentación del proyecto de software.
- Estandarización de datos a lo largo del programa.

- Tolerancia de errores en el programa.
- Eficiencia en la ejecución.
- Facilidad de expansión del diseño.
- Generalidad en la aplicación del programa.
- Independencia de hardware.
- Instrumentación en el funcionamiento e identificación de los propios errores del programa.
- Modularidad que implica la independencia funcional de los componentes del programa.
- Facilidad de operación del programa.
- Seguridad en la protección del programa y los datos.
- Auto-documentación.
- Simplicidad en el entendimiento del programa.
- Independencia del software con respecto al sistema operativo.
- Facilidad de traza para seguir la pista de la representación del diseño o de los componentes reales del programa hacia atrás; hacia los requerimientos.
- Formación en la aplicación para usuarios nuevos.

Los resultados de los factores y métricas para medir la calidad del modelo se dan en la tabla 4.4.

Factores de Calidad / Métricas de CS	Corrección	Fiabilidad	Eficacia	Integridad	Fac. Mant.	Flexibilidad	Fac. Prueba	Portabilidad	Reusabilidad	Interoperabil.	Fac. de Uso	Totales
Facilidad de Auditoría				3			4					7
Exactitud		10										10
Completitud	5											5
Consición		4				3	2					9
Consistencia	2	3			2	2						9
Estandarización										8		8
Tolerancia de Errores		10										10
Eficiencia en Ejecución			9									9
Facilidad de Expansión						8						8
Generalidad						2		2	2	2		8
Independencia del Hw								3	2			5
Instrumentación				5	2		3					10
Modularidad		1			1	1	1	1	1	1		7
Facilidad de Operación			4								4	8
Seguridad				6								6
Auto-documentación					2	1	2	1	1			7
Simplicidad		3			3	2	2					10
Independencia del Sw								2	2			4
Facilidad de Traza	8											8
Formación											8	8

Tabla 6.1. Factores y Métricas para medir la calidad del modelo

4.4.2 APLICACIÓN DE LA METODOLOGÍA PARA EL MODELO.

Al concluir la fase de análisis se realizaron las siguientes interrogantes para sustentar la correcta especificación del problema:

Es completo, consistente y exacto el análisis del dominio del problema?	Si
Es completa la partición del problema?	Si
Están definidas adecuadamente las interfaces internas y externas?	Si
Se pueden seguir todos los requerimientos a nivel del sistema?	Si

Para la fase de diseño se emplearon las siguientes interrogantes:

Se ha conseguido una modularidad efectiva?	85%
Depende de algunos factores la arquitectura del programa	Windows
Se han definido las interfaces para los módulos y elementos externos?	Si
Realizan los algoritmos las funciones deseadas?	Si
Son los algoritmos lógicamente correctos?	Si
Es consistente la interfaz con el diseño procedural?	Si
Es razonable la complejidad lógica?	Si
Han sido tratados los errores?	90%
Es sensible el nivel de detalle del diseño para el lenguaje de implementación	Si
Se han empleado características dependientes del sistema operativo?	Si

Para la fase de codificación se plantearon las siguientes preguntas para probar que se cumplieron los objetivos:

Se ha traducido adecuadamente el diseño al código?	Si
Se ha hecho uso adecuado de las convenciones del lenguaje?	Si
Hay comentarios incorrectos o ambiguos?	No
Son apropiadas las declaraciones de tipos y datos?	90%

4.5 RESULTADOS DE LA ENCUESTA.

4.5.1 CON RESPECTO A LA INTERFAZ DEL MODELO.

4.5.1.1 ¿La interfaz del usuario, te pareció agradable?

Condición	#	%
Si	35	83.33
No	7	16.67
Total	42	100

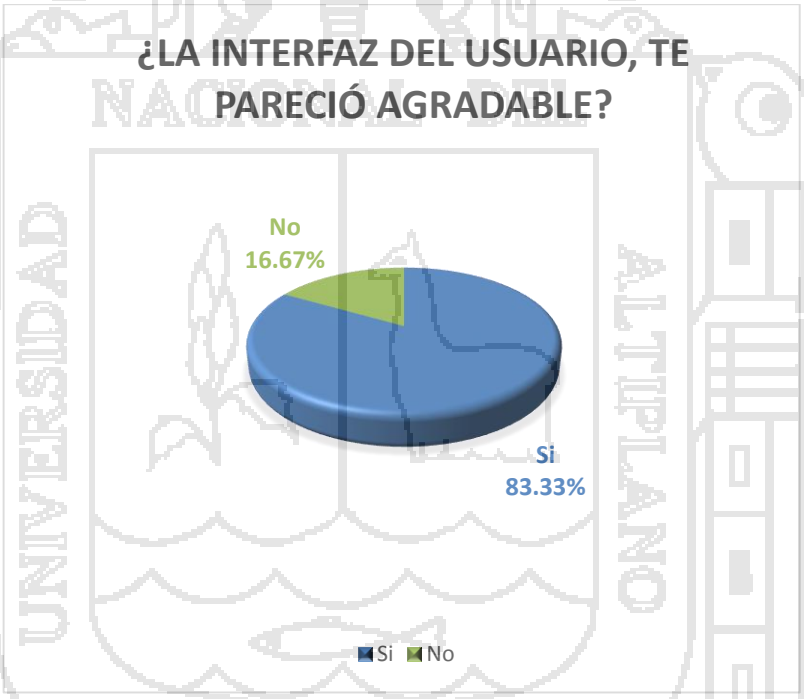


FIGURA 4.6: ACERCA DE LA INTERFAZ DEL USUARIO

El 83.33% de los encuestados consideran que la interfaz de usuario evaluada les pareció agradable, en tanto que el restante 16.67% considera que no les es agradable, debido principalmente a la poca flexibilidad de las ventanas.

4.5.1.2 ¿La distribución del menú, es comprensible?

Condición	#	%
Si	42	100
No	0	0
Total	42	100

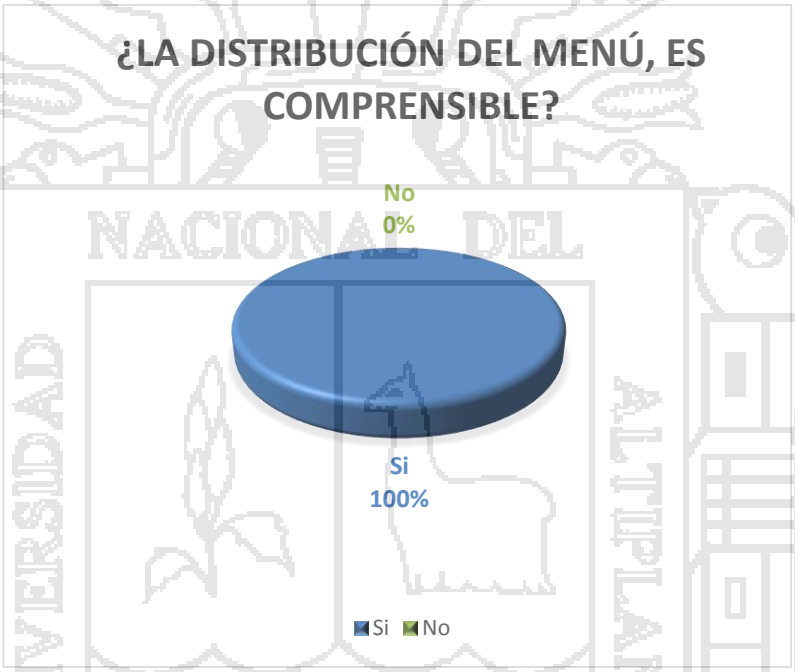


FIGURA 4.7: ACERCA DE LA DISTRIBUCIÓN DEL MENÚ

El 100% de los entrevistados, vale decir la totalidad de ellos respondió que la distribución del menú es totalmente comprensible, vale decir que ninguno de los entrevistados tuvo problemas de comprensión respecto a la distribución de opciones que ofrece el menú del modelo en cuestión.

4.5.1.3 ¿Los iconos, son claros en su semántica?

