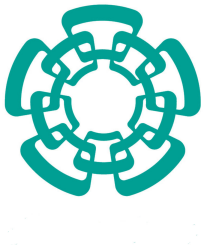

Centro de Investigación y de Estudios Avanzados
del Instituto Politécnico Nacional

Laboratorio de Tecnologías de Información

Diseño de un patrón arquitectónico de software mejorado, aplicable a proyectos de desarrollo ágil

Tesis que presenta:

Sergio Iván Ponce Ruiz

Para obtener el grado de:

Maestro en Ciencias en Computación

Director de la Tesis:
Dr. Eduardo Arturo Rodríguez Tello

Cd. Victoria, Tamaulipas, México. Mayo, 2014

Agradecimiento a SVAM International de México S. de R.L de C.V. por haber facilitado el acceso al caso de estudio utilizado en esta investigación

Acknowledgement to SVAM International de México S. de R.L de C.V. for providing access to the case of study used in this research

La tesis presentada por Sergio Iván Ponce Ruiz fue aprobada por:

Dr. Iván Lopéz Arevalo

Dr. Víctor Jesús Sosa Sosa

Dr. Eduardo Arturo Rodríguez Tello, Director

Cd. Victoria, Tamaulipas, México., 23 de Mayo de 2014

A Dios, mi maestro y principal fuente de voluntad para concluir con esta etapa de mi carrera.

Agradecimientos

- Agradezco primeramente a Dios por haberme dado la salud para afrontar este gran reto profesional y haber estado a mi lado cada día dándome ánimos y sabiduría para seguir adelante y concluir con éxito esta etapa de mi carrera.
- A mi madre *Guadalupe Ruiz,* por ser mi principal apoyo y por soportar el tiempo que estuve lejos de ella y mi familia por estar cumpliendo con esta labor.
- A mi padre *Alejandro Ponce,* por que quizás sin saberlo, ha sido una gran fuente de inspiración, motivándome a seguir sus pasos y su modelo de hombre trabajador y cosechador de éxitos.
- Al Dr. *Eduardo A. Rodríguez Tello,* mi director de tesis y gran amigo, por su comprensión, guía y sobre todo por su paciencia a lo largo del desarrollo de este proyecto.
- A mi novia Karina por soportar a mi lado todos los sacrificios de tiempo y esfuerzo que fueron necesarios para concluir con este reto, pero sobre todo por darme ánimos en los momentos más complicados y por creer en mí.
- Al Centro de Investigación y de Estudios Avanzados del IPN (CINVESTAV-IPN), Unidad Tamaulipas por permitirme formar parte de tan prestigiosa institución y por todo el apoyo recibido para concluir con este trabajo.
- Le agradezco también a todos los investigadores y al personal de CINVESTAV por haber compartido conmigo sus conocimientos y experiencias pero sobre todo por haberme brindado su valiosa amistad.
- Agradezco también al Sr. Robert J. Hart Director de Operaciones de SVAM International de México, por su valioso apoyo para el desarrollo de está investigación, sobre todo por su comprensión al permitirme tomar algunas horas de mi horario de trabajo para llevar a cabo esta tarea.
- Finalmente y no menos importante, al Sr. Othon Rodríguez y a todos mis compañeros de SVAM International de México por respaldarme a lo largo del desarrollo de esta investigación, especialmente a mi equipo de trabajo y a los involucrados con este trabajo.

Índice General

Índice de Figuras

Índice de Tablas

Índice de Algoritmos

Resumen

Diseño de un patrón arquitectónico de software mejorado, aplicable a proyectos de desarrollo ágil

por

Sergio Iván Ponce Ruiz

Maestro en Ciencias del Laboratorio de Tecnologías de Información
Centro de Investigación y de Estudios Avanzados del Instituto Politécnico Nacional, 2014
Dr. Eduardo Arturo Rodríguez Tello, Director

Actualmente en la industria del software existe aún cierta controversia entre las metodologías de desarrollo de software tradicionales y ágiles en cuanto al proceso de diseño. Por una parte los seguidores de las metodologías tradicionales defienden un enfoque de diseño predictivo basado en largas sesiones de planificación. Por otro lado los seguidores de las metodologías ágiles prefieren un enfoque de diseño incremental, argumentando que es imposible predecir con precisión los cambios que puedan llegar a surgir a lo largo del proceso de desarrollo. Respecto a este enfoque, los tradicionalistas resaltan entre otros riesgos, la carencia de un patrón arquitectónico homogéneo para la construcción del software. Esto deriva en costosas y repetitivas sesiones de reestructuración de la arquitectura del software en etapas avanzadas del proyecto.

El presente trabajo de investigación tiene como objetivo proponer un patrón arquitectónico que permita reducir el tiempo en el diseño de una arquitectura, pero que a la vez esté preparado para abordar escenarios de modificabilidad e integrabilidad.

Después de analizar diversos patrones arquitectónicos, se optó por tomar el patrón Datos, Contexto e Interacciones (DCI) como base y marco comparativo para este trabajo. Sobre este patrón se identificaron algunas áreas de oportunidad enmendadas mediante la aplicación de una serie de tácticas y patrones de diseño, resultando en un nuevo patrón arquitectónico denominado Datos, Contexto e Interacciones-Mejorado (EDCI), el cual se propone en esta tesis.

El diseño del patrón EDCI se realizó siguiendo el método de Diseño Dirigido por Atributos (ADD) debido a su enfoque hacia la satisfacción de los atributos de calidad requeridos por un sistema. El método ADD recibe como principales entradas, los requerimientos de un sistema, por lo cual se eligió como caso de estudio un proyecto de planificación de recursos actualmente en proceso de desarrollo para el Centro de Investigación y Estudios Avanzados del Instituto Politécnico Nacional (CINVESTAV) Unidad Tamaulipas.

Con la finalidad de validar las mejoras proporcionadas por EDCI, éste al igual que DCI fueron sometidos a un método de evaluación de arquitecturas de software conocido como Método de Comparación y Análisis de Arquitecturas de Software (SACAM). Este método se especializa en facilitar la elección de un patrón o arquitectura de software con base en la comparación de diversos candidatos.

La aplicación de este método permitió observar una mejora promedio del 40.35 %, proporcionada por EDCI respecto a escenarios de modificabilidad e integrabilidad resaltando la implementación de componentes intermediarios para la resolución de problemas de integrabilidad.

Abstract

Design of an enhanced architectural pattern for agile software development projects

by

Sergio Iván Ponce Ruiz

Master of Science from the Information Technology Laboratory
Research Center for Advanced Study from the National Polytechnic Institute, 2014
Dr. Eduardo Arturo Rodríguez Tello, Advisor

Currently in the software industry there is still controversy among the traditional and agile software development methodologies regarding architectural design process. On one side, traditional methodologies enthusiasts defend a predictive approach based on large planning sessions. On the other side, agile methodologies enthusiasts prefer rather an incremental approach, arguing that it is impossible to predict with enough accuracy the change requests that might come along the development process.

Regarding this approach, traditionalists highlight among other risks, the lack of an homogeneous architectural pattern, deriving in repetitive and expensive software architecture restructuring sessions in late stages of the project.

This research work has as objective to propose an architectural pattern that allows to reduce the invested time on architectural design, but that is well prepared to address modifiability and integrability scenarios.

After analyzing different architectural patterns, it was decided to take DCI pattern as basis and comparative framework for this research. Some areas of opportunity over this pattern were identified and amended through the application of a set of design tactics and patterns, resulting in a new architectural pattern called EDCI, that is proposed in this thesis.

The design of EDCI pattern was performed by following the Attribute Driven Design method (ADD) due to its approach oriented to the quality attributes required for a system. ADD method

receives system requirements as main inputs, so a real resources management project currently being developed for the Center of Research and Advanced Studies of the National Polytechnic Institute (CINVESTAV), Tamaulipas Unit was chosen as case of study.

With the goal of validating the improvements provided by EDCI, this pattern as well as DCI were submitted to a software architecture analysis method called Software Architecture Comparison and Analysis Method (SACAM). This method is specialized on enabling the selection of an architecture or pattern based on the comparison of several candidates. The implementation of this method allowed to observe an average improvement of 40.35 %, provided by EDCI regarding modifiability and integrability scenarios, highlighting the implementation of intermediary components for integrability issues resolution.

Nomenclatura

Acrónimos principales

ADD	*Attribute Driven Design*, método de diseño de software enfocado a la satisfacción de atributos de calidad.
ALMA	*Architecture-Level Modifiability Analysis*, método de evaluación de arquitecturas de software diseñado específicamente para evaluar escenarios de modificabilidad.
ATAM	*Architecture Trade-off Analysis Method*, método de evaluación de arquitecturas de software, que proporciona una técnica basada en el análisis del compromiso o Trade-off existente entre diversos atributos de calidad del software.
DCI	*Datos, Contexto e Interacción*, patrón arquitectónico compuesto por el patrón *Modelo, Vista, Controlador* y el patrón *Datos, Contexto e Interacción*.
EDCI	*Enhanced-DCI*, patrón DCI con patrones de diseño y tácticas aplicadas para su mejora.
GRP	*Governance Resource Planning*, sistemas de software diseñados con la finalidad de apoyar con tareas de gestión de recursos financieros públicos.
MVC	*Modelo-Vista-Controlador*, patrón arquitectónico, modelo, vista controlador.
RUP	*Rational Unified Process*, metodología de desarrollo de software enfocada en la participación continua de los diversos involucrados en el desarrollo de un proyecto.
SAAM	*Scenario-based Software Architecture Analysis Method*, método de evaluación de arquitecturas de software considerado el padre de los métodos de evaluación basados en escenarios.
SAAMCS	*SAAM for Complex Scenarios*, método para la evaluación de arquitecturas de software ante escenarios de calidad complejos.
SACAM	*Software Architecture Comparison and Analysis Method*, método de evaluación de arquitecturas de software cuyo objetivo es facilitar la selección entre diversas arquitecturas candidatas, mediante comparación.
SCRUM	Metodología de administración de proyectos utilizada comúnmente en proyectos ágiles de desarrollo de software.
XP	*Extreme Programming*, metodología ágil de desarrollo de software caracterizada por la técnica de programación en parejas.

1

Introducción

El objetivo de este capítulo es presentar una visión general del presente trabajo de investigación. El problema a resolver, la hipótesis a validar, los objetivos planteados y las contribuciones logradas, son presentados en este capítulo.

1.1 Planteamiento del problema

Hoy en día el software es ya parte esencial de toda organización, convirtiéndose de un lujo o ventaja competitiva en más bien una necesidad.

En los inicios de la industria del software (en la década de 1940), los sistemas a construir solían ser pequeños y de propósito muy específico por lo tanto eran liberados hasta que finalizaba su construcción. En ese entonces el tiempo para construir el software no rebasaba el umbral de un par de semanas; sin embargo, conforme la industria fue creciendo, los clientes demandaban la construcción de sistemas más robustos. Esto incrementó el tiempo necesario para la construcción del software ocasionando que los clientes cayeran en desesperación por no tener sus productos terminados en el tiempo que ellos requerían.

Más adelante, alrededor de los años 80, surgieron las metodologías ágiles para el desarrollo de software con el objetivo de reducir el tiempo de liberación del software y así poder brindar productos de valor al cliente de manera más rápida. Estas metodologías (en contraste con las tradicionales) se enfocaban en minimizar las tareas realizadas en cada fase del proceso de desarrollo con el objetivo de generar entregables de manera más rápida y así tener una mejor respuesta ante las necesidades del cliente. Esto sin embargo, conllevaba el riesgo de dejar de considerar aspectos clave para el desarrollo de un sistema.

Un claro ejemplo de lo anterior es el diseño de la arquitectura del software. Mientras las metodologías ágiles proponían un enfoque de diseño evolutivo en donde la arquitectura se iba ajustando conforme el sistema se iba construyendo, los seguidores de las metodologías tradicionales proponían un enfoque de antemano predictivo, argumentando que el no diseñar la arquitectura a conciencia en etapas tempranas conllevaría el riesgo de no estar preparado para adoptar cambios posteriores en los requerimientos, o bien, que el hacerlo significaría tener que invertir grandes cantidades de tiempo y de recursos en la reestructuración de la arquitectura.

Para los entusiastas de las metodologías ágiles, realizar predicciones precisas y completas acerca de las necesidades y problemas futuros para un sistema era imposible. Por lo tanto creían que no merecía la pena sacrificar la entrega rápida de entregables y poner en riesgo la satisfacción del cliente a cambio de un diseño por demás preparado.

La manera en que cada tipo de metodología de desarrollo (tradicional o ágil) aborda el problema del diseño arquitectónico de software ha causado gran controversia con base en lo antes descrito.

Actualmente existe un buen número de patrones arquitectónicos que pueden ser utilizados para el desarrollo de software siguiendo una metodología ágil. Sin embargo, estos patrones arquitectónicos carecen de la capacidad de mitigar el riesgo de invertir grandes cantidades de tiempo en corregir la arquitectura de un sistema en respuesta a requerimientos cambiantes debido a que no fueron diseñados para tal fin.

Como resultado de esta investigación se logra reducir este problema mediante la creación del patrón *Datos, Contexto e Interacción - Mejorado (EDCI, por sus siglas en inglés)* el cual fue diseñado con base en las principales características del software ágil: rápida entrega de productos de valor al

cliente, adaptación ante requerimientos cambiantes y escalabilidad mediante desarrollo incremental.

Para el diseño de EDCI se tomó como base el método de *Diseño Dirigido por Atributos (ADD, por sus siglas en inglés) [Wojcik et al., 2006]*, el cual tiene como objetivo primordial el asegurar la satisfacción de los atributos de calidad de software particularmente deseables para un sistema. En este caso, mediante la aplicación de este método se busca crear un diseño con elevados niveles de modificabilidad e integrabilidad, atributos de calidad característicos del software ágil.

Para iniciar el proceso de diseño ADD se requieren como entradas los requerimientos funcionales, no funcionales y restricciones de diseño del sistema para el cual se quiere diseñar la arquitectura [1]. Como fuente de obtención de dichas entradas se adoptó un caso de estudio que consiste en un *Sistema de Planificación de Recursos de Gobierno (GRP, por sus siglas en inglés)* desarrollado para el Centro de Investigación y de Estudios Avanzados del Instituto Politécnico Nacional (CINVESTAV), Unidad Tamaulipas.

Para el diseño del patrón EDCI se seleccionó un patrón base diseñado específicamente para aplicarse en proyectos ágiles de software denominado *Datos, Contexto e Interacción (DCI)* [Reenskaug y Coplien, 2009]. Sobre éste patrón se aplicaron una serie de técnicas de diseño con el fin de incrementar los niveles de modificabilidad e integrabilidad y así producir una versión mejorada del mismo.

Una vez terminado el diseño de dicha versión, se realizó una comparación entre ambos patrones para determinar si EDCI realmente incrementaba los niveles de modificabilidad e integrabilidad respecto a DCI.

Para tal comparación se tomaron tanto DCI como EDCI y se sometieron a un método de evaluación de arquitecturas de software conocido como *Método de Análisis y Comparación de Arquitecturas de Software (SACAM, por sus siglas en inglés)* [Stoermer, Bachmann, y Verhoef, 2003]. Como resultado de la aplicación de este método se asignó una puntuación a cada patrón de acuerdo a su desempeño al abordar las diferentes directrices arquitectónicas.

Los criterios de comparación consistían en una serie de directrices arquitectónicas de

[1] A pesar de que ADD es un método para el diseño de arquitecturas, en este trabajo se emplea en el diseño de un patrón arquitectónico con la intención de generalizar, de tal manera que el resultado de esta investigación pueda ser aplicable no solamente a un caso de estudio en particular.

modificabilidad e integrabilidad seleccionadas a partir de los requerimientos del caso de estudio y como indicadores de satisfacción se utilizaron diversas técnicas de diseño enfocadas a la satisfacción de estos mismos atributos.

1.2 Hipótesis

Es posible diseñar un patrón arquitectónico capaz de brindar un mayor nivel de modificabilidad e integrabilidad en comparación con los patrones arquitectónicos comúnmente aplicados para proyectos ágiles de desarrollo de software.

1.3 Objetivos

El objetivo general de esta investigación consisten en mejorar el nivel de los atributos de calidad de software característicos de las metodologías ágiles de desarrollo de software: modificabilidad e integrabilidad, brindado por los patrones arquitectónicos presentes en el estado del arte, mediante el diseño de un nuevo patrón arquitectónico mejorado.

Para alcanzar dicho objetivo se plantean los siguientes objetivos específicos:

- Identificar entre los patrones arquitectónicos presentes en el estado del arte, aquel con mayor nivel de modificabilidad e integrabilidad. Esto con el propósito de tomarlo como referencia para el diseño y evaluación del patrón arquitectónico propuesto como resultado de esta investigación.
- Diseñar un patrón arquitectónico que facilite el desarrollo de software ágil proporcionando niveles superiores de modificabilidad e integrabilidad, respecto al patrón de referencia identificado.
- Validar que el patrón arquitectónico propuesto efectivamente incrementa los niveles de modificabilidad e integrabilidad en relación al patrón de referencia. Dicha validación ha de llevarse a cabo mediante la aplicación de un método de evaluación de arquitecturas de software reconocido en la literatura.

1.4 Contribuciones

Las principales contribuciones de este trabajo de tesis son las siguientes.

Por un lado la identificación de aspectos de modificabilidad e integrabilidad que pueden ser mejorados en el patrón arquitectónico DCI. Esto es importante debido a que el resaltar estos aspectos permitirá a los arquitectos que echen mano de este patrón, ajustar el diseño de su arquitectura, poniendo especial atención en las áreas de oportunidad identificadas para DCI.

Por otro lado, a partir de dichas áreas de oportunidad, se propone una serie de mejoras con base en diversas técnicas de modificabilidad, resultando en la creación de un nuevo patrón arquitectónico al que denominamos Datos, Contexto e Interacción Mejorados (EDCI).

La intención de crear el patrón EDCI no es solamente cubrir los puntos en los que DCI puede considerarse débil, sino que además se pretende que éste pueda ser utilizado como plantilla para el desarrollo de sistemas de manera ágil. Esto permitirá también reducir la controversia existente entre los seguidores de las metodologías tradicionales y las metodologías ágiles, proporcionando un patrón preparado para enfrentar requerimientos cambiantes, pero que por ser un patrón podrá tomarse tal cual, reduciendo el tiempo invertido en el diseño arquitectónico (principal preocupación desde el enfoque ágil).

1.5 Organización de la tesis

El resto de este documento está conformado por cuatro capítulos más que abordan desde la revisión del estado del arte hasta las conclusiones obtenidas a partir de este trabajo, pasando por la implementación, la evaluación y el análisis de los resultados obtenidos.

El capítulo 2 presenta una serie de conceptos básicos relacionados al tema de investigación con el objetivo de contextualizar y conocer el trabajo relacionado en cuanto a metodologías ágiles de desarrollo de software, patrones arquitectónicos y métodos de evaluación de arquitecturas de software principalmente.

En el capítulo 3 se lleva a cabo el proceso de diseño del patrón EDCI. Para ello se realizó un

análisis del proyecto GRP (caso de estudio) con la finalidad de definir los requerimientos funcionales y no funcionales. Tales requerimientos fueron priorizados para seleccionar aquellos con mayor impacto arquitectónico y de negocio, dando pie a la definición de directrices arquitectónicas.

Posteriormente en el capítulo 4 DCI y EDCI se someten a evaluación por comparación, siguiendo el método de análisis y comparación de arquitecturas de software (SACAM). Este método basa la comparación en las directrices arquitectónicas antes mencionadas, consideradas como criterios de comparación y una serie de técnicas de diseño consideradas como indicadores.

