Universidad de las Ciencias Informáticas

Facultad 6

Título: Modelo de activos de software reutilizables.

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

Autora: Adriana Riera Pérez

Tutora: Ing. Lissete González Gallo

Junio 2011

"Año 53 de la Revolución"

Declaración de autoría

Declaro ser la autora de la presente tesis y reconozco a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

Para que así conste firmo la presente a los ____ días del mes de _____ del año _____.

Adriana Riera Pérez

_____

Firma del Autor

Ing. Lissete González Gallo

_____

Firma del Tutor

Datos de contacto.

Síntesis del Autor: Adriana Riera Pérez.

Correo Electrónico: ariera@estudiantes.uci.cu.

Síntesis del Tutor: Ing. Lissete González Gallo.

Profesión: Ingeniero en Ciencias Informáticas.

Años de graduado: 2008.

Correo Electrónico: lgallo@uci.cu.

Agradecimientos:

Agradezco a todas las personas que de una forma u otra me han ayudado a salir adelante, especialmente:

A mis amados padres por todos sus sacrificios en el transcurso de estos cinco años,

A mi querido Pablito por brindarme toda su energía, incluso en los momentos difíciles,

A mi tutora Lissete por todos sus consejos y su dedicación,

A mi abuelito Roberto por estar siempre apoyándome,

A mis abuelitos Orlando, Dulce y Nancy porque aunque no estén junto a mí, les agradezco todo su infinito amor y los bellos recuerdos,

A la querida Zule por darme todo su amor de abuela,

A mis amigas del alma Denita, Ani, Meily y Yeni por sus consejos y aliento,

A mi nueva familia, Anita, Ana, Alina, Riguito y Jorge Luis por brindarme todo su cariño,

A mis amigas Dayana y Yisel por hacer inolvidable mi primer año en la UCI,

A mis tíos Lino, Liudmila, Eddy, Maricela, Orlando, Niurka e Ileana por toda su ayuda en el transcurso de estos años,

A mis primitos adorados María Karla(Karlina), Mariolys(Mario) y el pequeño Lian(Muñeco) por brindarme tantas alegrías,

A todos ustedes, una y mil veces Gracias…

Dedicatoria:

Dedico este trabajo a mis padres, por ser la recompensa más grande que puedo ofrecerles después de tanto tiempo de sacrificios y ausencia.

Resumen:

Las Líneas de Productos de Software (LPS) constituyen una técnica de desarrollo de software basada en la reutilización de activos de software, previamente elaborados y almacenados en repositorios. Actualmente resulta muy difícil identificar los activos de software dentro de los sistemas ya desarrollados así como a los repositorios de activos de software, lo que dificulta la adopción de esta avanzada técnica en la Universidad de las Ciencias Informáticas (UCI).

Esta investigación se efectuó con el objetivo de proponer un modelo de activo de software que permita la clasificación de los activos de software reutilizables para facilitar el proceso de producción de software luego de aplicar las LPS. Un análisis profundo sobre los conceptos relacionados con la reutilización de activos de software y los diferentes conceptos de modelos existentes, sirvieron como punto de partida para el desarrollo del modelo propuesto.

Para llevar a cabo la clasificación se definieron un conjunto de características que los activos de software y los repositorios deben cumplir, además se puntualizaron los responsables de realizar la clasificación y las actividades a desarrollar por estos.

Se obtuvo un modelo que se espera sea aplicado en la universidad, para facilitar la identificación de los activos de software reutilizables y de sus repositorios.

PALABRAS CLAVES: Activos de software, Líneas de Productos de Software, Reutilización de software.

Índice

Índice

Introducción

El desarrollo de la industria de software crece a un ritmo vertiginoso, a medida que los sistemas de software requeridos se vuelven sumamente costosos, complejos y requieren resultados de una alta calidad, es por ello que deben utilizarse las más avanzadas técnicas y alternativas de la ingeniería de software para alcanzar productos con la combinación perfecta de rapidez, calidad y costo.

La Universidad de las Ciencias Informáticas (UCI) se dedica a la producción de software y tiene como paradigma convertirse en una de las primeras productoras de software del país, pero el desarrollo de software en la actualidad presenta problemas, entre los que se encuentra la falta de calidad, el incumplimiento con el tiempo planificado y existe un sobre esfuerzo por parte de los trabajadores, es por ello que se hace necesario emplear nuevas y avanzadas técnicas de producción de software, que garanticen una forma eficiente de trabajo.

El modelo Líneas de Productos de Software (LPS) se ha convertido en un alivio para las empresas productoras de software a escala mundial, pues se sustenta en el principio de la reutilización de software, que hace posible la composición de productos de software partiendo de un conjunto de activos ya desarrollados y probados, que son gestionados, almacenados y clasificados en los repositorios de software, por lo tanto puede ayudar en factores como:

- El costo de la ingeniería.
- El tiempo de entrega de los productos de forma económica.
- La entrega de productos con una alta calidad.
- El esfuerzo de los trabajadores.
- La erradicación de defectos en los productos.

La implantación de este nuevo modelo en la UCI representa una alternativa que reportaría grandes beneficios, pero a su vez es un proceso complejo y requiere de una alta organización. El desconocimiento de la identificación de los activos de software, así como la creación y trabajo con los repositorios,

representa una dificultad para la adopción eficiente de este nuevo modelo. Teniendo en cuenta lo anteriormente planteado surge como **problema científico**: ¿cómo determinar que un artefacto constituya un activo de software reutilizable?

A partir del problema científico se define como **objeto de estudio**: modelo de producción Líneas de Productos de Software y el **campo de acción:** el desarrollo de software basado en componentes.

Para dar solución al problema científico propuesto se define como **objetivo general:** desarrollar un modelo que permita clasificar los activos de software reutilizables para facilitar el proceso de producción de software basado en componentes.

Se definen como tareas de la investigación:

- Análisis de los conceptos asociados al desarrollo de software basado en componentes.
- Análisis de los modelos de activos de software existentes.
- Realización del diseño de la estructura del modelo de activos de software reutilizables.
- Definición de las actividades para identificar los activos de software reutilizables.
- Identificación de los responsables de las actividades propuestas para la clasificación de los activos de software reutilizables.
- Definición de los artefactos resultantes en la clasificación de los activos de software reutilizables.
- Descripción del almacenamiento de los activos de software reutilizables.
- Descripción de la recuperación de los activos de software reutilizables.
- Identificación de los responsables de las actividades propuestas para asegurar el almacenamiento y recuperación de los activos de software reutilizables.
- Definición de los artefactos resultantes de las actividades propuestas para asegurar el almacenamiento y recuperación de los activos de software reutilizables.

- Selección de los proyectos en los que se aplicará el modelo de activos de software reutilizables.
- Aplicación del modelo de activos de software reutilizables.
- Realización del análisis de los resultados obtenidos con la aplicación del modelo de activos de software reutilizables.

La investigación está estructurada de la siguiente manera:

Capítulo 1: Fundamentación Teórica.

En este capítulo se analizarán los conceptos fundamentales vinculados a la reutilización de software tales como, los activos de software reutilizables, los repositorios de activos de software reutilizables, las Líneas de Productos de Software y los modelos relevantes a esta investigación. Además se mencionarán las principales empresas que emplean las Líneas de Productos de Software y los beneficios de éstas.

Capítulo 2: Modelo de activos de software.

En este capítulo se presentará la estructura del modelo de activos de software propuesto. Se definirán las características que deben cumplir tanto los activos de software para ser reutilizables, como los repositorios para poder almacenar a dichos activos. Quedará puntualizado qué roles se responsabilizarán por el cumplimiento de tales características.

Capítulo 3: Validación del modelo de activos de software reutilizables.

En este capítulo se aplicará el modelo de activos de software reutilizable a los proyectos seleccionados y se realizará un análisis de los resultados obtenidos.

Capítulo I. Fundamentación Teórica

Introducción

En este capítulo se analizarán los conceptos fundamentales vinculados a la reutilización de software tales como, los activos de software reutilizables, los repositorios de activos de software reutilizables, las Líneas de Productos de Software y los modelos relevantes a esta investigación. Además se mencionarán las principales empresas que emplean las Líneas de Productos de Software y los beneficios de estas.

1.1 Activos de software

Los activos de software juegan un papel fundamental dentro de las fábricas de software, pues estos son los que permiten la composición segura y con calidad de los productos finales.

Un buen punto de partida para lograr un entendimiento claro de lo que son los activos de software, será el concepto planteado en la conferencia impartida por Jonás A Montilva: “Un activo reutilizable es un producto diseñado expresamente para ser empleado de forma recurrente en el desarrollo de muchos sistemas y aplicaciones. Ejemplos de activos reutilizables son: algoritmos, patrones de diseño, esquemas de base de datos, arquitecturas de software, especificaciones de requerimientos, de diseño y de prueba, entre otros”. [1]

Luego de realizar un análisis del concepto planteado podemos afirmar que los activos de software reutilizables pueden ser: algoritmos, patrones de diseños, esquemas de base de datos, entre otros, pero ¿Cuál sería la diferencia entre un artefacto de un sistema y un activo reutilizable? Los activos de software deben poseer un diseño que los haga únicos y que logre que su reutilización sea útil en el desarrollo de otros sistemas informáticos. Para alcanzar un entendimiento sobre la diferencia entre los artefactos y los activos de software se hace necesario realizar un análisis sobre el diseño que debe poseer un activo de software reutilizable.

1.2 Componentes de software

Para lograr un entendimiento a profundidad sobre los componentes de software realizaremos un estudio sobre los principales conceptos que se plantean al respecto.

Los expertos en el tema, Clements Szyperski y C. Pfister plantean que un componente es “una unidad de composición de aplicaciones software, que posee un conjunto de interfaces y un conjunto de requisitos, y que ha de poder ser desarrollado, adquirido, incorporado al sistema y compuesto con otros componentes de forma independiente, en tiempo y espacio”. [2]

En el sitio de Microsoft se plantea que los componentes son “paquetes software que ofrecen servicios a través de sus interfaces”. [3]

Un grupo de expertos plantean en el informe Desarrollo de Software Basado en Componentes: “como resultado de presiones crecientes sobre la industria del software orientadas a reducir drásticamente el costo y tiempo de desarrollo de sistemas y aplicaciones, sin afectar los niveles de calidad del producto, ha surgido un nuevo activo reutilizable denominado componente de software”. [4]

Muchos son los conceptos que se plantean de los componentes de software pero se tomará en cuenta la definición de F. Bachmann pues es la más reciente realizada con el propósito de consolidar todas las emitidas anteriormente, un componente de software es “una implementación opaca de funcionalidad, sujeta a composición por terceros y que cumple con un modelo de componentes”. [5]

El análisis de los conceptos planteados permitió comprender que los componentes de software constituyen un nuevo tipo de activo de software que permite la composición de sistemas con rapidez y asegurando su calidad. Los componentes además pueden ser tanto adquiridos por proveedores independientes como desarrollados internamente, por lo tanto se requiere que los componentes puedan interactuar entre ellos por lo que deben cumplir con un modelo de componentes que garantice la coordinación e interacción entre estos. Los componentes que se adquieren mediante proveedores son conocidos como componentes COTs y generalmente la implementación de estos es desconocida para los clientes. ¿Resultará segura la compra de componentes COTs?

1.2.1 Componentes COTS

¿Qué es un componente COTS[1]?

Los componentes COTS son componentes de software desarrollados de antemano y de índole comercial.

Actualmente existen muchas empresas resistentes a comprar componentes de software reutilizables a compañías externas y esto se debe a las causas planteadas por Ian Sommerville:

- **Confiabilidad de los componentes**: Los componentes son cajas negras de unidades de programas, y el código de los programas puede no estar disponible para los usuarios de dichos componentes. En tales casos,¿Cómo sabe un usuario si el componente es confiable?. Su comportamiento no funcional puede no ser el esperado y un problema más grave es que el componente podría ser un caballo de Troya que oculta código dañino que viola la seguridad del sistema.

- **Certificación de componentes**: Estrechamente relacionada con la confiabilidad se halla la cuestión de la certificación. Se ha propuesto que asesores independientes deberían certificar los componentes para asegurar a los usuarios que los componentes sean confiables. Sin embargo, no ha quedado claro quién la costeará y los límites de las responsabilidades. Desde nuestro punto de vista, la única solución viable es certificar que los componentes cumplen una especificación formal. Sin embargo la industria no parece dispuesta a pagar por esto.

- **Predicción de propiedades emergentes**: Los componentes son opacos, predecir sus propiedades emergentes es particularmente difícil. Como consecuencia, es posible encontrarse con que, cuando se integran componentes, el sistema resultante tiene propiedades no deseadas que limitan su uso. Propiedad emergente: No se puede atribuir a ninguna parte específica del sistema, más bien emergen solo cuando los componentes del sistema han sido integrados.

- **Equilibrio de requerimientos:** Normalmente se tiene que encontrar un equilibrio entre los requerimientos ideales y los componentes disponibles en el proceso de especificación y diseño del

[1] COTS: Commercial Off-The-Shelf, Comerciales fuera de la plataforma

sistema. Por el momento, alcanzar este equilibrio es un proceso intuitivo. Se necesita un método de análisis de equilibrio sistemático y más estructurado para ayudar a los diseñadores a seleccionar y configurar componentes. [6]

El desarrollo de activos de software dentro de la universidad, sería la solución más adecuada si se tiene en cuenta, la importancia que traería para el sistema de seguridad informática la implementación de sus propios activos.