Condición	#	%
Si	42	100
No	0	0
Total	42	100

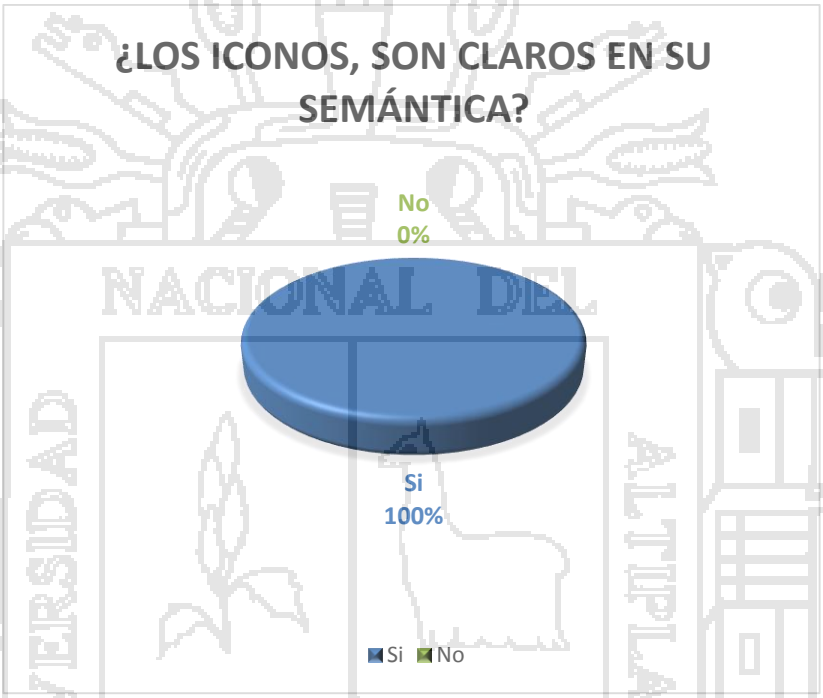


FIGURA 4.8: ACERCA DE LA SEMÁNTICA DE LOS ICONOS

La totalidad de los entrevistados concluyó que la semántica de los iconos era bastante clara.

4.5.1.4 ¿El diseño de pantallas es apropiado?

Condición	#	%
Si	30	71.43
No	12	28.57
Total	42	100

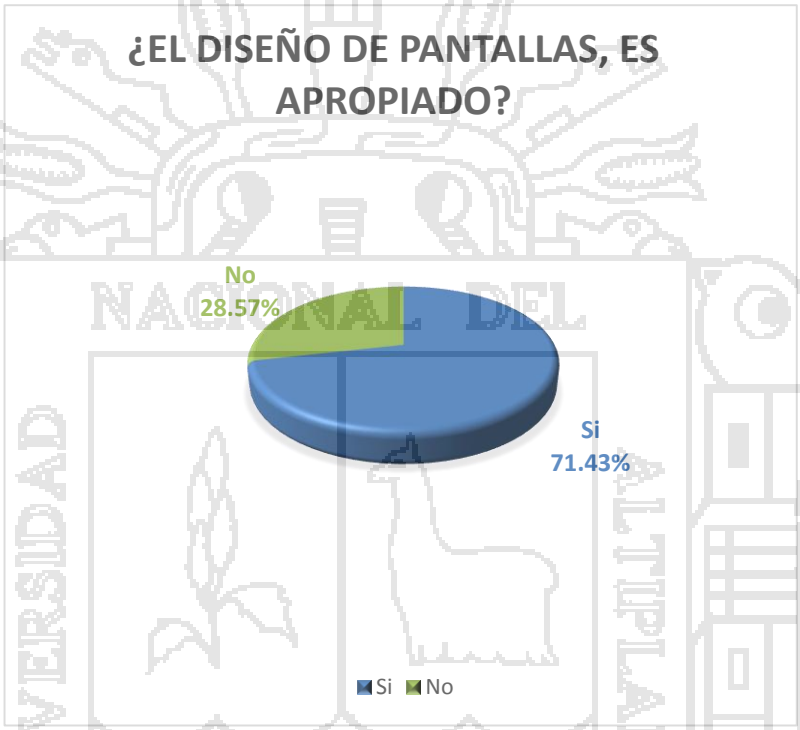


FIGURA 4.9: ACERCA DEL DISEÑO DE PANTALLAS

Un 71.43% de los encuestados coincidieron en afirmar que el diseño de pantallas era apropiado para este tipo de herramientas, mientras que un 28,57% de los encuestados mencionan que no es apropiado, indicando que los espacios no estaban muy bien distribuidos.

4.5.1.5 ¿La distribución de colores, es clara?

Condición	#	%
Si	42	100
No	0	0
Total	42	100

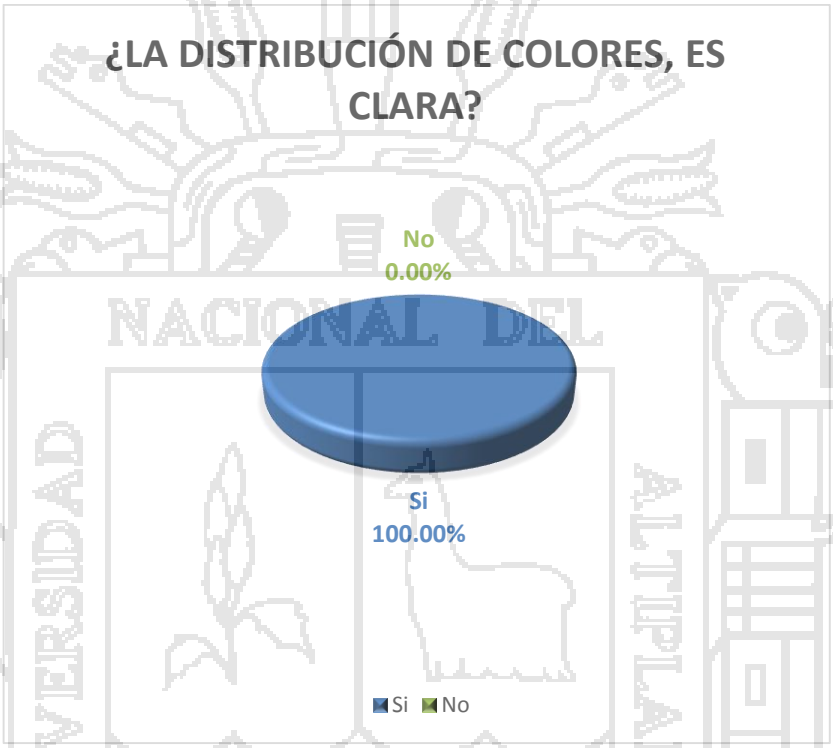


FIGURA 4.10: ACERCA DE LA DISTRIBUCIÓN DE COLORES

El 100% de los entrevistados respondieron que la distribución de colores si era clara, pues no se encontró ningún problema al respecto.

4.5.1.6 ¿Los mensajes son adecuados?