Finalmente en el capítulo 5 se presenta un resumen del trabajo realizado, las conclusiones de este trabajo con base en el análisis de los resultados obtenidos, además de un conjunto de áreas de oportunidad del mismo presentadas a manera de trabajo futuro.

2

Estado del arte

En este capítulo se presenta un análisis histórico, evolutivo y de la actualidad de los conceptos que componen el marco teórico concerniente al presente trabajo de investigación y como éstos se han ido relacionando hasta converger en el punto central de atención de esta tesis.

2.1 Introducción

En los inicios de la industria del software (en la década de 1940) los sistemas que se construían solían ser de tamaño muy pequeño en comparación con los que se desarrollan hoy en día. El propósito de dicho software solía ser tan específico (militar o científico) que para su construcción solía ser suficiente una sola persona o bien un equipo de trabajo muy pequeño.

Por mencionar ejemplos, algunos científicos desarrollaban su propio software con la finalidad de implementar, mediante algún lenguaje de programación, algoritmos que automatizaran sus experimentos o bien, facilitaran la validación de sus hipótesis. Del mismo modo, algunas compañías de hardware incluían el software como complemento de los dispositivos que fabricaban (simuladores o herramientas de configuración, por ejemplo); sin embargo, con el surgimiento de los sistemas

operativos esto cambió.

Poco a poco la industria del software dejó el enfoque de resolver problemas de índole exclusivamente científica, para adoptar problemas de la vida diaria, como automatización de procesos, monitorización de sensores, control automático de vehículos o de maquinaria pesada, administración de inventarios, automatización de transacciones bancarias, etc. Esto derivó en la necesidad de contratar especialistas en desarrollo de software para la resolución de dichos problemas y en la tendencia a construir sistemas de software cada vez más robustos.

Al inicio de la década de 1960 se comenzó a hacer notorio el hecho de que las técnicas de desarrollo de software se estaban quedando atrás en relación al tamaño y complejidad del mismo. No existía una base académica que garantizara la calidad en el desarrollo de software. En otras palabras, cada desarrollador adoptaba un estilo de programación y sus conocimientos se fundamentaban en su propia experiencia.

La falta de educación en cuanto a técnicas de programación y desarrollo de software, causaron diversos problemas, entre ellos, la entrega tardía del software a los clientes, discrepancias entre el producto final y el producto deseado, gran cantidad de defectos encontrados en producción y una nula capacidad de adaptación del software ante situaciones cambiantes del negocio. A este fenómeno se le denominó *crisis del software* [Naur y Randell, 1968].

Para discutir tal fenómeno, la Organización del Tratado del Atlántico Norte (NATO, por sus siglas en inglés) en su división de ciencias, patrocinó una conferencia sobre ingeniería de software [Naur y Randell, 1968] en dónde se discutió como tema principal la crisis del software y la aplicación de procesos de ingeniería que podrían resolver tal crisis.

Con la aplicación de diversas técnicas de diseño, construcción y validación, se pretendía prevenir los problemas englobados por la crisis del software, principalmente aquellos relacionados con la calidad y productividad. Estas técnicas en conjunto dieron origen a los procesos de desarrollo de software.

2.2 Proceso de desarrollo de software

Conforme se acentuaba la necesidad de desarrollar sistemas de software más robustos, surgía también la necesidad de involucrar una mayor cantidad de recursos en la elaboración de productos de software. Los grandes empresarios de la industria empezaban a preocuparse por las grandes inversiones realizadas para la construcción y el escaso retorno de inversión (ROI, por sus siglas en inglés).

Durante la era de mayor crecimiento en cuanto a la demanda del software y teniendo presente los problemas acarreados en la crisis del software, algunos expertos de esa época empezaron a detectar similitudes importantes entre la ingeniería de sistemas de diversa índole (*i.e.,* manufactura) y los sistemas de software. Esto dio paso a la implementación de modelos o marcos de trabajo, probados en otras áreas, para el desarrollo de sistemas de software, con la esperanza de replicar el éxito obtenido en la elaboración de otro tipo de productos.

Dada tal similitud entre la ingeniería de sistemas y el desarrollo de sistemas de software, es que nace la disciplina conocida como *ingeniería de software*.

La definición proporcionada por Ford [2009] del instituto de ingeniería de software de la Universidad de Carnegie-Mellon es la siguiente:

> "La ingeniería de software es una forma de ingeniería que aplica principios de ciencias computacionales y matemáticas a la realización de soluciones efectivas en cuanto a costos, a problemas de software".

Esta disciplina se dio a la tarea de replicar la implementación de procesos para la elaboración de productos en otras ramas de la ingeniería derivando en el *proceso de desarrollo de software*, al cual la IEEE [1990] define como:

> "El proceso mediante el cual las necesidades del usuario son transformadas en un producto de software. Este proceso involucra la transformación de dichas necesidades en requerimientos de software, transformados posteriormente en un diseño, el cual se implementa en un lenguaje de programación, para después ser probado, instalado y liberado para su uso operacional".

A continuación se brinda una breve descripción de cada una de las etapas que conforman el proceso de desarrollo de software.

2.2.1 Análisis de requerimientos

El objetivo de esta fase es obtener una visión completa del problema que se requiere resolver y de las condiciones y limitaciones del entorno en el cual el sistema a desarrollar va a estar operando.

La descripción del problema a resolver o en otras palabras la definición de requerimientos, suele estar compuesta principalmente por: requerimientos de funcionalidad, requerimientos de calidad (no funcionales) y restricciones de diseño o de negocio.

Parte del análisis de requerimientos también incluye un estudio de factibilidad, el cuál tiene como principal objetivo determinar si existe una solución al problema, económica y tecnológicamente viable.

La comunicación entre los distintos involucrados en el desarrollo del sistema es crucial en esta fase, ya que todo debe ser definido de manera bastante clara y concisa, evitando ambigüedades.

2.2.2 Diseño de la aplicación

Durante la fase de diseño, se construye un modelo del sistema el cual, cuando es implementado mediante la aplicación de ciertas herramientas tecnológicas (*i.e.,* un lenguaje de programación), resuelve el problema planteado en el análisis de requerimientos.

Las decisiones tomadas en la fase de diseño tienen un alto impacto en el resto del proceso de desarrollo de software y en la calidad del producto final. Estas decisiones se suelen asentar en una descripción global del sistema conocida como *arquitectura de software*. La arquitectura de software debe ser posteriormente evaluada y usada como plantilla para el desarrollo de sistemas con características similares o bien, como patrón para el desarrollo de componentes reutilizables.

Es a esta etapa del proceso de desarrollo hacia donde este trabajo de investigación se encuentra mayormente dirigido.

2.2.3 Implementación

Durante la fase de implementación, el trabajo se concentra en el "¿cómo?" del problema, es decir, en la definición y aplicación de diversas técnicas, algoritmos y programación para la realización de tareas concretas. Algunos expertos consideran que debe existir una fase intermedia entre el diseño y la implementación cuyo objetivo sea el diseño de algoritmos para cubrir ciertos requerimientos.

En esta fase se recibe como entrada una especificación de diseño y el resultado de la misma es un conjunto de programas ejecutables.

2.2.4 Pruebas o validación

En realidad es una idea completamente incorrecta el considerar esta fase como posterior a la implementación. De hecho, las pruebas, verificación y validación del sistema suelen aplicarse a diversos productos de trabajo a lo largo de todo el proceso de desarrollo.

En esta fase se realiza una validación del comportamiento de todos los componentes e interacciones de los mismos en conjunto. Como resultado de dicha validación se esperan una serie de salidas previamente definidas como resultados deseados.

El objetivo de realizar dicha validación es encontrar discrepancias entre la funcionalidad deseada y los resultados obtenidos de la operación del sistema. Tales discrepancias deben ser eliminadas mediante un proceso de corrección de errores.

2.2.5 Implantación

También denominada fase de instalación. Esta fase consiste en el montaje del sistema de software sobre una arquitectura de hardware en la cual estará operando de manera definitiva. Esta arquitectura de hardware, también es conocida como entorno de producción.

Es en esta fase en la que culmina el ciclo de desarrollo del software.

2.3 Metodologías de desarrollo de software

Una metodología de desarrollo de software tiene como finalidad ofrecer detalles acerca de qué actividades deben de ser realizadas para alcanzar los objetivos de cada una de las fases a lo largo del proceso de desarrollo [Mnkandla, 2009].

En la construcción de software de gran tamaño, como el que en la actualidad se desarrolla, suele involucrarse a un gran número de personas por un periodo largo de tiempo, situación que exige cuidadosa planificación y control. Es allí donde radica la importancia de las metodologías de desarrollo de software.

2.3.1 Metodologías dirigidas por planificación o tradicionales

Este tipo de metodologías se caracteriza por una larga etapa de planificación anticipada y por la generación de documentación extensiva.

El enfoque de este tipo de metodologías es predictivo y esta es la razón por la cual el tiempo invertido en etapas tempranas es elevado.

Es común que debido a este enfoque se genere desesperación por parte del cliente al no percibir un retorno de inversión en las primeras semanas o incluso meses de iniciado el proyecto.

En la mayoría de los proyectos de desarrollo de software, los requerimientos cambiantes suelen ser bastante comunes. Este tipo de metodologías, busca predecir dichos cambios y por ello el tiempo que se invierte en planificación es considerable, sin embargo, visto desde otros enfoques, esto es realmente imposible, ya que los requerimientos suelen ser impredecibles. Por ejemplo, en algunos casos el mismo cliente tiene sólo una idea vaga de lo que necesita del sistema, por lo cual realizar una larga planificación de forma anticipada podría perder un poco de sentido.

2.3.2 Metodologías ágiles de desarrollo de software

Las metodologías ágiles de desarrollo se enfocan en entregar un producto de valor al cliente de manera temprana, y en la rápida adaptación al cambio, dejando de lado la planificación y

documentación extensiva.

La naturaleza de estas metodologías se rige principalmente por los cuatro principios del manifiesto de desarrollo ágil [Beck *et al.*, 2001] anteponiendo:

- Los individuos y sus interacciones sobre los procesos y herramientas
- El software funcional sobre la documentación exhaustiva
- La colaboración del cliente sobre la negociación de contratos
- La rápida respuesta al cambio sobre el seguimiento de un plan

Entre las metodologías ágiles más populares en la actualidad podemos citar el Proceso Unificado de Rational (RUP, por sus siglas en inglés), Programación Extrema (XP, acrónimo comúnmente utilizado para referirse a éste) y SCRUM, mismas que se describen de manera breve a continuación.

2.3.2.1. The Rational Unified Process *(RUP)*

RUP es una metodología que guía el desarrollo de software de manera iterativa enfocada en la participación activa del cliente en la especificación de requerimientos [Vliet, 2008].

Durante la *fase de arranque* se definen los objetivos de la presente iteración. Se identifican los casos de uso críticos, se definen arquitecturas candidatas para el sistema y se estiman tiempos y costos para la siguiente fase. En la *fase de elaboración*, el dominio del problema es analizado, al igual que un prototipo de la arquitectura a implementar. Durante la *fase de construcción* es donde se desarrolla la funcionalidad del sistema y se realiza la integración de los componentes siguiendo la arquitectura definida en la fase anterior. Para finalizar, en la *fase de transición* y teniendo como base la retroalimentación por parte de los usuarios, se realizan correcciones a problemas o se finalizan requerimientos pendientes [Abrahamson, Salo, y Ronkainen, 2002].

2.3.2.2. Extreme Programming *(XP)*

XP se basa en la implementación de técnicas de desarrollo ágiles tales como: la inspección de código mediante programación en parejas, altos niveles de interacción entre el equipo de desarrollo y

el cliente, reducción de los tiempos de reuniones, involucramiento del cliente en algunas decisiones, entre otras.

Las fases del ciclo de vida de desarrollo de software consideradas por esta metodología son: *exploración, planificación, definición de iteraciones a liberar, publicación a producción, mantenimiento y muerte del sistema* (cuando deja de considerarse *en su vida útil*) [Abrahamson *et al.*, 2002].

Desde hace algunos años se han realizado algunos trabajos de investigación con la finalidad de integrar algunas prácticas de las metodologías tradicionales a esta metodología, con el objetivo de mejorarla manteniendo su naturaleza ágil [Nord y Tomayko, 2006].

2.3.2.3. SCRUM

SCRUM se enfoca en la realización de una serie de actividades de administración de proyectos, con la finalidad de detectar deficiencias e impedimentos para el proceso de desarrollo.

No existe en sí una clasificación formal de las actividades o etapas de esta metodología, sin embargo, algunos autores identifican tres de ellas:

La fase de *pre juego*, en donde se llevan a cabo la planeación y el diseño arquitectónico. La fase de *desarrollo*, en donde se prioriza y agrupa a los requerimientos en incrementos denominados *sprints*[1]. Y finalmente la fase de *post juego*, en donde no se recibe un requerimiento más y se enfocan esfuerzos en la integración de los componentes, pruebas del sistema y de integración, corrección de errores y documentación [Abrahamson *et al.*, 2002].

Actualmente, también para SCRUM existen algunos intentos en integrar prácticas de las metodologías tradicionales con el fin de complementar las ventajas que ofrece el propio SCRUM [Jeon, Han, Lee, y Lee, 2011].

2.4 Arquitectura de software

Existen diversas definiciones para el concepto de arquitectura de software, por ejemplo, Bass, Clements, y Kazman [2003] la definen como un conjunto de estructuras de distribución de

[1] Cada *sprint* es sometido a las fases tradicionales del ciclo de vida de desarrollo de software.

componentes e interacciones en un sistema de software. Algunos otros autores suman a la definición de este concepto, elementos como: protocolos de comunicación, mecanismos de sincronización, acceso a datos, distribución física y decisión entre diversas alternativas de diseño.

La IEEE [1990] por su parte integra un punto bastante importante a esta definición: los principios y restricciones que rigen la evolución del software.

Los atributos de calidad del software, así como algunas restricciones o políticas del proveedor o del cliente para el proceso de desarrollo de un sistema de software, son parte de dichos principios y restricciones.

El objetivo primordial de una arquitectura de software es garantizar la satisfacción de los atributos de calidad del software derivados de tales principios.

2.4.1 Atributos de calidad del software

Los atributos de calidad del software representan características deseables para la óptima operación de un sistema de software. Éstos son independientes de la funcionalidad y estrictamente dependientes entre ellos al no ser posible satisfacer uno de ellos sin impactar positiva o negativamente a otro. [Bass *et al.*, 2003].

Existen diversas clasificaciones para los atributos de calidad del software. Bass *et al.* [2003] identifican disponibilidad, modificabilidad, desempeño, seguridad, verificabilidad y usabilidad, como los atributos de calidad principales, de los cuales incluso pueden derivarse algunos otros.

Para cubrir los atributos de calidad deseados, el arquitecto de software debe identificar bien las necesidades de calidad del software, priorizarlas y con base en esa priorización aplicar técnicas de diseño que permitan cubrir las más importantes mediante la implementación de patrones arquitectónicos y de diseño.

2.4.2 Técnicas de diseño arquitectónico

Las técnicas de diseño arquitectónico son transformaciones previamente evaluadas que permiten ejercer cierto control sobre los parámetros de determinado atributo de calidad [Bachmann, Bass, y

Nord, 2007].

La identificación de técnicas aplicables a un sistema depende de la identificación de los parámetros del atributo de calidad a cubrir. No es factible definir las técnicas a utilizar sin antes identificar los parámetros que desean impactarse.

El proceso de la identificación de técnicas aplicables a un sistema, puede realizarse utilizando un modelado del mismo como herramienta (*i.e.* una vista arquitectónica). Tal modelo debe realizarse únicamente con el detalle preciso para la identificación de los parámetros de interés. El tener un modelo más detallado suma parámetros a considerar y en ocasiones puede derivar en una lista interminable y muy compleja de manejar. El arquitecto debe poseer una gran capacidad de abstracción para realizar esta tarea lo más eficientemente posible.

Para el presente trabajo se consideran únicamente técnicas para incrementar los niveles de modificabilidad e integrabilidad, ya que estos atributos de calidad representan las características principales de las metodologías ágiles de desarrollo de software según el manifiesto ágil [Beck *et al.*, 2001].

Habiendo identificado tanto los requerimientos de calidad del software como las técnicas de diseño aplicables para la satisfacción de los mismos, el siguiente paso para un arquitecto de software consiste en identificar los patrones arquitectónicos que implementan dichas técnicas y que en conjunto permitirán alcanzar el objetivo de cubrir los atributos de calidad deseados.

2.4.3 Patrones arquitectónicos

Aún cuando todos los conceptos revisados hasta este punto son esenciales para entrar en contexto, podríamos decir que la verdadera base de esta investigación radica principalmente en dos conceptos: patrones arquitectónicos y evaluación de arquitecturas de software.

De acuerdo con Bass *et al.* [2003] un patrón arquitectónico (también denominado estilo arquitectónico) es una descripción de elementos y relaciones entre ellos gobernados por una serie de restricciones acerca de cómo deben darse tales relaciones.

La importancia de usar patrones arquitectónicos consiste en que estos permiten asegurar que el

sistema sobre el cual son implementados cumple con ciertos atributos de calidad de software (*i.e.* modificabilidad e integrabilidad).

A continuación se describen algunos de los patrones arquitectónicos más comúnmente utilizados para cubrir escenarios de modificabilidad e integrabilidad.

2.4.3.1. Capas

El patrón arquitectónico en capas permite estructurar aplicaciones distribuyendo sus componentes en grupos de subtareas a diferentes niveles de abstracción (capas).

La interacción entre las diferentes capas es restringida, de manera que los componentes ubicados en capas superiores sólo pueden acceder a servicios en capas inferiores [Buschmann, Meunier, Rohnert, Sommerlad, y Stal, 1996].

La división de responsabilidades en diferentes capas permite que cada una pueda desempeñar sus funciones en cierto modo de manera independiente, dotando a este patrón de un grado aceptable de coherencia semántica [2].

Otra característica de este patrón arquitectónico es el escalamiento de niveles de abstracción, esto significa que para las capas inferiores responsables de lidiar con un mayor nivel de complejidad y especificidad existe una capa superior encargada de abstraer su funcionalidad.

La separación de responsabilidades en capas también puede brindar la facilidad de abstraer servicios comunes, de tal manera que se construya una capa específicamente para contener dichos servicios y, en caso de que uno de ellos se deba modificar, los cambios estarían limitados a dicha capa.

El acceso a las responsabilidades públicas de cada capa se logra a través de interfaces, mientras que las privadas quedan encapsuladas dentro del componente que las contiene.

La restricción de canales de comunicación establecida por este patrón permite prevenir la propagación de modificaciones a través de todo el sistema. Las modificaciones realizadas sobre una capa generalmente afectan sólo a sus capas adyacentes.

[2] El concepto de coherencia semántica se refiere a las relaciones entre las responsabilidades de un módulo. Para mantener un buen grado de coherencia semántica, las responsabilidades en un módulo deben operar con el mínimo de dependencias hacia otros módulos [Bass *et al.*, 2003].

2.4.3.2. Tuberías-filtros

El patrón arquitectónico Tuberías-filtros proporciona la estructura a sistemas encargados de procesar flujos de información. Cada etapa de procesamiento es encapsulada en un componente filtro y los datos fluyen a través de tuberías entre filtros adyacentes [Buschmann *et al.*, 1996].

Cada *filtro* equivale a una etapa de procesamiento de datos. Las responsabilidades de cada filtro son semánticamente coherentes, es decir, la manera en que cada filtro manipula los datos es independiente a la implementación de cualquier otro filtro.

Los mecanismos de procesamiento de datos de cada filtro se encuentran encapsulados dentro de éstos. La interacción con filtros adyacentes, se da a través de *interfaces*.

Los canales de comunicación en este patrón se encuentran restringidos en el sentido en que un filtro puede tener una sola entrada y una sola salida, lo cual limita el riesgo de propagación de cambios.