Se garantiza además la funcionalidad de los componentes de manera adecuada sin correr el riesgo de no saber qué hacer con las propiedades emergentes y la conveniencia de tener a mano al desarrollador de dichos activos y sistemas.

Los propios trabajadores del centro de producción serían los responsables del buen funcionamiento de los componentes y se lograría un entendimiento sobre qué hacer en un momento crítico.

El uso de los activos de software se ha convertido en una alternativa muy útil, pues como se ha visto las características que los distinguen brindan la posibilidad de ser almacenados y reutilizados en la producción de futuros sistemas, por ello surgen las interrogantes: ¿Dónde se almacenan los activos de software?

1.3 Repositorios de activos de software

Los repositorios de software surgen por la necesidad de almacenar y organizar los activos de software o componentes de software de forma tal que facilite su uso en el desarrollo de productos de software y existen varios conceptos relacionados con el tema.

Según la conferencia Desarrollo de Software Basado en Líneas de Productos de Software: “Un repositorio es una base de datos especializada que almacena activos de software y facilita la recuperación y el mantenimiento de los activos de software. Su objetivo es asegurar la disponibilidad de activos para apoyar el desarrollo de productos de la LPS”. [1]

En la tesis “La reutilización como parte de la calidad del software” realizada en el 2007 se define el siguiente concepto de repositorio de activo de software: “Un repositorio es una base de datos simple que

facilita el almacenamiento de los componentes de software reutilizables, que constituyen elementos centrales para el soporte operativo de la reutilización, y permiten además realizar búsquedas, gestionar los cambios y garantizar la calidad, aunque no es lo suficientemente seguro y confiable para soportar todas las responsabilidades que impone el desarrollo de software orientados a la reutilización". [7]

En la tesis "La reutilización como parte de la calidad del software" se exponen razones por las cuales es de vital importancia concebir un repositorio de componentes de software:

- Proporcionar un soporte para la información que se genera en los proyectos.
- Garantizar el acceso de todo el equipo de desarrollo y mantenimiento a los elementos reutilizables que lo componen.
- Soportar la composición de nuevos productos. [7]

Según los conceptos analizados se puede afirmar que los repositorios de activos de software deben ocuparse del almacenamiento, recuperación y mantenimiento de los activos de software de una forma segura y confiable. Además los repositorios de activos de software deben estar disponibles para múltiples proyectos que apliquen la reutilización de activos y de esta forma, compartir los activos de software incrementando su número y las funcionalidades de los sistemas de software que realizan. Casi todos los centros y proyectos de la universidad poseen bases de datos donde son almacenados sus expedientes de proyectos y otras herramientas, pero ¿Estarán preparadas las base de datos para gestionar los activos de software de la forma requerida?

1.4 Reutilización de software

La reutilización de software tiene como objetivo fundamental la producción de software a partir de software realizado de antemano y se ha convertido en una alternativa, que provee a las empresas de una forma más rápida y eficiente de producir software.

El experto M. Griss explica que "para que la reutilización del software ofrezca los beneficios que de ella se esperan, se debe producir un acercamiento sistemático e institucional en su implantación". [8]

El primer concepto de reutilización de software surge en 1968 por Dough McIlroy y la idea principal en aquel entonces era producir componentes de software como si de componentes electrónicos se tratara. El objetivo era reutilizar lo existente sin tener que volver a rediseñarlo desde el principio.

Según Ian Sommerville: “la reutilización es un enfoque de desarrollo [de software] que trata de maximizar el uso recurrente de componentes de software existentes”. [6]

El experto P. Freeman afirma que “la reutilización es cualquier procedimiento que produce o ayuda a producir un sistema mediante el nuevo uso de algún elemento procedente de un esfuerzo de desarrollo anterior”. [9]

Estudios realizados han demostrado la efectividad de la reutilización de software en particular Sametinger describe algunos de estos indicadores:

“Entre el 40 y 60% del código fuente de una aplicación es reutilizable en otra similar.

Aproximadamente el 60% del diseño y del código de aplicaciones administrativas es reutilizable.

Aproximadamente el 75% de las funciones son comunes a más de un programa.

Sólo el 15% del código encontrado en muchos sistemas es único y novedoso a una aplicación específica. El rango general de uso recurrente potencial está entre el 15% y el 85%”. [10]

La reutilización de software según los conceptos analizados ofrece incrementar las ganancias de las empresas productoras de software optimizando el desarrollo de producción, pero para lograr la eficiencia es necesario mantener un proceso organizado y controlado. Es necesario además crear el conjunto de activos de software a reutilizar y con estos su respectivo repositorio.

1.5 ¿Qué es un modelo?

Se realizará a continuación un estudio sobre los diferentes conceptos de modelos y luego se seleccionará el modelo más relevante a esta investigación y se realizará un análisis sobre este.

Un modelo según la Real Academia Española: “es un arquetipo o punto de referencia para imitarlo o reproducirlo. En las acciones morales y en las obras de ingenio, un modelo es un ejemplar que se debe seguir e imitar por su perfección”. [11]

1.5.1 Modelo Científico

Un modelo científico según la Real Academia Española: “es el resultado del proceso de generar una representación abstracta, conceptual, gráfica o visual para analizar, describir, explicar, simular y predecir fenómenos o procesos”. [11]

1.5.2 Modelo de Calidad

Un modelo de calidad según Manuel Bertoa y Antonio Vallecillo: “es el conjunto de características y sub-características, y de cómo estas se relacionan entre sí”. [12]

1.5.3 Modelo de Procesos

Un modelo de procesos según Jonás A Montilva: “es una representación del ciclo de vida del software que describe los procesos requeridos para desarrollar y/o mantener software además se puede considerar como un conjunto estructurado de actividades diseñado para alcanzar un objetivo establecido”. [12]

Luego de realizar un análisis de los diferentes conceptos de modelos se puede determinar que el modelo que se seleccionará para esta investigación es el modelo científico, debido a que permitirá realizar una representación conceptual, gráfica o visual que permitirá explicar el fenómeno que se analizará. Pero además se tomará en cuenta el modelo de procesos pues se definirán algunas actividades que deben realizarse para cumplir con el modelo que se propone.

1.6 Desarrollo de software basado en componentes

La idea básica del desarrollo de software basado en componentes es la implementación de sistemas utilizando componentes previamente elaborados y probados. Estos componentes se desarrollan mediante

la programación orientada a componentes y tienen como característica que son independientes pero pueden ser integrados con otros.

En el informe técnico de la universidad de Salamanca realizado en el 2002 se plantea que: “ofrece a las organizaciones una forma de desarrollo de soluciones de software, a escala de empresa, flexibles y con facilidad de incluir nuevas demandas de una forma eficiente en cuanto al coste y el tiempo”. [14]

1.7 Líneas de Productos de Software

Las LPS es una nueva técnica de ingeniería de software que se pretende adoptar en la UCI, se realizará un análisis de los principales conceptos relacionados con esta.

Actualmente en el mundo existen diversas definiciones de LPS se asumirá la expresada por Krueger: "...se refieren a técnicas de ingeniería para crear un portafolio de sistemas de software similares, a partir de un conjunto compartido de activos de software, usando un medio común de producción". [15]

Se tendrá en cuenta además el concepto de Clements y Northorp: "... es un conjunto de sistemas de software que comparten un conjunto común y gestionado de aspectos que satisfacen las necesidades específicas de un segmento de mercado o misión y que son desarrollados a partir de un conjunto común de activos fundamentales [de software] de una manera prescrita". [16]

Las Líneas de Productos de Software pertenecen a la segunda clase de reutilización y como plantea J. Bosch: “es donde los componentes son utilizados en una serie de versiones de productos y en una familia de productos que contienen funcionalidad relacionada, pero no idéntica”. [17]

Se puede concluir de estos conceptos que la idea básica de LPS es la producción de sistemas con un conjunto de funcionalidades comunes y algunas diferentes, también que los sistemas serán desarrollados a partir de activos de software previamente elaborados y almacenados en repositorios de activos de software. Las LPS brindan muchos beneficios, pero la adopción de estas de realizarse de una forma muy organizada para poder obtener las ganancias esperadas.

1.8 Procesos básicos de una LPS

Un proceso según Jonás A Montilva: “es un conjunto estructurado de actividades diseñado para alcanzar un objetivo establecido”. [13]

En la figura 1 se presentan los procesos de negocio de una LPS según Jonás A Montilva:

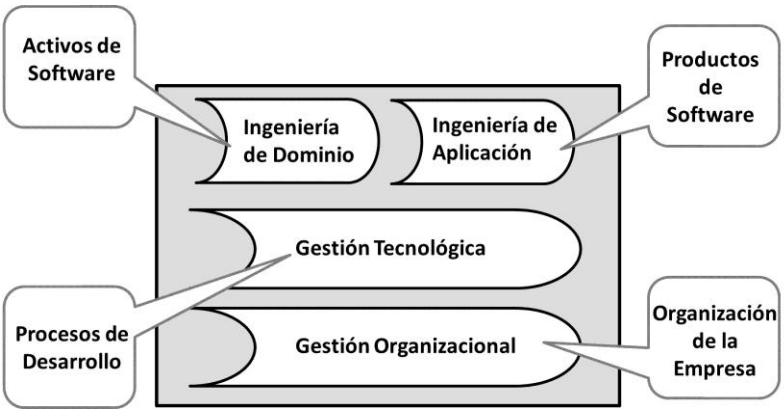

Figura 1: Procesos del negocio de una LPS. (1)

- Ingeniería de Dominio (ID): Captura información y representa el conocimiento sobre un dominio determinado, con el fin de crear activos de software reutilizables en el desarrollo de cualquier nuevo producto de una LPS.
- Ingeniería de Aplicaciones (IA): Se encarga del desarrollo de los productos de la LPS a través de la reutilización de activos de software y de los planes de producción.

Los procesos de una LPS de forma básica se dividen en el desarrollo de activos de software para la LPS, la producción de aplicaciones utilizando los activos almacenados en los repositorios, la gestión de los procesos de desarrollos tanto de los activos como de los productos y la gestión organizacional de la LPS.

1.9 Modelos de procesos para LPS

Según Jonás A Montilva los modelos de procesos para las Líneas de Productos de Software son:

- El Modelo TWIN
- El método WATCH

- El modelo ESPLEP
- El modelo del Software Engineering Institute (SEI)

Modelo TWIN extendido:

Este modelo es empleado en el desarrollo de software basado en componentes se muestra en la figura 3:

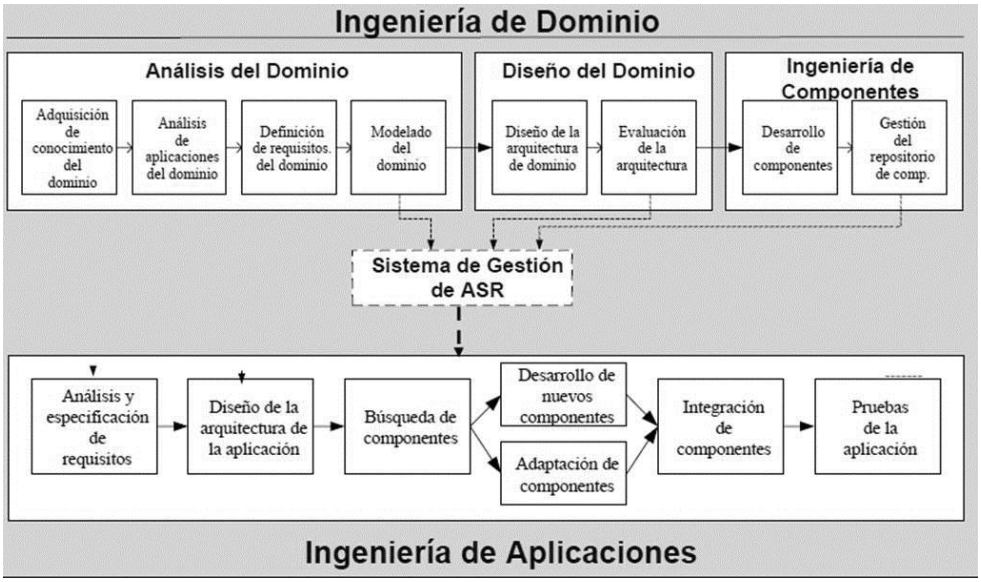

Figura 3: Modelo TWIN extendido. [1]

Método WATCH:

El método WATCH fue propuesto por la Universidad de los Andes (Venezuela) para el desarrollo de aplicaciones empresariales. Este consta de dos componentes metodológicos que pueden observarse en la siguiente figura 4:

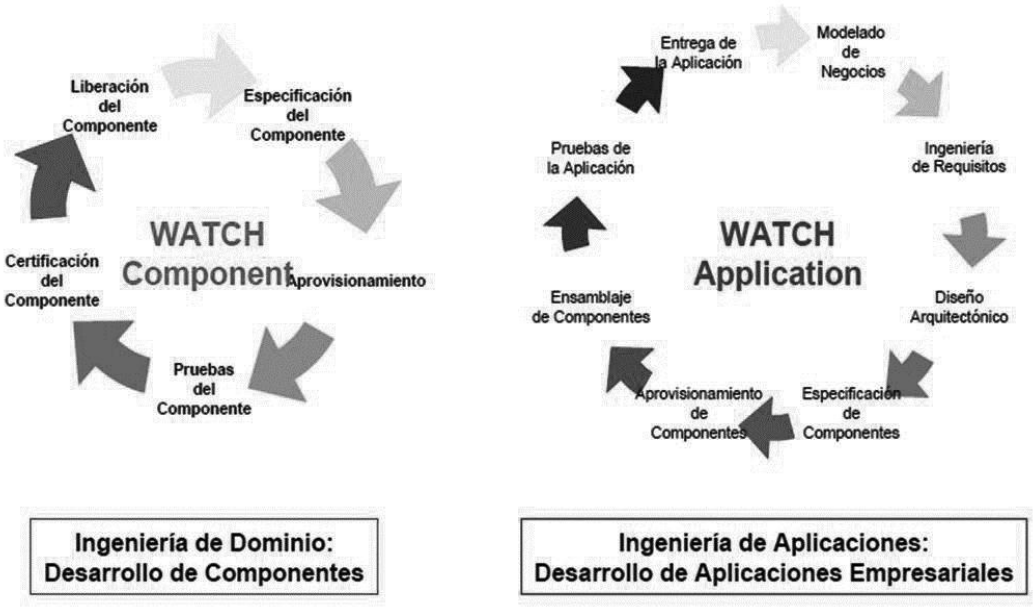

Figura 4: Método WATCH. [1]

Modelo ESPLEP:

El modelo ESPLEP (Evolutionary Software Product Line Engineering Process) puede observarse en la figura 5:

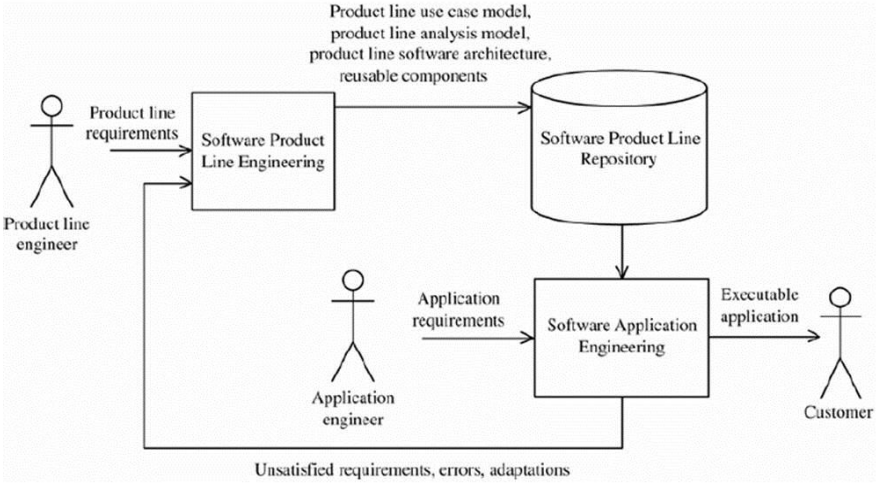

Figura 5: Modelo ESPLEP. [1]

Modelo del Software Engineering Institute

Este modelo de procesos para LPS fue desarrollado por el Software Engineering Institute (SEI)

Como se puede observar en la figura 6 el modelo cuenta con tres procesos fundamentales:

- Desarrollo de activos de software.
- Desarrollo de productos de software.
- Gestión de la línea de productos.

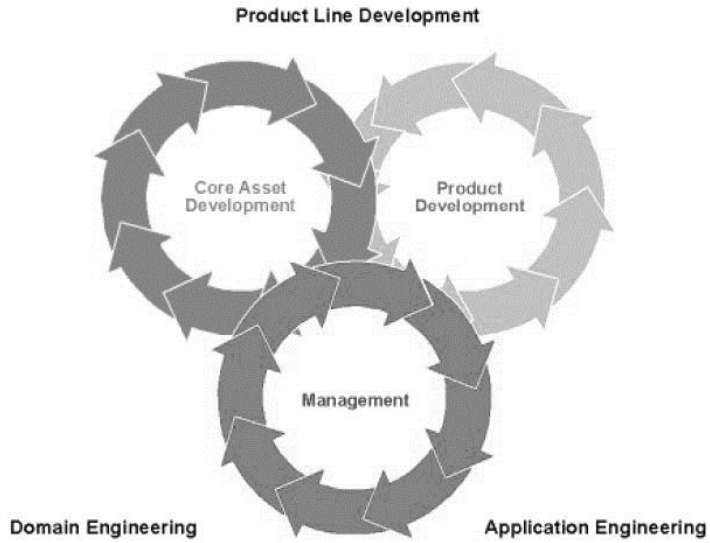

Figura 6: Modelo del SEI. [1]

Desarrollo de Activos de software (Ingeniería de Dominios)

En este proceso se establece la capacidad de producción para los productos mediante el desarrollo de activos de software reutilizables los cual se muestra en la figura 7.

Tiene como salidas:

- Alcance de la línea
- Activos de software
- Plan de producción

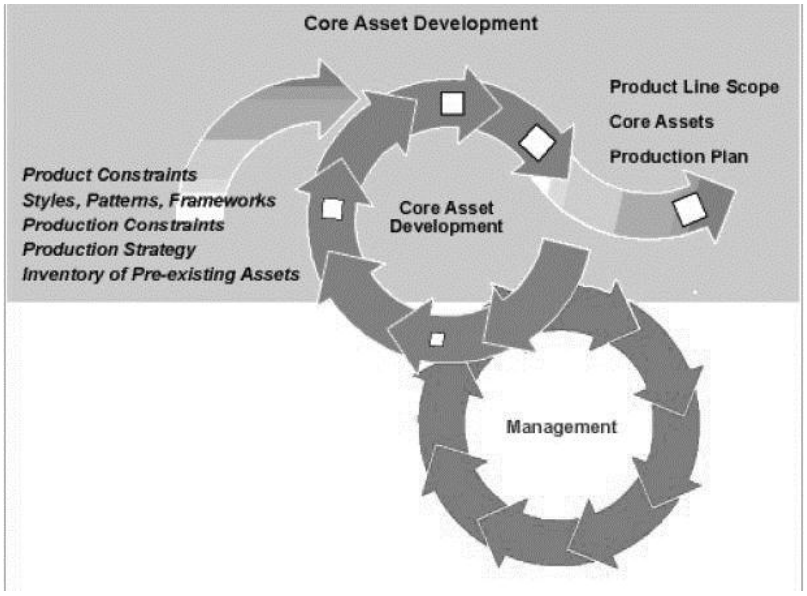

Figura 7: Desarrollo de Activos de Software. [1]

Desarrollo de productos (Ingeniería de Aplicaciones)

Este proceso se ocupa al ensamblaje de los activos de software siguiendo el plan de producción para desarrollar los productos de software el mismo se muestra en la figura 8.

Tiene como salida los productos de software de la línea

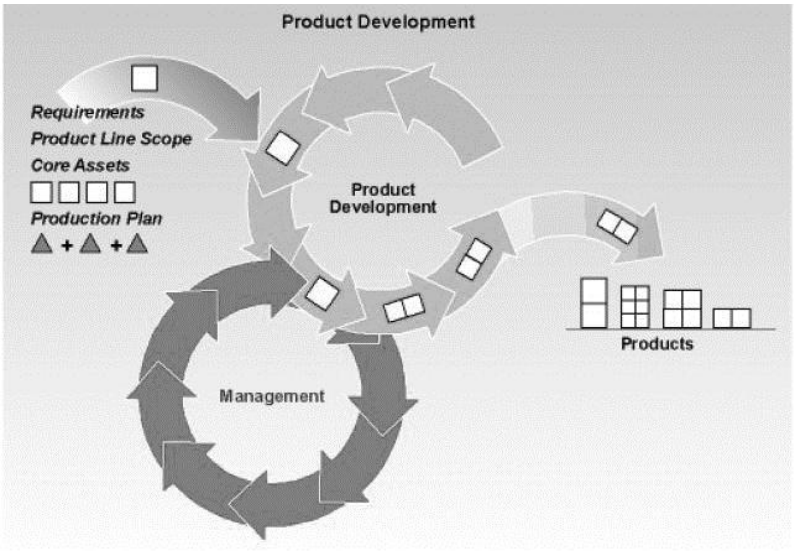

Figura 8: Desarrollo de Productos. [1]

Gestión de la línea de productos

Este proceso se encarga de proporcionar los recursos y de supervisar y coordinar el desarrollo de activos de software y productos de software.

Está dividido en:

- Gestión técnica: Está orientada a los grupos que desarrollan los activos y los productos de software.
- Gestión Organizacional: Está orientada a los aspectos organizacionales como estructura, recursos, relaciones, financiamiento, entre otros.

El modelo del SEI es de todos los modelos de procesos para LPS el que más se adecúa a las necesidades de la UCI, esto se debe a que maneja el desarrollo de activos y de productos de software de forma separada pero siempre manteniendo el vínculo entre ambos, esto brinda la posibilidad de que los

activos puedan ser desarrollados y mantenidos de forma independiente, lo cual hace que el proceso de producción de software con reutilización no sea afectado y se optimice.

En este modelo además se maneja la producción de forma que puedan ser reutilizados todos los activos de software no solo se reutilizan componentes de software. Se maneja de manera paralela la gestión organizacional de ambos procesos lo que facilita la organización dentro de las LPS.

1.10 Modelos de activos de software

La producción basada en Líneas de Productos de Software es una idea surgida hace poco tiempo y aún no se consolidan muchos criterios acerca de qué o cómo se realizarán muchos procesos necesarios dentro de la producción de software basada en componentes. Muchas empresas han optado por experimentar sus propios modelos y procesos, y de sus resultados obtenidos se retroalimenta el mundo.

En un caso muy similar se encuentran los modelos de activos de software como el que se propone en esta investigación, dado que la información de cómo se identifican los activos de software y los repositorios de éstos se encuentra dispersa por todo el mundo y en realidad no es explícita.

Es por ello que la presente investigación pretende realizar un modelo para agrupar las características necesarias para identificar los activos de software reutilizables y los repositorios asociados a éstos de manera que facilite el desarrollo de software.

1.11 Modelo de activos de software propuesto

Anteriormente se explicaba la difícil tarea que resultaba diferenciar los activos de software de los artefactos que comúnmente se generaban en el desarrollo de sistemas informáticos y se puntualizó que la única manera de hacerlo consistía en investigar detalladamente el diseño que debía poseer un activo de software para ser reutilizable. Se planteó también una interrogante acerca de la capacidad de las bases de datos de nuestra universidad de poder gestionar los activos de software que serán utilizados en el proceso de desarrollo de software con reutilización. La respuesta más acertada sería realizar una investigación sobre qué elementos debe poseer una base de datos para darle utilidad en una LPS.

El modelo que se propone en esta investigación tiene como objetivo facilitar el reconocimiento de los activos de software dentro de una aplicación informática y la identificación de un repositorio que presente las cualidades requeridas para el almacenamiento de los activos de software. Con este fin se realizará una representación gráfica y conceptual de los elementos que intervienen en este proceso. Se puntualizarán en este modelo las actividades que deben cumplir los responsables designados para lograr una acertada clasificación.

1.12 Empresas que aplican LPS

El modelo LPS actualmente es utilizado en numerosas empresas que se han convertido y son, reconocidas por sus grandes logros y ganancias en la rama de las tecnologías a la que se dedican. Se presentarán a continuación las más reconocidas en el mundo entero:

- Nokia.
- Philips (Software para aparatos de televisión).
- Philips (Sistema de telecomunicaciones de conmutación).
- Toshiba.

1.13 Beneficios de aplicar las LPS

Beneficios tácticos y estratégicos planteados por Krueger:

- Reducción en el tiempo promedio de creación y entrega de nuevos productos.
- Reducción en el número promedio de defectos por producto.
- Reducción en el esfuerzo promedio requerido para desarrollar y mantener los productos.
- Reducción en el costo promedio de producción de los productos.
- Incremento del número total de productos que pueden ser efectivamente desplegados y mantenidos. [15]

Conclusiones del capítulo

En el presente capítulo se realizó un análisis de los conceptos fundamentales relacionados con la reutilización de activos de software. El mismo permitió comprender que los artefactos de un software que son diseñados expresamente para ser reutilizados en el desarrollo de múltiples aplicaciones informáticas se determinan activos de software reutilizables. Los activos de software necesitan además para poder ser útiles en la composición de sistemas estar almacenados en repositorios especializados.

La reutilización de los activos de software constituye la base de las LPS, estas utilizan un conjunto de activos almacenados en repositorios para construir familias de software con un conjunto de características comunes y algunas diferencias. Se realizó además un estudio sobre los modelos de procesos de las LPS seleccionándose el modelo propuesto por el SEI por estar acorde a las necesidades de la UCI, debido a que primeramente en el proceso de producción pueden ser reutilizados todo tipo de activos de software no como en otros casos que solo pueden aplicarse a componentes de software.