Condición	#	%
Si	30	71.43
No	12	28.57
Total	42	100

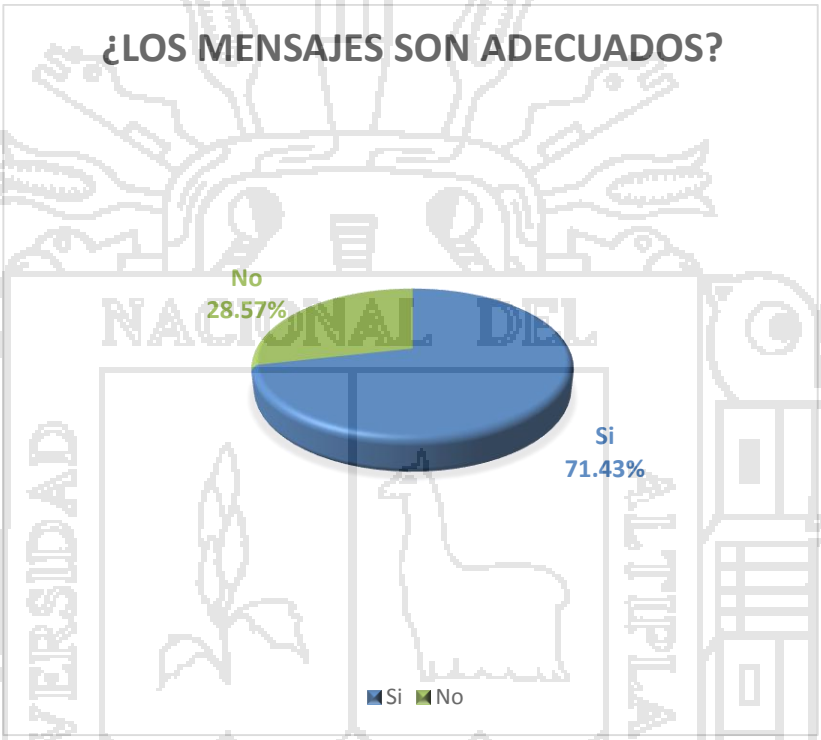


FIGURA 4.11: ACERCA DE LOS MENSAJES

Respecto a si los mensajes eran adecuados, un 71.43% de los entrevistados respondió que si eran adecuados, en tanto que un 28.57% de los encuestados respondió negativamente, ya que precisaron en su mayoría que eran insuficientes y en algunos casos poco descriptivos.

4.5.1.7 ¿La ayuda es adecuada?

Condición	#	%
Si	25	59.52
No	17	40.48
Total	42	100

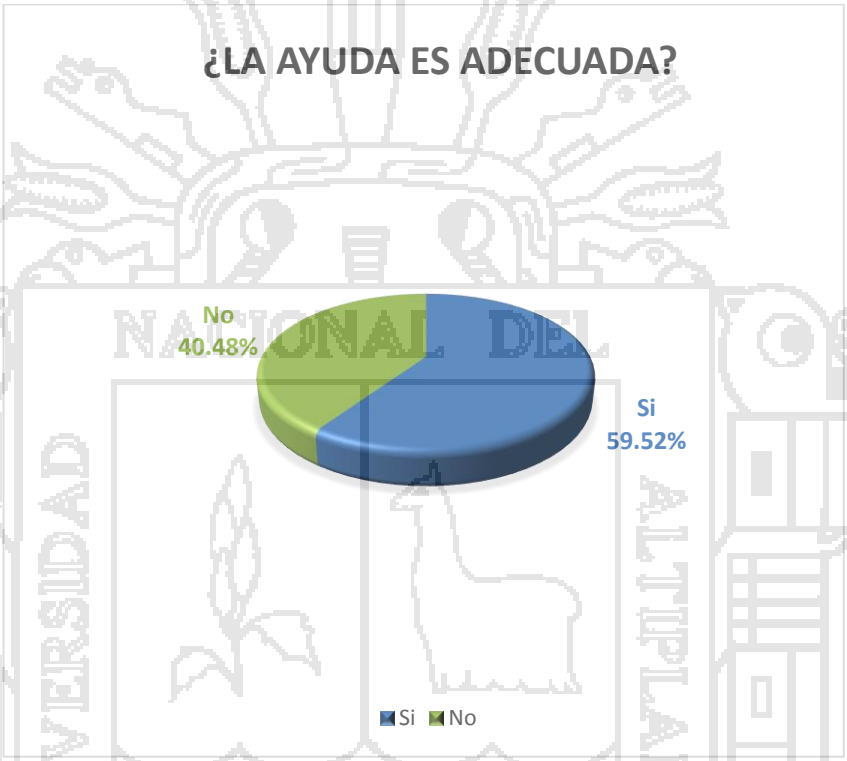


FIGURA 4.12: ACERCA DE LA AYUDA

Ante esta pregunta los encuestados respondieron en su mayoría (59.52%) afirmativamente, en tanto que un 40.48% de los encuestados respondieron negativamente precisando la mayoría de estos que la Ayuda era insuficiente y con ausencia de Ayuda interactiva.

4.5.2 CON RESPECTO A LA FUNCIONALIDAD DEL MODELO.

4.5.2.1 ¿Es comprensible la distribución del repositorio?

Condición	#	%
Si	38	90.48
No	4	9.52
Total	42	100

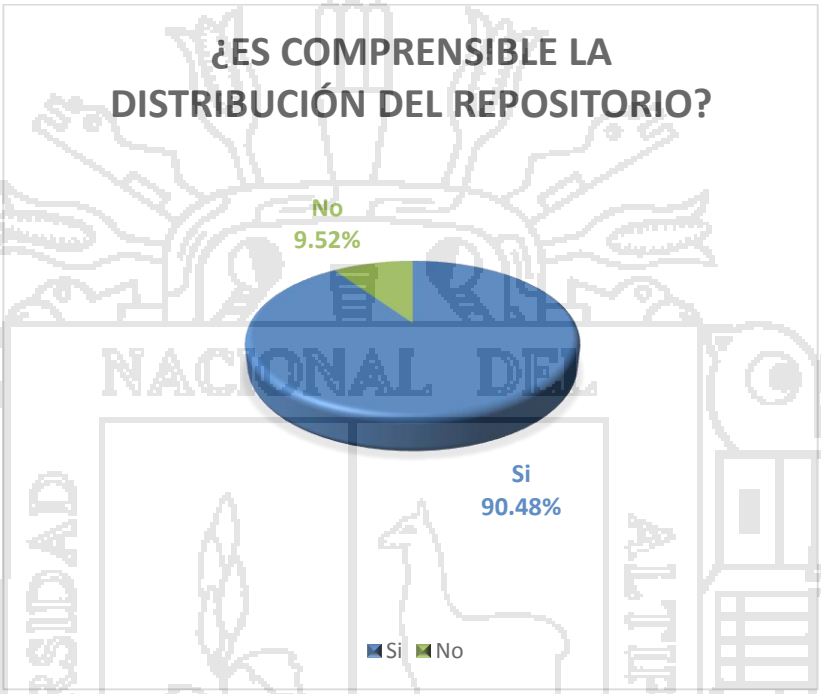


FIGURA 4.13: CON RESPECTO A LA DISTRIBUCIÓN DEL REPOSITORIO

Un 90.48% de los encuestados manifestaron que la distribución del repositorio era bastante comprensible frente a un 9.52% que tuvieron alguna dificultad para comprender la distribución del repositorio, debido principalmente a la nominación genérica que advirtieron.

4.5.2.2 ¿Las opciones del menú, realizan correctamente su función?

Condición	#	%
Si	34	80.95
No	8	19.05
Total	42	100

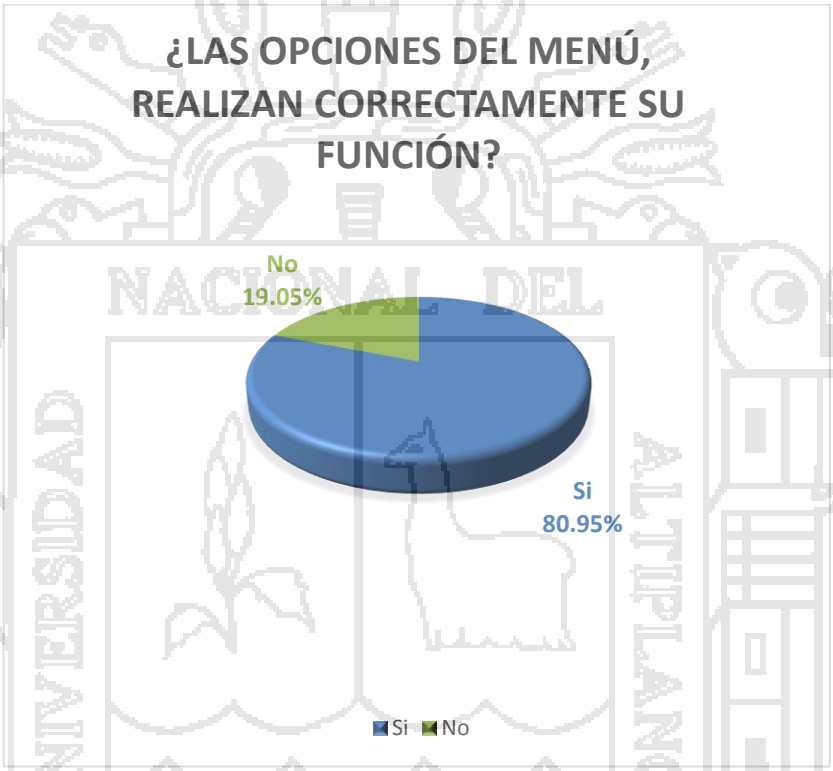


FIGURA 4.14: ACERCA DE LA CORRECTA FUNCIONALIDAD DE LAS OPCIONES DEL MENÚ

Un 80.95% de los encuestados respondió afirmativamente acerca de la correcta funcionalidad de las opciones del menú, en tanto que sólo un 19.05% de los encuestados respondieron negativamente, señalando en su mayoría que la recuperación de componentes no era lo suficientemente flexible con las rutas de destino. Además señalaron que la eliminación de componentes era muy estricta en su función.