Las *tuberías* actúan como intermediario que rompe algunas dependencias entre filtros adyacentes proporcionando mecanismos de sincronización y *buffering*.

El enlazado entre tuberías y filtros puede crearse al momento en que los filtros son invocados, es decir, realizando un enlazado en tiempo de inicialización.

2.4.3.3. Blackboard

Este patrón arquitectónico es útil para atacar problemas no deterministas mediante la transformación de datos crudos en estructuras de datos de alto nivel. La idea detrás de este patrón arquitectónico es la implementación de programas independientes que trabajan de manera cooperativa bajo una estructura de datos común para proporcionar una posible solución parcial o aproximada al problema [Bachmann *et al.*, 2007; Buschmann *et al.*, 1996; Craig, 1995; Engelmore y Morgan, 1988].

Un componente *blackboard* central evalúa el estado actual de procesamiento y se encarga de coordinar la actividad de los programas especializados.

Cada programa independiente, también denominado *fuente de conocimiento*, es semánticamente

coherente y se comunica con otros programas utilizando un componente blackboard como intermediario restringiendo los canales de comunicación.

Las fuentes de conocimiento suelen registrarse en tiempo de ejecución.

2.4.3.4. Broker

El patrón arquitectónico Broker es utilizado para la estructuración de sistemas distribuidos compuestos por componentes desacoplados que interactúan mediante invocaciones a servicios remotos. El componente *broker* es responsable de coordinar la comunicación, redireccionando peticiones y transmitiendo resultados y excepciones [Buschmann *et al.*, 1996].

Los distintos *servidores* distribuidos que componen el sistema en cuestión, agrupan funcionalidades semánticamente relacionadas y son registrados por el broker en tiempo de ejecución. De esta manera se publican las interfaces de acceso a los servicios expuestos por los servidores. Los *clientes* acceden a estos servicios enviando peticiones al broker, el cuál se encarga de localizar al servidor que atenderá la petición.

La restricción de canales de comunicación también se encuentra presente en este patrón arquitectónico debido a que los servicios son utilizados de manera restringida a través del componente broker.

2.4.3.5. Modelo-Vista-Controlador (MVC)

MVC divide el sistema en tres componentes principales. El *modelo*, que contiene la funcionalidad núcleo y de dominio de negocio; la *vista*, encargada de la presentación de los datos al usuario; y el *controlador*, responsable de administrar las entradas del usuario. La vista y los controladores comprenden la interfaz de usuario [Buschmann *et al.*, 1996; Reenskaug, 1979].

El principal objetivo de esta separación es el desacoplamiento de la vista y el modelo, ya que la primera es más susceptible a cambios y es deseable mantener al modelo exento de los mismos.

MVC asegura la coherencia semántica de cada uno de sus macro componentes que pueden verse también como capas mayormente desacopladas.

El modelo encapsula la funcionalidad y datos de dominio de negocio. El controlador actúa como intermediario entre las entradas del usuario y el modelo. Y finalmente las vistas pueden enlazarse en tiempo de ejecución a los datos.

2.4.3.6. Presentación-Abstracción-Control (PAC)

PAC define una estructura para sistemas de software interactivos en forma de una jerarquía de agentes cooperativos. Cada agente es responsable de algún aspecto de la funcionalidad y está compuesto por tres componentes: *presentación*, *abstracción* y *control*. Esta subdivisión separa los aspectos de interacción humano-computadora del agente, de su núcleo funcional y comunicación con otros agentes [Buschmann *et al.*, 1996].

Los *agentes* son por sí mismos unidades básicas de funcionalidad que procesan información y la comparten con otros agentes. Para realizar tal procesamiento, las responsabilidades del agente deben ser semánticamente coherentes desde el momento en que el agente no requiere interactuar con otros para procesar la información, sino sólo para la recepción y transmisión de los datos. Cada agente es responsable de encapsular las implementaciones privadas que posea.

Existe cierto escalamiento de niveles de abstracción en este patrón dado el simple hecho de tratarse de un sistema jerárquico de distribución de agentes (similar a lo que ocurre con el patrón en capas). Los niveles superiores en la jerarquía suponen un nivel de abstracción mayor correspondiente a los agentes ubicados en niveles inferiores.

PAC es un modelo jerárquico en el cual cada capa agente en cierto nivel de la jerarquía actúa como intermediario entre su propio agente padre e hijo.

2.4.3.7. Micronúcleo

Este patrón arquitectónico es aplicado a sistemas de software susceptibles a cambios constantes en la funcionalidad. Separa un núcleo de funcionalidad mínima, de otros componentes específicos a cierto dominio de negocio [Buschmann *et al.*, 1996].

El *micronúcleo* implementa servicios semánticamente coherentes que son extendidos por *servidores internos y externos.*

Los servicios implementados por el micronúcleo se encuentran de alguna manera centralizados para el consumo por parte de otros servidores a manera de servicios comunes.

El micronúcleo actúa como intermediario entre *clientes, servidores y adaptadores* restringiendo los canales de comunicación.

2.4.3.8. Reflexión

Este patrón arquitectónico proporciona un mecanismo para el manejo dinámico de cambios en la estructura y comportamiento de un sistema de software. En este patrón la aplicación se divide en un *meta nivel* que provee información acerca de ciertas propiedades del sistema y en un *nivel base* en donde la lógica de la aplicación es incluida [Buschmann *et al.*, 1996].

En este patrón la coherencia semántica se logra por medio de la división de los componentes del sistema en *meta-objetos* (susceptibles a cambios) y *objetos base* (lógica de negocio). La lógica de negocios por su parte, se encuentra encapsulada (oculta) de los meta objetos y es accesible sólo a través de interfaces de interacción.

2.4.3.9. Datos, Contexto e Interacción (DCI)

DCI es un patrón de reciente inclusión en el estado del arte, cuyo principal objetivo es mejorar la relación entre los modelos mentales del usuario y la implementación en código de los requerimientos funcionales simplificando la comprensión del código y tareas como las pruebas unitarias y la auditoría del mismo [Coplien y Bjrnvig, 2010].

DCI es considerado un complemento de MVC y está compuesto por cinco capas:

Los *Datos*, que representan las entidades del negocio (lo que el sistema es). La *Interacción* en donde se implementa la lógica del negocio de la aplicación (lo que el sistema hace) en términos de roles en lugar de objetos puros. El *Rol* encargado de definir el comportamiento que puede adoptar un objeto bajo cierto contexto determinado. El *Contexto* responsable de relacionar los objetos participantes en una interacción con los roles que representan y que es ejecutado desde el *Controlador*. A su vez el controlador tiene la responsabilidad de procesar las peticiones y eventos generados por la vista.

Finalmente, la *Vista* es la responsable de capturar eventos de usuario para ser procesados por el controlador.

Este patrón garantiza coherencia semántica al separar lo que el sistema hace de lo que el sistema es, manteniendo los componentes susceptibles a cambios separados del modelo y fácilmente localizables. Además, provee encapsulamiento al ocultar las implementaciones privadas, dejando accesibles únicamente las interfaces de comunicación entre sus componentes. Estas interfaces están restringidas por las reglas de implementación del patrón, es decir, mantienen un orden de flujo específico.

Al ser un complemento de MVC, comparte con éste el mismo nivel de abstracción ubicado en el controlador, sin embargo, en contraste, DCI agrega más capas intermedias entre la vista y el modelo (controladores, contextos e interacciones), facilitando la localización de los puntos de impacto a raíz de cambios en los requerimientos, además del establecimiento de las relaciones entre los objetos y los roles que éstos pueden llegar a tener bajo diversos contextos.

2.4.4 Métodos de evaluación de arquitecturas de software

Al igual que el concepto de patrones arquitectónicos, el concepto de evaluación de arquitecturas de software es también de vital importancia ya que una vez finalizado el diseño del nuevo patrón EDCI, mediante la aplicación de ciertos métodos de evaluación de arquitecturas, fue posible medir y validar las mejoras proporcionadas por éste en relación al patrón DCI.

Roy y Graham [2008] proporcionan una visión general acerca de los métodos de evaluación de arquitecturas de software definiendo a la evaluación de arquitecturas de software como una técnica que determina el grado en el que una arquitectura o bien un patrón arquitectónico satisface un conjunto específico de atributos de calidad de software. En algunos casos la evaluación de arquitecturas suele ser también empleada para determinar el impacto que produce la satisfacción de cierto atributo de calidad en relación a otro, lo que se conoce como compromiso o *Trade-off*.

El proceso de evaluación de arquitecturas toma como entradas el problema que se pretende atacar, los objetivos del negocio, la especificación de requerimientos y la documentación de diseño existente,

arrojando como resultado además de la valoración de la arquitectura o patrón, documentación valiosa y mejor organizada acerca los mismos, lo cual es muy útil para las fases restantes del proceso de desarrollo.

El equipo responsable de realizar la evaluación suele estar conformado por diferentes clases de expertos como son: administradores de proyecto, administradores de sistemas, arquitectos de software, analistas, diseñadores, programadores y en algunos casos incluso usuarios finales.

Una arquitectura de software puede ser evaluada antes (evaluación temprana) o después (evaluación posterior) de su implementación. La evaluación temprana suele emplearse con el objetivo de seleccionar la arquitectura que en mayor grado satisfaga los requerimientos de un sistema a desarrollar, mientras que la evaluación posterior es más utilizada para evaluar decisiones de diseño ya implementadas con la finalidad de medir el desempeño y detectar áreas de mejora.

El presente trabajo de investigación se enfoca al diseño previo a la construcción del software, por lo tanto, los métodos de evaluación temprana son los que mejor se alinean a nuestros objetivos.

La mayoría de los métodos de evaluación temprana se clasifican como métodos basados en escenario [3]. Esta clase de métodos evalúa la arquitectura de un sistema con base en su capacidad de respuesta ante un escenario en particular. Si la capacidad de respuesta resulta ser pobre, algunos de estos métodos pueden proveer ideas o sugerir técnicas que permitan elevar el nivel de dicha capacidad de respuesta.

A continuación, se listan los métodos de evaluación temprana más comunes, comenzando con el que según Roy y Graham [2008] es considerado el padre de los métodos de evaluación de arquitecturas basados en escenarios.

2.4.4.1. Scenario-based Software Architecture Analysis Method (SAAM)

SAAM fue inicialmente propuesto por [Kazman, Bass, Abowd, y Webb, 1994]. Su objetivo es cotejar los principios y supuestos de las arquitecturas en contra de la documentación que describe la funcionalidad y requerimientos de calidad deseados para el sistema de software a construir.

Los productos resultantes de la aplicación de este método incluyen, escenarios sensibles a la

[3] Un escenario es una descripción breve de una interacción entre un actor con un sistema.

calidad del software, una relación entre dichos escenarios y elementos de una arquitectura, además de una estimación de esfuerzo requerido para la aplicación de técnicas que garanticen la cobertura de dichos escenarios.

Aún cuando SAAM es un buen método de evaluación y pudiera ser usado para evaluar EDCI, existen algunos métodos de evaluación más concretos que pueden ser más adecuados para realizar un análisis con base en escenarios de modificabilidad.

2.4.4.2. Architecture Trade-off Analysis Method (ATAM)

En contraste con SAAM, ATAM [Bass *et al.*, 2003] proporciona una técnica basada en principios para evaluar una arquitectura con respecto a atributos de calidad opuestos [4].

Este método consiste en nueve actividades comenzando con la definición de objetivos de negocio que son posteriormente refinados en atributos de calidad, detallados en escenarios generales de calidad y refinados nuevamente en escenarios concretos de calidad para ser finalmente priorizados.

En esta tesis nuestro enfoque se dirige a los atributos de calidad de software de modificabilidad e integrabilidad, por lo que, al no ser opuestos, las características de este método no serían del todo aprovechadas.

2.4.4.3. Architecture-Level Modifiability Analysis (ALMA)

ALMA Bengtsson, Lassing, y Bosch [2002] es un método de evaluación específicamente diseñado para evaluar escenarios de modificabilidad. Está dirigido a la evaluación de riesgos y comparación de arquitecturas con base en la medición del costo de asegurar la satisfacción de escenarios de modificabilidad y el costo total de esfuerzo de mantenimiento, el cual es calculado con base en el tamaño del componente y pesos asignados a cada escenario.

Este método es un serio candidato para la evaluación de EDCI.

[4] Contrarios en el sentido que el satisfacer un atributo de calidad compromete el grado de satisfacción de otro atributo.

2.4.4.4. SAAM for Complex Scenarios (SAAMCS)

SAAMCS fue propuesto por Lassing, Rijsenbrij, y Vliet [1993] con la intención de evaluar escenarios con un alto nivel de complejidad. Está basado en el nivel de impacto del escenario sobre la arquitectura del sistema, necesidades de coordinación y la presencia de conflictos de versiones.

El caso de estudio utilizado en nuestra investigación no considera escenarios altamente complejos, de tal manera que la aplicación de SAAMCS para su evaluación puede no ser adecuada.

2.4.4.5. Software Architecture Comparison and Analysis Method (SACAM)

SACAM [Stoermer *et al.*, 2003] es un método que al igual que ALMA, fue diseñado con fines comparativos y que proporciona las bases para la selección de arquitecturas de software.

El proceso de SACAM inicia con la definición de criterios de comparación (objetivos de negocio) que son posteriormente refinados en escenarios de calidad (directrices arquitectónicas) los cuales se priorizan.

Para poder llevar a cabo la aplicación de SACAM se requiere contar con una serie de vistas, métricas y documentación homogéneas entre las arquitecturas o patrones candidatos con el fin de tener parámetros claramente comparables.

Tomando los artefactos antes mencionados como herramientas se modela cada una de las directrices arquitectónicas para evaluar el comportamiento de cada patrón. Esto se realiza asignando en consenso (por un equipo de expertos) una puntuación a cada patrón que indica el grado de satisfacción de cada directriz.

Dado que SACAM es también un método de evaluación comparativo, es otro candidato para la evaluación del resultado de este trabajo de investigación.

Existen algunos otros métodos dirigidos a evaluar ciertos atributos de calidad en particular; SALUTA (usabilidad)[Folmer, Van Gurp, y Bosch, 2005] por ejemplo. Sin embargo, estos métodos no han sido considerados, dado que los atributos de calidad fijados como meta en esta investigación son modificabilidad e integrabilidad.

Resumen del capítulo

En este capítulo se revisaron algunos antecedentes históricos relacionados a la investigación con la finalidad de contextualizar en el tema. Una vez en contexto se definieron algunos conceptos clave para el desarrollo de esta tesis. Finalmente, se profundizó en los conceptos de patrones arquitectónicos y métodos de evaluación de arquitecturas de software, presentando los trabajos relacionados a esta investigación. En el siguiente capítulo se detalla el proceso de diseño de un nuevo patrón arquitectónico que representa el enfoque de solución al problema de investigación planteado en esta tesis.

3

Diseño del patrón arquitectónico EDCI

Una vez analizado el estado del arte y con el conocimiento base acerca de los patrones arquitectónicos relacionados a los atributos de modificabilidad e integrabilidad es tiempo de abordar el proceso de diseño del patrón arquitectónico propuesto en esta tesis (EDCI).

3.1 Introducción

En este capítulo se presenta el proceso de diseño del patrón arquitectónico EDCI. Para llevar a cabo dicho proceso, se involucran dos elementos clave.

El primero consiste en un método de diseño de software denominado Diseño Dirigido por Atributos (ADD). Este método fue elegido para llevar a cabo el diseño de EDCI debido a su enfoque a la satisfacción de atributos de calidad de software específicamente requeridos para un proyecto. En este caso, los atributos de calidad que se persiguen son modificabilidad e integrabilidad por representar adecuadamente algunas de las características más importantes del software ágil.

El método ADD fue propuesto por Wojcik *et al*. [2006], consta de siete fases y recibe los requerimientos funcionales, no funcionales y algunas restricciones de diseño de especial consideración

como entradas principales. Estas entradas son clasificadas y priorizadas para, a partir de ellas, generar directrices arquitectónicas. Con base en dichas directrices se llevó a cabo la selección de un patrón arquitectónico base, para el diseño de un nuevo patrón capaz de satisfacer las directrices generadas.

Después de la selección de DCI como patrón base se realizó un análisis para identificar posibles áreas de oportunidad de dicho patrón, con el objetivo de cubrirlas mediante la aplicación de algunas técnicas propuestas por Bachmann *et al*. [2007], para así derivar en el nuevo patrón EDCI.

El segundo elemento clave consiste en un Sistema de Planificación de Recursos de Gobierno (GRP), el cual funge como caso de estudio y fuente para la obtención de las entradas necesarias por el método ADD. El caso de estudio GRP es un proyecto de software actualmente en proceso de desarrollo para el Centro de Investigación y de Estudios Avanzados del Instituto Politécnico Nacional (CINVESTAV) en México [1].

Como parte del plan de desarrollo de la unidad Tamaulipas, CINVESTAV busca incrementar la eficiencia de sus procesos en áreas tales como: control presupuestal, tesorería y caja, recursos humanos, adquisiciones, inventarios, contabilidad, administración de proyectos, entre otras, a través de la automatización de los procesos involucrados en dichas áreas a través del sistema GRP.

Actualmente en CINVESTAV se llevan a cabo alrededor de 120 procesos administrativos agrupados en 13 áreas. La gran cantidad de procesos por automatizar como parte del desarrollo del sistema GRP, ha llevado a los directivos del Instituto a considerar que el proyecto GRP se realice siguiendo una metodología ágil de desarrollo de software. De esta manera, sería posible construir un núcleo básico de funcionalidad operable al cien por ciento, en un lapso de tiempo conveniente y posteriormente poder ir agregando nuevos módulos y componentes de software, con la finalidad de extender las capacidades funcionales de GRP.

La Tabla 3.1 muestra el acta de constitución del proyecto en donde se especifican entre otros puntos la justificación, los objetivos, una descripción a alto nivel, los participantes, los entregables y los principales riesgos para el desarrollo del proyecto GRP.

[1] CINVESTAV es un instituto de investigación reconocido a nivel internacional, enfocado en un amplio conjunto de disciplinas científicas y tecnológicas. La unidad de más reciente creación de esta institución, se encuentra localizada en Tamaulipas, México y se especializa en líneas de investigación afines a las ciencias de la computación.

Titulo del Proyecto:	**Governance Resource Planning**		
Administrador de Proyecto:	Emigdio Hernández	**Cliente:**	CINVESTAV
Liberación Fase 1:	8 Semanas	**Liberación Fase 2:**	56 Semanas
Justificación:	Actualmente CINVESTAV en su Unidad Tamaulipas, lleva acabo alrededor de 121 procesos divididos en 13 áreas. La ejecución de algunos de estos procesos o parte de ellos, se lleva a cabo de manera manual (incluyendo el manejo de documentos físicos y procesos manuales de autorización de flujos de trabajo) o mediante software legado de la institución (para algunos procesos en las áreas académica, financiera y de inventarios). En la búsqueda de aprovechar al máximo la tecnología con la que se cuenta, CINVESTAV requiere de un sistema de tipo GRP que le permita automatizar sus procesos de gestión financiera y administrativa mejorando la eficiencia de sus recursos, procesos, usabilidad y garantizando una sutil pero efectiva transición a la implementación de este GRP: ▪ Simplificación y estandarización de procesos ▪ Administración de la arquitectura y transición de datos ▪ Tomar ventaja del enfoque *Software as a Service (SaaS)* ▪ Facilidad de aceptación al cambio y administración efectiva del mismo		
Objetivos:	▪ Contar con una herramienta que permita optimizar los procesos de CINVESTAV ▪ Implementar de manera efectiva la nueva herramienta garantizando la aceptación por parte de los usuarios el banco de información con el que actualmente se cuenta		
Descripción de Proyecto a Alto Nivel:	El sistema GRP consiste en una solución integral de tipo para la automatización de procesos de gestión financiera y administrativa de CINVESTAV en su Unidad Tamaulipas enfocado a las áreas: financiera, administrativa, académica, de adquisiciones y administración de proyectos, siguiendo los principios de gobierno de tecnologías de la información.		

