ESTUDIO COMPARATIVO DE PROPUESTAS DE ANÁLISIS DE DOMINIO

JORGE HERNAN VELEZ URBANO

UNIVERSIDAD DE NARIÑO

FACULTAD DE INGENIERÍA

PROGRAMA DE INGENIERÍA DE SISTEMAS

SAN JUAN DE PASTO

2015

ESTUDIO COMPARATIVO DE PROPUESTAS DE ANÁLISIS DE DOMINIO

JORGE HERNAN VELEZ URBANO

TRABAJO DE GRADO PRESENTADO COMO REQUISITO PARCIAL PARA OPTAR AL TITULO DE INGENIERO DE SISTEMAS

Director
ING. LUIS VICENTE CHAMORRO MARCILLO

UNIVERSIDAD DE NARIÑO
FACULTAD DE INGENIERÍA
PROGRAMA DE INGENIERÍA DE SISTEMAS
SAN JUAN DE PASTO
2015

NOTA DE RESPONSABILIDAD

Las ideas y conclusiones aportadas en este Trabajo de Grado son responsabilidad de los autores.

Artículo 1 del Acuerdo No. 324 de octubre 11 de 1966, emanado del honorable Concejo Directivo de la Universidad de Nariño.

NOTA DE ACEPTACIÓN

_____

_____

_____

_____

Jurado

_____

Jurado

San Juan de Pasto, 2015

AGREDECIMIENTOS

A Dios por bendecirme con mi familia y amigos.

RESUMEN

La reutilización del software dentro de un dominio de aplicación incluye el descubrimiento de elementos comunes a los sistemas pertenecientes a dicho dominio. Cuando se utiliza este enfoque se produce un cambio de un desarrollo orientado a un único producto software a un desarrollo centrado en varios productos que comparten un conjunto de características formando una familia. Esto genera una reestructuración del proceso software de forma que surgen dos procesos distintos: la ingeniería de dominio y la ingeniería de aplicación.

En el presente trabajo se presenta un estudio comparativo de diferentes propuestas de análisis de dominio, con el propósito de proveer a los investigadores de un referente que permita tomar una decisión acertada sobre el enfoque que más se adapta a los requisitos particulares del problema que se pretende resolver.

ABSTRACT

Software reuse within an application domain includes the discovery of elements common to the systems belonging to that domain. When this approach is used a change -oriented development of a unique software product development centered on several products that share a set of characteristics occurs forming a family. This software generates a restructuring process so that there are two distinct processes: domain engineering and application engineering.

In this work a comparative study of different proposals domain analysis is presented for the purpose of providing researchers a reference enabling make a decision on the approach that best suits the particular requirements of the problem to be solved.

CONTENIDO

LISTA DE TABLAS

Pag.

LISTA DE FIGURAS

Pag.

INTRODUCCIÓN

La reutilización del software dentro de un dominio de aplicación incluye el descubrimiento de elementos comunes a los sistemas pertenecientes a dicho dominio. Cuando se utiliza este enfoque se produce un cambio de un desarrollo orientado a un único producto software a un desarrollo centrado en varios productos que comparten un conjunto de características formando una familia. Esto genera una reestructuración del proceso software de forma que surgen dos procesos distintos: la ingeniería de dominio y la ingeniería de aplicación.

La ingeniería de dominio se centra en el desarrollo de elementos reutilizables que formarán la familia de productos, mientras que la ingeniería de aplicación se orienta hacia la construcción o desarrollo de productos individuales, pertenecientes a la familia de productos, y que satisfacen un conjunto de requisitos y restricciones expresados por un usuario específico, reutilizando, adaptando e integrando los elementos reutilizables existentes y producidos en la ingeniería de dominio.

La Ingeniería de dominio incluye todas las actividades requeridas para la construcción de elementos reutilizables de software. Estas actividades son: el análisis de dominio, el diseño de dominio y la implementación del dominio. Los productos o activos de software de estas actividades son modelo del dominio, modelo de diseño del dominio, lenguajes de dominio especifico, código generado y componentes de código.

El análisis del dominio es el proceso de identificación, colección, organización y representación de la información relevante en un dominio, basado en el estudio de sistemas existentes y sus desarrollos históricos, conocimiento capturado de expertos del dominio, la teoría relacionada y la tecnología emergente dentro de un dominio. [INT1]

Este trabajo busca comparar enfoques de análisis de dominio con el fin de identificar aspectos comunes y diferenciales.

1. GENERALIDADES

1.1 TITULO

Estudio comparativo de propuestas de análisis de dominio

1.2 LINEA DE INVESTIGACION

Este proyecto corresponde a la línea de Investigación de Ingeniería de Software.

1.3 ALCANCE Y DELIMITACION

Este proyecto de investigación se limita al estudio comparativo de tres propuestas de análisis de dominio.

El término “Análisis de Dominio” tiene su origen en los trabajos de Neighbords a comienzos de la década de los 80, quien lo definió como “La actividad que consiste en identificar los objetos y operaciones de un tipo de sistemas similares dentro de un dominio de problema particular”. (Neighbords, 1981).

Otra definición clásica es la propuesta por Prieto- Diaz (1990), como “el proceso por el cual la información utilizada para el desarrollo de sistemas software se identifica, captura y organiza con el fin de hacerla reutilizable en la creación de nuevos sistemas”.

Estas dos definiciones nos permiten identificar las principales características del análisis de dominio:

- ✓ Se trata de una disciplina orientada a la captura y gestión de información y conocimiento.
- ✓ Pretende abstraer los aspectos que describen y caracterizan un área de actividad o proceso, es decir, un “dominio” específico.
- ✓ Parte de un conjunto de sistemas software existentes.
- ✓ Tiene como objetivo identificar información que pueda reutilizarse en el diseño de futuros sistemas.

1.4 MODALIDAD

Este proyecto corresponde a la modalidad trabajo de investigación.

1.5 DESCRIPCIÓN DEL PROBLEMA

1.5.1 Planteamiento del problema

Actualmente no existe un estudio comparativo de propuestas de análisis de dominio, por lo tanto los investigadores que utilizan enfoques de análisis de dominio como insumo en sus trabajos no tienen referentes para tomar una decisión acertada sobre el enfoque que más se adapta a los requisitos particulares del problema que el investigador pretende resolver.

El Modelamiento basado en el Dominio o Domain-Driven Modeling, es un nuevo paradigma de Software que destaca la construcción de herramientas de software sobre la base del Modelo del negocio o dominio. El modelo debe abstraer y mostrar de forma clara los principios fundamentales que rigen al dominio particular de los detalles tecnológicos tales como lenguajes, herramientas de comunicación, etc., que pueda requerir el producto final; permitiendo así que el modelo original sea independiente de la plataforma de desarrollo PIM (Platform Independent Model)

El problema que se puede observar en los paradigmas de modelamiento de sistemas tradicionales, es que estos se basaban en la generación de documentos que eran muy difíciles de mantener en sincronía con los cambios realizados en el código que atendían a los cambios en los requerimientos iníciales del sistema. O sea que, a menos que no hubiese cambios dentro de estas especificaciones iníciales, el modelo no correspondía al sistema desarrollado. Una de las causas principales de este problema es que estos métodos de modelamiento y los ambientes de desarrollo existentes no son muy buenos en separar los detalles arquitectónicos y tecnológicos del espacio conceptual del problema. Como consecuencia de lo anterior los sistemas producidos son extremadamente sensibles a los cambios en el ambiente de desarrollo, y al mismo tiempo, se vuelve muy difícil, sino imposible, reflejar de forma directa en la implementación del sistema los cambios producidos en el dominio.

Además, aparte de lo citado anteriormente, esta gran dependencia de la infraestructura tecnológica requiere un alto nivel de experiencia y conocimiento por parte de los equipos de desarrollo; lo que incrementa los costos de contratación de personal o entrenamiento del mismo.