4.5.2.3 ¿Es suficiente la información asociada a los componentes?

Condición	#	%
Si	40	95.24
No	2	4.76
Total	42	100

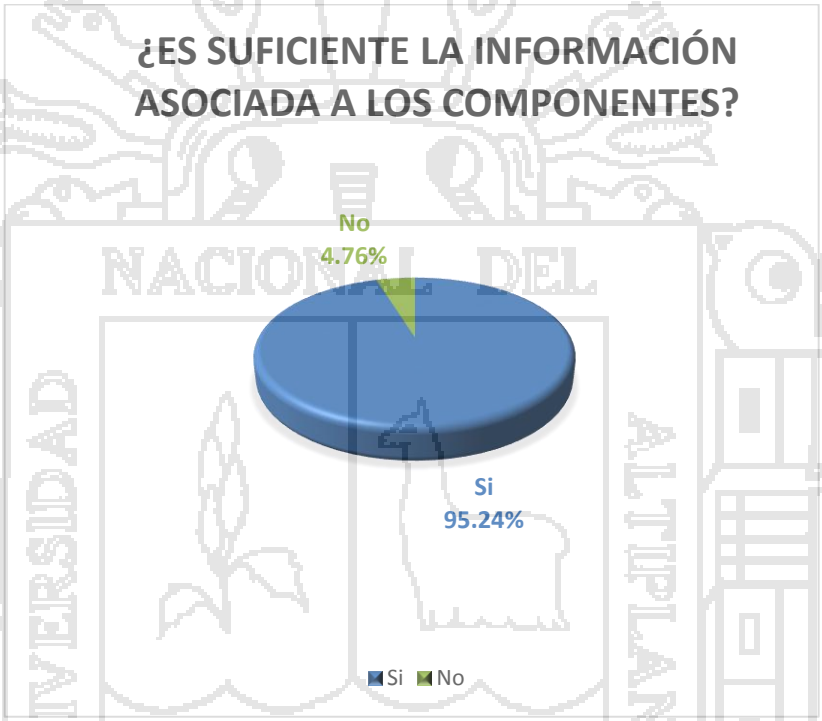


FIGURA 4.15: ACERCA DE LA INFORMACIÓN ASOCIADA A LOS COMPONENTES

Un gran porcentaje de los encuestados (95.24%) resolvió en responder que la información asociada a los componentes era suficiente, mientras que sólo un 4.76% respondió negativamente, señalando que era insuficiente, pues faltaban especificaciones técnicas acerca del lenguaje de los componentes, así como información acerca del número de líneas que ocupaban los componentes entre otras.

4.5.2.4 ¿La búsqueda de componentes candidato se realiza correctamente?

Condición	#	%
Si	42	100
No	0	0
Total	42	100

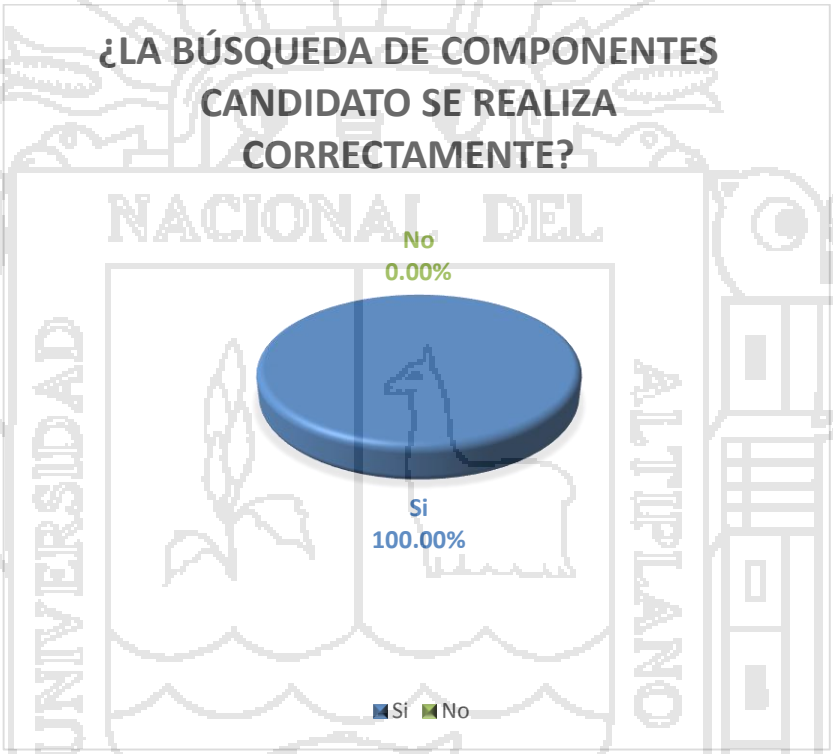


FIGURA 4.16: ACERCA DE LA BÚSQUEDA DE COMPONENTES

La totalidad de los encuestados coincidieron en responder que la búsqueda de componentes candidato se realizaba correctamente en sus dos modalidades.

4.5.2.5 ¿El resultado de la búsqueda de componentes, es congruente con su especificación numérica?

Condición	#	%
Si	42	100
No	0	0
Total	42	100

FIGURA 4.17: CON RESPECTO AL RESULTADO DE LA BÚSQUEDA Y SU ESPECIFICACIÓN NUMÉRICA

El 100% de los encuestados respondieron que el resultado de la búsqueda de componentes era totalmente congruente con la especificación numérica que ofrecía la herramienta.

4.5.2.6 ¿El formato de ingreso de los datos del componente es adecuado?

Condición	#	%
Si	30	71.43
No	12	28.57
Total	42	100

FIGURA 4.18: ACERCA DEL INGRESO DE DATOS DEL COMPONENTE

71.43% de los encuestados respondieron que el formato de ingreso de los datos del componente era adecuado, mientras que un 28.57% de los encuestados no les pareció adecuado dicho formato, aduciendo principalmente la falta de automatización de algunos campos, así como la excesiva extensión del mismo, entre otros.

4.5.2.7 ¿La exploración del repositorio y sus catálogos es la más adecuada?

Condición	#	%
Si	40	95.24
No	2	4.76
Total	42	100

FIGURA 4.19: ACERCA DE LA EXPLORACIÓN DEL REPOSITORIO Y SUS CATÁLOGOS

Un gran porcentaje de los encuestados (95.24%) coincidieron en afirmar que la exploración del los repositorios y sus respectivos catálogos era la más adecuada, frente a un escaso 4.76% de los encuestados que respondieron negativamente, indicando que no estaban acostumbrados a la propuesta.

4.5.2.8 ¿Existe congruencia entre el componente activo y los datos mostrados en la ficha múltiple?

Condición	#	%
Si	42	100
No	0	0
Total	42	100

FIGURA 4.20: ACERCA DE LA CONGRUENCIA ENTRE EL COMPONENTE Y LA FICHA MÚLTIPLE

El 100% de los encuestados respondieron que si existe congruencia entre el componente activo y la ficha múltiple mostrada por la herramienta.

4.5.2.9 ¿Midiendo en función del tiempo, que porcentaje cree Ud. que ahorra utilizando este modelo?