Continúa en la siguiente página...

...Continúa de la página anterior

Participantes:	▪ Dr. Arturo Díaz Pérez - Director de CINVESTAV. ▪ Diego de la Rosa - Administrador de proyecto CINVESTAV. ▪ Emigdio Hernández - Administrador de proyecto. ▪ Diego Torres - Líder técnico de proyecto. ▪ Calixto Conde - Analista de negocios. ▪ Fernando Castañeda - Líder de pruebas de calidad.
Entregables:	Todos los entregables deberán ser firmados y autorizados por el Dr. Arturo Díaz y Diego de la Rosa. ▪ Documentación de descripción de procesos de negocio ▪ Plan de proyecto detallado por fases según implementación ▪ Especificaciones de requerimientos ▪ Documento de diseño de alto y bajo nivel ▪ Estrategia de integración y migración de datos ▪ Reporte de casos de prueba ▪ Entorno para la ejecución de pruebas de aceptación de usuario ▪ Sistema instalado en ambiente de producción ▪ Ejecución de estrategia de entrenamiento
Riesgos:	▪ Alto - Complejidad de los procesos para finalizar en el tiempo estipulado. ▪ Alto - Estimaciones afectadas por posibles variaciones o solicitudes de cambio en los requerimientos iniciales. ▪ Medio - Disponibilidad de los participantes del proyecto.
Patrocinador del Proyecto:	Dr. Arturo Díaz Pérez.

Tabla 3.1: Acta de constitución del proyecto.

Para llevar a cabo este proyecto, CINVESTAV contrató los servicios de consultoría y desarrollo de software de la empresa SVAM International de México.

Al momento en el que se inició esta investigación el desarrollo del proyecto GRP se encontraba

apenas en proceso de análisis. Esto complicó un poco el arranque de la investigación dado que aún había muchas cuestiones, relacionadas al proyecto, por definir.

No se contaba con un documento formal de especificación de requerimientos que sirviera como entrada al proceso de diseño. Tampoco se tenía un documento de diseño que diera alguna idea de la arquitectura que se pretendía implementar. Únicamente se contaba con una serie de diagramas y listados de los procesos de negocio de CINVESTAV.

Estos diagramas y listados significaron el punto de partida para el desarrollo de la investigación, sin embargo, fue necesario invertir tiempo en la revisión de los mismos, además de en algunas tareas para completar las entradas requeridas por ADD. Un ejemplo de estas tareas son los talleres de atributos de calidad (QAW, por sus siglas en inglés) [Barbacci *et al.*, 2002] para la obtención y clasificación de los requerimientos no funcionales del GRP.

A continuación se detalla el proceso de diseño de EDCI, desde las tareas realizadas para completar las entradas requeridas, hasta la obtención del patrón arquitectónico.

3.2 Confirmar suficiencia de requerimientos

Acorde al método ADD, la primera actividad a realizar consistía en verificar que los requerimientos del proyecto GRP fueran suficientes para llevar acabo el proceso de diseño.

3.2.1 Definición de requerimientos funcionales

Considerando el basto número de procesos a automatizar y las restricciones temporales para la realización de esta investigación, se optó por trabajar únicamente con un conjunto muestra de los requerimientos funcionales, que incluyera los requerimientos de mayor prioridad para CINVESTAV y que además permitiera la representación de escenarios de modificabilidad e integrabilidad.

Para obtener este conjunto muestra se realizó un análisis independiente[2] sobre la documentación existente con el fin de identificar las relaciones existentes entre los procesos, definir el orden en

[2]El método de análisis independiente consiste en realizar una revisión de la documentación existente con la finalidad de transformar dicha información en datos de interés acorde a un objetivo en particular[Runeson y Höst, 2009].

el que éstos debían ser implementados y asignar un esfuerzo estimado en horas-hombre para su implementación.

El grafo de la Figura 3.1 ilustra un ejemplo de la metodología propuesta para este análisis. Los vértices del grafo representan procesos del sistema etiquetados con el nombre y un par de valores que denotan el orden de prioridad de implementación y un peso asignado acorde al esfuerzo estimado de implementación del proceso. Las aristas denotan la relación entre los procesos y su dirección (flechas) representa el sentido de la dependencia. Una arista saliente de un vértice significa que el proceso origen, depende del proceso hacia el cuál se dirige la arista. Una arista entrante denota que uno o varios procesos dependen del proceso representado por el vértice en cuestión.

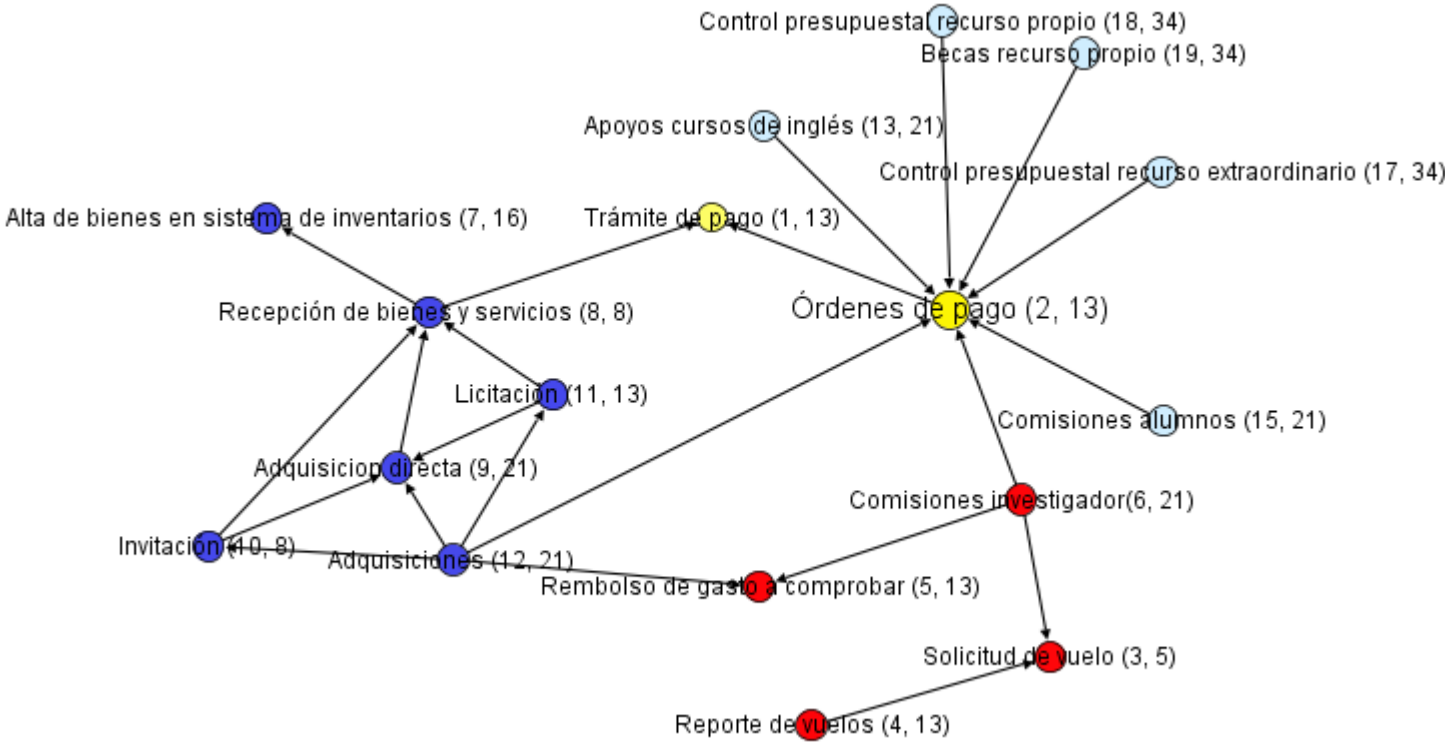

Figura 3.1: Grafo que representa la metodología empleada para el análisis de interdependencias de los procesos del proyecto GRP.

El primer paso en esta metodología consistió en identificar de entre los procesos del proyecto GRP, aquel con mayor número de procesos dependientes de él (se toma este proceso dado que para

poder implementar la mayoría del resto de los procesos, sería necesario que éste ya se encuentre implementado), en este caso el proceso *órdenes de pago* cuyos procesos dependientes se encuentran caracterizados en color azul claro.

Visitando el proceso identificado, se verificó que éste no dependiera de algún otro proceso en cuyo caso se le debía asignar un nivel de prioridad y un peso acorde al esfuerzo estimado para su implementación, de lo contrario, había que visitar el proceso hacia el cual existiera dependencia. Acorde al ejemplo ilustrado, órdenes de pago depende de el proceso *trámite de pago*, por lo que se visitó entonces éste y se analizó si a su vez dependía de otro proceso.

Dado que trámite de pago no depende de ningún otro proceso, entonces se le asigna un nivel de prioridad de 1 y un peso de 13, que representa el número de horas-hombre estimado para su implementación. Una vez asignados la prioridad y el peso para trámite de pago, éste se agrega a la muestra de procesos a utilizar[3]. La asignación del peso fue realizada siguiendo la técnica de *Planning Poker* (ver Anexo A).

Habiendo agregado trámite de pago a la muestra, se procedió a visitar nuevamente el proceso de órdenes de pago y se analizó si dependía de otro además de trámite de pago, en cuyo caso debió visitarse dicho proceso como en el caso anterior, simulando un recorrido de árbol en profundidad. Sin embargo, dado que no existía otro proceso del cuál dependiera órdenes de pago, se le asignó un nivel de prioridad de 2 y un peso de 13 y se agregó a la muestra. Esta primera parte de la metodología es representada en el grafo por los vértices color amarillo.

Ya que hasta este punto las dependencias de órdenes de pago se encontraban ya resueltas, se optó por comenzar a visitar aquellos procesos que dependieran de él. Para determinar cuál de los procesos dependientes de órdenes de pago se analizaría en primer lugar, se le dio preferencia a aquellos procesos que en un pre análisis realizado por el equipo de desarrollo[4], resultaron de menor complejidad. Si en lugar de esto, se hubiese dado prioridad a procesos con alto nivel de complejidad, el tamaño de la muestra de procesos hubiese sido menor. Tomando lo anterior en consideración, el

[3]Este proceso se realizó de manera recursiva hasta no encontrar dependencias

[4]Es importante destacar que dicho pre análisis fue útil para conducir esta metodología, sin embargo, las estimaciones oficiales fueron resultado de la aplicación del método de Planning Poker. Después de aplicar dicha técnica, algunas estimaciones fueron confirmadas, mientras que otras tuvieron que ser ajustadas.

siguiente proceso a analizar fue *comisiones investigador*.

Para éste proceso se identificaron tres dependencias: hacia *rembolso de gastos a comprobar*, hacia *solicitud de vuelo* y hacia *órdenes de pago* (dependencia que ya había sido resuelta previamente). Primeramente se procedió a analizar el proceso de *solicitud de vuelo* y ya que éste último no dependía de algún otro proceso, se le asignó una prioridad de 3 y un peso de 5 y se agregó a la muestra.

El siguiente proceso a considerar fue una excepción a la metodología seguida. La razón por la cuál se hizo esta excepción, es debido a que era de vital importancia dado, el tipo de proyecto del caso de estudio, considerar un proceso que implicara la realización de reportes. Es por ello que se tomó el proceso de *reportes de vuelo*, el cuál también fue visitado en busca de dependencias. Al verificar que reportes de vuelo no dependía de otros procesos, se le asignó una prioridad de 4, un peso de 13 y se agregó a la muestra de procesos.

A continuación se visitó nuevamente el proceso de solicitud de vuelo mismo que ya había sido analizado y agregado a la muestra de procesos. Posteriormente se visitó nuevamente el proceso de *comisiones investigador* y se procedió a analizar su dependencia hacia *rembolso de gasto a comprobar*. Ya que éste último no dependía de otro proceso se le asignó una prioridad de 5, un peso de 13 y se agregó a la muestra, para posteriormente visitar el proceso de *comisiones investigador*, al cual se le asignó una prioridad de 6 y un peso de 21 para así agregarlo también a la muestra de procesos. Esta parte de la metodología está representada en el grafo por los vértices en color rojo.

Finalmente se retornó al proceso de *órdenes de pago* y se procedió a seleccionar otro nodo que dependiera de éste (en este caso el proceso de *adquisiciones*) y se procedió siguiendo la misma metodología. A esta parte de la metodología seguida se le caracterizó con un color azul.

Para delimitar el número de procesos a incluir en el conjunto muestra se fijó un umbral aproximado de 360 horas-hombre, es decir que, la suma de los pesos asignados a los procesos de la muestra no debe exceder esta cantidad. Este umbral de tiempo fue definido con base en el tiempo planeado para la fase de implementación y experimentación de esta investigación.

La Tabla 3.2 lista los requerimientos considerados dentro del conjunto muestra. La prioridad representa el orden en el que deben de ser implementados los procesos y el paso, la estimación de esfuerzo requerido para implementar cada proceso.

Área	Proceso	Prioridad	Peso
Control presupuestal	Requisiciones	16	8
	Ordenes de pago	2	13
	De recurso propio	18	34
	De recurso extraordinario	17	34
Académicos	Becas rec. propio	19	34
	Apoyos inglés	13	21
Tesorería y caja	Tramite de pago	1	13
	Rembolso de gasto por comprobar	5	13
Recursos humanos	Solicitudes de vuelo	3	5
	Reportes de vuelos	4	13
	Comisiones administrativo	14	21
	Comisiones investigadores	6	21
	Comisiones alumnos	15	21
Adquisiciones	Recepción de bienes y servicios	8	8
	Adquisición directa	9	21
	Licitación	11	13
	Invitación	10	8
	Adquisiciones	12	21
Inventarios	Alta en inventarios	7	16

Tabla 3.2: Conjunto muestra de requerimientos funcionales resultante del análisis de interdependencias.

Además del análisis de interdependencias, otro criterio a considerar para la priorización de requerimientos funcionales fue la prioridad establecida por el cliente.

Para cubrir este criterio, una vez adquirida la muestra ordenada de procesos a implementar, ésta se validó con el administrador del proyecto GRP por parte de CINVESTAV. Como resultado, se realizaron algunos ajustes en las prioridades iniciales y algunos procesos fueron descartados o bien remplazados por otros[5]. La Tabla 3.3 muestra el resultado de la revisión con el cliente.

[5]El descarte y remplazo de procesos se dio, ya que el administrador del proyecto consideró que existían ciertos procesos no incluidos en el análisis de interdependencias que tenían mayor prioridad, así como algunos incluidos, que desde su perspectiva no eran tan necesarios en primera instancia.

Área	Proceso	Prioridad	Peso
Control presupuestal	Ordenes de pago	2	13
	De recurso propio	20	34
	De recurso extraordinario	16	34
Tesorería y caja	Tramite de pago	1	13
	Rembolso de gasto por comprobar	5	13
	Tramite recibo de recurso extraordinario	14	8
	Tramite recibo de recurso propio	18	8
Recursos humanos	Solicitudes de vuelo	3	5
	Reportes de vuelos	4	13
	Comisiones investigadores	6	21
Adquisiciones	Recepción de bienes y servicios	8	8
	Adquisición directa	9	21
	Licitación	11	13
	Invitación	10	8
	Adquisiciones	12	21
Inventarios	Alta en inventarios	7	16
Contabilidad	Recibo institucional	13	5
	Factura electrónica	17	21
Administración de proyectos	De recurso propio	19	55
	De recurso extraordinario	15	55

Tabla 3.3: Conjunto muestra de requerimientos funcionales resultante del análisis de interdependencias después de la revisión por parte del cliente.

Una vez validados los requerimientos funcionales fue necesario realizar una segunda clasificación para reflejar tanto el descarte de procesos como la asignación de valores cualitativos (A = Alto; M = Medio; B = Bajo) [Wojcik *et al.*, 2006].

Esta asignación de valores cualitativos tiene el efecto de ordenar parcialmente los requerimientos con base en dos parámetros, el impacto a nivel de negocio (definido por el cliente)[6] y el impacto

[6]En función a la Tabla 3.3 en donde los primeros siete valores representan un valor alto, los siguientes siete un valor medio y el resto un valor bajo.

a nivel arquitectónico (definido por el arquitecto de software)[7]. Con ello fue posible identificar aquellos requerimientos que pudieran representar directrices arquitectónicas para el diseño del patrón propuesto.

En la Tabla 3.4 se listan los requerimientos funcionales que conforman el conjunto muestra final, incluyendo valores cualitativos.

Área	Proceso	Impacto Negocio	Impacto Arquitectura
Control presupuestal	Ordenes de pago	A	A
	De recurso propio	B	M
	De recurso extraordinario	B	M
Tesorería y caja	Tramite de pago	A	A
	Rembolso de gasto por comprobar	A	A
	Tramite recibo de recurso extraordinario	M	M
	Tramite recibo de recurso propio	B	M
Recursos humanos	Solicitudes de vuelo	A	M
	Reportes de vuelos	A	A
	Comisiones investigadores	A	B
Adquisiciones	Recepción de bienes y servicios	M	A
	Adquisición directa	M	A
	Licitación	M	B
	Invitación	M	B
	Adquisiciones	M	A
Inventarios	Alta en inventarios	A	A
Contabilidad	Recibo institucional	M	B
	Factura electrónica	B	M
Administración de proyectos	De recurso propio	B	M
	De recurso extraordinario	B	M

Tabla 3.4: Priorización de requerimientos funcionales acorde al método ADD..

[7]En función de la relación existente entre los requerimientos y los atributos de modificabilidad e integrabilidad.

3.2.2 Definición de requerimientos no funcionales

Una vez validadas las necesidades de negocio (requerimientos funcionales), se definieron las necesidades de calidad (requerimientos no funcionales) para el caso de estudio. Para estos requerimientos fue necesario generar nueva documentación, dado que no se tenía especificación alguna de los mismos, ni material que pudiera ayudar a inferirlos, como en el caso de los requerimientos funcionales.

Para generar dicha información se llevó a cabo una técnica de recolección de información directa [Runeson y Höst, 2009] conocida como *Taller para la Definición de Atributos de Calidad (QAW)* [Barbacci *et al.*, 2002]. Esta técnica consiste en el planteamiento de una lluvia de ideas en donde los involucrados en el proyecto externan tantos escenarios de calidad como crean que se pudieran presentar en el sistema. Una vez hecho esto, los escenarios obtenidos son clasificados de tal manera que a partir de un grupo de ellos pueda obtenerse una generalización (un único escenario representativo de dicho grupo).

La Tabla 3.5 muestra una plantilla utilizada para generar escenarios de calidad durante el QAW.

Cabe destacar que la ejecución de esta técnica fue intencionalmente orientada a la identificación de requerimientos no funcionales relacionados con modificabilidad e integrabilidad, debido a la relevancia que estos atributos de calidad tienen para la realización de esta investigación. Además, el tiempo de respuesta para los escenarios identificados fue establecido indirectamente de forma relativa al tiempo de respuesta del patrón arquitectónico de referencia o base para el diseño del patrón propuesto en esta investigación.

La Tabla 3.6 muestra el resultado del QAW, en donde se obtuvo una serie de requerimientos no funcionales en forma de escenarios de calidad [Bass *et al.*, 2003], que fueron priorizados de forma cualitativa similar a los requerimientos funcionales (A = Alto; M = Medio; B = Bajo).