El modelamiento basado en el dominio, también conocido como Desarrollo basado en el Modelo -MDD por sus siglas en inglés Model Driven Development- o como Desarrollo basado en el Dominio –DDD por sus siglas en inglés Domain Driven Development-; toma como base la idea de que la complejidad de un sistema de información está dada por la complejidad del contexto del problema. Entonces, dando una especial atención a comprender los detalles esenciales del dominio, antes que a las especificaciones del sistema en sí, los desarrolladores serán capaces de lidiar de una mejor manera con la complejidad del sistema. Basado en esta idea, el MDD plantea la construcción y uso de un modelo independiente de la plataforma, que represente los conceptos principales del dominio y las reglas del mismo. Por

supuesto, la construcción de un sistema no puede estar respaldada únicamente en un modelo del problema, por lo que encima del modelo independiente de la plataforma, debe existir al menos un modelo que represente los detalles tecnológicos del sistema a construir. Este último modelo se conoce como PSM –Platform Specific Model- por sus siglas en inglés. La figura 1 representa la arquitectura básica de un sistema modelado usando el paradigma MDD.

La mayoría de los métodos que soportan este paradigma definen su propia subdivisión de la capa superior; pero en todas ellas se mantiene constante la existencia de un modelo que representa el dominio y la lógica del mismo.

1.5.2 Formulación del problema

¿De qué manera se pueden identificar los aspectos comunes y diferenciales entre propuestas de análisis de dominio?

1.5.3 Sistematización del problema

¿Cuáles son las propuestas de análisis de dominio que deben ser incluidas en el estudio?

¿Cómo se identifican los aspectos que deben ser estudiados en cada una de las propuestas de análisis de dominio?

¿Cuál es el proceso para estudiar las propuestas de análisis de dominio?

1.6 OBJETIVOS

1.6.1 Objetivo general

Identificar aspectos comunes y diferenciales entre propuestas de análisis de dominio por medio de un estudio comparativo.

1.6.2 Objetivos específicos

- ✓ Identificar las propuestas de análisis de dominio que hacen parte del estudio comparativo.

- ✓ Definir los criterios de análisis empleados para el desarrollo del estudio comparativo.

- ✓ Aplicar los criterios de análisis a cada una de las propuestas de análisis de dominio.

1.7 JUSTIFICACIÓN

El diseño y construcción de sistemas informáticos se basa en la aplicación de una serie de técnicas y metodologías que aseguran la evolución de unas especificaciones formuladas en lenguaje natural hasta su representación en código escrito en un lenguaje de programación determinado. Para facilitar esta evolución, se han propuesto distintas metodologías y técnicas de diagramación que representan la estructura y el comportamiento de los sistemas de información. Se pretende así evitar cualquier malinterpretación y ambigüedad en las especificaciones y en su evolución posterior. Estos formalismos no solo facilitan la representación de los sistemas de información y los hace más comprensibles para las distintas personas que participan en su desarrollo, también hacen viable su posterior mantenimiento y reutilización en futuros proyectos.

La comunidad académica se encuentra en un proceso de aprendizaje continuo, por ello este proyecto de investigación ofrecería agilidad en otros proyectos de investigación que utilizan la ingeniería de dominio y enfoques de análisis de dominio como insumo para sus investigaciones.

El análisis de dominio pretende capturar la información crítica sobre las entidades, los datos y los procesos que caracterizan un área de negocio en particular, con el objetivo de facilitar la especificación, el diseño y la construcción de aplicaciones que soporten dichos procesos. A partir de la información procedente de estos documentos – y de los sistemas informáticos existentes – se podrá elaborar una conceptualización del área de actividad y de los procesos para los que se idearon los sistemas informáticos tomados como base del análisis.

El análisis de dominio está estrechamente vinculado a la reutilización de software, que busca la construcción de nuevos sistemas informáticos a partir de componentes, especificaciones o diseños creados en el pasado. La relación entre análisis de dominio y reutilización es evidente: disponer de una representación sistematizada del conocimiento característico de un área, hará más sencilla la creación de sistemas, módulos y componentes que puedan aplicarse a múltiples escenarios y en la resolución de distintos problemas.

2. MARCO TEORICO

2.1 REUTILIZACIÓN DE SOFTWARE

La reutilización de software es el proceso mediante el cual se usa cualquier tipo de artefacto o activo, o parte del mismo, creado con anterioridad, en un nuevo proyecto [REUS1].

La reutilización del software es un tópico de la ingeniería del software que cada día cobra más importancia, debido a las evidentes ventajas que genera su aplicación. Las buenas prácticas asociadas a la reutilización permiten: la reducción de los costes de desarrollo, el aumento de la calidad de los productos, el aumento de la productividad, mejoras en las actividades de mantenimiento y mejoras en las actividades de control y planificación.

2.1.1 La evolución de la reutilización

Hasta hace algunos años, resultaba difícil poder representar artefactos de software en un catálogo principalmente los asociados a etapas tempranas del ciclo de vida, lo cual impedía que los desarrolladores pudieran buscarlos sin conocimiento previo de su existencia.

Por este motivo, para reutilizar hubo que realizar el proceso inverso: primero parar la organización para ver qué podía ser reutilizable, después hacerlo reutilizable, lo siguiente era obligar a que todos conocieran lo que existe para reutilizar en la organización y luego intentar obligar a que reutilizaran lo existente. Tras estas consideraciones, puede deducirse que aplicar la reutilización implicaba un cambio en el proceso y en la cultura organizacional en el desarrollo de software.

Para minimizar los inconvenientes que se presentaban cuando una organización pretendía iniciar procesos de reutilización de activos de software, muchas de las prácticas aplicadas hasta ese momento fueron eliminadas o modificadas y otras más fueron creadas para soportar el nuevo paradigma. Por ejemplo: la reutilización debe aplicarse de manera natural dentro del ciclo de vida que soporta el desarrollo de software en la organización, se deben crear artefactos reutilizables cuando se tenga la seguridad de que serán útiles en oportunidades posteriores, no tener restricciones en lo que se desee buscar, cualquier artefacto, en cualquier etapa del proceso es factible de ser reutilizado.

2.1.2 La reutilización en las etapas tempranas del proceso de desarrollo de software

Hasta hace poco tiempo, el único artefacto reutilizable era el código fuente, aunque reutilizar código fuente es realmente problemático debido a la dependencia con la plataforma. Hoy en día los objetivos de la reutilización

apuntan también a las llamadas etapas tempranas del ciclo de vida del software, donde el activo a reutilizar tiene más valor, puesto que se ha requerido un mayor poder de abstracción para crearlo; y hay más independencia de la plataforma tecnológica, lo que evita riesgos de obsolescencia antes de la reutilización.

Los patrones de software representan la estrategia más reconocida por la comunidad de ingeniería de software para aplicar la reutilización. El concepto de patrones fue introducido por Christopher Alexander quien los define de la siguiente manera: “Cada patrón describe un problema que ocurre una y otra vez en un entorno dado, describiendo el núcleo de la solución al problema de tal forma que pueda ser empleada un millón de veces sin hacerlo dos veces de la misma forma”.

2.1.2.1 La reutilización en la fase de requisitos

Para aplicar la reutilización de software en la etapa de requisitos normalmente se utilizan los patrones de requisitos. Un patrón de requisitos puede ser visto como un conjunto de requisitos reutilizable [REUS2]. Al igual que en otras fases del ciclo de vida, pueden existir patrones horizontales, es decir, interoperables entre varios dominios, como por ejemplo los requisitos de seguridad, y patrones verticales, es decir, propios de un dominio dado. Un patrón de requisitos está compuesto por la siguiente información:

- El conjunto de requisitos, ya sean estos horizontales o verticales, junto con sus correspondientes tipos.

- Los riesgos que acecharán al proyecto por la simple y sencilla razón de que tiene que abordar uno o más de esos requisitos.

- La especificación de las pruebas que han de llevarse a cabo para validar que el nuevo sistema a construir cumple con lo indicado en los requisitos del patrón.