Intervalo	f	%
0 a 10%	2	4.76
11 a 20%	4	9.52
21 a 30%	9	21.43
31 a 40%	8	19.05
41 a 50%	14	33.33
más de 50%	5	11.90
Total	42	100

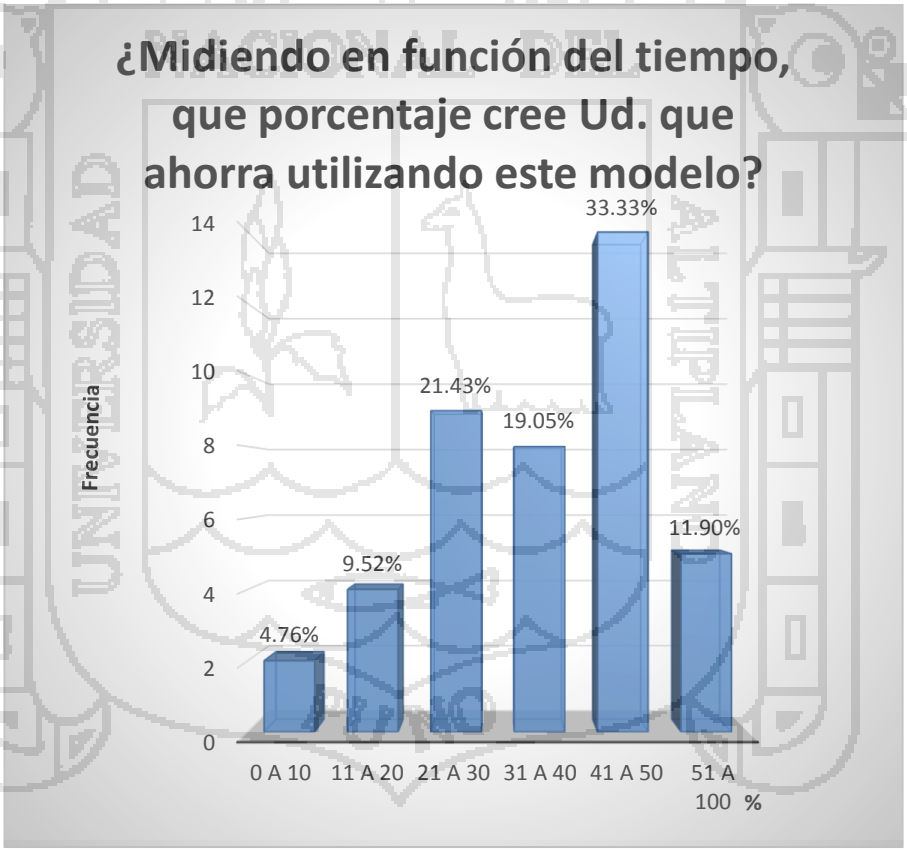


FIGURA 4.21: PORCENTAJE DE AHORRO DE TIEMPO

Un 4.76% de los encuestados señalan que ahorran menos del 10% de tiempo utilizando el modelo, un 9.52% de los encuestados señalan que ahorran de 11 a 20% de tiempo utilizando el modelo, en tanto que 21.43%

de los encuestados indican que el ahorro de tiempo es de 21 al 30%. Por otra parte, un 19.05% de los encuestados señalan que el ahorro de tiempo es de 31 a 40%, así mismo la tercera parte de los encuestados señalan que el ahorro de tiempo es de 41 a 50% y solo un 11.90% de los encuestados indican que ahorran más del 50% de tiempo utilizando el modelo en cuestión.

CAPÍTULO V
CONCLUSIONES Y RECOMENDACIONES

5 CONCLUSIONES.

5.1 CON RESPECTO A LA HERRAMIENTA.

- Los resultados obtenidos por los instrumentos de validación y la forma como se planificó la herramienta nos demuestran la total factibilidad de su construcción.
- El enfoque de la reutilización nos facilita grandemente la concepción de la herramienta.
- Los resultados de la encuesta nos demuestra fehacientemente que el presente modelo nos permite un ahorro sustancial de tiempo.
- Los repositorios de componentes se constituyen en los elementos centrales para el soporte operativo en la reutilización de software.

5.2 CON RESPECTO AL MODELO.

- Un porcentaje superior al 70% de los encuestados coinciden en afirmar que tanto el diseño, así como la distribución y semántica de sus elementos, son los más adecuados para este tipo de herramienta.
- Más del 80% de los encuestados respondió afirmativamente acerca de la correcta funcionalidad del modelo y la congruencia de sus resultados, esto se traduce en un alto grado de aceptación.
- El 100% de los encuestados afirman que ahorran tiempo (porcentajes variables) utilizando el modelo, lo cual nos demuestra que en cualquier caso, el ahorro de tiempo es considerable.

5.3 RECOMENDACIONES.

- Investigar sobre avances en la adquisición, representación y organización del conocimiento; tratamiento del lenguaje natural; tecnologías multimedia; repositorios distribuidos; motores de búsqueda; agentes inteligentes; comunicaciones –fundamentalmente internet-, etc., nos ofrecerán alternativas importantes para la reutilización efectiva de componentes software.
- Las organizaciones deben evaluar su nivel de reutilización y deberán trabajar para que ésta se convierta en una disciplina involucrada en el proceso mismo de desarrollo de software y cuente con la tecnología apropiada.
- Se recomienda se complemente el modelo para aprovechar un uso completo de la reutilización del software.
- Se recomienda que esta herramienta sea usada por los alumnos del curso de Ingeniería de Software, para que logren entender el problema de la reutilización de software por medio de repositorios.

5.4 BIBLIOGRAFÍA.

Altmeyer, J., & Otros. (1997). *Application of a Generator - Based Software Development Method Supporting Model Reuse.* Barcelona España: Antoni Olive.

Ambrosio, A. (1996). *Applying similarity vectors for the selection of reusable schemas.* Colombia: Memorias XXII Conferencia Latinoamericana de Informatica.

Arango, G. (1991). *Domain Analysis - From Art Form to Engineering Discipline.* Santiago Chile: Domain Analisys and Software Systems Modeling. IEEE Computer Society Press.

Baumer, D., & Otros. (1997). *Framework Development for Large Systems.* Brasil: Comunications of the ACM.

Bellinzona, R., Fugini, M., & Pernici, B. (1995). *Reusing Specifications in OO Applications.* California USA: IEEE Software.

Bernstein, P. A., & Dayal, U. (1994). *An Overview of Repository Technology.* Santiago - Chile: In the proceedings of the 20th International Conference on Very Large Data Bases.

Caldiera, G., & Basili, V. (1991). *Identifying and qualifying reusable software components* (Vol. 24). España: Computer.

Castaño, S., & De Antonellis, V. (1997). *Engineering a library of reusable conceptual components* (Vol. 39). USA: Information and Software Technology.

Constantopoulos, P., & Pataki, E. (1992). *A browser for software reuse.* USA: Proceeding 4th International Conference.

Fowler, M., & Scott, K. (1999). *UML – GOTA A GOTA.* may: ADDISON WESLEY LONGMAN.

Frakes, W., & Fox, C. (1995). *Sixteen Questions Abaut Software Reuse* (Vol. 38). California USA: ACM Comunications.

Frakes, W., & Terry, C. (1996). *Software Reuse: Metrics and Models* (Vol. 28). USA: ACM Computing Surveys.

Grady, B. (1996). *ANÁLISIS Y DISEÑO ORIENTADO A OBJETOS.* DIAZ DE SANTOS: ADDISON-WESLEY.

Hall, P. (1992). *Overwiew of reverse engineering and reuse search* (Vol. 34). USA: Information and Software Technology.

Isakowitz, T. (1996). *Supporting search for reusable objects* (Vol. 22). USA: Transactions on software engineering.

James, R., Michael, B., William, P., Frederick, E., & William, L. (1996). *MODELADO Y DISEÑO ORIENTADO A OBJETOS.* HALL.

Kara, D. (1997). *The Repository's Role in Component Development.* California USA: ACM Computing Surveys.