Además la producción de activos de software y de productos se gestionan de forma independiente pero manteniendo el vínculo entre ambos procesos, lo que facilita la gestión de los activos de software sin perjudicar el desarrollo de productos.

Se mostraron todos los beneficios que brindan las LPS en la producción de software y las principales empresas que lo aplican en la actualidad. Se definió además el modelo de activo de software a desarrollar el cual tiene como objetivo lograr una correcta clasificación de los activos de software dentro de un sistema y de los repositorios de estos.

Capítulo 2: Desarrollo del modelo de activos de software reutilizables

Introducción

En este capítulo se presentará la estructura del modelo de activos de software propuesto. Se definirán las características que deben cumplir tanto los activos de software para ser reutilizables, como los repositorios para poder almacenar a dichos activos. Quedará puntualizado además que roles se responsabilizarán por el cumplimiento de tales características.

2.1 Estructura del modelo de activos de software

En el presente capítulo se presentará el desarrollo del modelo de activos de software reutilizables pero primeramente debe presentarse la estructura del mismo de una forma gráfica y textual con el fin de lograr que los usuarios entiendan perfectamente cómo utilizarlo:

La estructura del modelo de activos de software reutilizable se presentará en la figura 10:

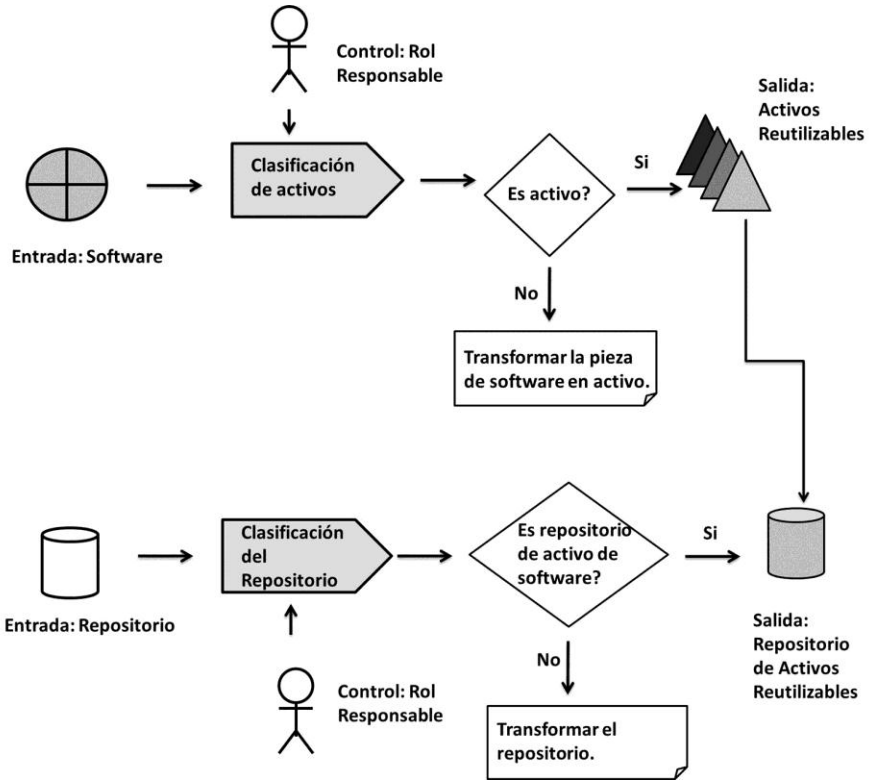

Fig 9: Modelo de activos de software reutilizables.

El modelo que se propone en la presente investigación tiene como entrada los softwares que serán desmantelados para ser convertidos en activos reutilizables. Se describirán las características que deben cumplir los activos de software reutilizables y se definirán los responsables que controlarán el cumplimiento de dichas características.

Al mismo tiempo se identificará además el almacén de datos existente como repositorio de activos de software reutilizables, lo cual se garantizará si éste cumple con las características para el almacenamiento y recuperación de dichos activos. Las características que se proponen serán verificadas por los responsables designados.

2.1.1 Estructura de la clasificación de los activos de software reutilizables

- Entrada: Software.

Primeramente se realizará la elección del software a desmantelar, el mismo será descompuesto en piezas funcionales para clasificarlo como activo de software reutilizable.

- Clasificación de activos de software reutilizables. (Ver sub-epígrafe 2.2.2)

Se presentarán las características necesarias para clasificar una pieza de software como activo de software reutilizable y las relaciones que se establecen entre estas.

- Control: Rol Responsable. (Ver sub-epígrafe 2.2.3)

Se definirá que roles deben asumir responsabilidades sobre la validación de las características propuestas.

Si el artefacto cumple con todas las características propuestas:

- Salida: Activos de software reutilizables.

Luego de realizar la clasificación se obtendrán los activos de software reutilizables listos para ser almacenados en los repositorios correspondientes. En la figura 11 se presentarán los elementos que componen la clasificación de activos de software reutilizables.

Si el artefacto no cumple con todas las características propuestas:

- Salida: Transformar el activo de software.

Sería una tarea del equipo de desarrollo valorar la transformación del artefacto a activo de software.

2.1.2 Estructura de la clasificación de los repositorios de activos de software reutilizables

- Entrada: Repositorio.

Se seleccionará el almacén de datos a validar como repositorio de activos de software reutilizable.

- Clasificación de Repositorios. (Ver sub-epígrafe 2.2.4)

Se presentarán las características fundamentales que debe cumplir un repositorio en cuanto a almacenamiento y recuperación para que puedan utilizarse los activos de software reutilizables en una Línea de Productos de Software.

- Control: Rol Responsable. (Ver sub-epígrafe 2.2.5)

Se indicará que roles son los responsables de la clasificación del repositorio seleccionado.

Si el repositorio cumple con todas las características propuestas:

- Salida: Repositorio de activos de software reutilizables.

Luego de validar que el repositorio cumpla con las características indicadas el repositorio será capaz de almacenar los activos de software reutilizables. En la figura 12 se muestran los elementos que componen la clasificación del repositorio de activos de software.

Si el repositorio no cumple con todas las características propuestas:

- Salida: Transformas el repositorio.

Le corresponde al equipo de desarrollo valorar las ventajas de convertir el repositorio existente en el proyecto por un repositorio de activos de software.

2.2 Modelo de activos de software reutilizables

En este sub-epígrafe se describirán tanto las características que deben cumplir los artefactos para ser considerados activos de software como las características que deben cumplir los almacenes de datos para ser clasificados como repositorios de activos de software.

2.2.1 Objetivos

- Clasificar los artefactos de software de un proyecto en activos de software, para facilitar su reutilización en la producción de software.
- Clasificar los repositorios comunes en repositorios de activos de software para facilitar el almacenamiento y recuperación de activos de software.

2.2.2 Clasificación de los activos de software reutilizables

Los activos de software reutilizables son diseñados expresamente para ser reutilizados en la producción de otros sistemas, también pueden ser resultado del proceso de producción de un software.

En los sistemas informáticos pueden ser clasificados numerosos artefactos como activos de software reutilizables, se ha realizado una selección de los más significativos para el desarrollo de esta investigación:

- Componentes
- Arquitectura
- Especificaciones de requisitos
- Código
- Patrones de diseño
- Documentación

2.2.2.1 Componentes de software reutilizables

Los componentes de software que son resultado de la producción interna, no son inmediatamente reutilizables, debido a que estos poseen características específicas del software y de su interfaz, esto los

hace poco reutilizables en otras aplicaciones. Debido a esto, se hace necesario adaptarlos y hacerlos más genéricos con el fin de incrementar su reutilización.

Granularidad de los componentes de software reutilizables.

La granularidad de los componentes es el grado en que se descomponen los sistemas de software. Generalmente los sistemas de grano grueso son los que implementan un mayor número de funciones esto trae como consecuencia que tengan menos dependencias de otros componentes por lo que se hace muy fácil de utilizar pero poco reutilizables.

Por el contrario los componentes de grano fino implementan normalmente una sola funcionalidad lo que los hace depender de otros componentes y resultan más difíciles de usar que los de grano grueso, pero resultan mucho más reutilizables.

Peso de los componentes de software reutilizables

El peso de los componentes de software reutilizables es el grado de dependencia de estos a un entorno operativo. Los componentes pesados son aquellos que dependen completamente de un sistema operativo lo cual los hace muy fáciles de usar pero no tienen grandes posibilidades de ser reutilizados.

Por otro lado los componentes ligeros evitan toda dependencia de un sistema operativo, esta cualidad los hace muy reutilizables pero resultan muy difíciles de utilizar debido a que necesitan ser configurados para cada entorno.

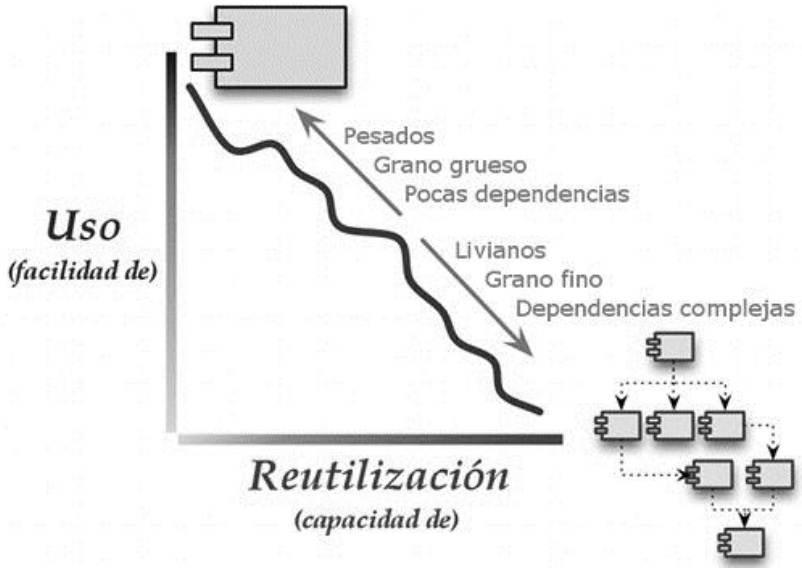

Figura 10: Granularidad y Peso en los componentes contra uso y reutilización.

Características de un componente de software reutilizable

Existen muchos autores que mencionan las características fundamentales que debe tener un activo de software o componente para que sea considerado reutilizable, como se muestra en la tabla 1, además también plantean cualidades que favorecen su posible reutilización en una empresa.

Tabla 1: Características que deben poseer los componentes de software, según diferentes autores.

Autores	**(Maribel Ariza Rojas, 2004)**	**(Szyperski, 2002)**	**(Sommerville, 2005)**	**(Montilva, 2006)**	**(Desarrollo de Software Basado en Componentes, 2003)**
	Identificable			Identificable	Identificable

CARACTERÍSTICAS	Auto contenido	Dependencias de contexto explícitas únicamente.	Independiente	Auto contenido	Auto contenido
	Remplazado por otros componentes			Reemplazable por otro componente	
	Acceso solamente a través de su interfaz		Componible	Accesible solamente a través de su interfaz	Accesible sólo a través de su interfaz
	Servicios invariantes			Inmutabilidad de sus servicios	Sus servicios son invariantes
	Bien Documentado		Documentado	Documentación de sus servicios	Documentado
	Es genérico				Genérico
	Reutilizado dinámicamente				Reutilizado dinámicamente
		Sujeto a la composición por terceras partes.			

			Estandarizado		
		Desplegado de forma independiente.	Desplegable		
				Rastreable a través de su ciclo de desarrollo	
				Mantenido sistemáticamente	Mantenido
	Independiente de la plataforma		Independiente de plataforma		Independiente de la plataforma
		Interfaces especificadas			
					Certificado
					Accedido sin importar su localidad

- Nombres genéricos: el componente no debe poseer nombres específicos del sistema.

- Métodos con manejo de excepciones consistente: todos los métodos deben manejar las excepciones consistentemente.
- Métodos deben ser genéricos: los métodos deben estar generalizados es decir no debe ser específico del sistema.
- Interfaz adaptable: el componente debe poseer una interfaz de “configuración” que lo haga adaptable a diferentes situaciones de uso.
- Independiente: los componentes requeridos deben estar integrados para incrementar su independencia.
- Desplegado de forma independiente: debe ser independiente y funcionar como una unidad autónoma.
- Mantenido: es deseable que los componentes de software estén inmersos en un proceso de mejoramiento continuo que le garantice al integrador nuevas versiones que incluyan correctivos, optimizaciones y nuevas características. Esto contribuye a que dicho componente sea seleccionado con mayor frecuencia para formar parte de sistemas de software. [4]
- Documentado: un componente debe tener una documentación adecuada que facilite su búsqueda dentro de repositorios de componentes, evaluación, adaptación a nuevos entornos, integración con otros componentes y acceso a información de soporte. [4]
- Identificable: deben tener una identificación que permita acceder fácilmente a sus servicios y que permita su clasificación. [18]
- Auto contenido: un componente no debe necesitar de otros para realizar la función para la cual fue diseñado, de ser así debe especificarse en su interfaz.
- Accesible solamente a través de su interfaz: el componente debe mostrar al usuario únicamente el conjunto de operaciones que lo caracteriza. Esta característica permite que el componente sea remplazado por otro que implemente la misma interfaz. Esta definición trae implícita la

característica de reemplazable por lo tanto no se tomará como necesaria, si el componente cumple que es accesible solo por la interfaz entonces puede ser reemplazado. [4]

- Servicios Invariantes: las operaciones que ofrece un componente a través de su interfaz no deben variar. La implementación de estos servicios puede ser modificada pero no debe afectar la interfaz. [4]

Si finalmente la pieza de software no cumple con alguna de estas características no puede ser considerado como componente de software reutilizable, puede ser transformado validando que cumpla con las características antes descritas.