- Los diagramas de análisis que responden a estos requisitos: estructuras de clases, modelos de datos, diagramas de actividad o de interacción que resuelvan los requisitos.

- El conjunto de documentos que permita entender el contenido del patrón y que ayude a implementarlo correctamente.

- Las herramientas que soportan la ingeniería de requisitos.

2.1.2.2 La reutilización en la fase de Análisis

Como en las otras fases del ciclo de vida del software, los patrones representan la estrategia más reconocida para aplicar la reutilización de software y proveer a los proyectos de las ventajas que esta genera.

Particularmente en la fase de Análisis, los patrones de análisis de Martín Fowler son el trabajo más representativo. Fowler en [REUS3] Identifica problemas repetitivos y los transforma en modelos reutilizables que son agrupados en un catálogo de patrones de análisis que tiene aplicación en un amplio rango de dominios de aplicación incluyendo comercio, medición, contabilidad y relaciones organizacionales. Reconociendo que los patrones conceptuales no pueden existir de manera aislada, el autor también presenta una serie de patrones de soporte que describen la forma de convertir modelos conceptuales en software, que a su vez, encaja en una arquitectura de un sistema de información. Incluye en cada patrón el razonamiento seguido para su diseño, las normas para cuando se debe y no se debe utilizar, y consejos para la aplicación.

2.1.2.3 La reutilización en la fase de diseño

Históricamente, el siguiente nivel de reutilización por encima del código fuente son los diseños. La idea de la reutilización de diseños se inicia en el trabajo de Christopher Alexander quien introduce el concepto de patrones en el dominio de la arquitectura, Alexander define los patrones de la siguiente manera: “Cada patrón describe un problema que ocurre una y otra vez en un entorno dado, describiendo el núcleo de la solución al problema de tal forma que pueda ser empleada un millón de veces sin hacerlo dos veces de la misma forma”.

Posteriormente la idea de patrones es adaptada al diseño de software, encontrando importantes trabajos al respecto como el presentado por Gamma y su equipo, en su libro Elementos de Software Reutilizable Orientado a Objetos [REUS4] en el cual hacen una clasificación en patrones creacionales, estructurales y de comportamiento.

Más adelante, surgen una gran cantidad de patrones. Algunos de ellos han sido ideados para ser reutilizados horizontalmente, es decir, con posibilidad de aplicar en diferentes dominios o sectores, otros patrones han sido creados para dominio específicos, lo que se conoce como reutilización vertical.

2.1.3 El proceso de la recuperación de información

Basili [REUS2] creó una famosa taxonomía que definía las actividades a acometer para llevar a cabo la reutilización de software; sin embargo, esta idea sigue siendo la clave en los modernos catálogos de activos de software.

Tal y como puede verse en la figura 1, el proceso de reutilización se fundamentaba en la recuperación de activos en un catálogo y su posterior adaptación para el proyecto donde se deseen aplicar. Para lo cual debemos partir de un catálogo que nos permita identificar sus activos, de cara a poder evaluarlos y seleccionar el candidato ideal a formar parte del nuevo proyecto. Este proceso de identificación, a su vez, consta de los procesos de

caracterización y del proceso de matching que permite en sí la localización de los activos clasificados.

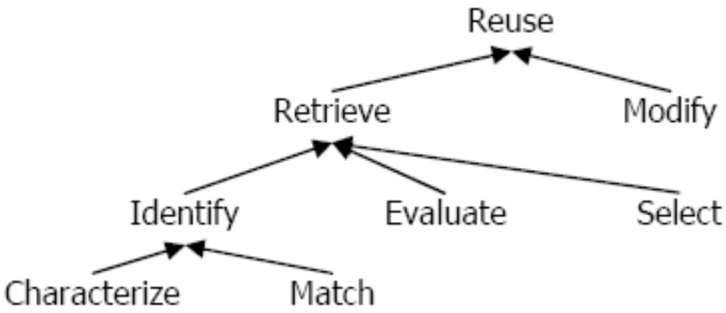

Figura 1: Proceso de reutilización de Software

2.2 INGENIERÍA DE DOMINIO

La ingeniería de dominio es el proceso empleado para el desarrollo de aplicaciones para una familia de sistemas similares. La Ingeniería de dominio incluye todas las actividades requeridas para la construcción de los activos principales del software. Estas actividades son: el Análisis de dominio (identificación de uno o más dominios, capturar la variación dentro de un dominio), el diseño de dominio (construcción de un diseño adaptable) y la implementación del dominio (definir los mecanismos para trasladar los requisitos a sistemas creados para la reutilización). Los productos o activos de software de estas actividades son modelo del dominio, modelo de diseño del dominio, lenguajes de dominio especifico, código generado y componentes de código.

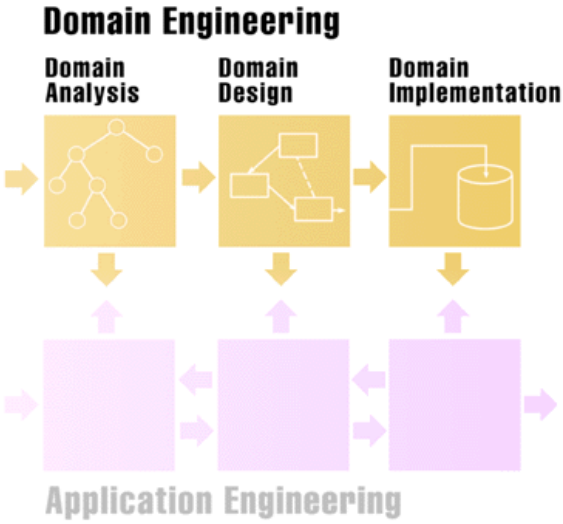

Figura 2: Actividades de la ingeniería de dominio

- **Análisis de Dominio.** El análisis de dominio es el proceso de identificación, colección, organización y representación de la información

relevante en un dominio, basado en el estudio de sistemas existentes y sus desarrollos históricos, conocimiento capturado de expertos del dominio, la teoría relacionada y la tecnología emergente dentro de un dominio [INDO1].

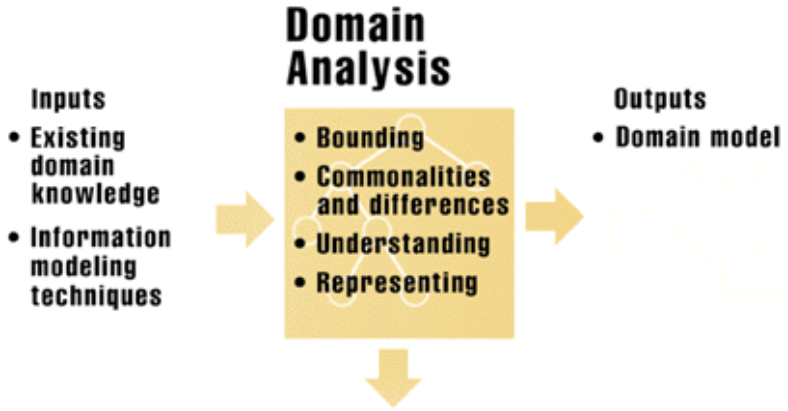

Figura 3: Proceso de análisis de dominio.

El Análisis de dominio describe detalladamente el dominio estudiado, considerando los elementos comunes y los elementos diferenciales, de los sistemas en el dominio, permite la organización y la comprensión de las relaciones entre los elementos del dominio y representa la información recolectada de manera que sea útil para el desarrollo de la aplicación actual y de las aplicaciones futuras que pertenezcan a la misma familia o dominio.

En la actualidad existen numerosas técnicas de análisis de dominio. Cada una se centra en incrementar el nivel de comprensión del dominio capturando la información y representándola en modelos formales.

- **Diseño del Dominio.** Actualmente muchos investigadores están estudiando y clasificando arquitecturas de software, con el fin de identificar estilos de arquitectura y patrones de sistemas que permitan la reutilización en el desarrollo de nuevas aplicaciones dentro de un dominio.