Krueger, C. W. (1992). *Software Reuse* (Vol. 24). New York USA: ACM Computing Surveys.

Marqués Corral, J. M. (1998). *Soporte Operativo para la Reutilización del Software: Repositorios y Clasificación.* Santiago Chile: En las actas del curso Ingeniería de Software y Reutilización: Aspectos Dinámicos y Generación Automática.

Meijler, T., & Otros. (1997). *Making Design Patterns Explicit in Face. A Framework Adaptive Composition Environment.* Madrid: ACM SOGSOFT Sym´psium.

Mili, H., Mili, F., & Mili, A. (1995). *Reusing Software: Issues and Research Directions.* USA: IEEE Transactions on Software Engineering.

Moore, J. (1991). *Domain Analysis Framework for Reuse.* USA: IEEE Computer Society Press.

Motro, A., & Goullioud, S. (1995). *Knowledge organization for exploration.* Bruselas España: Memorias DEXA95.

NATO. (1992). *NATO Standard for Management of a Reusable Software Component Library* (Vol. 2). USA: NATO Communications and Information Systems Agency.

Neighbors, J. D. (1989). *A method fpr engineering reusable software systems.* USA: Software Reusability, ACM Press.

Pree, W. (1994). *A Means for Capturing the Essentials of Reusable Object Oriented Design.* California USA: Springer Verlag.

Prieto Díaz, R. (1991). *Implementing Faceted Classification for Software Reuse* (Vol. 34). USA: Comunications of the ACM.

Prieto Díaz, R. (1993). *Status Report: Software Reusability.* USA: IEEE Software.

Prieto Díaz, R., & Arango, G. (1991). *Domain Analysis and Software Systems Modeling.* USA: IEEE Computer Society Press.

Roger S., P. (1997). *Ingeniería del software.* MC. GRAW HILL.

Rubén, P. D., & Guillermo, A. (1991). *Domain Analysis and Software Systems Modeling.* IEEE Computer Society Press.

Ruggia, R., & Ambrosio, A. P. (1997). *A toolkit for Reuse in Conceptual Modeling. Proceeding Advanced Information System Engineering.* Barcelona España: Antoni Olive.

Tracz, W. (1990). *Tutorial: Software Reuse: Emerging Technology.* USA: The Computer Society of IEEE.

Viasoft Inc. (1997). *The Rochade Repository Enviromment. The Fundation for the Information Asset Warehouse.* USA: Viasoft Incorporation White Paper.

Wilhelm, S., & Rubén, P.-D. (1994). *Software Reusability.* Ellis Horwood Limited.

ANEXOS

5.5 ANEXO 1.

ENCUESTA

EMPRESA: _____
ENTREVISTADO: _____
CARGO: _____**FECHA:**_____

1. CON RESPECTO A LA INTERFAZ DEL MODELO

1.1. ¿La interfaz con el usuario, te pareció agradable?

Si	
No	

Si su respuesta es No, ¿Por qué?_____
_____
_____

1.2. ¿La distribución del menú es comprensible?

Si	
No	

Si su respuesta es No, ¿Por qué?_____
_____
_____

1.3. ¿Los iconos, son claros en su semántica?

Si	
No	

Si su respuesta es No, ¿Por qué?_____
_____
_____

1.4. ¿El diseño de pantallas es apropiado?

Si	
No	

Si su respuesta es No, ¿Por qué?_____
_____
_____

1.5. ¿La distribución de colores, es clara?

Si	
No	

Si su respuesta es No, ¿Por qué?_____
_____
_____

1.6. ¿Los mensajes son adecuados?

Si	
No	

Si su respuesta es No, ¿Por qué?_____
_____
_____

1.7. ¿La ayuda es adecuada?

Si	
No	

Si su respuesta es No, ¿Por qué?_____
_____
_____

2. CON RESPECTO A LA FUNCIONALIDAD DEL MODELO

2.1. ¿Es comprensible la distribución del repositorio?

Si	
No	

Si su respuesta es No, ¿Por qué?_____
_____
_____

2.2. ¿Las opciones del menú, realizan correctamente su función?

Si	
No	

Si su respuesta es No, ¿Por qué?_____
_____
_____

2.3. ¿Es suficiente la información asociada a los componentes?

Si	
No	

Si su respuesta es No, ¿Por qué?_____
_____
_____

2.4. ¿La búsqueda de componentes candidato se realiza correctamente?

Si	
No	

Si su respuesta es No, ¿Por qué?_____
_____
_____

2.5. ¿El resultado de la búsqueda de componentes, es congruente con su especificación numérica?

Si	
No	

Si su respuesta es No, ¿Por qué?_____
_____
_____

2.6. ¿El formato de ingreso de los datos del componente es adecuado?

Si	
No	

Si su respuesta es No, ¿Por qué?_____
_____
_____

2.7. ¿La exploración del repositorio y sus catálogos es la más adecuada?

Si	
No	

Si su respuesta es No, ¿Por qué?_____
_____
_____

2.8. ¿Existe congruencia entre el componente activo y los datos mostrados en la ficha múltiple?

Si	
No	

Si su respuesta es No, ¿Por qué?_____
_____
_____

2.9. ¿Midiendo en función del tiempo, que porcentaje cree Ud. que ahorra utilizando este modelo?

_____
_____
_____

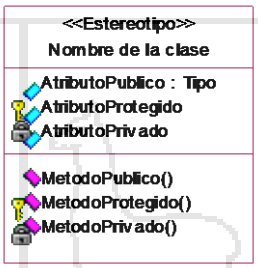
5.6 ANEXO 2.

NOTACIÓN UML

ELEMENTOS.

- **Clases.**

Una clase es representada como un rectángulo, el cual incluye su nombre, además de sus atributos y operaciones de manera opcional. Las clases además pueden incluir un estereotipo. Se puede pensar en un estereotipo como un metatipo y se usa para extender la semántica de los iconos en UML.



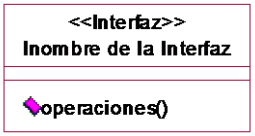
- **Objetos.**

Los objetos se representan de manera similar a las clases, pero el nombre del objeto va subrayado, además se puede incluir el nombre de la clase a la que pertenece.

Nombre del Objeto: Nombre de la Clase

- **Interfaces.**

A diferencia de una clase abstracta una interfaz no puede tener atributos.



- **Actores.**

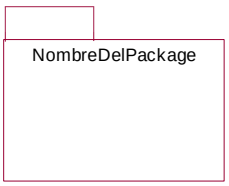
Un actor se puede representar con el mismo icono de una clase, o una representación simple de una persona.

- **Casos de uso.**

Un caso de uso se representa mediante una elipse

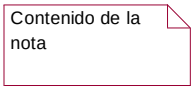
- **Package.**

Un package es un mecanismo de propósito general para organizar elementos en grupos.



- **Notas.**

Las notas son partes aclaratorias de los modelos en UML.



TIPOS DE DIAGRAMAS EN UML.

- **Diagrama de Clases.**

Un diagrama de clases muestra un conjunto de clases, interfaces, colaboraciones y sus relaciones. Estos diagramas proporcionan una vista estática del sistema.

- **Diagrama de Casos de Uso.**

Un diagrama de casos de uso muestra un conjunto de casos de uso, actores y sus relaciones. Este diagrama muestra una vista estática de las funcionalidades del sistema. Este diagrama es especialmente importante para la organización y modelamiento del comportamiento del sistema.

- **Diagramas de Interacción.**

Los diagramas de secuencia y los diagramas de colaboración son géneros de diagramas de interacción.

- **Diagramas de secuencia.**

Un diagrama de secuencia muestra una interacción, consistente de un conjunto de objetos y sus relaciones, incluyendo los mensajes que se envían. Este diagrama muestra una vista dinámica del sistema.

- **Diagrama de colaboración.**

Un diagrama de colaboración, enfatiza la organización y estructura de los objetos que envían y reciben mensajes. Los diagramas de secuencia y colaboración son isomorfos, esto significa que se pueden transformar diagramas de un tipo al otro y viceversa.