Componente	Valor
Fuente de estímulo	Entidad (humano, sistema computacional, o algún otro actor) que genera el estímulo.
Estímulo	Condición que debe ser considerada al arribar al sistema (al presentarse).
Entorno	El estímulo se presenta bajo ciertas condiciones. El sistema puede estar sobrecargado o en plena ejecución, son algunos ejemplos de entorno.
Artefacto	Es la entidad que es afectada por el estímulo. Puede ser todo el sistema o alguno de sus componentes.
Respuesta	Es la actividad que tiene lugar al presentarse el estímulo en el sistema.
Tiempo de respuesta	Cuando el estímulo ocurre, la respuesta al mismo debe ser medible con el objetivo de determinar si la directriz se satisface o no. Por ejemplo, para escenarios de desempeño pueden aplicarse técnicas para medir el tiempo de ejecución del escenario de calidad propuesto, mientras que para escenarios de modificabilidad es más común medir el tiempo en el que el cambio descrito por un escenario de calidad en particular es implementado.

Tabla 3.5: Plantilla para documentación de escenarios de calidad de software utilizada en QAW.

3.2.3 Definición de restricciones de diseño

En cuanto a restricciones de diseño específicas para el caso de estudio se definieron como tales la implementación incremental de procesos y el uso de tecnología Microsoft (dada la experiencia del equipo de desarrollo en plataformas Microsoft).

La Tabla 3.7 presenta las restricciones antes mencionadas.

Atributo	Escenario	Impacto Negocio	Impacto Arquitectura
Modificabilidad	Edición de especificación de reportes	A	A
	Edición de flujos de autorización	A	A
	Implementación de firma electrónica para cheques y pólizas	B	A
	Agregar nuevos tipos de beca a tramitar	B	B
	Envío de documentos por correo electrónico	M	M
	Almacenamiento de facturas XML y PDF	M	M
Integrabilidad	Modificación de funcionalidad en un sistema de terceros	A	A
	Integración del módulo de contabilidad	A	A
	Integración con sistema de control escolar	A	A
	Integración con portal COMPRANET	B	B
	Integración con sistema de inventarios	B	M
	Generación de facturas desde sistema CONTPAQ	M	M

Tabla 3.6: Priorización de requerimientos no funcionales acorde al método ADD.

Restricción	Impacto Negocio	Impacto Arquitectura
Implementación incremental de procesos	A	A
Plataforma de desarrollo Microsoft (.Net Framework)	A	A

Tabla 3.7: Priorización de restricciones de diseño acorde al método ADD.

3.3 Selección del elemento a descomponer

Cuando el sistema que se pretende diseñar es muy grande, el método de diseño ADD sugiere descomponerlo de manera que pueda ser diseñado de manera iterativa.

Inicialmente se suele considerar al sistema completo como el elemento a descomponer, partiéndolo de acuerdo a diversos criterios dependiendo de las necesidades del proyecto. Una vez identificados cada uno de sus componentes se procede a seleccionar uno de ellos (comúnmente el de mayor prioridad) para aplicarle el método y obtener el diseño del mismo. Posteriormente se evalúa si el diseño resultante se puede aplicar al siguiente componente como tal o si hay necesidad de reajustar el diseño actual. Una vez hechos los ajustes necesarios se aplica el diseño al segundo componente y nuevamente se

evalúa si hay necesidad de ajustar el diseño para un tercer componente y así sucesivamente.

Como ya se ha mencionado con anterioridad, el sistema GRP tuvo que descomponerse por motivos de restricciones temporales. Se optó por tomar una muestra del conjunto total de procesos que permitiera caracterizar de forma eficiente un conjunto de escenarios de modificabilidad y que además fuera acorde con las prioridades establecidas por el cliente (CINVESTAV).

En la Figura 3.2 se ilustra el conjunto de procesos que componen la operación administrativa de CINVESTAV y las relaciones establecidas entre ellos. De igual manera, en color amarillo se puede apreciar el subconjunto muestra de procesos considerados para el modelado de escenarios de modificabilidad que sirvieron de apoyo para el diseño del nuevo patrón y para la evaluación del mismo.

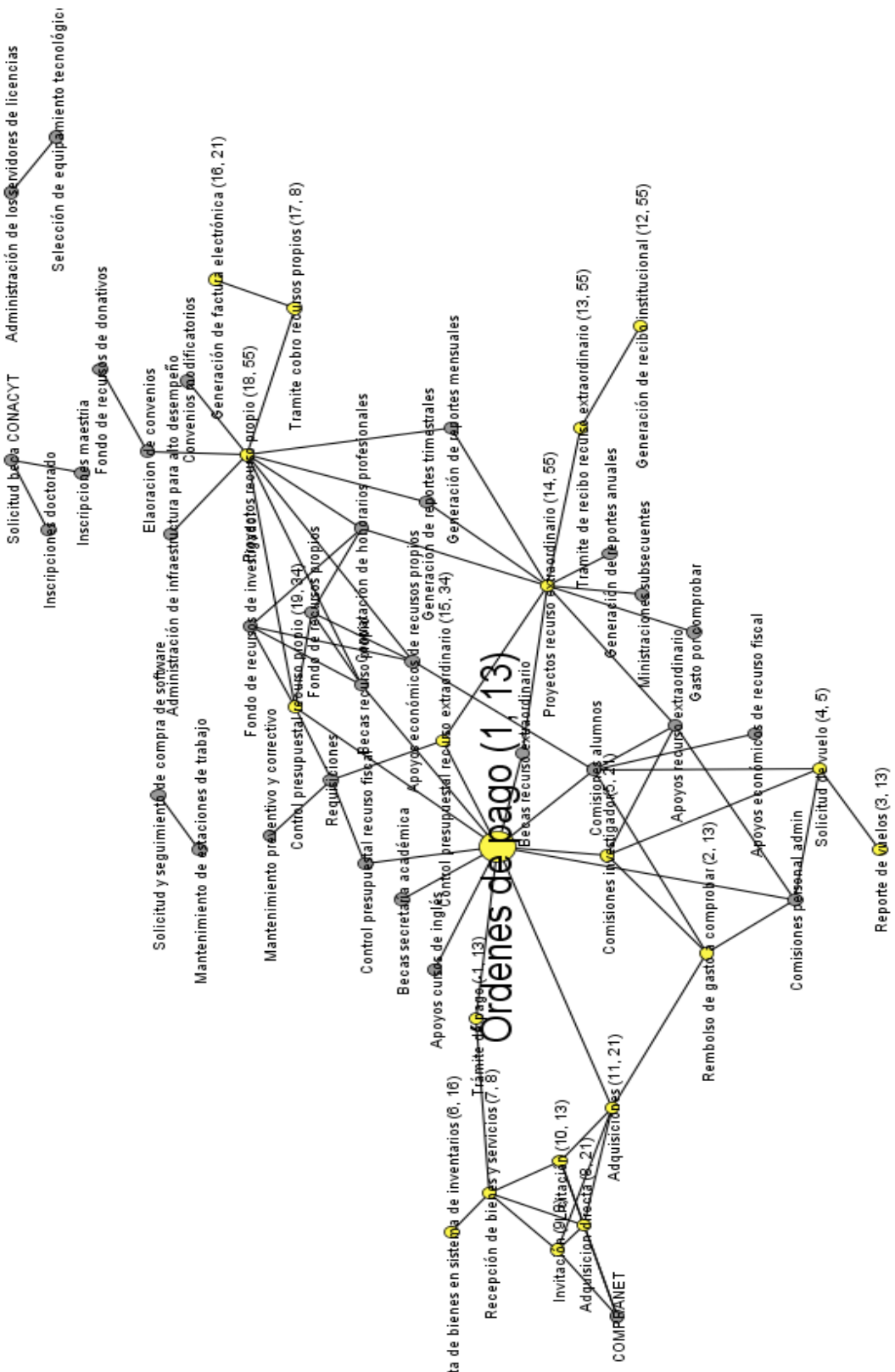

Figura 3.2: Representación gráfica de los procesos del proyecto GRP, resaltando los procesos seleccionados como parte del conjunto muestra.

Una vez definido el subconjunto de procesos del primer componente fue necesario definir un conjunto de aquellos requerimientos de con un mayor impacto a nivel de negocio y arquitectónico que guiarían el diseño. A dicho conjunto se le denomina directrices arquitectónicas.

3.4 Definición de directrices arquitectónicas

Las directrices arquitectónicas son escenarios de calidad de software que representan los requerimientos con mayor nivel de impacto tanto desde la perspectiva de negocio como de la arquitectura del sistema [Bass *et al.*, 2003]. Se aconseja considerar únicamente de cuatro a seis directrices arquitectónicas para el diseño de un sistema, ya que seleccionar un número mayor de las mismas, puede derivar en directrices arquitectónicas contrarias por satisfacer, lo cual haría el trabajo de diseño mucho más complejo [8] [Wojcik *et al.*, 2006].

Al igual que para el caso de los escenarios de calidad identificados en la sección 3.2.2 las directrices arquitectónicas definidas también están orientadas a modificabilidad e integrabilidad. Además, el tiempo de respuesta para cada directriz, es implícito[9] y relativo al desempeño del patrón arquitectónico de referencia. Por ejemplo, si para la directriz arquitectónica A, el tiempo de respuesta de DCI es de ocho horas-hombre, la respuesta para esta directriz se considera satisfactoria si el tiempo de respuesta de EDCI es menor a ocho horas-hombre.

En la Tabla 3.8 se listan las directrices arquitectónicas definidas para el caso de estudio de esta investigación, de acuerdo a su atributo de calidad objetivo.

[8] En ocasiones cubrir un atributo de calidad significa sacrificar algún otro atributo.

[9] En la Tabla 3.8 se ha omitido el tiempo de respuesta en el enunciado, dado que para todas ellas se considera un tiempo de respuesta relativo al tiempo de respuesta del patrón arquitectónico de referencia.

Atributo de calidad	Directriz arquitectónica
Modificabilidad	Edición de flujos de autorización (DA1). El desarrollador requiere editar algún flujo de autorización ya implementado en el sistema.
Modificabilidad	Modificación de datos en reportes (DA2). El desarrollador requiere modificar los campos a mostrar en un reporte.
Integrabilidad	Integración de módulos externos (DA3). El desarrollador requiere construir los diferentes módulos del sistema de manera que puedan integrarse entre sí.
Integrabilidad	Integración de un sistema externo (DA4). El desarrollador requiere integrar alguno de los módulos del sistema con un sistema externo.
Integrabilidad	Modificación en la funcionalidad de un sistema externo (DA5). El equipo de desarrollo de un sistema externo realiza una modificación sobre alguno de sus procesos utilizados por el sistema objeto de la investigación.

Tabla 3.8: Directrices arquitectónicas acorde al atributo de calidad objetivo.

3.5 Elección de un patrón para satisfacer las directrices arquitectónicas

La elección de un patrón que satisfaga las directrices arquitectónicas definidas en la Sección 3.4 tuvo dos propósitos principales.

El primero de ellos consistía en identificar un patrón arquitectónico cuyas características cubrieran las necesidades representativas del desarrollo de software ágil y que sirviera como base para el diseño de EDCI. Una vez seleccionado este patrón se debieron identificar áreas de mejora relativas a modificabilidad e integrabilidad sobre éste, y aplicar diversas técnicas de diseño para mejorarlo.

El segundo propósito de esta elección era que el patrón identificado fungiera como marco comparativo para la validación de las mejoras resultantes en EDCI, proceso que se detalla en el Capítulo 4.

En 2007 Bachmann *et al.* [2007] analizaron diferentes patrones arquitectónicos en relación a

diversas técnicas de diseño que los arquitectos de software podían aplicar con la finalidad de garantizar altos niveles de modificabilidad [10] en un sistema de software. Con este análisis se establecían los principios básicos para comprender cómo los diversos patrones satisfacen el atributo de modificabilidad y cómo pueden ser ajustados para mejorar sus propiedades en relación a dicho atributo.

Si se toma en cuenta que un sistema que implemente las técnicas sugeridas en el trabajo de Bachmann *et al*. [2007] sería un sistema altamente modificable [11], estas técnicas podrían también ser consideradas como indicadores para determinar qué patrones arquitectónicos son más aptos para ser aplicados a desarrollos ágiles.

La Tabla 3.9 ilustra los resultados obtenidos por Bachmann *et al*. [2007] en su análisis. En la parte superior se muestran técnicas de diseño orientadas a modificabilidad y del lado izquierdo, los patrones arquitectónicos analizados.

Después de revisar los resultados del análisis anterior, los patrones Broker y Blackboard se perfilaron como los que mayor número de técnicas de modificabilidad implementaban. Sin embargo, a pesar de esto, ninguno de ellos fue diseñado específicamente para ser aplicado a proyectos ágiles de software (como se ha mencionado anteriormente).

Un patrón arquitectónico no incluido en este análisis es el propuesto por Reenskaug y Coplien [2009] denominado Datos, Contexto e Interacción o DCI. Este patrón además de ser altamente modificable fue diseñado específicamente para desarrollos ágiles, cubriendo así las características adicionales de este tipo de software.

La principal característica de DCI consiste en reforzar la relación entre los modelos mentales del usuario y el código, separando el modelo de dominio (lo que el sistema es) de la funcionalidad (lo que el sistema hace). Esto trae consigo las siguiente ventajas:

[10] Algunos autores consideran a la integrabilidad como un atributo de calidad derivado de modificabilidad, por lo tanto algunas de las técnicas propuestas por Bachmann *et al*. [2007] pueden ser aplicadas a integrabilidad de igual manera.

[11] La expresión, altamente modificable, se refiere a que su nivel de modificabilidad es elevado y no a la probabilidad que pudiera tener de ser modificado.

Modificabilidad										
	Cohesión		Acoplamiento					Tiempo de enlace		
Patrón arquitectónico	Aseguramiento de coherencia semántica	Abstracción de servicios comunes	Uso de encapsulamiento	Uso de envolturas	Restricción de canales de comunicación	Uso de intermediarios	Escalamiento de niveles de abstracción	Uso de registro en tiempo de ejecución	Uso de enlazado en tiempo de inicialización	Uso de enlazado en tiempo de ejecución
Capas	✓	✓	✓		✓	✓	✓			
Tuberías-Filtros	✓		✓		✓	✓			✓	
Blackboard	✓	✓			✓	✓	✓	✓		✓
Broker	✓	✓	✓		✓	✓	✓	✓		
Modelo-Vista-Controlador (MVC)	✓		✓			✓	✓			
Presentación-Abstracción-Control (PAC)	✓		✓			✓	✓			
Micronúcleo	✓	✓	✓		✓	✓				
Reflexión	✓		✓							

Tabla 3.9: Patrones arquitectónicos y técnicas de modificabilidad implementadas.

1. Incrementa la legibilidad del código. Ahora los métodos y funciones se leen como algoritmos y no como un conjunto de instrucciones de sintaxis compleja. Basta con visualizar el código de una interacción para entenderlo, analizarlo e incluso crear funciones en código para ejecución de pruebas unitarias de manera más sencilla.

2. Adaptarse a requerimientos cambiantes y a la evolución acelerada de la funcionalidad del sistema (lo que el sistema hace) sin afectar a la parte estable del código (lo que el sistema es). Al surgir la necesidad de un cambio, el impacto de éste es fácilmente localizable dado que para cada caso de uso existe un componente contexto. Si el requerimiento es agregar una nueva funcionalidad, simplemente se tiene que agregar un nuevo contexto para el caso de uso, lo cual facilita el desarrollo incremental.

3. Mejorar la interacción entre individuos (proveedores y clientes) facilitando la participación de los usuarios finales (característica esencial del desarrollo de software ágil). Gracias a esto es posible comunicar y entender los requerimientos de mejor manera y obtener retroalimentación que permita controlar la evolución del sistema.

4. Garantizar la entrega de software de valor al cliente, incrementando la confianza en que lo que se desarrolla es lo que el cliente realmente necesita, dado que éste ha tenido participación activa en el proceso de construcción.

Una posible debilidad de este patrón consiste en la estricta regla de mantener al modelo exento de cualquier tipo de responsabilidad más allá de modelar a los objetos de negocio. Una práctica común en el diseño a nivel modelo de negocio, consiste en incluir funcionalidad simple dentro de las clases del modelo con la finalidad de evitar, en caso de que el modelo no requiera mayor lógica de negocios, la construcción de una larga cadena de clases para la representación de cada capa.

DCI es muy claro en cuanto a la regla de mantener al modelo de dominio exento de responsabilidades; sin embargo, el hacer esto implica que incluso para implementar funcionalidad meramente sencilla, tendría que crearse una larga cadena de componentes que soporte la funcionalidad. Para un sistema pequeño esto significaría añadir complejidad innecesaria que llevaría a perder el enfoque del desarrollo ágil.

A pesar de esta clara desventaja, se seleccionó a DCI como el patrón base para el diseño de un nuevo patrón, dado que en cuanto a la capacidad de cubrir las características del software ágil va más allá de lo que ofrecen otros patrones arquitectónicos. Además, al ser complemento de MVC, incluye también los beneficios que éste proporciona, sin dejar de mencionar su relativamente reciente inclusión al estado del arte. La Figura 3.3 muestra una vista modular del patrón DCI.

El siguiente paso en el proceso de diseño de EDCI consiste en la identificación de áreas de oportunidad y la aplicación de ciertas técnicas de diseño que cubren dichas áreas. Para esto fue

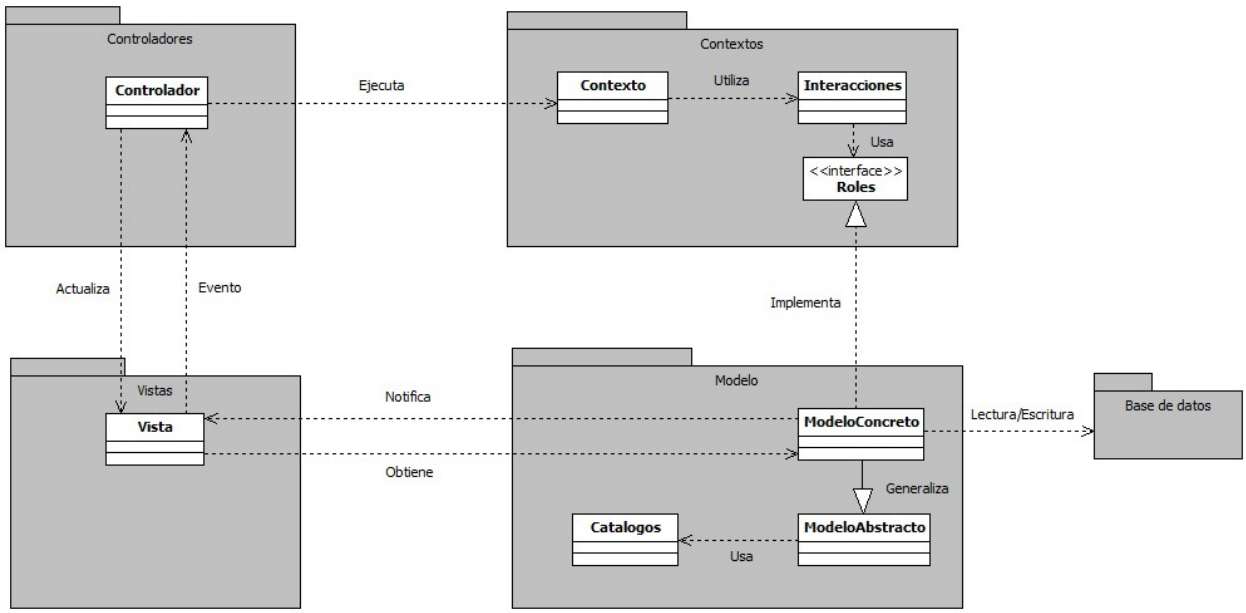

Figura 3.3: Vista módular del patrón arquitectónico DCI.

necesario modelar las directrices arquitectónicas una a una bajo el patrón DCI. A continuación se presentan los detalles de este proceso de modelado.