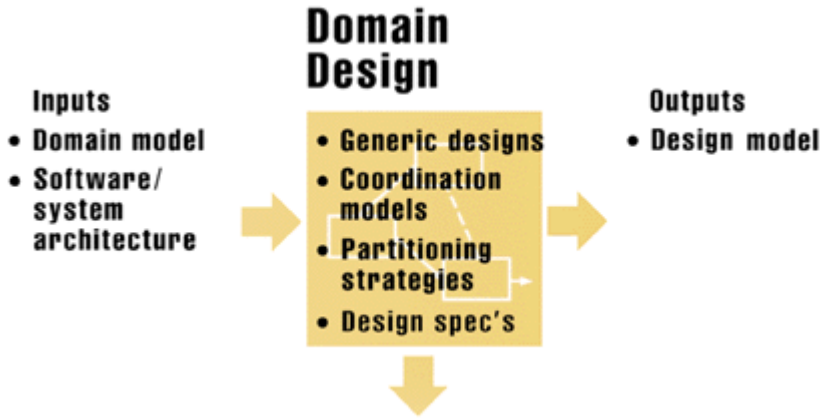

Figura 4: Proceso de diseño de dominio

El diseño del dominio es el proceso de desarrollo de un modelo de diseño a partir de los productos del análisis del dominio y del conocimiento adquirido a partir del estudio del reusO de los requisitos, diseños y arquitecturas genéricas de software de software [INDO1].

La clave para la reutilización es el desarrollo y documentación de arquitecturas genéricas para las aplicaciones de un dominio. El modelo de diseño representa la arquitectura genérica desarrollada para el dominio analizado y provee el marco de trabajo para el desarrollo de los componentes reutilizables en la implementación del dominio.

La arquitectura genérica es representada por un estilo arquitectónico que es el más apropiado para la familia de productos. Esta arquitectura define la plataforma computacional para todas las aplicaciones de la familia y provee las pautas para la construcción de un diseño genérico. El resultado es una estrategia de partición para la asignación de características del dominio a elementos del software y un modelo de coordinación que describe como los elementos son activados y como comparten información.

Un modelo de coordinación define los protocolos de activación y comunicación para elementos del software que implementan el modelo de comportamiento. Un modelo de coordinación típicamente define el tipo de relaciones asignadas a los elementos de software y los mecanismos que soportan las interacciones.

Una estrategia de partición define los elementos de software y como las características del dominio son asignadas a ellos. Ejemplos de estrategias de partición incluyen asignación por entidades del mundo real, asignación por procesos de control, asignación por tipos de usuarios o asignación por objetos. Algunas de estas estrategias pueden ser más apropiadas para unas familias de productos que para otras.

Aplicar el modelo de coordinación al modelo de dominio, permite identificar las operaciones internas del sistema, además, se especifican los mecanismos del sistema y sus parámetros de desempeño.

La especificación del diseño formaliza el comportamiento interno del sistema, identifica la información del dominio y sus restricciones. Si las aplicaciones de una familia de productos son embebidas en un sistema hardware especifico, las restricciones impuestas por el modelo de coordinación y por el hardware adquieren una importancia mayor. Si la familia de productos es un sistema de información, las reglas de negocio se capturan como restricciones.

- **Implementación de Dominio.** La creación de componentes reutilizables ha cobrado gran trascendencia en la comunidad de ingeniería de software actual, principalmente con el uso de librerías de componentes de código. Muchos de estos repositorios no son el resultado de esfuerzos en torno a

la ingeniería de dominio, por tanto, activos reutilizables de alto nivel tales como modelos de dominio o arquitecturas de referencia. El trabajo en el área de librerías de componentes tiende a centrarse en cuestiones de cualificación, clasificación, recuperación y retiro de componentes.

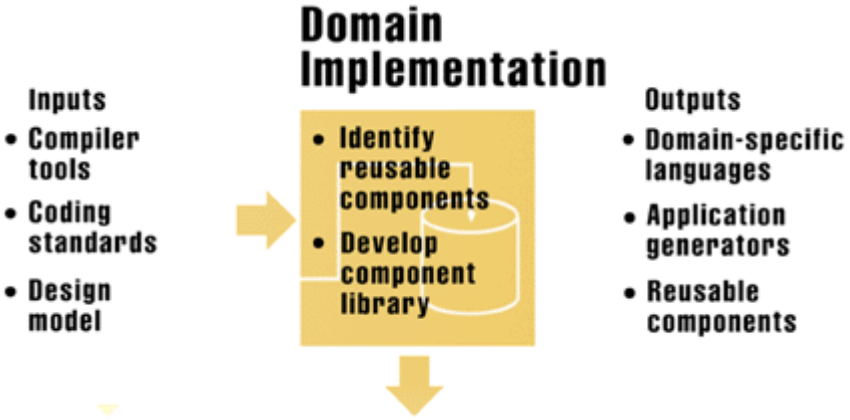

Figura 5: Proceso de implementación de dominio

La implementación del dominio es el proceso de identificar componentes reutilizables basados en el modelo del dominio y una arquitectura genérica [INDO1]. Usando el conocimiento obtenido durante el análisis del dominio, y la arquitectura genérica desarrollada durante el diseño del dominio, los ingenieros del dominio adquieren y, donde sea necesario, crean los activos reutilizables que son catalogados en una librería de componentes para el uso de los ingenieros de aplicación. Estos componentes reutilizables, así como los generadores de aplicación y los lenguajes de dominio son los principales productos de esta fase en la ingeniería de dominio. La creación, gestión, y mantenimiento de un repositorio de activos reutilizables, son también tareas de las cuales se encarga la implementación de dominio.

2.2.1 FODA

2.2.1.1 Objetivo de la propuesta FODA

FODA tiene como objetivo, identificar y modelar los elementos comunes y variables del sistema en un dominio durante el análisis de dominio, enfocado en las características (aspecto, calidad o característica de un sistema visible al usuario final) de los dominios del sistema.

FODA permite tener más control de los resultados de cada fase y conocimiento del sistema, debido a que cada fase del proceso ya tiene definidos los artefactos que debe generar. Las características que más se ven influenciadas al aplicar el método FODA son las características de mantenibilidad (facilidad de un sistema o componente de ser corregido, adaptado y/o mejorado [FODA01]), entendimiento (grado de claridad de un sistema para su evaluador [FODA01]) y reutilización (grado de un módulo software o un producto de trabajo de ser usado en más de un programa o sistema software [FODA01]).

FODA se fundamenta en dos conceptos: abstracción y refinamiento. La abstracción es usada para crear productos de dominio de aplicaciones específicas y el refinamiento para mejorar estos productos.

El producto primario de FODA es un framework estructurado de modelos relacionados que catalogan los resultados del análisis de dominio.

2.2.2 ODM

2.2.2.1 Objetivo de la propuesta ODM

El objetivo principal de ODM es la transformación sistemática de artefactos de sistemas heredados en activos que pueden ser utilizados en múltiples sistemas en un contexto de dominio específico centrado en la arquitectura [ODM1]. Para ello, ODM busca:

- Proveer una base sistemática para la reingeniería de activos y su reutilización basada en la documentación explicita de su alcance.

- Maximizar el uso de artefactos heredados existentes como base para el conocimiento del dominio y la reingeniería.

El elemento clave es el modelo de dominio adquirido empíricamente, que establece el rango de variabilidad requerida que el modelo debe abarcar. ODM enfatiza en el uso de artefactos y conocimiento como fuente de conocimiento del dominio y de potenciales recursos para reutilización. Sin embargo, uno de sus objetivos es evitar la incorporación de restricciones ocultas que pueden existir en los sistemas heredados, en los modelos de dominio y en los activos reutilizados [ODM2].