- **Diagramas de Actividad.**

Son un género especial de diagramas de estados que muestra el flujo entre actividades del sistema. Los diagramas de actividad muestran una vista dinámica del sistema y son especialmente importantes en el modelamiento del funcionamiento del sistema y enfatizan el flujo de control entre objetos.

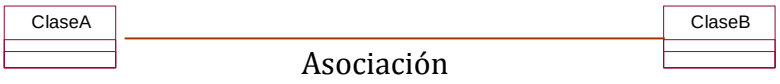
- **Diagrama de Componentes.**

Estos diagramas muestran la organización y dependencia entre conjuntos de componentes. Los diagramas de componentes muestran una vista estática de la implementación del sistema. Estos diagramas están relacionados con los diagramas de clases debido a que los componentes se derivan de clases, interfaces y colaboraciones.

TIPOS DE RELACIONES EN UML.

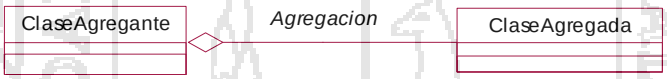
- **Relación de Asociación.**

Es una relación estructural, que indica que un clasificador esta conectado a otro. La relación es bidireccional, a menos que se indique lo contrario.



- **Asociación de agregación.**

Una asociación simple indica que las clases relacionadas tienen el mismo nivel (igual importancia). Agregación es una asociación todo/parte. Representa una relación "tiene un". Esto no cambia el sentido de la navegación.

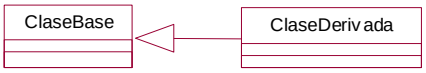


- **Asociación de Composición.**

Composición, es una forma de agregación, con un fuerte sentido de propiedad y tiempo de vida coincidente como parte del todo.

- **Relación de generalización.**

Muestra relaciones de generalización/especialización. Las clases virtuales y operaciones virtuales se representan con itálica.



- **Relación de dependencia.**

Esta relación indica que el cambio en una clase podría afectar a otra que la usa, pero no necesariamente al revés. Se debe usar esta relación cuando: 1) Se desee indicar que una cosa depende de la otra. 2) El cambio en una cosa podría afectar a la otra, pero no necesariamente al revés. 3) Se quiera mostrar que una cosa usa otra. 4) Para indicar que una clase usa a otra como argumento en la firma de una operación (parámetro).

5.7 ANEXO 3.

CODIFICACIÓN DE LOS PRINCIPALES MODULOS DEL MODELO

- **MODULO EXPLORAR COMPONENTES.**

```
Dim Cons As String

Private Sub cmdAceptar_Click()

   'Cierra la Ventana Actual
   Unload Me

End Sub

Private Sub DBGrid1_Click()
Dim XCodCom As String * 3

If Not (Data1.Recordset.EOF Or Data1.Recordset.BOF) Then
   'Ubicacion del codigo para el componente elegido
   XCodCom = Data1.Recordset.Fields("CodCom")
   'Filtro para mostrar los usuarios del componente
   Data6.RecordSource = "Select * from Usuarios Where [CodCom]='" & XCodCom & "'"
   Data6.Refresh
   DBList1.ListField = "NomUsu"
Else
   'Limpia si no hay componentes
   DBList1.ListField = ""
End If

   DBList1.Refresh

End Sub

Private Sub Dir1_Change()
   Dim UbiCom As String * 2

   'Solo se podrá grabar en Repositorios de la unidad C
   If Dir1.Path = "C:\" Then Dir1.Path = "C:\Repositorios"

   lblRuta.Caption = Dir1.Path

   'Obtener el codigo del catalogo
   UbiCom = "C" + Right(Dir1.Path, 1)

   'Filtro para mostrar los componentes segun el tipo de catalogo donde se ubique
   Data1.RecordSource = "Select * from Repositorio Where [CodCat]='" & UbiCom & "'"
   Data1.Refresh

   DBGrid1_Click

End Sub

Private Sub Form_Activate()

   'Activar el menu eliminar y recuperar (Componentes)
   MDIForm1.MnuEliminar.Enabled = True
   MDIForm1.MnuRecuperar.Enabled = True
   MDIForm1.Toolbar1.Buttons.Item(5).Enabled = True
   MDIForm1.Toolbar1.Buttons.Item(3).Enabled = True

End Sub
```

```
Private Sub Form_Load()

    'Propiedades del formulario
    frmExplorar.WindowState = vbMaximized

    'Ubicacion en la estructura de catalogos
    Dir1.Path = "C:\Repositorios"
    lblRuta.Caption = Dir1.Path

    'Al inicio no hay nada en el Filtro (repositorios y usuarios)
    Data1.RecordSource = ""
    Data1.Refresh

    Data6.RecordSource = ""
    Data6.Refresh

End Sub

Private Sub Form_Unload(Cancel As Integer)

    'Desactivar el menu insertar y recuperar al cerrar
    MDIForm1.MnuEliminar.Enabled = False
    MDIForm1.MnuRecuperar.Enabled = False
    MDIForm1.Toolbar1.Buttons.Item(5).Enabled = False
    MDIForm1.Toolbar1.Buttons.Item(3).Enabled = False

End Sub
```