3.6 Directrices arquitectónicas de modificabilidad

3.6.1 Edición de flujos de autorización (DA1)

El desarrollador requiere editar algún flujo de autorización ya implementado en el sistema.

Una aplicación de tipo GRP, similar al caso de estudio, tiene como una de sus principales características la ejecución de procesos de negocio únicamente mediante previa autorización por parte de un usuario con privilegios suficientes. Dicho de otro modo, para avanzar de un subproceso a otro en este tipo de sistemas, se suele requerir que las salidas de cada subproceso, sean validadas antes de continuar.

Esta directriz arquitectónica presenta el escenario en el que un flujo de autorización definido en tiempo de diseño debe ser modificado a petición del cliente del sistema. Este cambio puede llegar a impactar a más de un componente en caso de que el flujo a modificar sea el mismo para varios procesos.

DCI implementa algoritmos de autorización de manera independiente para cada proceso en las interacciones. De esta manera, por cada proceso de negocio que utiliza el flujo de autorización a modificar, sería necesario realizar las mismas modificaciones de manera individual, lo que resulta en dificultad de mantenimiento del código.

Además de lo anterior, para hacer persistentes los cambios de estado de una entidad al realizar la autorización correspondiente, sería necesario interactuar con la base de datos a través del modelo, lo cual va en contra de la propia filosofía DCI de mantener esta capa exenta de modificaciones.

Con base en la forma en que DCI cubre esta directriz, las siguientes áreas de oportunidad fueron identificadas para este patrón arquitectónico *(ver Figura 3.4)*:

1. Bachmann *et al.* [2007] sugiere separar el código susceptible a modificaciones del código estable. En este escenario, las interacciones pueden contener código estable además de los flujos de autorización que en algún momento se vean afectados por esta directriz. Separar estas dos clases de código incrementaría los niveles de coherencia semántica.

2. Es posible realizar abstracción de servicios comunes, de tal forma que los cambios necesarios para modificar un flujo de autorización compartido por diferentes procesos de negocio se localicen en un único componente. De esta manera se evitaría la propagación de cambios a cada uno de los procesos de negocio afectados.

3. En lo que respecta al accesos a datos, dado que el modelo se supone debe quedar exento de cambios, los algoritmos de interacción con la fuente de datos podrían ser extraídos y ubicados en un componente responsable específicamente del acceso a datos.

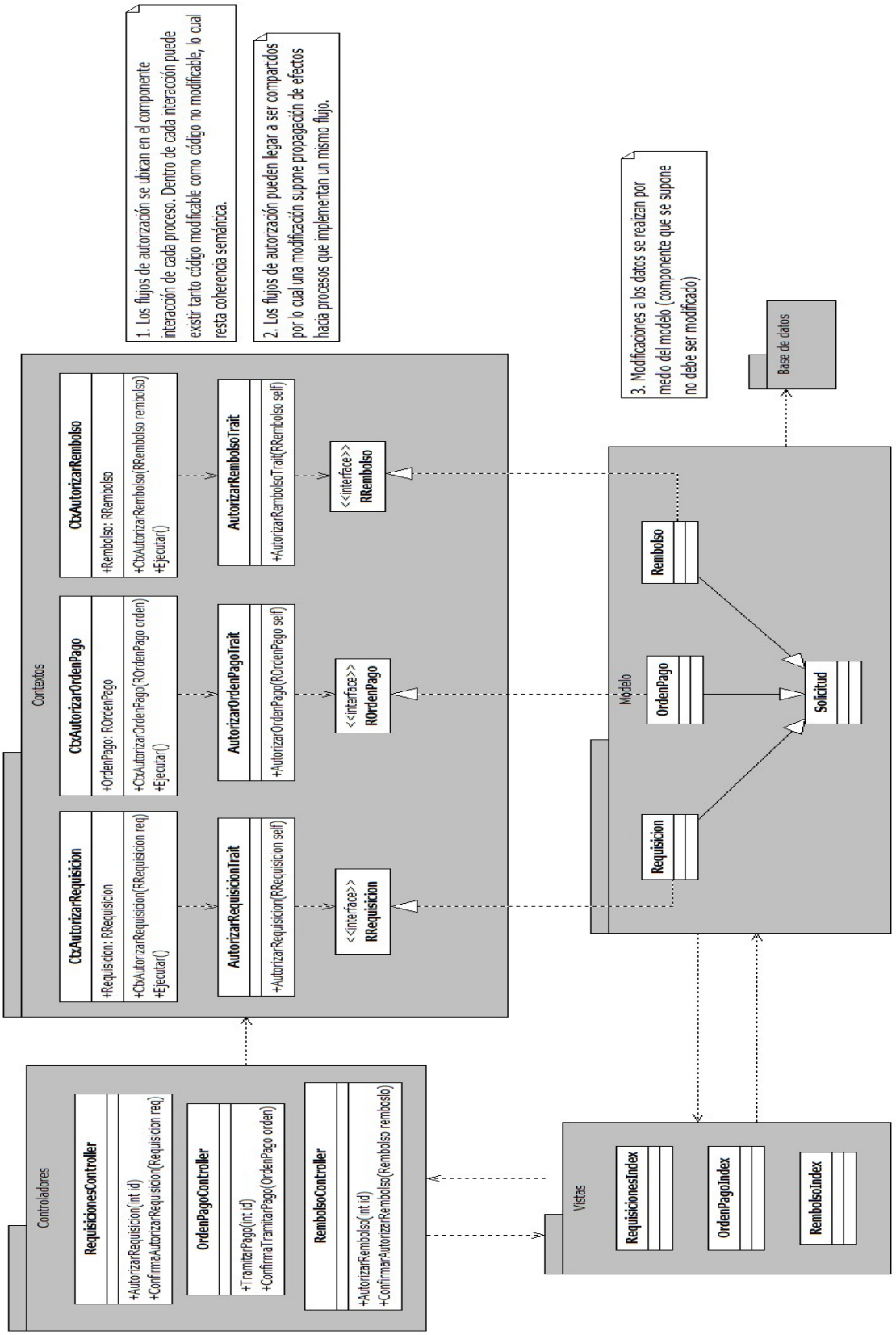

Figura 3.4: Modelado de DA1 bajo el patrón arquitectónico DCI.

Después de la aplicación de las técnicas sugeridas, las áreas de oportunidad identificadas podrían ser cubiertas de la siguiente manera en el nuevo patrón EDCI *(ver Figura 3.5)*:

1. El código correspondiente a los flujos de autorización (código modificable) es separado y localizado en un componente diferente utilizando el patrón de diseño *estrategia* [Gamma, Helm, Jhonson, y Vlissides, 2002]. De esta manera cada flujo de autorización es implementado mediante una estrategia que incluso puede ser intercambiable en tiempo de ejecución en caso de ser requerido.

2. Al hacer la separación de los flujos de autorización a estrategias se abstrae cada uno de ellos como un servicio común que puede ser empleado por diferentes procesos de negocio en las interacciones. Esto permite además, que los posibles cambios queden aislados y fácilmente localizables en cada estrategia.

3. El acoplamiento entre el modelo y la fuente de datos se rompe mediante la adición de una capa intermediaria de acceso a datos. A esta capa se le asigna la responsabilidad de interactuar con la fuente de datos en lugar de que el modelo tenga que hacerlo directamente.

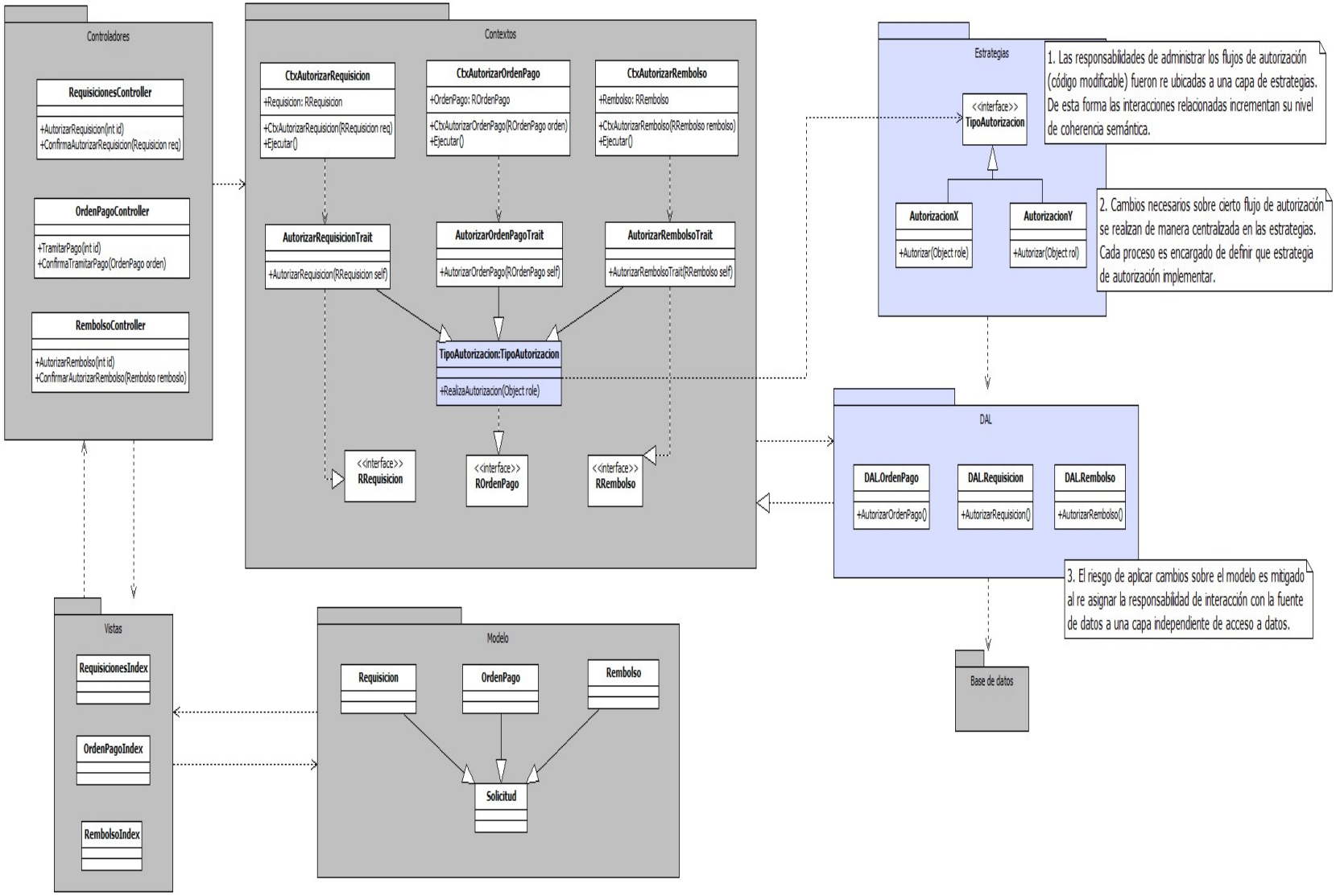

Figura 3.5: Modelado de DA1 bajo el patrón arquitectónico EDCI.

3.6.2 Modificación de datos en reportes (DA2)

El desarrollador requiere modificar los campos a mostrar en un reporte.

Otra de las características de los sistemas GRP es la capacidad de presentar información valiosa en reportes. Esta directriz arquitectónica está relacionada con dicha característica, ya que modela el escenario en que una vez definida la estructura de un reporte se requiere modificar las columnas que forman parte del mismo.

En DCI los reportes son construidos con base en una serie de atributos parte de una entidad del modelo. Esta entidad puede ser incluso una entidad compuesta para fusionar entidades con el fin de modelar los datos a presentar en un reporte (modelos de vistas o view-models).

Con la finalidad de cargar los datos a mostrar en un reporte la vista debe enviar una petición a través de cada capa de DCI hasta llegar al modelo en donde una consulta a la base de datos es ejecutada.

La materialización de esta directriz arquitectónica puede generar las siguientes áreas de oportunidad *(ver Figura 3.6)*:

1. Construir una entidad compuesta o un modelo de vista, sólo para modelar los datos a cargar en un reporte es una opción pero no es recomendable. Las entidades originales pueden llegar a requerir modificaciones propagadas hacia el modelo de vista, que dificulten su mantenimiento.

2. Preparar la entidad original de manera que de antemano cuente con toda la información cargada, sea o no necesaria, tampoco es precisamente una buena práctica. Cargar información sólo en caso de que pueda ser necesaria (sin tener la certeza), en cierto modo carece de sentido, además, es improbable de antemano conocer la información que se llegará a requerir en un reporte en determinado momento.

3. Realizar modificaciones en tiempo de diseño cada vez que un usuario necesita modificar las columnas de un reporte es poco práctico y hace necesaria la recompilación y publicación del código.

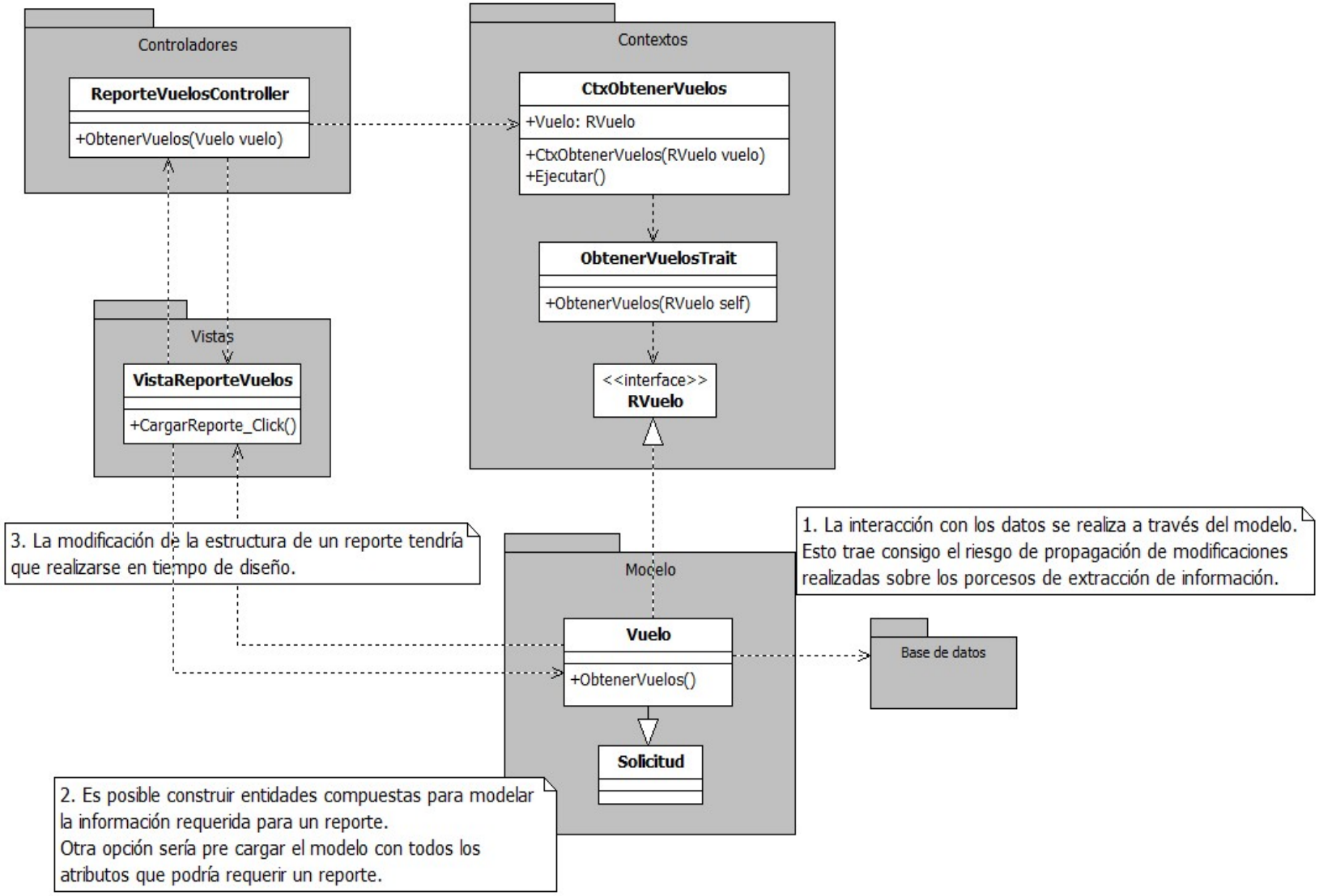

Figura 3.6: Modelado de DA2 bajo el patrón arquitectónico DCI.

Aplicando algunas técnicas de diseño, estos problemas podrían evitarse en el patrón propuesto EDCI *(ver Figura 3.7)*:

1. La creación de una capa de acceso a datos es útil también para cubrir esta directriz. Moviendo la carga de datos fuera del modelo es más simple evitar efectos de propagación. Esto también evita la necesidad de crear una entidad especializada para la carga de información en un reporte ya que se estaría generando una tabla de datos dinámica en la propia capa de acceso a datos que contendría los resultados de la consulta realizada sobre la base de datos y a partir de dicha tabla se generarían los reportes.

2. Mediante el agregado de un componente dedicado a la visualización de reportes, el usuario podría definir en tiempo de ejecución, qué columnas del reporte necesita cargar.

3. Finalmente, un beneficio extra es la habilidad de cargar únicamente la información requerida mediante la creación y ejecución de consultas dinámicas y parametrizadas en la capa de acceso a datos. Dependiendo de los parámetros recibidos se determinaría qué atributos deben cargarse y cuáles no.

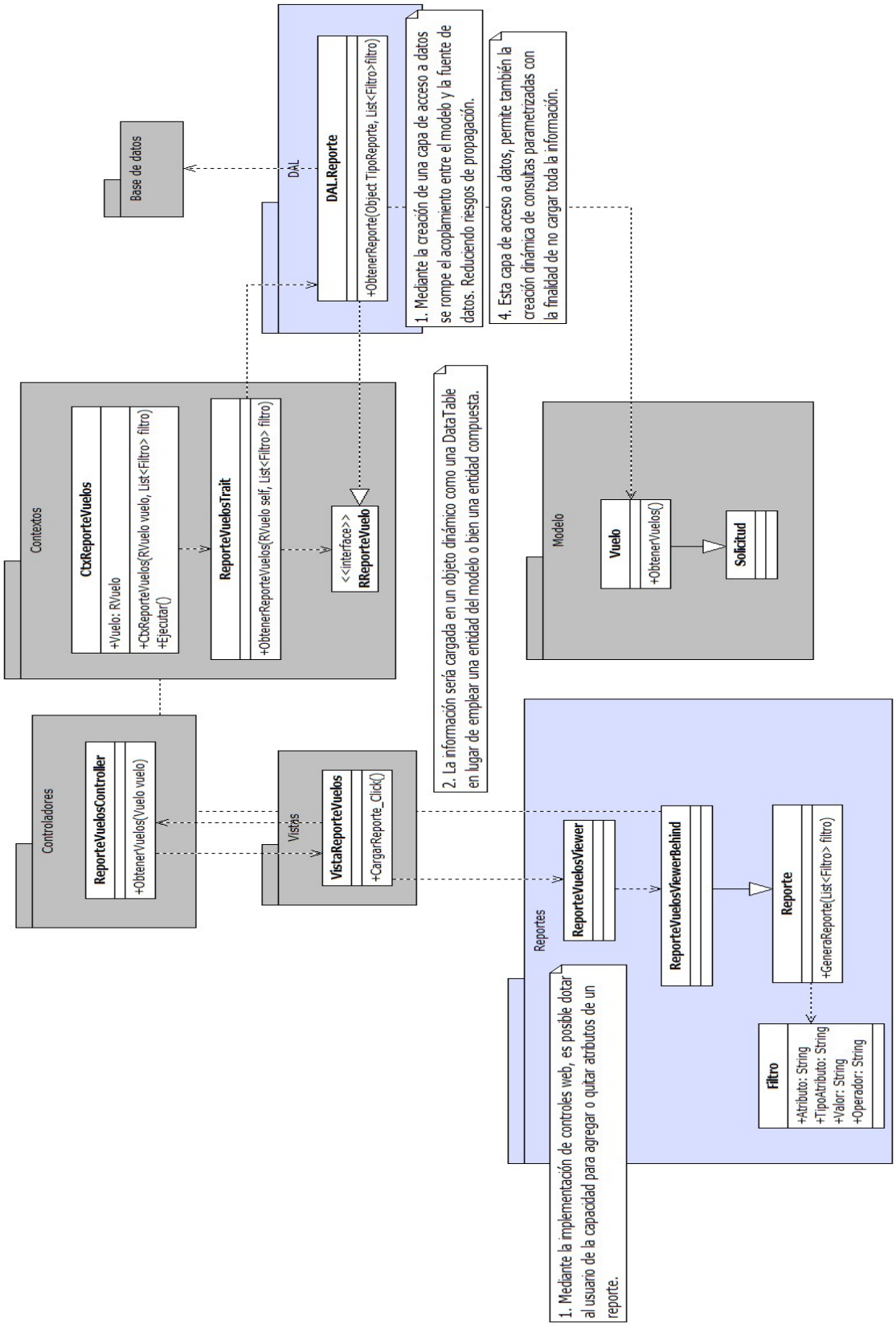

Figura 3.7: Modelado de DA2 bajo el patrón arquitectónico EDCI.