ODM influye sobre las características de mantenibilidad, entendimiento y reusabilidad de un sistema o familia de sistemas. La mantenibilidad hace referencia a la facilidad con que un sistema de software o componente puede ser modificado para corregir fallas, incrementar desempeño, adaptarlo a los cambios del entorno, etc. La entendibilidad es el grado con que el propósito de un sistema o componente es claro para el evaluador. La reusabilidad es el grado con que un módulo de software u otro producto de trabajo puede ser usado en más de un programa de computador o sistema de software [ODM2].

Otros objetivos de ODM son:

- Promover la exploración de la máxima variabilidad dentro del dominio, incluyendo:

    - Reutilización de artefactos en todo el ciclo de vida.

    - Reutilización de procesos así como de activos de software.

- Reutilizar de forma que facilite el uso de componentes estáticos y técnicas generativas tales como estrategias hibridas para implementación de activos.

- Proveer estrategias efectivas para definir el alcance de aplicabilidad para activos base que son estratégicamente apropiados para la organización.

- Soportar la evolución de los activos base y la tecnología requerida para soportar nuevos tipos de organizaciones.

2.2.3 DARE

2.2.3.1 Objetivo de la propuesta DARE

Es una herramienta CASE que proporciona un entorno de trabajo y una metodología que facilita las tareas propias del análisis del dominio: extracción de información sobre el ámbito de aplicación y adquisición y almacenamiento de dicha información. Se basa en las herramientas disponibles en la actualidad de recuperación de información (IR), inteligencia artificial (IA), e ingeniería inversa. [DARE01]

Las principales ventajas respecto a otros entornos de trabajo como FODA y ODM es su alta modularidad que la hace compatible con dichos entornos. [DARE02]

El objetivo primario cuando se usa DARE es la creación de una arquitectura genérica que describe elementos de arquitectura y sus relaciones, para una familia de sistemas. En la arquitectura generada se destacaran los componentes reusables sin embargo será capaz de adaptarse a componentes variables. Como resultado del análisis de dominio se obtendrá, también, documentación sobre éste y un sistema de clasificación automatizado organizado por etapas. [DARE02]

3 CRITERIOS DE ANALISIS PARA LA COMPARACIÓN DE ENFOQUES DE ANÁLISIS DE DOMINIO.

Para determinar el conjunto de criterios que guían el análisis de los enfoques de análisis de dominio, exploramos los empleados en estudios anteriores similares.

En [CRIT1] se describe el examen de aproximaciones orientadas por aspectos en las fases de análisis, arquitectura y diseño. Este estudio aplica criterios generales para determinar el comportamiento de los enfoques en características propias de la fase del ciclo de vida que soportan. También se utilizan criterios particulares que determinan la forma como las propuestas abordan algunas calidades sistémicas del software, como la trazabilidad a través del ciclo de vida, la Componibilidad, la evolucionabilidad y la escalabilidad.

En [CRIT2] se desarrolla un estudio para la comparación de notaciones de modelado de la variabilidad de requisitos para líneas de productos, este estudio presenta criterios de análisis como: legibilidad, simplicidad y expresividad, representación de dependencias, evolucionabilidad, adaptabilidad, escalabilidad, etc. Básicamente es un trabajo destinado a identificar la notación más apropiada para la representación de requisitos en el contexto particular de las líneas de productos.

En [CRIT3] se presenta el examen de enfoques de análisis de dominio, este estudio aplica criterios de comparación para determinar la aproximación más apropiada de acuerdo al contexto particular de aplicación. Los criterios de comparación incluyen: conceptos de ingeniería de dominio y aspectos relacionados con el proceso y los resultados del análisis de dominio.

3.1 Criterios de comparación aplicados

Se han definido como criterios de comparación base los presentados en [CRIT3]. Algunos de estos criterios no se tienen en cuenta porque se considera que no son relevantes para esta investigación, otros fueron reformulados y algunos tomados tal y como los plantea el autor original. Los criterios de comparación que rigen este estudio se presentan en detalle a continuación.

3.1.1 Definición de dominio

Tal vez la diferencia más fundamental entre los enfoques de análisis de dominio es la falta de un acuerdo general sobre lo que es un dominio. En lo que hay consenso es que un dominio está integrado entre otros elementos por un conjunto de problemas relacionados. Un dominio maduro también integra las soluciones a esos problemas, es decir, existen aplicaciones en el dominio que

permiten resolver esos problemas. Esta dicotomía en el espacio del problema/solución es común a todos los enfoques de análisis de dominio. Sin embargo, todavía se plantea la pregunta, de que fuerzas dirigen estos problemas y estas soluciones. Dos definiciones de "dominio" utilizadas en las aproximaciones de análisis de dominio, expresan lo siguiente:

3.1.1.1 Área de aplicación

Un dominio puede ser visto como un área de aplicación. El dominio está integrado por aplicaciones que resuelven los problemas que se presentan en el dominio.

3.1.1.2 Área de negocio

Un dominio puede ser también considerado como un área de negocio. El dominio está dado por el proceso de negocio en donde se presentan problemas que las aplicaciones resuelven.

3.1.2 Determinación de problemas en el dominio

Todas las aproximaciones contienen una vista de un dominio como un conjunto de problemas con las correspondientes soluciones para cada problema. Es posible determinar los problemas del dominio de tres maneras:

3.1.2.1 Orientada al Problema

El análisis del dominio puede primero definir un conjunto de problemas, para luego generar las soluciones. El modelo construido utilizado en las etapas iníciales del análisis del dominio enfatiza en los conceptos de niveles del problema. El analista subsecuentemente los refina en conceptos apropiados para especificar la solución, usualmente en el diseño y construcción de software.

3.1.2.2 Orientada a la solución

El analista de dominio puede iniciar por examinar aplicaciones, a partir de las cuales se determinan los problemas comunes. Esto puede sonar como un paso hacia atrás, pero tiene mucha justificación en la realidad del análisis de dominio. Algunos investigadores asumen que el análisis del dominio es útil solo después de cierto número de aplicaciones construidas. Específicamente, después de tener suficiente aplicaciones para construir el dominio. En este punto de vista, el análisis de dominio sucede después de que se han construido las aplicaciones del dominio para proporcionar una firme comprensión de los principios que soportan el dominio.

3.1.2.3 Orientada al problema/solución

La aproximación utiliza orientación al problema para algunos productos y orientación a la solución para otros, sobre la base de una evaluación de lo que parece más apropiado. Análisis orientado al problema y orientado a la solución, por lo tanto, se producen simultáneamente.

3.1.3 Permanencia de resultados de análisis de dominio

Cada aproximación tiene un proceso asociado que define cuando el analista de dominio desarrolla un producto, define si puede evolucionarlo, y en caso afirmativo, establece cómo. Para esto existen dos posibilidades:

3.1.3.1 Permanente

El proceso no establece la evolución del producto, la aproximación asume que el resultado sería completado y corregido antes del primer uso. La presunción es que cualquier dominio lo suficientemente maduro como para aplicar análisis de dominio, es estable. Los problemas que plantea no evolucionan, ni su terminología, ni las normas que lo rigen. En resumen, no es necesaria la evolución. Esto no excluye la evolución de las aplicaciones en el dominio. Los avances tecnológicos y económicos pueden dar lugar a nuevas soluciones para los problemas en un dominio, lo que en este modelo implica nuevos productos. Estas soluciones pueden cambiar con el tiempo, siempre y cuando continúen resolviendo la especificación de problemas. Para ello, se espera que los requisitos hayan sido los correctos.

3.1.3.2 Variable

El proceso tiene pasos que permiten la evolución de los productos en el tiempo. El conocimiento adquirido del dominio, tanto del uso de los productos en el dominio y de las influencias externas, es decir, las nuevas tecnologías, es la base para la evolución.