Responsables de la identificación de los componentes de software reutilizables

- Analista del sistema.
- Revisor técnico o documentador.
- Arquitecto de software.
- Ingeniero de componentes o implementador.
- Líder de proyecto.

2.2.2.2 Patrones de diseño

Un patrón de diseño no es más que la descripción de un problema y su solución, de forma que esta solución pueda ser reutilizada en diferentes situaciones.

¿Por qué re-usar patrones?

- Es una buena forma de compartir las buenas prácticas.
- Ofrece un vocabulario común entre diseñadores.

- Permite re-usar diseño no solo código.
- Facilitan el aprendizaje al diseñador inexperto.
- Ayuda a documentar el diseño.

Características que deben poseer los patrones de diseño para ser considerados activos de software reutilizables

Los cuatro elementos que deben poseer los patrones para incrementar su reusabilidad y facilitar su identificación, los describe Gamma como:

- Nombre: es una referencia significativa del patrón.
- Descripción del área del problema: explica cuándo puede aplicarse el patrón.
- Descripción de las partes de la solución del diseño, sus relaciones y sus responsabilidades: no es una descripción detallada del diseño, es una plantilla para una solución de diseño que puede instanciarse de diferentes formas. Esta generalmente muestra las relaciones entre los objetos y las clases de los objetos en la solución.
- Declaración de las consecuencias, los resultados y compromisos de aplicar el patrón: esto servirá de ayuda a los diseñadores para saber cuándo un patrón podrá ser aplicado de forma efectiva en una situación particular. [19]

El uso de estos patrones es una forma eficiente de reutilización, pero se puede afirmar, que sólo los ingenieros más experimentados y con amplios conocimientos de patrones pueden utilizarlos de forma eficaz, estos son capaces de reconocer situaciones en las que se puede utilizar un patrón. Los programadores sin experiencia aún cuando hayan estudiado bibliografía referente a los patrones y su uso les resultará muy difícil tomar una decisión entre re-usar un patrón o si necesitan desarrollar una solución específica.

Responsables de la clasificación de los patrones de diseño

- Arquitecto de software.

2.2.2.3 Especificación de Requisitos

La identificación de requisitos es una actividad consumidora de recursos y expuesta a errores, es por ello que se considera un factor crítico en el proceso de desarrollo. Cuando se poseen elementos reutilizables de requisitos, almacenados y clasificados en un repositorio se reducen grandemente las tazas de errores en los mismos, además de aprovecharse eficientemente los recursos del desarrollo.

El experto Creel ha definido patrones de requisitos a un nivel de abstracción más alto, se proponen los tres siguientes:

- **Patrón Especificar:** el objetivo de este patrón es aconsejar cómo se puede seleccionar (especificar) una determinada información (para modificarla, eliminarla o consultarla) en un requisito separado y hacer referencia a dicho requisito cuando sea necesario.

  Adaptando el patrón podría formularse para que aconsejara la forma en que un actor selecciona o identifica una determinada información en los casos de uso en que fuera necesario, sino sacar factor común en un caso de uso abstracto e incluirlo en los casos de uso que fuera necesario.

- **Patrón Presentación:** este patrón recomienda limitarse a indicar qué datos debe solicitar o presentar el sistema sin entrar en detalles concretos de interfaz de usuario.

- **Patrón Priorizar:** este patrón sugiere que en caso de que el usuario desee ordenar la información que brinda el sistema, se separen las formas de ordenar en un requisito a parte y luego se referencien desde los requisitos donde sea necesario, muy similar al patrón Especificar. [20]

El uso de estos patrones conlleva a riesgos potenciales tales como intentar forzar la realidad del problema para que los requisitos encajen dentro los patrones identificados; esto es un gran problema de la reutilización en cualquier fase de desarrollo de software.

Estos riesgos deben ser conocidos por los ingenieros de requisitos y consultar primeramente con los clientes y usuarios la posibilidad de ajustar los requisitos a los patrones para reducir los costes de desarrollo.

Las actividades generales a desarrollar para introducir la reutilización en el proceso de ingeniería de requisitos han sido propuestas por W. Lam las cuales han influido en los resultados de esta investigación:

- Identificar familias de sistemas en los que los requisitos suelan coincidir.
- Desarrollar requisitos parametrizables abstractos.
- Separar los aspectos específicos de los generales.
- Intentar identificar patrones de requisitos al trabajar en dominios específicos.
- Intentar reutilizar también los procesos de obtención de ciertos tipos de requisitos, es decir las preguntas a realizar a los clientes y usuarios, las consideraciones a tener en cuenta a la hora de especificarlos. [21]

Características que debe poseer la especificación de requisitos para ser considerada activo de software reutilizable

- No deben existir referencias específicas tanto en aspectos de la implementación como de la interfaz de usuario. Por ejemplo: Si se está desarrollando el sistema “Y”, evitar las frases como “el sistema Y deberá…” mejor utilizar “el sistema deberá…” con esto se lograría reutilizar este requisito en sistemas similares.
- No deben utilizarse términos específicos dentro de un dominio de problemas. Estos elementos pueden pasarse a los glosarios de términos de sus respectivos proyectos.
- Deben separarse las partes de los requisitos generales y reutilizables de las específicas en uno o más requisitos aparte.

Responsables de la clasificación de la especificación de requisitos

- Analista del sistema.

2.2.2.4 Código Fuente

La reutilización de software surge de la idea de programar un software de tal manera que posteriormente pueda ser reutilizada su implementación en otro software similar. La reutilización de código de programación es una técnica que tiene como objetivo ahorrar tiempo y energía al reducir el trabajo redundante.

Características que debe poseer el código fuente para ser considerado un activo de software reutilizable

- Código del sistema debe ser genérico: el código debe realizarse lo más genérico posible para poder ser reutilizado, aunque debe generalizarse solo lo que va a ser reutilizado debido a que la mayoría de las veces se pierde tiempo con código que más adelante no se utiliza. Se propone realizar un estudio sobre el código de las aplicaciones del dominio para determinar cual tendrá más utilidad en futuras aplicaciones.
- Usable: el código seleccionado para reutilizar debe ser sencillo y fácil de usar para asegurar su entendimiento por todos los programadores.
- Documentado: el código debe estar bien documentado para hacerlo comprensible y facilitar su uso.
- Debe asegurársele mantenimiento y capacitación.
- El código debe ser robusto.
- El código debe estar empaquetado.

Responsables de la clasificación del código fuente

- Ingeniero de componentes o implementador.

2.2.2.5 Arquitectura de Software

Los sistemas siempre se descomponen en pequeños subsistemas que poseen una sola funcionalidad y brindan algún servicio, estos son identificados en un proceso inicial de diseño donde se establece un

marco para su control y comunicación. Como resultado de este proceso se obtiene la descripción de la arquitectura de software.

Una LPS está basada en la producción de software con una arquitectura común, es por ello que la arquitectura es reutilizada en una misma línea de productos, cada vez que se realiza una modificación en el producto. En estos sistemas que poseen una misma arquitectura la diferencia radica en su funcionalidad.

Características que debe poseer la arquitectura de software para ser considerada un activo de software reutilizable

- Debe ser lo más generalizada posible: la arquitectura del sistema debe rediseñarse haciéndola lo más genérica posible que permita que un sistema sea utilizado múltiples veces y pueda sufrir modificaciones.

Responsables de la clasificación de la arquitectura

- Arquitecto de software.

2.2.2.6 Documentación

En un proceso de desarrollo de software basado en componentes intervienen actividades como diseño de la arquitectura de software, la búsqueda y selección de componentes de software, la validación de componentes, entre otras, estas dependen de la realización de una buena especificación de componentes de software sobre la cual actúa cada una de estas actividades.

Características que debe poseer la documentación de un software para ser considerada un activo de software reutilizable

En la bibliografía de forma general no aparece ninguna especificación de cómo y qué elementos debe contener una especificación de componentes, pero al parecer ya se están consolidando caracterizaciones

de este tipo de activos de software como las planteadas por Han Jun en las cuales se considera que una especificación de componentes de software reutilizable puede contener la siguiente información:

- Información sintáctica de los atributos, las operaciones(o métodos) y los eventos de las interfaces de un componente.
- Información semántica acerca del comportamiento de los atributos, las operaciones(o métodos) y los eventos de las interfaces de un componente.
- Información de protocolos que describe la interoperabilidad del componente con otros componentes.
- Propiedades Extra-Funcionales que describe otro conjunto de características que debe poseer el componente reutilizable como propiedades de seguridad, fiabilidad o rendimiento, entre otras. [22]

Todo el conjunto de información presentada (sintáctica, semántica y de protocolos) es conocida como información funcional de los componentes pues hace referencia a aspectos del funcionamiento de los componentes de software.

Existe otro tipo de información relevante a los componentes y que no tienen que ver precisamente con su funcionamiento como:

- Información de atributos de calidad.
- Información de implementación.
- Información del fabricante del componente (en caso de ser COT).

Responsables de la clasificación de la documentación

- Revisor técnico o documentador.
- Ingeniero de componentes o implementador.

2.2.3 Responsables de la clasificación de los activos de software reutilizables

La tabla 2 contiene a cada uno de los responsables implicados en la clasificación de los activos de software, así como las actividades que realizan.

Tabla 2: Roles y responsabilidades de la clasificación de los activos de software.

Roles	Responsabilidades
Arquitecto	Es el encargado de definir e identificar los patrones de diseños que constituyen activos de software reutilizables, realiza las siguientes tareas: ➢ Definir nombre del patrón. ➢ Describir el área del problema. ➢ Describir las partes de la solución del diseño, sus relaciones y sus responsabilidades. ➢ Declarar las consecuencias, los resultados y compromisos de aplicar el patrón. Es el encargado de identificar la arquitectura de software como activo de software y tiene la tarea siguiente: ➢ Verificar que la arquitectura de software sea genérica.
Revisor técnico o Documentador	Se encargan de la documentación del software es por esto que sus tareas en la identificación y elaboración de la documentación de los componentes son: ➢ Identificar la información sintáctica de los atributos, las operaciones(o métodos) y los eventos de las interfaces de un componente. ➢ Identificar la información semántica acerca del comportamiento de los

	atributos, las operaciones(o métodos) y los eventos de las interfaces de un componente. ➢ Identificar la información de protocolos que describe la interoperabilidad del componente con otros componentes. ➢ Identificar las propiedades Extra-Funcionales como propiedades de seguridad, fiabilidad o rendimiento, entre otras. ➢ Identificar información de atributos de calidad. ➢ Identificar información de implementación. ➢ Identificar información del fabricante del componente (en caso de ser COT).
Ingeniero de componentes o Implementador	Es el encargado de implementar los componentes sus tareas en la identificación del código fuente como activo de software son: ➢ Verificar si existe código específico del sistema. ➢ Evaluar usabilidad del código. ➢ Verificar que el código este documentado. ➢ Verificar que el código esté empaquetado y que sea robusto. También se encargan de tareas en la identificación y elaboración de documentación de componentes estas tareas son: ➢ Identificar la información sintáctica de los atributos, las operaciones(o métodos) y los eventos de las interfaces de un componente. ➢ Identificar la información semántica acerca del comportamiento de los atributos, las operaciones(o métodos) y los eventos de las interfaces de

	un componente. ➢ Identificar la información de protocolos que describe la interoperabilidad del componente con otros componentes. ➢ Identificar información de implementación.
Líder de proyecto	Es el encargado de gestionar los recursos del proyecto por lo tanto será el encargado de realizar la tarea siguiente referente a la identificación del código fuente como activo de software: ➢ Verificar que exista la posibilidad de dar capacitación a los implementadores para que exista comprensión del código de otros programadores y además que exista la posibilidad de dar soporte al código. ➢ Evaluar usabilidad del código.
Analista	Es el encargado de encontrar los actores y los casos de uso, sus tareas en la especificación de requisitos son: ➢ Identificar términos específicos dentro de un dominio de problemas. ➢ Identificar las partes de los requisitos, generales y reutilizables de las específicas y separarlas en uno o más requisitos aparte. ➢ Identificar referencias específicas tanto en términos de la implementación como de la interfaz de usuario. En la identificación de la documentación de los componentes también tiene responsabilidades y sus tareas son: ➢ Identificar las propiedades Extra-Funcionales como propiedades de

	seguridad, fiabilidad o rendimiento, entre otras. ➢ Identificar información del fabricante del componente (en caso de ser COT).
Especificador de Caso de Uso	Es el responsable de describir los caso de uso, sus tareas en la especificación de requisitos son: ➢ Identificar las partes de los requisitos, generales y reutilizables de las específicas y separarlas en uno o más requisitos aparte. ➢ Identificar referencias específicas tanto en términos de la implementación como de la interfaz de usuario.
Asegurador de la Calidad	Tiene como principal responsabilidad asegurar que el componente cumpla con las normas de calidad requeridas, por ello en la identificación de la documentación del componentes realiza la tarea siguiente: ➢ Identificar información de atributos de calidad.