3.7 Directrices arquitectónicas de integrabilidad

3.7.1 Integración de módulos/sistemas externos (DA3 y DA4)

El desarrollador requiere integrar nuevos módulos/sistemas a la aplicación, de manera bidireccional [12].

Una característica deseable en cualquier tipo de aplicación es la integrabilidad. Sistemas de mediano a gran tamaño como los de tipo GRP (caso de estudio de esta investigación) suelen requerir la participación de varios equipos de trabajo desarrollando componentes en paralelo, con la finalidad de reducir los tiempos de entrega.

Otros ejemplos de necesidades de integración son, la integración con módulos del sistema desarrollados posteriormente con fines de expansión y la integración con sistemas legado implementados por terceros.

Las directrices arquitectónicas DA3 y DA4 son muy semejantes y fueron seleccionadas para modelar los escenarios antes mencionados. La diferencia principal entre ambas consiste en que la primera modela la integración de módulos externos y la segunda modela la integración de sistemas externos.

Para soportar estos escenarios DCI necesitaría establecer mecanismos de interacción con el módulo o sistema externo a través del modelo lo cual va en contra de las reglas de DCI de mantener el modelo sin funcionalidad.

Otra posibilidad consiste en crear modelos desde cero para las entidades de los módulos o sistemas externos y se establezca la conexión con sus fuentes de datos para poblar estos modelos. Esto a pesar de ser viable implicaría tener que duplicar funcionalidad ya presente en otros módulos o sistemas. Un ejemplo de ello sería el crear funcionalidad para realizar operaciones de creación, lectura y actualización en el sistema caso de estudio, aún cuando en el módulo o sistema externo ya cuenta con estas capacidades.

[12] Esto quiere decir, que la integración debe permitir al sistema consumir y proporcionar funcionalidad e información desde y hacia los módulos o sistemas externos.

Para soportar estos escenarios DCI necesita crear mecanismos de interacción entre el modelo y las fuentes externas de datos. Una vez más, realizar la interacción a través del modelo no es la mejor opción ya que sería necesario construir todas las capas (desde el modelo hasta la vista) para modelar las entidades externas participantes.

Los puntos antes mencionados derivan en las siguientes áreas de oportunidad para DCI *(ver Figura 3.8 y 3.9)*:

1. La responsabilidad asignada al modelo para interactuar con fuentes externas de datos reduce el grado de coherencia semántica. Esté podría incrementarse liberando al modelo de dicha responsabilidad.

2. Encapsular la funcionalidad externa a consumir por el sistema caso de estudio, de manera que pueda ser consumida desde un punto centralizado, localizando así las posibles modificaciones que puedan requerirse en la implementación de esta directriz.

3. De la misma manera, dado que la interacción debe ser bidireccional, la funcionalidad provista por dicho sistema a los módulos (sistemas) externos, puede encapsularse con la misma finalidad.

Figura 3.8: Modelado de DA3 bajo el patrón arquitectónico DCI.

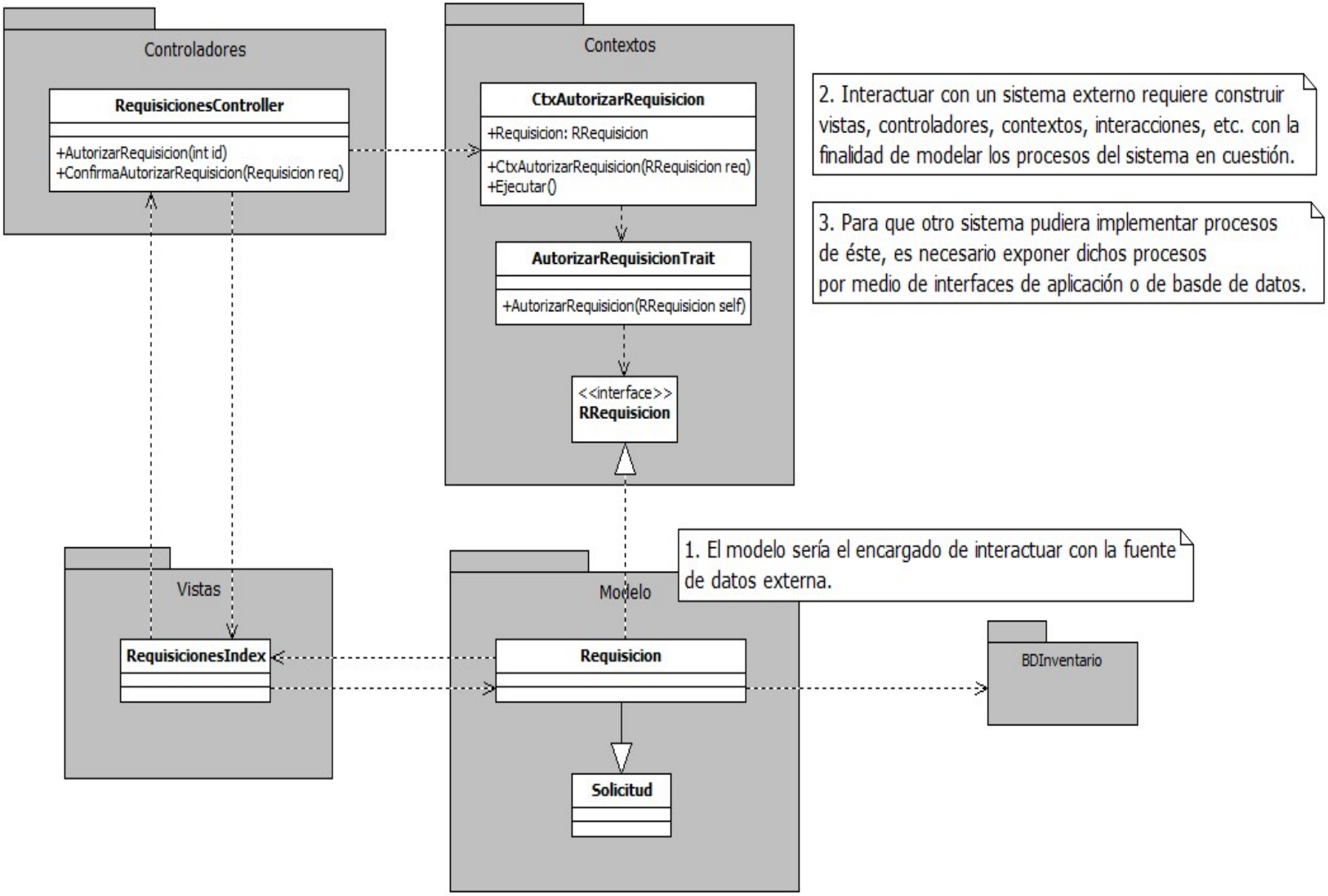

Figura 3.9: Modelado de DA4 bajo el estilo arquitectónico DCI.

EDCI, cubre estas áreas de oportunidad de la siguiente manera *(ver Figura 3.10 y 3.11)*:

1. Agrega una capa de acceso a datos que rompe el acoplamiento entre el modelo y las fuentes externas de datos. El acceso a los datos externos ahora fluye desde de las interacciones a las fuentes externas de datos teniendo como intermediarios la propia capa de acceso a datos y una capa de servicios web.

2. Añade una capa de servicios web para encapsular los servicios de los módulos (sistemas) externos a consumir. Estos servicios web deben contener además la definición de las entidades necesarias para su implementación como objetos de transferencia de datos *(data transfer objects)*.

3. Dado que la interacción con los módulos (sistemas) externos debe ser bidireccional, también se creó una capa para encapsular los servicios provistos por el sistema caso de estudio.

4. Estos servicios son publicados vía Internet para evitar límites relacionados con la red, lo cual es una necesidad común en este tipo de sistemas.

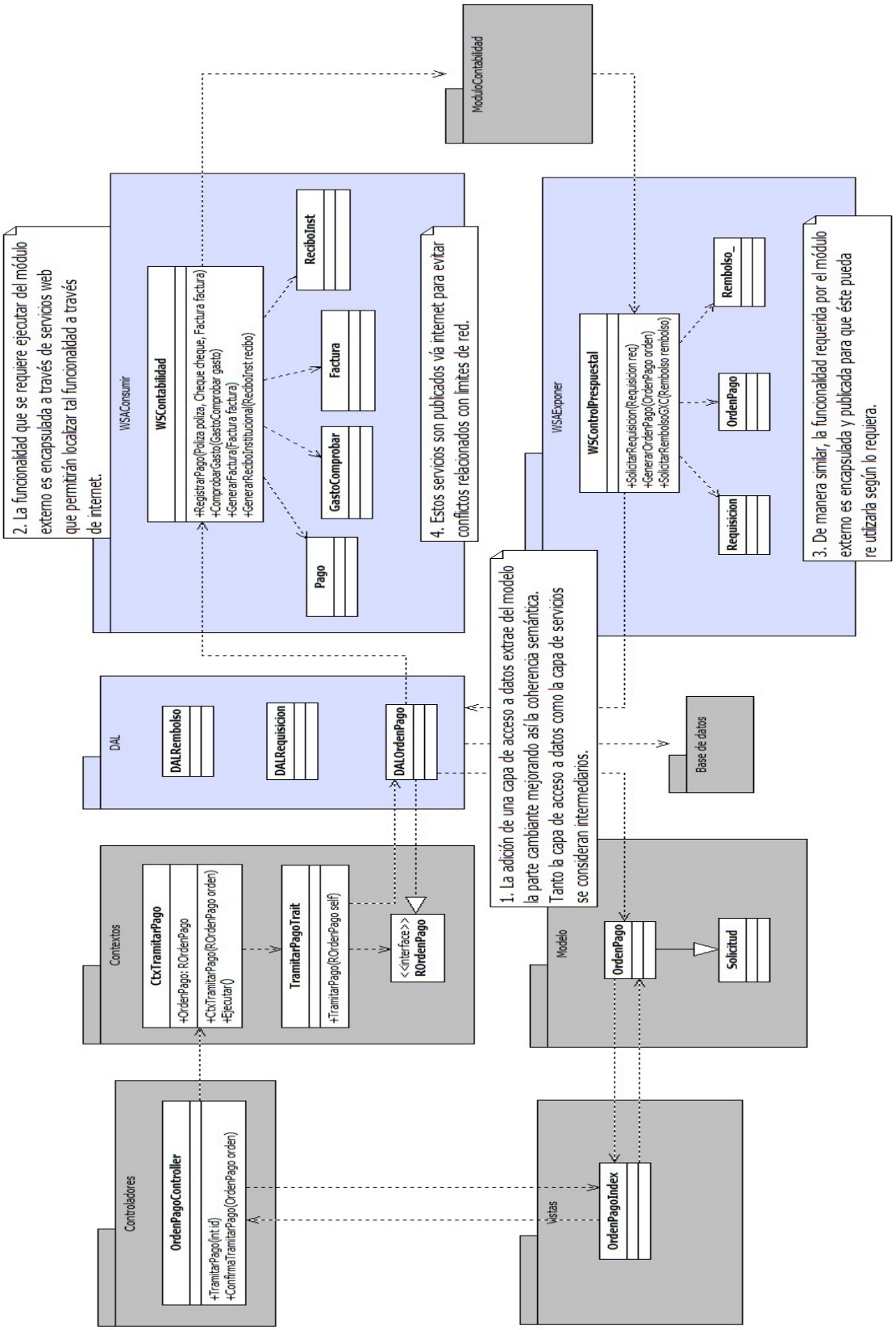

Figura 3.10: Modelado de DA3 bajo el patrón arquitectónico EDCI.

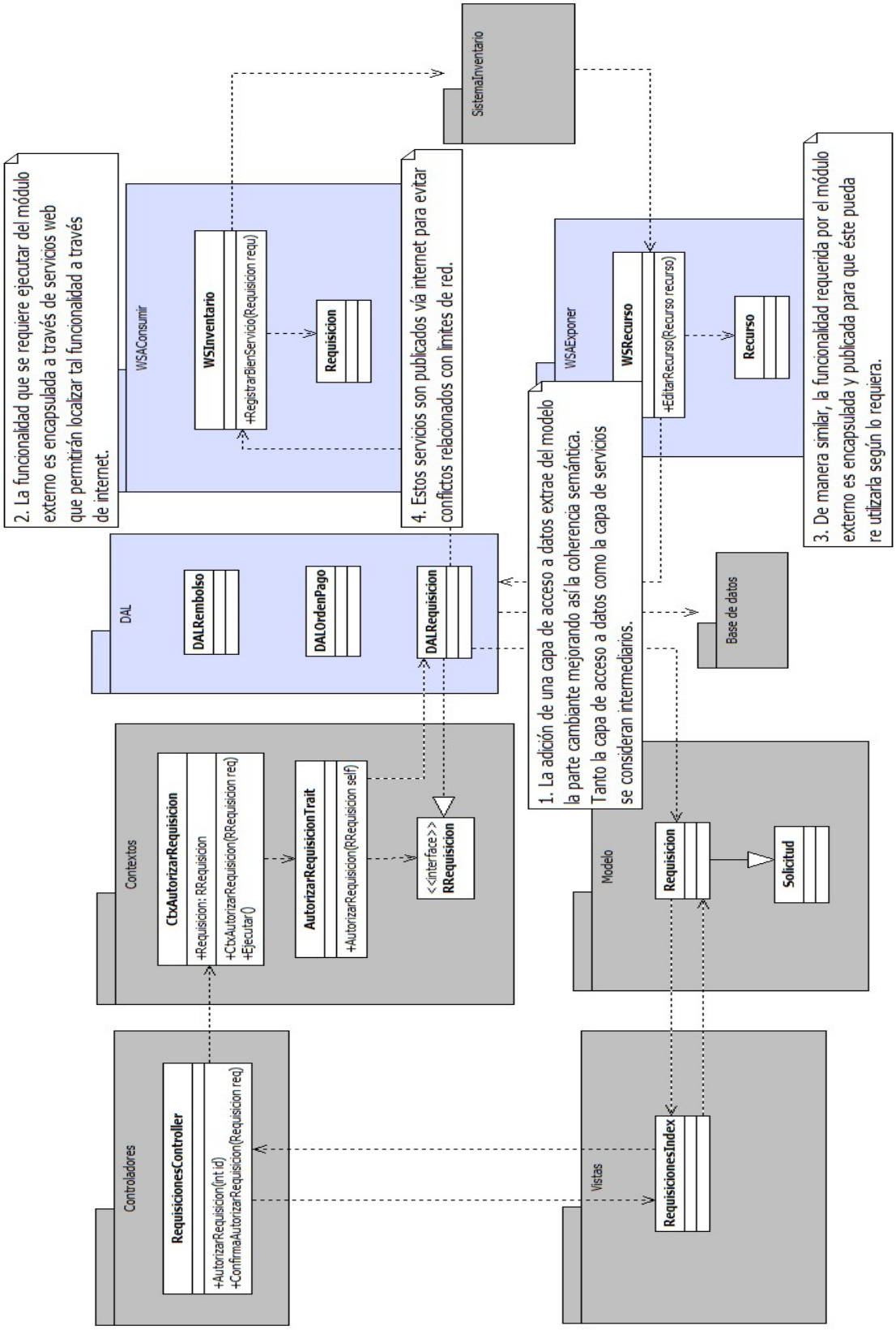

Figura 3.11: Modelado de DA4 bajo el estilo arquitectónico EDCI.

3.7.2 Modificación en la funcionalidad de un sistema externo (DA5)

El equipo de desarrollo de un sistema externo realiza una modificación sobre alguno de sus procesos utilizados por el sistema objeto de la investigación.

Este escenario es similar a los anteriores ya que también modela un escenario de integrabilidad.

Una vez establecida la integración entre el sistema y una fuente externa, se puede presentar el escenario en el cual es requerido que el equipo de desarrollo de algún sistema externo modifique uno de sus procesos, de tal forma que la información devuelta sea diferente y el sistema caso de estudio no esté preparado para procesarla. Esto es bastante común ya que la comunicación con equipos de desarrollo externos no es tan constante como durante la fase de desarrollo y por ende las notificaciones referentes a cambios realizados no se llevan acabo de manera oportuna.

Considerando lo anterior es aconsejable intentar prevenir o al menos reducir la posibilidad de impacto ocasionado por estos cambios al sistema caso de estudio. La respuesta de DCI ante esta directriz, deriva en la siguiente área de oportunidad *(ver Figura 3.12)*:

1. Dado que los mecanismos de interacción con datos de sistemas externos son implementados por cada entidad, puede ser necesario para más de una de éstas realizar alguna corrección a raíz de las modificaciones realizadas.