3.1.4 Relación con el proceso de desarrollo de software

Los productos de análisis de dominio no son un fin en sí mismas. Las organizaciones los usarían como parte de otro proceso. ¿Qué proceso puede ser, y donde encaja el análisis de dominio?, son también cuestiones a considerar. Las posibilidades identificadas son las siguientes:

3.1.4.1 Dependiente

Una aproximación, puede hacer análisis de dominio en actividades de pre-requisitos de un proceso de desarrollo de software. La aproximación integra

actividades de análisis de dominio dentro de las actividades del proceso de desarrollo. Las salidas de cada actividad de análisis de dominio son insumos para las actividades específicas del proceso de desarrollo de software. El análisis de dominio se desarrolla durante o después de la fase de requisitos del sistema, pero antes de la fase de requisitos de software (de ahí, "pre-requisitos"). La implementación del dominio puede preceder, o ser parte las posteriores actividades de diseño de la aplicación. El resultado son partes reutilizables disponibles durante las fases de requisitos y diseño del software de cualquier otro proyecto de software para el mismo dominio.

3.1.4.2 Independiente

El análisis de dominio puede ser una actividad de pre-requisitos, pero independiente del modelo de ciclo de vida. Los productos de análisis de dominio están destinados a ser lo suficientemente genéricos como para ser usado en gran variedad de procesos. Sin embargo, el análisis de dominio debe desarrollarse antes del análisis de requisitos de software, por lo que el modelo debe tener una fase de requisitos.

3.1.5 Objeto de análisis

El análisis de dominio se esfuerza por descubrir ciertos puntos de vista fundamentales de un dominio. Cada enfoque identifica varios tipos de puntos de vista. Entre estos se define el "objeto de análisis." El objeto de análisis es el punto de vista que, de todos los tipos identificados, es central para la aproximación de análisis de dominio. El objeto de análisis determina la forma en que los analistas de dominio llevarán a cabo sus tareas y la forma de los productos de las actividades de análisis. Los diferentes objetos de análisis son los siguientes:

3.1.5.1 Objetos y Operaciones

El análisis puede centrarse en los objetos y las operaciones entre sistemas similares. Dependiendo del enfoque, esto puede significar centrado en los requisitos, el diseño arquitectónico, o el diseño detallado. En los requisitos, el analista se concentra en la identificación de objetos en el dominio (dispositivos, usuarios, etc.) y en los requisitos funcionales y no funcionales para esos objetos. El analista que se centra en el diseño arquitectónico busca modelos de procesos y diseño de estructuras que caracterizan a todas las aplicaciones en el dominio. El analista que se centra en el diseño detallado descubre módulo de interfaces y el funcionamiento de las mismas.

No importa en qué nivel este centrado el analista, lo importante es que debe concentrase en lo que es común en el dominio. Todas las aplicaciones comparten los objetos y las operaciones. El análisis se centra en las semejanzas.

3.1.5.2 Decisiones

El análisis puede centrarse en las decisiones que los desarrolladores deben afrontar para obtener una solución aceptable a un problema en un dominio. El analista de dominio utiliza los conceptos del dominio para definir los medios para diferenciar los problemas en términos de decisiones que lleven a las soluciones. Este es un objeto de análisis en el nivel de requisitos únicamente. Las consideraciones de diseño e implementación de las decisiones son un objeto secundario del análisis.

4 ANÁLISIS

4.1 Definición de dominio

Definición de dominio	FODA	ODM	DARE
• Área de aplicación:	X		NA
• Área de negocio:		X	

Tabla 1: Definición de dominio

4.1.1 Definición de dominio FODA

FODA define dominio simplemente como un conjunto de aplicaciones [FODA01]. Las entradas al análisis de dominio orientado a las características son fundamentalmente de carácter funcional (restricciones de ambiente, requisitos de software); el proceso no establece la forma de relacionarlos con las necesidades del negocio. Los resultados son vistas de software desde diferentes perspectivas: de usuario final, analista de requisitos y diseñados o desarrollador de software.

4.1.2 Definición de dominio ODM

ODM utiliza el modelado explícito de contextos sociales y organizacionales en todas las fases del ciclo de vida, articulando los objetivos de la ingeniería de dominio con los criterios de selección de dominio, la identificación del dominio, definición y alcance del mismo. El contexto organizacional también establece los criterios de las entradas y las salidas para las actividades de modelado.

Los contextos organizacionales en ODM son analizados a partir de su estructura como áreas de negocio, a partir de las cuales se desarrolla el proceso ODM. Las áreas del negocio y su articulación en organizaciones permiten desarrollar el análisis de dominio ODM, en tal sentido, la definición del dominio para ODM se fundamenta en las áreas del negocio y las organizaciones.

4.1.3 Definición de dominio DARE

El modelo de proceso de DARE especifica que la definición del dominio debe desarrollarse previamente, es decir, esta es una tarea que esta fuera del contexto de DARE.

4.2 Determinación de problemas en el dominio

Determinación de problemas en el dominio	FODA	ODM	DARE
• Orientada al Problema	X		

• Orientada a la solución		X	
• Orientada al problema/solución			X

Tabla 2: Determinación de problemas en el dominio

4.2.1 Determinación de problemas en el dominio FODA

Para el análisis de dominio FODA sigue el enfoque top-down partiendo de un conjunto de problemas, la función del analista es buscar la solución en un conjunto de soluciones genéricas. FODA es por tanto, orientado al problema.

4.2.2 Determinación de problemas en el dominio ODM

Los problemas de selección del alcance del dominio son menos probables de presentarse cuando el esfuerzo del desarrollo es guiado por un espacio de problema y/o un espacio de solución claramente definido. En un contexto de ingeniería dirigido por el problema, los ingenieros buscan soluciones teniendo en cuenta que tan estable es el problema. En un contexto dirigido al producto o a la tecnología, el problema es estable. En este último caso, el espacio de la solución se emplea para que los ingenieros busquen oportunidades de aplicación. ODM reconoce que en el contexto de ingeniería de dominio los espacios del problema y de la solución son simultáneamente dinámicos, en tal sentido el proceso ODM plantea a los analistas, la libertad de partir de un conjunto de problemas asociados a un dominio generar las soluciones o de las soluciones propuestas indagar los problemas que estas solucionan. ODM es orientada al problema/solución.

4.2.3 Determinación de problemas en el dominio DARE

Una de las entradas del proceso DARE son los sistemas existentes, en tal sentido se puede catalogar DARE como una aproximación orientada a la solución, sin embargo, en DARE el proceso de análisis de dominio se guía por la organización de objetivos de la organización definidos como los problemas a solucionar, por esto, DARE también se identifica como un enfoque orientado al problema.

De lo anterior se puede deducir que DARE utiliza orientación al problema para algunos productos y orientación a la solución para otros, sobre la base de una evaluación de lo que parece más apropiado. Por lo tanto se concluye que DARE es un enfoque de análisis de dominio orientado al problema y orientado a la solución, es decir, se producen simultáneamente.

4.3 Permanencia de resultados de análisis de dominio

Permanencia de resultados de análisis de dominio	FODA	ODM	DARE
• Permanente	X	X	
• Variable			X

Tabla 3: Permanencia de resultados de análisis de dominio

4.3.1 Permanencia de resultados de análisis de dominio FODA

En FODA, una vez que el analista de dominio define un conjunto de modelos de dominio (es decir, las vistas), siguen siendo en gran medida permanentes, ya que representan características genéricas en un área de aplicación. Estos modelos actúan como marcos de referencia para instanciaciones de nuevas aplicaciones. Cambiarlos podría producir incompatibilidades para la reutilización de los componentes existentes. Por lo tanto en FODA los resultados son permanentes.

4.3.2 Permanencia de resultados de análisis de dominio ODM

Uno de los objetivos de ODM es: soportar la evolución de los activos base y la tecnología requerida para soportar nuevos tipos de organizaciones. El proceso ODM provee los elementos requeridos para ello. El producto final de ODM es una librería de activos dinámicos reutilizables, a los cuales se les puede aplicar reingeniería. Estos hechos permiten deducir que la permanencia de los resultados del análisis del dominio en ODM es variable.