2.2.4 Clasificación del repositorio de activos de software reutilizables

El almacenamiento y recuperación de activos de software emplean mecanismos que están relacionados directamente pues dependen del mismo objeto. La clasificación se encarga de encontrarle una ubicación a los activos dentro de un repositorio y la recuperación debe adecuarse a esa organización para facilitar el uso de estos. En la figura 14 podemos encontrar la relación del almacenamiento y recuperación con el repositorio de activos de software reutilizables.

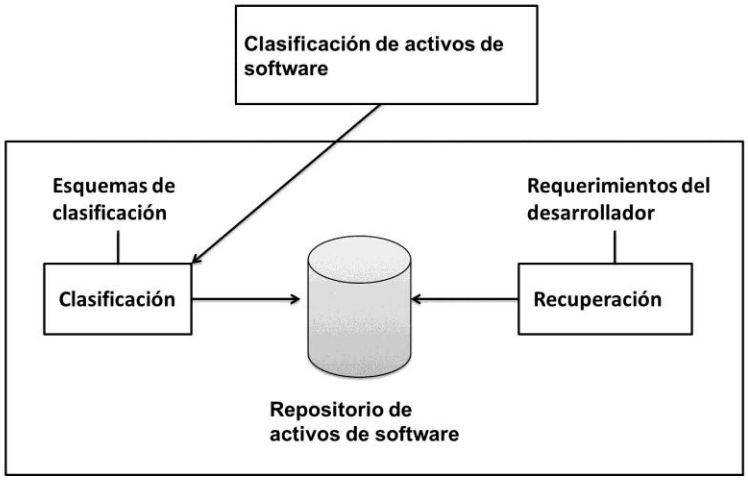

Figura 11: Funciones básicas que se realizan sobre el repositorio de activos de software.

2.2.4.1 Almacenamiento de activos de software

Puede considerarse que el almacenamiento de activos de software es la catalogación o clasificación de estos en un repositorio mediante una traducción directa del activo a una estructura de representación interna.

Se considera que los elementos que no deben faltar en un repositorio para poder realizar el almacenamiento de activos son:

- Esquema de clasificación.
- Estándar de metadatos a seguir.

1. Definición del esquema de clasificación

Una definición de esquema de clasificación puede ser la planteada por José Luis Barros Justo “los esquemas especifican algunos atributos llamados facetas que deben ser usados como descriptores del componente software generalmente con énfasis en la acción ejecutada por el componente y por los objetos manipulados”. [23]

La clasificación se realiza especificando términos claves para cada atributo en el esquema. Las relaciones entre términos se expresan mediante la combinación de atributos, permitiendo una gran precisión.

La aparición de las facetas se debe a R Prieto Diaz su esquema fue adoptado o generalizado por muchas otras propuestas en el área de la reutilización. El esquema consiste en cuatro facetas:

Función realizada por el activo.

- Objetos manipulados por la función.
- Estructura de datos donde la función tiene lugar (medio o localidad).
- Sistema al que pertenece la función. [24]

Otro ejemplo de clasificación por facetas puede encontrarse en Asset Source for Software Engineering Technologic donde se consideran los siguientes:

- Datos específicos.
- Tipo de distribución.
- Tipo de componente.
- Colección.
- Dominio.
- Función. [25]

Otro interesante esquema de clasificación para repositorios orientados a objeto ha sido propuesto por M.R. Girardi y R.T. Pierce donde las facetas representan los siguientes conceptos:

- Dependencias: cuales son las restricciones aplicadas al componente (limitaciones).
- Opera sobre: sobre que objetos actúa el componente (relación, comunicación, etc.).
- Operaciones: que hace el componente (operaciones, funciones, procesos).

- Abstracción: que es el componente (nombre del objeto en un acercamiento OO). [26]

Otro elemento muy importante requerido para el almacenamiento de activos de software es:

2. Definición del estándar de metadatos a seguir

Los metadatos son la información relevante a un objeto determinado como se muestra en la figura 15, con la cual puede identificarse el mismo dentro de un conjunto de objetos semejantes. Los metadatos pueden estar adheridos a los objetos pero esto no resultaría de gran utilidad cuando existen grandes cantidades de objetos pues la búsqueda sería muy difícil dado que es necesario revisar objeto por objeto para encontrar el que se necesita. Por esta razón muchas veces se realiza una copia de los metadatos que son ubicadas en índices de búsquedas para facilitar el proceso.

Existen diferentes estándares de metadatos ejemplo de estos son:

Estándares World Wide Web Consortium (W3C): el World Wide Web Consortium desarrolló el RDF (Resource Description Framework) y las especificaciones para la Plataforma para la Selección de Contenido en Internet (PICS).

- RDF es un modelo de datos y sintaxis para los metadatos de los recursos web dígase recurso a una página web, un sitio web completo y cualquier elemento en la web que contenga información en cualquier formato.

- PICS fue desarrollado para relacionar los metadatos con la información en internet. Al principio de su creación fue altamente utilizado por los padres para controlar el acceso de sus hijos a internet pero actualmente con algunas pequeñas modificaciones puede codificar y transmitir metadatos derivado del estándar Dublin Core.

Dublin Core: este estándar pretende ser un conjunto de elementos básicos requeridos para facilitar la recuperación de objetos. Las propiedades extrínsecas que describen el contexto donde se utiliza el objeto no se utiliza. Dublin Core se concentra en la descripción de las propiedades intrínsecas del objeto tales como:

- Contenido intelectual (título, autor o fuente).

- Forma Física (formato).

Tabla 3: Elementos del estándar Dublin Core.

Nombre	Etiqueta	Descripción
Título	TITLE	Nombre del objeto
Autor	CREATOR	Persona responsable del contenido intelectual del objeto.
Tema y palabras clave	SUBJECT	Tópico tratado por el trabajo
Descripción	DESCRIPTION	Descripción textual del contenido del objeto
Editor	PUBLISHER	Entidad responsable de poner a disposición el objeto en su formato actual
Colaborador	CONTRIBUTOR	Personas tales como editores o correctores que hicieron contribuciones intelectuales significativas al trabajo
Fecha	DATE	Fecha asociada con la creación o publicación del objeto.
Tipo de objeto	TYPE	Categoría del objeto tal como poesía, novela, diccionario o reporte técnico
Formato	FORMAT	Formato de los datos. Se utiliza para identificar el software y posible hardware necesarios para su

		uso.
Identificador	IDENTIFIER	Cadena o número utilizado para identificar el objeto de manera única.
Fuente	SOURCE	Objetos, impresos o electrónicos, a partir de los cuales se derivó este objeto (si aplica)
Idioma	LANGUAJE	Idioma del contenido intelectual
Relación	RELATION	Relación del presente objeto con otros
Cobertura	COVERAGE	Características espaciales y/o temporales del contenido intelectual del objeto
Derechos	RIGHTS	Enunciado de los derechos sobre el objeto

Luego de analizar los diferentes metadatos se proponen los siguientes:

- Nombre.
- Descripción.
- Versión.
- Categoría.
- Autor.
- Relación.

- Lenguaje.
- Formato.
- Fecha.
- Palabras clave.

2.2.4.2 Recuperación de activos de software

Los sistemas gestores de base de datos se basan en la coincidencia total de los ítems de información con los criterios de búsqueda de las consultas realizadas, lo cual para la recuperación de información de los activos resulta difícil debido a que es imposible realizar la consulta precisa y los ítems de información recuperados pueden coincidir total o parcialmente con los criterios de la consulta realizada. Los Sistemas de Recuperación de Información (SRI) permiten la coincidencia total o parcial de los ítems de información lo cual resulta necesario para la recuperación de los activos de software.

1. Sistemas de recuperación de información

- Su principal función consiste en satisfacer la necesidad de información.
- La necesidad de información es transmitida al SRI en forma de consulta.
- El SRI responderá la solicitud con una lista de ítems del repositorio.

Ítems: Se refiere a los productos de desarrollo de software almacenados en los repositorios.

Principales métodos de recuperación de información

Indización

El almacenamiento de activos de software consiste en el proceso de clasificación que es conocido también como mapeo. La clasificación o mapeo se ocupa de catalogar a los diferentes activos en un repositorio a través de una traducción directa del activo a una estructura de representación interna por la cual el activo podrá ser recuperado en el futuro.

La representación interna consiste en un conjunto de términos que describen al activo, teniendo en cuenta las categorías preestablecidas en el esquema de clasificación o en el lenguaje de representación.

Para la asignación de términos o también llamada indización puede utilizarse un lenguaje libre o un lenguaje controlado (lenguaje específico del dominio).El vocabulario controlado determina los términos que pueden utilizarse en el proceso de indización y su significado, es algo así como un glosario de términos.

En la indización se utilizan dos tipos de métodos:

- Método estadístico: se basan en la frecuencia de ocurrencia de las palabras en un texto, las palabras que más se repiten no son tomadas en cuenta pues estas no permiten distinguir con claridad los conjuntos de términos de información.

  Las principales desventajas de los métodos estadísticos son:

  - Efectividad reducida en la recuperación debido a términos fuera de contexto o términos muy generales.
  - Requieren grandes muestras para que el análisis de frecuencia sea apropiado.

- Método Lingüístico: investigaciones realizadas demuestran que si se realizará un tratamiento sobre el lenguaje natural podría mejorar el trabajo con la información textual y así surgen los métodos lingüísticos que prometen resolver los problemas relacionados con la recuperación de información pues permitirían el entendimiento total de los ítems de información.

  Pero esta idea no se puede llevar a cabo pues entender actualmente un lenguaje natural no es posible y esto se debe en gran medida:

  - Las técnicas disponibles de procesamiento del lenguaje natural no son suficientes.
  - No existe el conocimiento necesario.
  - Se requiere un esfuerzo humano muy grande para realizar una aplicación de este tipo.

Actualmente existe un creciente interés en la unión de ambas técnicas la estadística y la lingüística pues generalmente no es necesario realizar una interpretación exacta de la información lingüística extraída.

Marcos de conocimientos

Los marcos de conocimientos son técnicas que emplean el análisis sintáctico, semántico y lexicográfico de las especificaciones en lenguaje natural de los activos de software sin aspirar a un entendimiento completo de los documentos.

Estos sistemas son más poderosos que los sistemas por palabras clave pero requieren de mucho más recursos humanos para construir las bases de conocimientos y poblarlas.

Especificaciones formales

Las consultas constituyen las representaciones formales de requerimientos, el sistema devuelve los activos relevantes desde el repositorio que se encuentran especificados formalmente. La recuperación se realiza mediante teoremas que verifican si las especificaciones de los activos satisfacen los requerimientos de las consultas.

Dado que las especificaciones formales se centran en el funcionamiento del activo brindan las siguientes ventajas:

- No presentan ambigüedad.
- Presentan mayor precisión que los métodos informales.

También presenta desventajas como son:

- Solo una parte muy pequeña del software disponible para reutilización se encuentra especificado formalmente.
- El tiempo de proceso del algoritmo de búsqueda resulta excesivo.

Redes Neuronales

Las redes neuronales artificiales se utilizan para estructurar el repositorio de acuerdo a la similitud funcional de los activos de software almacenados. Los activos de software que presentan una similitud funcional son almacenados cerca por lo tanto la similitud entre componentes queda explícita. Las funciones de vecindad usadas por el algoritmo se basan en las medidas tradicionales de similitud en los modelos de espacio vectorial.

Hojear

El uso de hojear o visualización rápida como mecanismo secundario es muy frecuente en las aplicaciones de recuperación. El proceso comienza con el conjunto de candidatos recuperado a partir de la consulta del usuario. La principal desventaja de esta técnica es el manejo del proceso de navegación por el propio usuario lo cual puede resultar impreciso y engorroso en grandes repositorios. Aunque la característica de vuelta atrás podría facilitar la navegación pero no es muy común en las aplicaciones.

Hipertexto

En la búsqueda sobre la base de la tecnología de hipertexto el experto J. Savoy plantea que “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” [27]. 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 activo a la red es tremendamente complicado, pues deben considerarse todos los posibles enlaces que se relacionen con el nuevo componente.

Un ejemplo de Sistema de Recuperación de Información es:

Lucene (anteriormente denominada Lucene Java), proporciona una indexación basada en Java así como la implementación de la búsqueda, la corrección ortográfica, el resaltado y el análisis avanzado de capacidades de tokenización.

El concepto de Documento (Document) que contiene campos de texto se encuentra en el centro de la arquitectura lógica de Lucene y esta flexibilidad le permite ser independiente del formato del fichero.