- **MODULO INSERTAR COMPONENTES.**

```
Dim UbiComCat As String * 2

Private Sub cmdCerrar_Click()

    Unload Me
    'frmInsertar.Hide
End Sub

Private Sub cmdInsertar_Click()

    Dim NroCom As Integer
    'Generar el numero correlativo del componente
    If Data1.Recordset.EOF Then
        NroCom = 1
    Else
        Data1.Recordset.MoveLast
        NroCom = (Data1.Recordset.RecordCount) + 1
    End If

    'Buscar el Tipo de componente
    Data2.Recordset.FindFirst "[DesCom]='" & DBCombo1.Text & "'"
    'Buscar el Tipo de Licencia
    Data3.Recordset.FindFirst "[DesLic]='" & DBCombo2.Text & "'"
    'Buscar el Tipo de Dominio
    Data4.Recordset.FindFirst "[DesDom]='" & DBCombo3.Text & "'"
    'Buscar el Tipo de Area
    Data5.Recordset.FindFirst "[DesArea]='" & DBCombo4.Text & "'"
```

```
    'Abrir un Nuevo Registro
    Data1.Recordset.AddNew

    'Actualizar datos en los campos
    Data1.Recordset.Fields("CodCom") = NroCom
    Data1.Recordset.Fields("NomCom") = Text12.Text
    Data1.Recordset.Fields("AutCom") = Text4.Text
    Data1.Recordset.Fields("DesCom") = Text8.Text
    Data1.Recordset.Fields("Obs01") = IIf(Text9.Text = "", "ninguna", Text9.Text)
    Data1.Recordset.Fields("Obs02") = IIf(Text10.Text = "", "ninguna", Text10.Text)
    Data1.Recordset.Fields("Obs03") = IIf(Text11.Text = "", "ninguna", Text11.Text)
    Data1.Recordset.Fields("DirAut") = Text7.Text
    Data1.Recordset.Fields("MailAut") = Text6.Text
    Data1.Recordset.Fields("TelAut") = Text5.Text
    Data1.Recordset.Fields("TipCom") = DBCombo1.Text 'Data2.Recordset.Fields("TipCom")
    Data1.Recordset.Fields("TipLic") = DBCombo2.Text 'Data3.Recordset.Fields("TipLic")
    Data1.Recordset.Fields("TipDom") = DBCombo3.Text 'Data4.Recordset.Fields("TipDom")
    Data1.Recordset.Fields("TipArea") = DBCombo4.Text 'Data5.Recordset.Fields("TipArea")
    Data1.Recordset.Fields("TamCom") = Val(Text1.Text)
    Data1.Recordset.Fields("FecCom") = CDate(Text2.Text)
    Data1.Recordset.Fields("VerCom") = Text3.Text
    Data1.Recordset.Fields("CodCat") = UbiComCat

    'Actualiza los Datos en la Base
    Data1.Recordset.Update

    'Grabar el Usuario que uso/creo el componente
    Data6.Recordset.AddNew
    Data6.Recordset.Fields("CodCom") = NroCom
    Data6.Recordset.Fields("NomUsu") = Text4.Text
    Data6.Recordset.Update

    'Grabar la Ubicacion del componente
    Data8.Recordset.AddNew
    Data8.Recordset.Fields("CodCom") = NroCom
    Data8.Recordset.Fields("CodCat") = UbiComCat
    Data8.Recordset.Update

    cmdInsertar.Enabled = False
    cmdCerrar.SetFocus
End Sub

Private Sub Dir1_Change()

    If Dir1.Path = "C:\" Then Dir1.Path = "C:\Repositorios"

    UbiComCat = "C" + Right(Dir1.Path, 1)

    If Not (UbiComCat = "C1" Or UbiComCat = "C2" Or UbiComCat = "C3" Or UbiComCat = "C4") Then
        'Si no se elije una ubicacion se graba en el catalogo 1
        UbiComCat = "C1"
    End If
End Sub

Private Sub Form_Activate()

    'Se carga la fecha del sistema en el campo
    Text2.Text = Date
End Sub
```

```
Private Sub Form_Load()

   'Dimensiones del formulario
   frmInsertar.WindowState = vbMaximized

   'Ubicacion de la estructura de directorios
   Dir1.Path = "C:\Repositorios"
End Sub

Private Sub Text1_Validate(Cancel As Boolean)
   'Consistencia para que el campo no quede sin datos
   If Text1.Text = "" Then
      MsgBox "No dejar Vacio"
      Cancel = True
   End If
End Sub

Private Sub Text10_LostFocus()
   'Por ser Observacion puede estar sin datos
   If Text10.Text = "" Then
      Text10.Text = "ninguna"
   End If
End Sub

Private Sub Text11_LostFocus()
   'Por ser Observacion puede estar sin datos
   If Text11.Text = "" Then
      Text11.Text = "ninguna"
   End If
End Sub

Private Sub Text12_Validate(Cancel As Boolean)
   'Consistencia para que el campo no quede sin datos
   If Text12.Text = "" Then
      MsgBox "No dejar Vacio"
      Cancel = True
   End If
End Sub

Private Sub Text2_Validate(Cancel As Boolean)
   'Consistencia para que el campo no quede sin datos
   If Text2.Text = "" Then
      MsgBox "No dejar Vacio"
      Cancel = True
   End If
End Sub

Private Sub Text3_Validate(Cancel As Boolean)
   'Consistencia para que el campo no quede sin datos
   If Text3.Text = "" Then
      MsgBox "No dejar Vacio"
      Cancel = True
   End If
End Sub

Private Sub Text4_Validate(Cancel As Boolean)
   'Consistencia para que el campo no quede sin datos
   If Text4.Text = "" Then
      MsgBox "No dejar Vacio"
      Cancel = True
   End If
```

```
End Sub

Private Sub Text5_Validate(Cancel As Boolean)
   'Consistencia para que el campo no quede sin datos
   If Text5.Text = "" Then
      MsgBox "No dejar Vacio"
      Cancel = True
   End If
End Sub

Private Sub Text6_Validate(Cancel As Boolean)
   'Consistencia para que el campo no quede sin datos
   If Text6.Text = "" Then
      MsgBox "No dejar Vacio"
      Cancel = True
   End If
End Sub


Private Sub Text7_Validate(Cancel As Boolean)
   'Consistencia para que el campo no quede sin datos
   If Text7.Text = "" Then
      MsgBox "No dejar Vacio"
      Cancel = True
   End If
End Sub

Private Sub Text8_Validate(Cancel As Boolean)
   'Consistencia para que el campo no quede sin datos
   If Text8.Text = "" Then
      MsgBox "No dejar Vacio"
      Cancel = True
   End If
End Sub

Private Sub Text9_LostFocus()
   'Por ser Observacion puede estar sin datos
   If Text9.Text = "" Then
      Text9.Text = "ninguna"
   End If
End Sub
```

- **MODULO BUSCAR COMPONENTES.**

```
Private Sub cmdBuscar_Click()
Dim Clave As Byte

'Determinar que se eligio (Dominio, Componente o una cadena) a buscar

If Text16.Text = "" And Combo1.Text = "Todos" And Combo2.Text = "Todos" Then
   Clave = 1
ElseIf Text16.Text = "" And Combo1.Text = "Todos" And Combo2.Text <> "Todos" Then
   Clave = 2
ElseIf Text16.Text = "" And Combo1.Text <> "Todos" And Combo2.Text = "Todos" Then
   Clave = 3
ElseIf Text16.Text = "" And Combo1.Text <> "Todos" And Combo2.Text <> "Todos" Then
```

```
   Clave = 4
ElseIf Text16.Text <> "" And Combo1.Text = "Todos" And Combo2.Text = "Todos" Then
   Clave = 5
ElseIf Text16.Text <> "" And Combo1.Text = "Todos" And Combo2.Text <> "Todos" Then
   Clave = 6
ElseIf Text16.Text <> "" And Combo1.Text <> "Todos" And Combo2.Text = "Todos" Then
   Clave = 7
ElseIf Text16.Text <> "" And Combo1.Text <> "Todos" And Combo2.Text <> "Todos" Then
   Clave = 8
End If

'Filtro para mostrar los componentes segun lo que se elija

Cadena = "*" & Text16.Text & "*"
Select Case Clave
Case 1
   Data1.RecordSource = "Select * from Repositorio"
Case 2
   Data1.RecordSource = "Select * from Repositorio Where [TipCom]='" & Combo2.Text & "'"
Case 3
   Data1.RecordSource = "Select * from Repositorio Where [TipDom]='" & Combo1.Text & "'"
Case 4
   Data1.RecordSource = "Select * from Repositorio Where [TipDom]='" & Combo1.Text & "' and [TipCom]='" &
Combo2.Text & "'"
Case 5
   Data1.RecordSource = "Select * from Repositorio Where [DesCom] Like '" & Cadena & "'"
Case 6
   Data1.RecordSource = "Select * from Repositorio Where [TipCom]='" & Combo2.Text & "' and [DesCom] Like
'" & Cadena & "'"
Case 7
   Data1.RecordSource = "Select * from Repositorio Where [TipDom]='" & Combo1.Text & "' and [DesCom] Like
'" & Cadena & "'"
Case 8
   Data1.RecordSource = "Select * from Repositorio Where [TipDom]='" & Combo1.Text & "' and [TipCom]='" &
Combo2.Text & "' and [DesCom] Like '" & Cadena & "'"
End Select

Data1.Refresh

'Contar el Numero de registros (componentes)
NroCom = Data1.Recordset.RecordCount
lblResultado.Caption = "Resultado de la Búsqueda: Se encontró " + Str(NroCom) + " " + " Registro (s)"

DBGrid1_Click

End Sub

Private Sub DBGrid1_Click()
Dim XCodCom As String * 3

If Not (Data1.Recordset.EOF Or Data1.Recordset.BOF) Then
   'Ubicacion del codigo para el componente elegido
   XCodCom = Data1.Recordset.Fields("CodCom")
   'Filtro para mostrar los usuarios del componente
   Data6.RecordSource = "Select * from Usuarios Where [CodCom]='" & XCodCom & "'"
   Data6.Refresh
   DBList1.ListField = "NomUsu"
Else
   'Limpia si no hay componentes
   DBList1.ListField = ""
End If
```

```
    DBList1.Refresh

End Sub

Private Sub Form_Activate()

    'Activar el menu eliminar
    MDIForm1.MnuEliminar.Enabled = True

End Sub


Private Sub Form_Load()

    'Propiedades del formulario
    frmBuscar.WindowState = vbMaximized

    'Al inicio se visualizan todos los componentes
    Data1.RecordSource = "Select * From Repositorio"
    Data1.Refresh

    Data6.RecordSource = "Select * From Usuarios"
    Data6.Refresh

End Sub

Private Sub Form_Unload(Cancel As Integer)

    'Desactivar el menu insertar al cerrar
    MDIForm1.MnuEliminar.Enabled = False

End Sub

Private Sub Text16_KeyPress(KeyAscii As Integer)
    If KeyAscii = 13 Then
        cmdBuscar.SetFocus
    End If
End Sub
```