4.3.3 Permanencia de resultados de análisis de dominio DARE

En DARE, los modelos de dominio proporcionan la base para el diseño y la aplicación de componentes reutilizables. Para el uso de los modelos de dominio se aplica la reutilización basada en el desarrollo, que permite la retroalimentación de la librería reutilizables. Por lo tanto los modelos de dominio se encuentran en un proceso de continua evolución, en otras palabras los resultados de análisis de dominio son variables.

4.4 Relación con el proceso de desarrollo de software

Relación con el proceso de desarrollo de software	FODA	ODM	DARE
• Dependiente			
• Independiente	X	X	X

Tabla 4: Relación con el proceso de desarrollo de software

4.4.1 Relación con el proceso de desarrollo de software FODA

FODA, define un conjunto de procesos para su aplicación, pero no mapea a los modelos de proceso software para el desarrollo de aplicaciones. En lugar de ello, sugiere estrategias generales de utilización de los productos de análisis de dominio. Al no asociarse directamente con los modelos de desarrollo de software o con una fase del ciclo de vida, se concluye que FODA es independiente del modelo de proceso de desarrollo de software.

4.4.2 Relación con el proceso de desarrollo de software ODM

ODM es estructurado en términos de un ciclo de vida principal de ingeniería de dominio que es independiente al ciclo de vida de ingeniería de sistemas

[ODM1]. El ciclo de vida de ODM es dividido en tres fases: plan de ingeniería del dominio, modelado del dominio e ingeniería de la base de activos. Este proceso puede ser adaptado para acompañar a cualquier modelo de proceso de software. Los resultados del proceso ODM, pueden ser empleados como insumos de cualquiera de las fases del proceso de desarrollo que rige la construcción de software. Por lo tanto ODM es independiente del proceso de desarrollo de software empleado por las organizaciones.

4.4.3 Relación con el proceso de desarrollo de software DARE

Los productos de análisis de dominio en DARE son lo suficientemente genéricos para garantizar su uso en cualquier modelo de proceso de software.
De lo anterior se deduce que DARE es independiente del proceso de desarrollo de software.

4.5 Objeto de análisis

Objeto de análisis	FODA	ODM	DARE
• Objetos y Operaciones		X	X
• Decisiones	X		

Tabla 5: Objeto de análisis

4.5.1 Objeto de análisis FODA

El análisis de dominio orientado a las características usa una estructura simple para describir los elementos comunes y variables. La generación de esta estructura es una de las primeras tareas del análisis del dominio en FODA. Aunque la estructura muestra los aspectos comunes y variables, su propósito es diferenciar las aplicaciones en el dominio. Para esto, FODA muestra un modelo jerárquico que facilita la representación de decisiones acerca de cómo las aplicaciones difieren. En FODA, entonces, el objeto de análisis son las decisiones que deben tomar los desarrolladores de aplicaciones. FODA permite tomar estas decisiones oportunamente puede ser en tiempo de compilación, de carga o de ejecución, creando un modelo de decisión que refleja la información capturada.

4.5.2 Objeto de análisis ODM

El objetivo principal de ODM es la transformación sistemática de artefactos de sistemas heredados en activos que pueden ser utilizados en múltiples sistemas en un contexto de dominio específico centrado en la arquitectura [ODM1]. Para ello, ODM busca: Proveer una base sistemática para la reingeniería de activos y su reutilización basada en la documentación explicita de su alcance y; maximizar el uso de artefactos heredados existentes como base para el conocimiento del dominio y la reingeniería. El objeto de análisis en ODM son los objetos y las operaciones del contexto organizacional analizado.

4.5.3 Objeto de análisis DARE

El objetivo primario cuando se usa DARE es la creación de una arquitectura genérica que describe elementos de arquitectura y sus relaciones, para una familia de sistemas. En la arquitectura generada se destacaran los componentes reusables sin embargo será capaz de adaptarse a componentes variables. Esto indica que DARE es un enfoque orientado a la arquitectura, en este caso el analista busca modelos de procesos y diseños de estructuras que caracterizan a todas las aplicaciones en el dominio. Por lo tanto, DARE tiene como objeto de análisis los objetos y las operaciones en la fase de arquitectura.

4.6 Resumen del análisis

A continuación se presenta la tabla 4 en la cual se sintetiza el resultado el análisis desarrollado.

Criterios	**Enfoque**		
	FODA	**ODM**	**DARE**
Definición de dominio			
• Área de aplicación:	X		NA
• Área de negocio:		X	
Determinación de problemas en el dominio			
• Orientada al Problema	X		
• Orientada a la solución		X	
• Orientada al problema/solución			X
Permanencia de resultados de análisis de dominio			
• Permanente	X	X	
• Variable			X
Relación con el proceso de desarrollo de software			
• Dependiente			
• Independiente	X	X	X
Objeto de análisis			
• Objetos y Operaciones		X	X
• Decisiones	X		

Tabla 6: Resultados del análisis desarrollado

5 CONCLUSIONES

5.1 La comunidad de la ingeniería del software ha generado enfoques de análisis de dominio con orientaciones muy diversas en muchos aspectos.

5.2 Aunque el análisis de dominio puede ser valioso para un solo proyecto, su verdadero aporte está en su aplicación en todos los proyectos de la organización, integrados en familias de productos o en líneas de productos de software.

5.3 La ingeniería del software utiliza los catálogos de patrones como un instrumento válido para la identificación, descripción y distribución de soluciones probadas empíricamente, las cuales pueden ser reutilizadas para la solución de problemas concretos que aparecen comúnmente en determinadas fases de desarrollo y dentro de dominios de aplicación específicos.

5.4 Este trabajo permite poner de manifiesto que la integración del análisis de dominio y específicamente los patrones de análisis en los procesos y métodos de desarrollo de software facilita:

- La reutilización del conocimiento que ha demostrado su validez en proyectos anteriores.

5.5 Uno de los periodos más complejos y críticos dentro del proceso de desarrollo de software es el que está relacionado con la obtención, análisis y especificación de requisitos. Durante estas etapas tempranas se suelen construir modelos conceptuales con el objetivo de facilitar el análisis y la comprensión del dominio del problema.

6 REFERENCIAS

[INT1] http://www.sei.cmu.edu/domain-engineering/domain_anal.html
[REUS1] “A pattern language”. Christopher Alexander et al. Oxford University Press, 1977.

[REUS2] “Support for comprehensive reuse”. V. R. Basili, H. D. Rombach. IEEE Software Engineering Journal, 6(5):303–316, September 1991.

[REUS3] “Analysis Patterns: Reusable Object Models”. Martin Fowler. Addison-Wesley Professional, 1996.

[REUS4] “Design Patterns: Elements of Reusable Object-Oriented Software”. Eric Gamma et al. Addison-Wesley Professional, 1995

[SPL1] García Peñalvo, F.J., Barras, J.A., Laguna Serrano, M.A., Marqués Corral, J.M., Líneas de Productos, Componentes, Frameworks y Mecanos, Departamento de Informática y Automática Universidad de Salamanca, 2002.

[SPL2] Clements, P., Northrop, L. M., Bachmann, F., Bass, L., Bergey, J., Chastek, G., Cohen, S., Donohoe, P., Jones, L., Krut, R., Little, R., Smith, D., Tilley, S., Weiderman, N., Withey, J. y Woods, S. (1998). “*A Framework for Software Product Line Practice – Version 1.0*”. Product Line Systems Program, Software Engineering Institute. Carnegie Mellon University, Pittsburgh, Pennsylvania 15213 (USA).

[SPL3] Bosch, J. (2000). “*Design & Use of Software Architectures. Adopting and Evolving a Product-Line Approach*”. Addison-Wesley.

[SPL4] Jacobson I., Griss M. y Jonsson P. (1997). “*Software Reuse. Architecture, Process and Organization for Business Success*”. ACM Press. Addison Wesley Longman.