2. Seguimiento de los activos de software almacenados en el repositorio

Los siguientes elementos que se proponen pueden ser controlados en cada uno de los activos de software con el propósito de medir cuan usables y reutilizables resultan en el desarrollo de software. Resultaría muy útil mantener el control de los elementos que se almacenan en el repositorio pues muchas veces estos se sobrecargan con información que no resulta relevante y no se utiliza. Los elementos que se proponen son:

- Frecuencia de reutilización.
- Accesos sin reutilización.
- Accesos sin modificación.
- Uso sin modificación.
- Uso con adaptaciones.
- Calidad de la documentación.
- Versión.
- Lenguaje.
- Complejidad (tamaño, número de entradas/salidas).

2.2.5 Responsables de la clasificación de los repositorios de activos de software reutilizables

Los responsables de verificar que el repositorio de activos de software cumpla con las características planteadas pueden observarse en la tabla 4.

Tabla 4: Roles y responsabilidades de la clasificación del repositorio de activos de software reutilizables.

Roles	Responsabilidades
Diseñador de base de datos Implementador	Verificar que exista un esquema de clasificación que garantice el orden de los activos de software dentro del repositorio.
	Verificar que se inserten los metadatos correspondientes a cada activo para facilitar su recuperación en el proceso de producción de software.
	Verificar de se utilice un Sistema de Recuperación de Información adecuado en la recuperación de activos de software.
	Verificar que se realiza un seguimiento de los activos de software que se encuentran almacenados en el repositorio.

Conclusiones del capítulo

El modelo de activos de software reutilizables que se desarrolló en este capítulo contiene el número de clasificaciones de activos que mayores beneficios reportan en la producción de software, como: componentes, patrones de diseño, especificación de requisitos, arquitecturas, código fuente, documentación.

Se realizó una estructura detallada y visual del modelo, que permite a los usuarios ubicarse y entender el funcionamiento del modelo de activos de software. El mismo cuenta con la clasificación de los activos de software reutilizables y la clasificación de los repositorios de activos de software reutilizable y dentro de cada una de estas se detallan los responsables y salidas del modelo.

Las características de los activos de software y de los repositorios que se definen en esta investigación para lograr una correcta clasificación, son de obligatorio cumplimiento, pues cada una de estas representa un beneficio en el desarrollo de sistemas con reutilización.

Para lograr el cumplimiento de las características descritas se puntualizaron los responsables de llevar a cabo la clasificación tanto de los activos de software como de los repositorios y se le asignaron a estos las actividades que deben cumplir.

Capítulo 3: Validación del modelo de activos de software reutilizables

Introducción

En este capítulo se aplicará el modelo de activos de software reutilizables a los proyectos seleccionados y se realizará un análisis de los resultados obtenidos.

3.1 Resultados de la aplicación del modelo de activos de software reutilizables

En el centro Tecnologías de Gestión de Datos (DATEC) existen varios proyectos que se encuentran dando sus primeros pasos hacia la reutilización de software ejemplos de ellos son: Lycan Génesis-Component Builder (IDE para el desarrollo de aplicaciones web), Sistema de Información de Gobierno (SIGOB) y Sistema Integrado de Gestión Estadística (SIGE). Las pruebas serán realizadas en el SIGOB y el SIGE.

3.1.1 Resultados obtenidos en el proyecto Sistema de Información de Gobierno

En el centro DATEC se encuentra el software SIGOB. Esta aplicación informática contiene sub-módulos que se desarrollan con la programación orientada a componentes, lo que significa un avance hacia la reutilización de activos de software. Es por esto que se decidió aplicar el modelo de activos de software en este proyecto.

Centro: DATEC.

Línea: Integración de soluciones.

Sistema: Sistema de Información de Gobierno (SIGOB).

Módulo: tablero digital (Dashboard).

Sub-Modulo: diseñador de modelos.

Capítulo 3. Validación del modelo de activos de software reutilizables

El diseñador de modelos sigue el desarrollo basado en componentes, lo cual facilita la reutilización en el centro DATEC, pero es necesario y adecuado conocer si esos componentes cumplen con las características requeridas, para que puedan en un futuro ser utilizados en el desarrollo de otros sistemas, es por ello que se propone aplicar el modelo de activos de software a los siguientes componentes:

- Crear origen de datos.
- Visualizar origen de datos.
- Navegador del diseñador de modelos.

Tabla 5: Resultado de la aplicación del modelo de activos de software al componente Crear origen de datos.

Tipo de activo: componente	
Nombre: Crear origen de datos	
Nombres genéricos	x
Manejo consistente de excepciones	x
Métodos genéricos	x
Interfaz adaptable	x
Desplegado de forma independiente	x
Mantenido	x
Documentación	-
Identificable	x
Auto contenido	x
Accesible solamente a través de su interfaz	x

Servicios invariantes	x

Tabla 6: Resultado de la aplicación del modelo de activos de software al componente Navegador del diseñador de modelos.

Tipo de activo: componente	
Nombre: Navegador del diseñador de modelos	
Nombres genéricos	x
Manejo consistente de excepciones	x
Métodos genéricos	x
Interfaz adaptable	x
Desplegado de forma independiente	x
Mantenido	x
Documentación	-
Identificable	x
Auto contenido	x
Accesible solamente a través de su interfaz	x
Servicios invariantes	x

Los componentes Crear origen de datos y Navegador de diseñador de modelos tienen una documentación incompleta debido a que no presentan información semántica del comportamiento de los atributos,

métodos y los eventos de las interfaces del componente. Carece de información que describa la interoperabilidad del componente con otros componentes y no presenta información extra-funcional del componente como características de seguridad, fiabilidad o rendimiento.

Tabla 7: Resultado de la aplicación del modelo de activos de software al componente Data source wizard.

Tipo de activo: componente	
Nombre: Data source wizard	
Nombres genéricos	x
Manejo consistente de excepciones	x
Métodos genéricos	x
Interfaz adaptable	x
Desplegado de forma independiente	-
Mantenido	x
Documentación	-
Identificable	x
Auto contenido	x
Accesible solamente a través de su interfaz	x
Servicios invariantes	x

El componente Data source wizard al igual que los componentes planteados anteriormente, presenta problemas con la documentación pero además contiene dos sub componentes que dependen el uno del

otro para su despliegue, elemento muy importante que afecta la reutilización futura de los componentes. Se recomienda en este caso dejar documentado las dependencias de estos componentes y así sería más reutilizable debido a su granularidad fina; pero existe la alternativa de tratar el componente de manera general como un solo activo, y en este caso el componente sería de granularidad media y aún reportaría beneficios.

3.1.2 Resultados obtenidos en el proyecto Sistema Integrado de Gestión Estadística

El Sistema Integrado de Gestión Estadística (SIGE) se encuentra desarrollado orientado a componentes es por ello que sería útil aplicar el modelo de activos de software reutilizables, pues reportaría datos importantes para esta investigación.

Centro: DATEC

Línea: Integración de soluciones

Sistema: Sistema Integrado de Gestión Estadística (SIGE)

Módulos:

- Gestión de configuración
- Diseñador de formularios
- Entrada de datos

Tabla 8: Resultado de la aplicación del modelo de activos de software al componente Actualizar formulario.

Modulo: Entrada de datos
Tipo de activo: componente
Nombre: Actualizar formulario

Nombres genéricos	-
Manejo consistente de excepciones	x
Métodos genéricos	-
Interfaz adaptable	x
Desplegado de forma independiente	x
Mantenido	x
Documentación	-
Identificable	x
Auto contenido	x
Accesible solamente a través de su interfaz	x
Servicios invariantes	x

Tabla 9: Resultado de la aplicación del modelo de activos de software al componente Diseñar encuesta.

Modulo: Diseñador de formularios	
Tipo de activo: componente	
Nombre: Diseñar encuesta	
Nombres genéricos	-
Manejo consistente de excepciones	x

Métodos genéricos	-
Interfaz adaptable	x
Desplegado de forma independiente	x
Mantenido	x
Documentación	-
Identificable	x
Auto contenido	x
Accesible solamente a través de su interfaz	x
Servicios invariantes	x

Tabla 10: Resultado de la aplicación del modelo de activos de software al componente Actualizar sistemas de información.

Módulo: Gestión de configuración	
Tipo de activo: componente	
Nombre: Actualizar sistemas de información	
Nombres genéricos	-
Manejo consistente de excepciones	x
Métodos genéricos	-

Interfaz adaptable	x
Desplegado de forma independiente	x
Mantenido	x
Documentación	-
Identificable	x
Auto contenido	x
Accesible solamente a través de su interfaz	x
Servicios invariantes	x

Los componentes Actualizar formulario, Diseñar encuesta y Actualizar sistemas de información presentan problemas comunes debido a que en el proyecto se siguen los mismos estilos y estándares de desarrollo. Las características del modelo de activos de software que se incumplen en estos componentes son:

- La inexistencia de documentación referente a los métodos, atributos e interfaces de los componentes, además no se describe la relación de estos componentes con otros y no se plantean los atributos de seguridad, fiabilidad, usabilidad, entre otros.
- Los componentes presentan nombres específicos del sistema por lo que disminuye en gran medida la posibilidad de ser reutilizado en el desarrollo de futuros sistemas informáticos. Se recomienda utilizar nombres que no comprometan el sistema para aumentar la reutilización de los componentes.
- Los métodos de los componentes no están desarrollados de forma genérica de manera que se puedan reutilizar, aunque no necesariamente todos los métodos deben estar implementados genéricamente debido a que es una tarea costosa que requiere de tiempo y esfuerzo. Es

recomendable en este caso escoger qué métodos son o podrían ser los más usados en futuras aplicaciones y generalizarlos.

Tabla 11: Resultado de la aplicación del modelo de activos de software al patrón de diseño Vista detalle.

Tipo de activo: patrón de diseño	
Nombre: Vista detalle	
Nombre	x
Descripción del área del problema	x
Descripción de las partes de la solución del diseño, sus relaciones y sus responsabilidades	x
Declaración de las consecuencias, los resultados y compromisos de aplicar el patrón	x

Tabla 12: Resultado de la aplicación del modelo de activos de software al patrón de diseño Editor subscriptor.

Tipo de activo: patrón de diseño	
Nombre: Editor subscriptor	
Nombre	x
Descripción del área del problema	x
Descripción de las partes de la solución del diseño,	x

sus relaciones y sus responsabilidades	
Declaración de las consecuencias, los resultados y compromisos de aplicar el patrón	x

Los patrones que actualmente se utilizan en el sistema SIGE son patrones que son reconocidos y contienen todas las características necesarias para que sean considerados activos de software reutilizables. Realizar un patrón de diseño genérico que pueda ser reutilizado por múltiples sistemas es un proceso que consume tiempo y esfuerzo, es por ello que lo más común es utilizar patrones ya desarrollados que se ajusten a las necesidades del sistema a realizar. Se debe tener presente en los proyectos que, cuando se utilizan patrones de diseño, en cualquiera que sea el caso y se pretende almacenarlos en un repositorio de activos, deben contener toda la documentación necesaria para que sea comprensible por todos.

Tabla 13: Resultado de la aplicación del modelo de activos de software al código fuente WP.Templates.js.

Módulo: Entrada de datos	
Tipo de activo: código fuente	
Nombre: WP.Templates.js	
Código del sistema debe ser genérico	-
Usable	x
Documentado	-
Debe asegurársele mantenimiento y capacitación	x
El código debe ser robusto	x

El código debe estar empaquetado	x

Tabla 14: Resultado de la aplicación del modelo de activos de software al código fuente WP.Models.js.

Módulo: Entrada de datos	
Tipo de activo: Código fuente	
Nombre: WP.Models.js	
Código del sistema debe ser genérico	-
Usable	x
Documentado	-
Debe asegurársele mantenimiento y capacitación	x
El código debe ser robusto	x
El código debe estar empaquetado	x

El código fuente que se encuentra en las clases WP.Templates.js y WP.Models.js no se encuentra generalizado ni documentado según lo establecido en el modelo que se aplica, este código fuente es solo un ejemplo de todo el código fuente del sistema que se encuentra en esta misma situación. Se recomienda al equipo de desarrollo de este sistema, escoger cuidadosamente el código candidato a ser reutilizado en un futuro y que éste sea generalizado correctamente. No obstante se aclara que todo el código no debe ser generalizado, sólo el de mayor probabilidad de ser reusado. El código fuente del sistema no cuenta con una debida documentación, elemento clave para la futura reutilización tanto del

código como del componente, es recomendable cuando se generalice el código a reutilizar se especifique como funciona cada línea de código para facilitar el trabajo con éste.

3.1.3 Resultados de la aplicación del modelo de activos de software reutilizables en los repositorios de la universidad.

Los repositorios que actualmente se utilizan en los proyectos SIGOB y SIGE para almacenar los activos de software que son identificados, no cumplen con ninguna de las características que se puntualizan en el modelo propuesto. Estos repositorios no gestionan la clasificación y el almacenamiento de los activos, así como tampoco se ocupan de la recuperación de los activos para su reutilización y mantenimiento. Por el momento puede que no sea un problema para los proyectos, debido a que el número de activos de software que se almacenan en los repositorios es muy bajo, pero cuando se adopte el modelo de LPS se incrementará en gran medida el número de activos y se dificultará la gestión de estos.