[INDO1] http://www.sei.cmu.edu/domain-engineering/domain_anal.html

[FODA01] Institute of Electrical and Electronics Engineers. IEEE Standard Computer Dictionary: A Compilation of IEEE Standard Computer Glossaries.New York, NY: 1990

[FODA02] Boehm, Barry W.; Brown, John R.; Kaspar, Hans; Lipow, Myron; MacLeod, Gordon J. & Merritt, Michael J. Characteristics of Software Quality. New York, NY: North-Holland Publishing Company, 1978.

[FODA03] Fei Cao, Barrett R. Bryan, Carol C. Burt, Zhisheng Huang, Rajeev R. Raje, Andrew M. Olson, MukhailAuguston. Automating feature-Oriented Domain Analysis.
[FODA04] Kang, K., et al. Feature-Oriented Domain Analysis (FODA) Feasibility Study (CMU/SEI-90-TR-21, ADA 235785). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1990.

[FODA05] K. Czarnecki, U.W. Eisenecker. Generative Programming: Methods, Tools, and Applications. Addison Wesley, 2000.

[FODA06] A. van Deursen and P. Klint. Domain-specific Language Design Requires Feature Descriptions. Journal of Computing and Information Technology 10(1), pp. 1-17, 2002.

[FODA07] M.G.J. van den Brand, J. Heering, H. A. de Jong, M. de Jonge, T. Kuipers, P. Klint, L. Moonen, P.A. Olivier, J. Scheerder, J.J. Vinju, E. Visser, J. Visser. The ASF+SDF Meta-Environment: a Component-Based Language Development Environment. Compiler Construction (CC '01), vol. 2027, Lecture Notes in Computer Science, pp. 365-370, Springer-Verlag, 2001.

[FODA08] Cohen, Sholom G., et al. Application of Feature-Oriented Domain Analysis to the Army Movement Control Domain (CMU/SEI-91-TR-28, ADA 256590). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1992.

[FODA09] Petro, James J.; Peterson, Alfred S.; & Ruby, William F. In-Transit Visibility Modernization Domain Modeling Report Comprehensive Approach to Reusable Defense Software (STARS-VC-H002a/001/00). Fairmont, WV: Comprehensive Approach to Reusable Defense Software, 1995.

[FODA10] Devasirvatham, Josiah, et al. In-Transit Visibility Modernization Domain Scoping Report Comprehensive Approach to Reusable Defense Software (STARS-VC-H0002/001/00). Fairmont, WV: Comprehensive Approach to Reusable Defense Software, 1994.

[FODA11] Hassan Gomaa. "A Software Design Method for Real-Time Systems." Communications of the ACM, Vol. 27(9): 938-949, September 1984.

[FODA12] Krut, Robert W. Jr. Integrating 001 Tool Support into the Feature-Oriented Domain Analysis Methodology (CMU/SEI-93-TR-11, ESC-TR-93-188) Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1993.

[FODA13] Schnell, K.; Zalman, N.; & Bhatt, Atul. Transitioning Domain Analysis: An Industry Experience (CMU/SEI-96-TR-009). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1996

[FODA14] Kang Kye C, Lee Jaejson. Feature-Oriented Product Line Engineering. IEEE Software. Pp 58, 2002.

[FODA15] Spencer Peterson A, Stanley Jr Jay L. Mapping a Domain Model and Architecture to a Generic Design (CMU/SEI-94-TR-8,ESC–TR-94-008). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1994.

[ODM1] http://www.sei.cmu.edu/domain-engineering/domain_anal.html

[ODM2] Simos, M., Creps, D., Klinger, C., Levine, L., Organization Doman Modeling (ODM*) Guidebook* Version 2.0 (STARS-VC-A025/001/00). Manassas, VA: Lockheed Martin Tactical Defense Systems, 1996.

[DARE01] Prieto-Díaz, Rubén; Frakes, Bill;Gogia, B.K. DARE: A Domain Analysis and Reuse Environment, Phase I Final Report. Reuse, Inc. Agosto 20, 1992.

[DARE02] Frakes, William; Prieto-Diaz, Rubén; Christopher, Fox. "DARE: Domain Analysis and Reuse Environment.," Annals of Software Engineering, Volume 5, Pag. 125-141. 1998

[DARE03] Vegas Hernández, J. Un Sistema de Recuperación de Información sobre Estructura y Contenido. PhD tesis, Universidad de Valladolid, Valladolid, España, 1999.

[DARE04] PINTO, María., Análisis Documental. Fundamentos y procedimientos, Madrid, Eudema, 241 p., 1991, 1993 (2ª ed.). ISBN: 84-7754-070-5.

[DARE05] Frakes, W.B., Gandel, P.B., "Representing Reusable Software", Information and Software Technology , **32**(10), Diciembre, 1990.

[DARE06]Frakes, W.B. and Pole, T., "An Empirical Study of Representation Methods for Reusable Software Components", submitted to IEEE Transactions on Software Engineering. Junio,1992.

[PAT1] Alexander, C., S. Ishikawa, MSilverstein, M. Jacobson, I. Fiksdahl'King, and S. Angel. "A Pattern Language". New York: Oxford University Press, 1977.

[PAT2] Coplien J.O. "Agenerative Development-process Pattern Language". In Pattern Language of program design. J.O. Coplien and D.C. Schmidt, ED. Reading, MA:Addison-Weley, 1995.

[PAT3] Gamma, E., R. Helm R Johnson, and J. Vlissides. Design Patterns: Elements of Reusable Objects-Oriented Software Reading, MA: Addison Weley, 1995.

[PAT4] Hay, D. Datta Model Patterns: Conventions of Thought. New York: Dorset House, 1996

[PAT5] Fowler, M. Analysis Patterns: Reusable Object Model. Addison Wesley, 1997

[PAT6] Isla, J. L. "Modelado conceptual de sistemas cooperativos en base a patrones en amenities". Universidad de Granada, 2007.

[PAT7] Schmidt, D. C Stephenson, P. "Experiences Using Design Patterns to Evolve System Software Across Diverse OS Platforms," in *Proceedings of the*

9th *European Conference on Object-Oriented Programming*, (Aarhus, Denmark), ACM, agosto 1995.

[AOSD1] R. Chitchyan, A. Rashid, P. Sawyer, A. Garcia, M. Pinto Alarcón, J. Bakker, B. Tekinerdogan, S. Clarke y A. Jackson. “Survey of Aspect-Oriented Analysis and Design Approaches”, AOSD-Europe, 2005.

[AOSD2] Mohamed M. Kandé, JörgKienzle and Alfred Strohmeier, “From AOP to UML: Towards an Aspect-Oriented Architectural Modeling Approach”, Software Engineering laboratory Swiss Federal Institute of Technology Lausanne.

[AOSD3] REINA, Antonia. Visión general de Aspectos. Sevilla, España.2000.

[CRIT1] “Survey of Aspect-oriented Analysis and Design Approaches”, Report of the EU Network of Excellence on AOSD by RuzannaChitchyan, Awais Rashid, Pete Sawyer, Allesandro Garcia, Monica Pinto Alarcon, Jethro Bakker, BedirTekinerdogan, Siobhan Clarke, Andrew Jackson, 2005.

[CRIT2] Djebbi, O., Salinesi C. University Paris 1 – Sorbonne 90, Paris, France. Criteria for Comparing Requirements Variability Modeling Notations for Product Lines

[CRIT3] S. Wartik and R. Prieto-Díaz, "Criteria for Comparing Domain Analysis Approaches," In International Journal of Software Engineering and Knowledge Engineering, vol. 3, pp. 403-431, 1992.

7 RECOMENDACIONES

7.1 Antes de definir el enfoque de análisis de dominio a utilizar, se debe tener un conocimiento claro de cada uno, para poder determinar cuál es el más adecuado.

7.2 Es importante identificar el enfoque de análisis de dominio más adecuado a una organización, y utilizarlo en todos los proyectos de la organización, integrados en familias de productos o en líneas de productos de software.