Conclusiones del capítulo

Para la validación del modelo de activo de software propuesto se realizaron pruebas en el centro DATEC y dentro de este, a dos proyectos diferentes al SIGOB y al SIGE. Ambos proyectos emplean el desarrollo de software basado en componentes, por esto se decidió aplicar el modelo a varios componentes, patrones de diseño y código fuente.

Luego de realizar un análisis a los resultados obtenidos, se comprobó que el modelo fue capaz de detectar errores existentes en los diferentes artefactos y determinar exactamente de qué tipo error se trataba.

Algunos de los problemas detectados en las piezas de software no eran considerados significativos en sus respectivos proyectos, sin embargo el modelo fue capaz de alertar sobre el peligro de almacenar estos artefactos sin estar listos para reutilizar. Por lo tanto la validación realizada prueba que resultaría muy útil para el proceso de adopción de las LPS aplicar este modelo a los proyectos de la UCI.

Conclusiones Generales

El análisis de los conceptos relacionados con la reutilización de activos de software, fue fundamental para entender que los activos de software reutilizables son piezas de software que tienen un diseño específico que les permite ser reutilizadas múltiples veces en el desarrollo de otros sistemas. Para que un activo pueda ser utilizado debe estar almacenado en un repositorio que cumpla con la seguridad y las características necesarias.

También se analizaron las LPS, estas sustentan sus bases en la reutilización de activos de software para crear una familia de sistemas informáticos. Se seleccionó el modelo de procesos del SEI debido a que cumple con las necesidades de la UCI de reutilizar los activos de software y de poderlos mantener de una forma separada al proceso de producción de aplicaciones.

En el desarrollo del modelo de activos de software reutilizables se mencionaron, algunos de los activos que más beneficios reportan con su reutilización como: componentes, patrones de diseño, especificación de requisitos, arquitectura, código fuente y documentación. Se definió la estructura detallada y visual del modelo de activo de software reutilizable, que permite a los usuarios ubicarse y entender el funcionamiento del mismo.

Se definieron que características deben ser cumplidas tanto por los diferentes artefactos como los repositorios de un proyecto para ser clasificados como activos de software reutilizables y repositorio de activo de software. Se puntualizaron además los responsables de realizar la clasificación y las actividades específicas que cada uno de ellos debe cumplir.

Luego de realizar un análisis a los resultados de las pruebas realizadas al modelo de activos de software reutilizables, en los proyectos SIGOB y SIGE se determinó que el modelo de activo de software que se propone puede ser útil en el proceso de identificación de activos de software para la adopción de las LPS, debido a que el mismo fue capaz de detectar errores en artefactos de diferentes proyectos confirmando que no estaban listos para ser reutilizados. El modelo no solo detecto estos errores sino que estuvo preparado para informar de que clase de errores se trataba y que elementos había que modificar para rectificarlos.

Recomendaciones

- Dar continuidad a la presente investigación en futuros trabajos ampliando la clasificación de activos de software.
- Incorporar el uso de la reutilización de activos de software de forma organizada en todos los proyectos productivos de la universidad.
- Desarrollar los repositorios de activos de software con todas las características que se proponen para facilitar la gestión de un mayor número de activos de software.

Referencias Bibliográficas

1. Jonás A Montilva, “Desarrollo de Software Basado en Líneas de Productos de Software”. Mérida, Venezuela, Universidad de los Andes. (2006). Fecha de consulta: 20 de octubre del 2010.
2. C.Szyperski y C. Pfister, “Component-Oriented Programming”. M. Muhlhauser, editor, Special Inssues in Object-Oriented Programming-ECOOP96 Worksop Reader (1997). Dpunkt Verlag, Heidel. Fecha de consulta 20 de octubre del 2010.
3. Microsoft.(1997). “Microsoft Repository Documentation”. http://www.microsoft.com/repository. Fecha de consulta 30 de octubre del 2010.
4. Jonás A Montilva, Nelson Arapé y Juan Andrés Colmenares, “Desarrollo de software basado en componentes”. Mérida: Publicado en las actas del 4to congreso de automatización y control(CAC03), 2003. Fecha de consulta: 20 octubre del 2010.
5. F. Bachmann, L. Bass, Ch. Buhman, S. Comella-Dorda, F. Long, J. Robert, R. Seacord y K. Wallnau. “Volume II: Technical concepts of component-based software engineering”, 2nd edition. Institute, Carnegie Mellon University: s.n.,2000. Fecha de consulta 10 de noviembre del 2010.
6. Ian Sommerville. “Software engineering”. s.l.:Addison-Wesley Pub Co, 6ta edición, 2000. Fecha de consulta: 15 noviembre 2010.
7. Zamira Segoviano y Yusmila Vidiaux. “La reutilización como parte de la calidad del software”.Habana: s.n.,2007. Fecha de consulta: 28 de noviembre del 2010.
8. M. Griss. “Software reuse: from library to factory” IBM System Journal (1993). Fecha de consulta: 10 diciembre del 2010.
9. P. Freeman. “tutorial: Software reusability”, Washington, IEEE Computer Society Press (1987). Fecha de consulta: 11 de enero del 2011.
10. Sametinger, J. “Software engineering with reusable components”. s.l.: Springer Verlag, (1997). Fecha de consulta: 11 de enero del 2011.
11. Real Academia Española(RAE). (2005). Disponible en: http://www.rae.es. Fecha de consulta 20 de marzo del 2011.
12. Bertoa, M y Vallecillo, A. “Atributos de calidad para componentes COTs”. Málaga: s.n. Fecha de consulta: 12 de enero 2011.

13. Montilva, Jonás A. “Modelo de procesos para el desarrollo de software orientado a objetos”. Maracaibo: s.n.: 2000. Fecha de consulta: 22 de marzo del 2011.
14. Francisco J. García, Juan A. Barras, José M. Marqués. “Informe Técnico: Líneas de productos, Componentes, Frameworks y mecanos”. Marzo 2002. Departamento de Informática y Automática, Universidad de Salamanca. Fecha de consulta: 12 de enero 2011.
15. Krueger, ch. “Introduction to Software Product Lines”. Software Product Lines. (2006). Disponible en: http://www.softwareproductlines.com. Fecha de consulta: 2 de diciembre del 2010.
16. Clements, P. y Northrop, L. “Software Product lines: Practices and Patterns”. s.l.: Addison Wesley. (2002). Fecha de consulta: 24 de enero del 2011.
17. Bosch, J. “Design & Use of Software Architectures. Adopting and Evolving a Product-Line Approach”. Addison Wesley (2000). Fecha de consulta: 25 de enero del 2011.
18. Maribel Ariza, Juan Carlos Molina. “Introducción y principios básicos del desarrollo de software basado en componentes”. (2004). Fecha de consulta: 24 de enero del 2011.
19. Gamma, E., R., et al. “Design Patterns: Elements of Reusable Object- Oriented Software Reading”. s.l.: Addison Wesley, 1995. Fecha de consulta: 5 de febrero del 2011.
20. C. Creel. “Requirements by Pattern. Software Development”. (2000). Fecha de consulta: 5 de febrero del 2011.
21. W. Lam, J. A. McDermid, and A. J. Vickers. “Ten Steps towards Systematic Requirements Reuse”. Requirements Engineering Journal. (1997). Fecha de consulta: 7 de febrero del 2011.
22. Han, Jun. “Characterization of Components”. Kyoto, Japan: s.n., (1998), Vols. International Workshop on Component-Based Software Engineering. Fecha de consulta 9 de febrero del 2011.
23. José Luis Barros Justo. “Técnicas para la clasificación/recuperación de componentes software reutilizables y su impacto en la calidad”. Universidad de Vigo, Vigo, España. Fecha de consulta: 22 de marzo del 2011.
24. R. Prieto. “Implementing Faceted Classification for Software Reuse”. Vol. 34 (1991). Fecha de consulta: 15 febrero del 2011.
25. ASSET. “Asset Source for Software Engineering Technologic”. Disponible en: http://source.asset.com. Fecha de consulta: 24 de marzo del 2011.

26. M. R. Girardi, R. T. Pierce. “A Toll for Software Reuse in Object-Oriented Development”. Río de Janeiro: XII Brazilian Computer, (1992). Fecha de consulta 4 marzo del 2011.
27. J. Savoy. “Effectiveness of Information Retrieval Systems Used in a Hypertext Enviroment”, Hipermedia, Vol. 5, Nº 1, (1993). Fecha de consulta: 20 de marzo del 2011.

Bibliografía

1. **EnterpiseDB.** ItilV3. ItilV3. [En línea] 2009. [Citado el: 16 de Febrero de 2011.] Disponible en: http://itil.com.

2. **ASSET.** Asset Source for Software Engineering Technologic. [En línea] [Citado el: 24 de marzo de 2011.] Disponible en: http://source.asset.com.

3. **Bertoa, Manuel y Vallecillo, Antonio.** Atributos de calidad para componentes COTS. Málaga: s.n.

4. **C.Creel.** Requirements by Pattern. Software Development. 2000.

5. **Gamma, E., R., et al.** Design Patterns: Elements of Reusable Object-Oriented Software Reading. s.l.: Addison Wesley, 1995.

6. **M.R. Girardi, R.T. Pierce.** A Toll for Software Reuse in Object-Oriented Development. Río de Janeiro: XII Brazilian Computer, 1992.

7. **RAE.** Real Academia Española. [En línea] abril de 2005. [Citado el: 20 de marzo de 2011.] disponible en: http://www.rae.es.

8. **Sanchez, Maria A. Mendoza.** Metodologias de Desarrollo de Software.

9. **W.Lam, J.A.McDermid, and A.J.Vickers.** Ten Steps towards Systematic Requirements Reuse. Requirements Engineering Journal. 1997.

10. **Szyperski, C.** Component software: Beyond Object-Oriented Programming, 2nd edn. s.l.: Harlow:Adinson-Wesley.(Chs.12,19), 2002.

11. **Sommerville, Ian.** Ingeniería de software séptima edición. s.l.: Addison-Wesley, 2005.

12. **Sommerville, I.** Software engineering. s.l.: Addison-Wesley Pub Co, 6ta edición, 2000.

13. **Sametinger, J.** Software engineering with reusable components. s.l.: Springer Verlag, 1997.

14. **Montilva, Jonás A.** Desarrollo de Software Basado en Líneas de Productos de Software. Mérida-Venezuela: Universidad de Los Andes, 2006.

15. **Maribel Ariza Rojas, Juan Carlos Molina García.** Introducción y principios básicos del desarrollo de software basado en componentes. 2004.

16. **Krueger, Ch.** Introduction to Software Product Lines. Software Product Lines. [En línea] 2006. [Citado el: 2 de diciembre de 2010.] Disponible en: http://www.softwareproductlines.com.

17. **J. Sodhi, y P. Sodhi.** Software reuse: Domain analysis and design process. s.l.: McGraw-Hill, 1999.

18. **F. Bachmann, L. Bass, Ch. Buhman, S.Comella-Dorda, F. Long, J. Robert, R.Seacord y K. Wallnau.** Volume II:Technical concepts of component-based software engineering,2nd edition. Institute, Carnegie Mellon University: s.n., 2000.

19. **Clements, P. and Northrop, L.** Software Product Lines: Practices and Patterns. s.l.: Addison Wesley, 2002.

20. **Szyperski, C. Pfister.** Component-Oriented Programming. Dpunkt Verlag, Heidel: M. Muhlhauser, 1997. Special Inssues in Object-Oriented Programming-ECOOP96 Worksop Reader:.

21. **Microsoft.** Microsoft Repository Documentation. [En línea] 1997. [Citado el: 30 de octubre de 2010.] Disponible en: http://www.microsoft.com/repository.

22. **Griss, M.** Software reuse: from library to factory. s.l.: IBM System Journal, 1993.

23. **Freeman, P.** Tutorial: Software reusability. Washington: IEEE Computer Society Press, 1987.

24. **Francisco J. García, Juan A. Barras, José M. Marqués.** Informe Técnico: Líneas de productos, Componentes, Frameworks y mecanos. s.l.: Universidad de Salamanca., 2002.

25. **Bosch, J.** Design & Use of Software Architectures. Adopting and Evolving a Product-Line Approach. s.l.: Addison Wesley, 2000.

26. **Justo, José Luis Barros.** Técnicas para la clasificación/recuperación de componentes software reutilizables y su impacto en la calidad. Vigo, España: Universidad de Vigo.

27. **Prieto, R.** Implementing Faceted Classification for Software Reuse. 1991. Vol. 34.

28. **Savoy, J.** Effectiveness of Information Retrieval Systems Used in a Hypertext Enviroment. 1993. Vol 5, Nº 1.

29. **Jonás A. Montilva, Nelson Arapé y Juan Andrés Colmenares.** Desarrollo de Software Basado en Componentes. Mérida: Publicado en las actas del 4to congreso de automatización y control(CAC03), 2003.

30. **Díaz, R.Prieto.** Implementing Faceted Classification for Software Reuse. 1991.

31. **Han, Jun.** Characterization of Components. Kyoto, Japan: s.n., 1998.

32. **Montilva, Jonás A.** Modelo de procesos para el desarrollo de software orientado a objetos. Maracaibo: s.n., 2000.

33. **Zamira Segoviano, y Yusmila Vidiaux.** La reutilización como parte de la calidad del software. Habana: s.n., 2007.