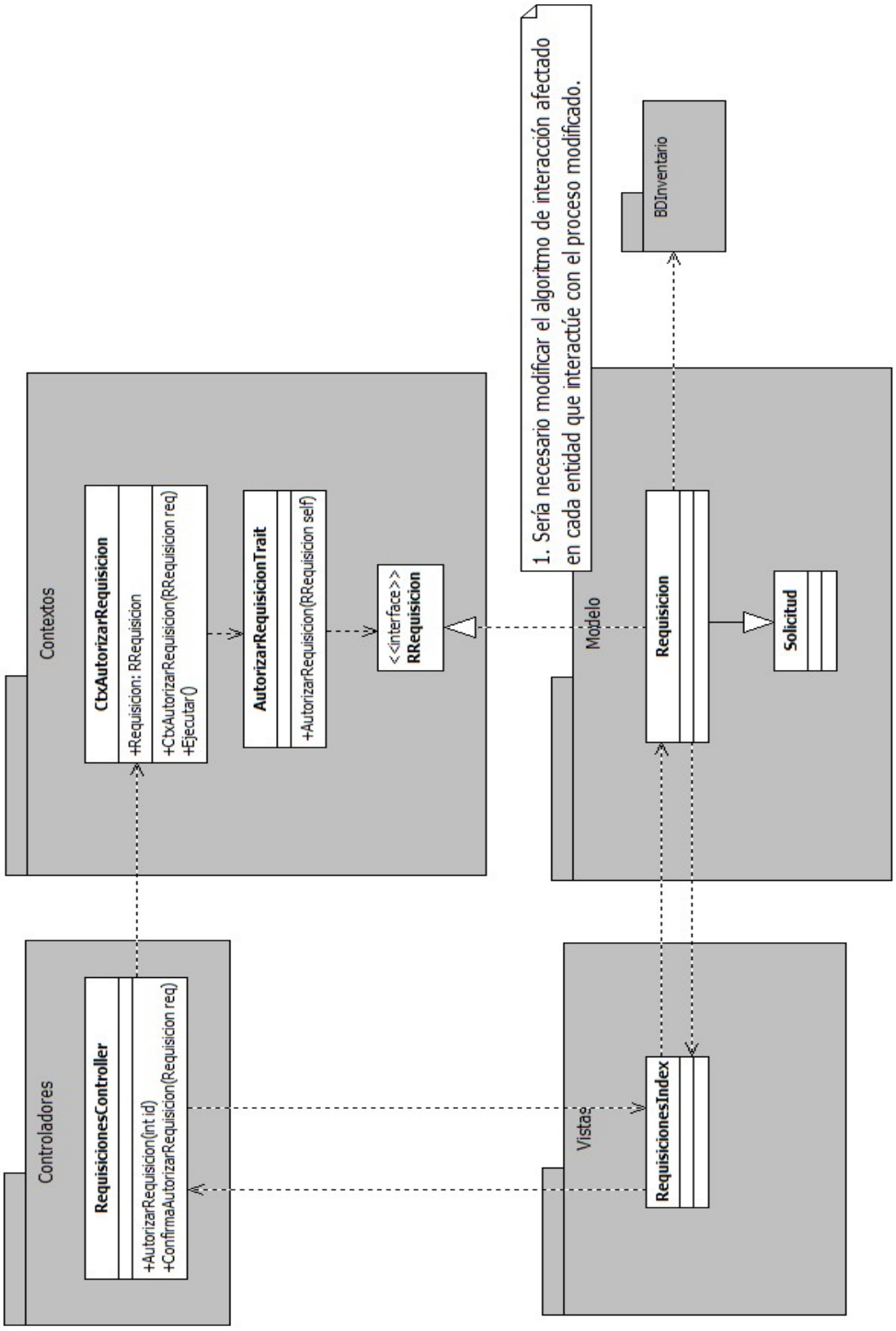

Figura 3.12: Modelado de DA5 bajo el patrón arquitectónico DCI.

Desde la perspectiva de EDCI, esta directriz es cubierta mediante la aplicación de las siguientes técnicas *(ver Figura 3.13)*:

1. Encapsular los algoritmos de interacción con el sistema externo, de tal forma que los cambios potenciales sean localizados únicamente a nivel de la capa de acceso a datos.

2. Además de lo anterior, cada componente en la capa de acceso a datos encargado de interactuar con el sistema externo puede diseñarse a manera de envoltura *(wrapper)*. Estas envolturas serían responsables de prevenir la propagación de cambios, transformando los datos recibidos a una representación que pueda ser procesada por el componente que consume tales datos.

Figura 3.13: Modelado de DA5 bajo el patrón arquitectónico EDCI.

Resumen del capítulo

En este capítulo se presentan dos importantes herramientas para el desarrollo de esta tesis. La primera de ellas, un método de diseño enfocado a la satisfacción de atributos de calidad y la segunda, un proyecto (caso de estudio) el cual fungió como fuente de entradas en las diferentes fases del método de diseño. El proceso de diseño del patrón EDCI fue también descrito a detalle fase a fase en este capítulo.

4

Evaluación del patrón arquitectónico EDCI

En este capítulo se ilustra el proceso de comparación al cual fueron sometidos los patrones arquitectónicos DCI (marco comparativo) y EDCI (patrón arquitectónico propuesto). Los resultados obtenidos de dicha comparación son presentados de manera general con base en directrices arquitectónicas y a mayor detalle con base en indicadores (técnicas) de modificabilidad.

4.1 Introducción

Una vez cubiertas las áreas de mejora de DCI para cada directriz se obtuvo como resultado el patrón: *Datos, Contexto e Interacción-Mejorados (EDCI)*. El siguiente diagrama *(Figura 4.1)* presenta la distribución de los componentes de este patrón arquitectónico mediante una vista modular de descomposición y dependencias.

En color claro se muestran: un componente de carga de reportes que facilita su manipulación, una capa de estrategias encargada de alojar la lógica de autorización, una capa de acceso a datos que libera al modelo de interactuar con las fuentes de datos y dos capas más de servicios web que centralizan la funcionalidad brindada por el sistema caso de estudio y la que éste necesita consumir.

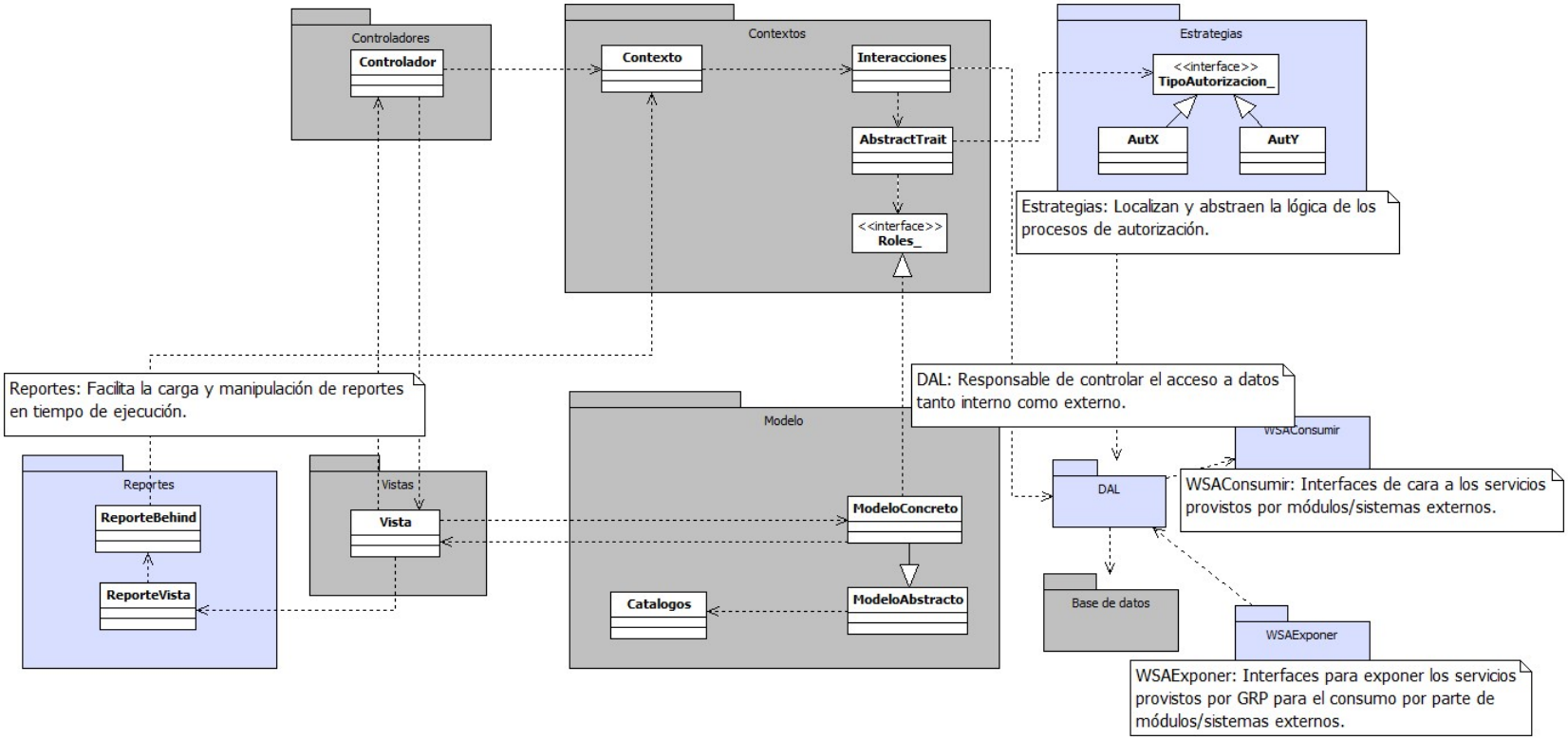

Figura 4.1: Vista modular del patrón arquitectónico EDCI.

Como recordaremos, en el estado del arte se hacía mención de dos conceptos clave para esta investigación, el primero de ellos, patrones arquitectónicos, fue importante para el proceso de diseño de EDCI detallado en el Capítulo 3, mientras que el segundo, métodos de evaluación de arquitecturas de software, es clave para el análisis de EDCI, proceso que se detalla en este capítulo.

Con la intención de validar el grado de mejora alcanzado por EDCI, se aplicó un método de evaluación de arquitecturas de software seleccionado de entre los revisados en el Capítulo 2, Sección 2.4.4.

Esta selección fue hecha considerando las características de cada método de evaluación, buscando principalmente algún método que permitiera distinguir el grado de mejora proporcionado por EDCI en relación con el patrón DCI.

Tomando ésto en cuenta, se determinó que el método de evaluación a emplear debía facilitar la comparación de diferentes patrones arquitectónicos, es así como los métodos ALMA [Bengtsson *et al.*, 2002] y SACAM [Stoermer *et al.*, 2003] se perfilaron como los principales candidatos a implementarse.

Ambos métodos fueron diseñados con fines comparativos (a diferencia del resto) y son bastante similares entre sí; sin embargo, SACAM es de más reciente inclusión al estado del arte, lo que significó un parámetro de decisión entre uno y otro método.

El método SACAM se lleva a cabo en seis fases. La Figura 4.2 ilustra las fases del método y su orden de ejecución.

A continuación, se describen las actividades realizadas en cada fase del método SACAM para la comparación de DCI y EDCI en el caso de estudio utilizado en esta investigación.

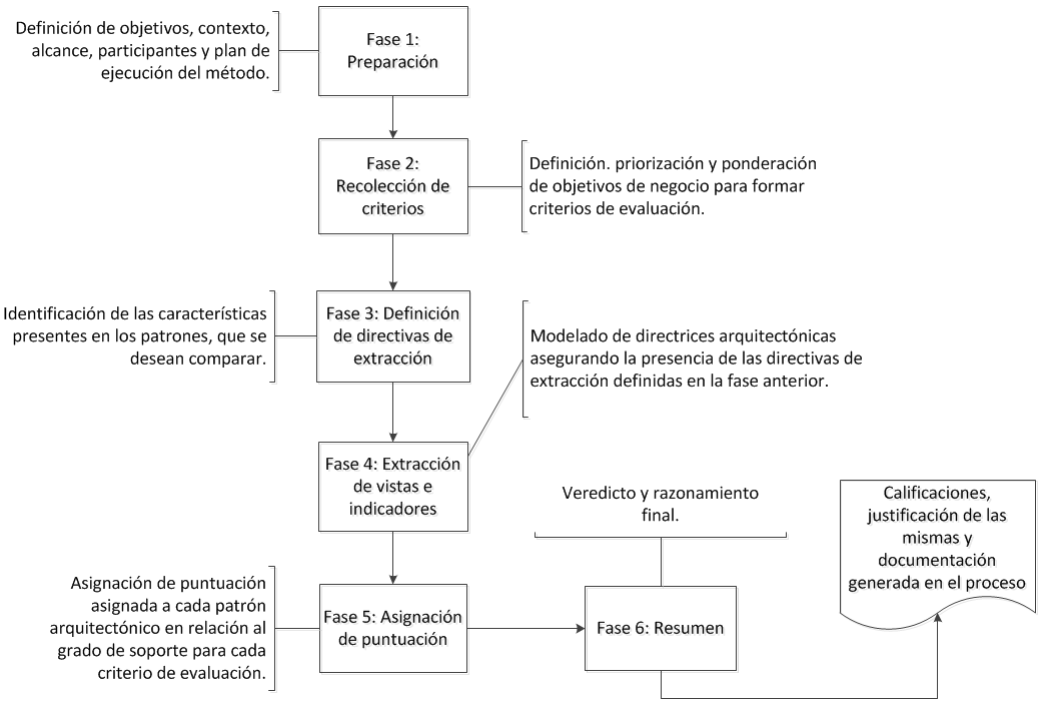

Figura 4.2: Representación gráfica del método SACAM.

4.2 Fase 1. Preparación

La primera fase es denominada *preparación* y proporciona las bases para la ejecución del método. En ésta se definen, los objetivos, el contexto, el alcance y los participantes del proceso de evaluación. Además, se realiza una presentación de los patrones arquitectónicos a ser comparados y se elabora un plan de ejecución del método.

4.2.1 Entradas

4.2.1.1. Objetivos de negocio

A pesar de que las metas de negocio del caso de estudio ya se encontraban previamente definidas, para esta investigación dichas metas fueron ligeramente adaptadas, de tal manera que empataran con el objetivo de diseñar un patrón arquitectónico altamente modificable e integrable.

De esta manera se adoptaron las principales características del software ágil como metas de

negocio, las cuales como recordaremos son: rápida entrega de productos de software funcionales al cliente, facilidad de adaptación al cambio y escalabilidad mediante integración de componentes.

4.2.1.2. Sujetos de comparación

En el Capítulo 3 se ilustra el proceso de diseño de EDCI (ver Figura 4.1), un patrón arquitectónico diseñado para mejorar los niveles de modificabilidad e integrabilidad ofrecidos por los patrones arquitectónicos presentes en el estado del arte. Su principal característica consiste en que está basado en DCI (ver Figura 3.3), otro patrón arquitectónico de reciente inclusión en el estado del arte, diseñado con la finalidad de facilitar la comprensión y la validación del código, elevar las capacidades de modificabilidad y simplificar el mantenimiento.

Estos dos patrones representan los sujetos de comparación en esta evaluación, en la que se perseguía la meta de identificar qué características de DCI habían sido mejoradas por EDCI y en qué proporción. De esta manera se demostraría o refutaría la hipótesis planteada en esta investigación, dependiendo de si los resultados de la aplicación de SACAM son favorables o no para EDCI.

4.2.2 Salidas

4.2.2.1. Contexto y objetivos de evaluación

La presente evaluación se realiza bajo un contexto de comparación en donde el principal objetivo es determinar el grado de mejora obtenida después de aplicar algunas técnicas de modificabilidad sobre el patrón DCI. Para lograr ésto, cubrir los siguientes objetivos específicos fue esencial.

- Ponderar las directrices arquitectónicas (criterios de comparación) definidas para evaluar a los sujetos de comparación.
- Definir los indicadores (técnicas, patrones y/o métricas) de validación para la satisfacción de los criterios de comparación.
- Homologar las vistas arquitectónicas actuales con la finalidad de que sean comparables asegurando que la presencia o ausencia de indicadores se vea reflejada.

- Calificar el grado en el que cada patrón arquitectónico aborda cada directriz.

4.2.2.2. Participantes

Los participantes en el proceso de evaluación fueron seleccionados de entre los involucrados en el caso de estudio, considerando sus conocimientos sobre los requerimientos funcionales, no funcionales, los patrones arquitectónicos candidatos y el método de evaluación seleccionado. La Tabla 4.1 muestra un listado de los participantes y su responsabilidad en el proceso de evaluación.

Participante	**Responsabilidades**
Analista de requerimientos	El analista de requerimientos mantuvo la responsabilidad de aclarar cualquier duda o supuesto que pudiera surgir a raíz de una mala interpretación de los requerimientos tanto funcionales como no funcionales del sistema.
Arquitecto de software (tesista)	Fue el encargado de presentar los patrones arquitectónicos candidatos (DCI y EDCI) al resto de los participantes, con la finalidad de obtener retroalimentación acerca de las directrices arquitectónicas por parte de los análistas y de brindar las herramientas necesarias al equipo de evaluadores para presentar un juicio de cada patrón.
Evaluadores	Los evaluadores fueron los responsables de analizar la manera en que cada patrón comparado abordaba cada directriz, con el objetivo de asignar una puntuación a cada uno, con respecto a indicadores definidos en fases posteriores del proceso de evaluación.

Tabla 4.1: Participantes en el proceso de evaluación SACAM.

4.2.2.3. Presentación de sujetos de comparación

Una vez definidos el contexto, los objetivos y los participantes del proceso de evaluación, se procedió a presentar a los sujetos de comparación, con el objetivo de darlos a conocer a los participantes, de modo que adquirieran el conocimiento necesario para poder evaluar.

En las Figuras 3.3 y 4.1 del Capítulo 3 se muestra una vista modular para cada sujeto.

4.2.2.4. Plan de ejecución de SACAM

Con la finalidad de dar seguimiento a la ejecución del método SACAM, los participantes elaboraron en conjunto un plan de trabajo semanal para el desarrollo de cada fase del proceso, mismo que se detalla a continuación.

	Junio - Julio			
Actividad	**9 - 15**	**16 - 22**	**23 - 29**	**30 - 6**
1. Preparación	•			
2. Recolección de criterios de evaluación	•			
3. Definición de directivas de extracción de vistas		•		
4. Extracción de vistas e indicadores		•	•	•
5. Asignación de puntuación (calificación)				•
6. Resumen (veredicto final)				•

Tabla 4.2: Plan de ejecución de SACAM sobre los sujetos de comparación.

4.3 Fase 2. Recolección de criterios de evaluación

Esta fase consiste básicamente en la definición, priorización y ponderación de objetivos de negocio con la finalidad de elegir aquellos de mayor impacto arquitectónico y de negocio como criterios de evaluación (comparación).

4.3.1 Entradas

4.3.1.1. Objetivos de negocio

En la sección anterior se habló de la transformación de los objetivos de negocio originales con el fin de adaptarlos al propósito de esta investigación y se definió también cuáles serían finalmente dichos objetivos (ver Sección 4.2).

En esta fase del proceso de evaluación, dichos objetivos son presentados como criterios de evaluación. Estos criterios, ayudaron a determinar si un patrón arquitectónico es capaz de reaccionar y en qué grado, a la ocurrencia de un escenario de modificabilidad en particular.

Estos escenarios han sido previamente identificados, priorizados y presentados a manera de directrices arquitectónicas (ver Tabla 4.3 para un listado de estas directrices).

4.3.2 Salidas

4.3.2.1. Ponderación de directrices arquitectónicas

En el proceso de diseño de EDCI (Capítulo 3) estas directrices arquitectónicas fungieron únicamente como guías para asegurar el cumplimiento del objetivo.

Para esta fase de la evaluación fue necesario contar con una métrica de ponderación de la importancia o impacto de cada directriz arquitectónica. Esta métrica permitió asignar una calificación al desempeño de cada sujeto de comparación considerando su relevancia en relación a los objetivos de negocio. El valor de ponderación se asignó en escala de 1 a 5, en donde 5 representa un nivel de impacto muy alto de la directriz sobre el comportamiento del sistema en cuestión de modificabilidad.

Estos valores fueron definidos por los participantes (ver Tabla 4.1) en ese rango dado que representan valores cualitativos: muy bajo, bajo, medio, alto y muy alto.

En la Tabla 4.3 se listan las directrices arquitectónicas y el valor de ponderación asignado a cada una.

Atributo de calidad	Directriz arquitectónica	Valor de ponderación
Modificabilidad	Edición de flujos de autorización (DA1). El desarrollador requiere editar algún flujo de autorización ya implementado en el sistema.	5
Modificabilidad	Modificación de datos en reportes (DA2). El desarrollador requiere modificar los campos a mostrar en un reporte.	3
Integrabilidad	Integración de módulos externos (DA3). El desarrollador requiere construir los diferentes módulos del sistema de manera que puedan integrarse entre sí.	5
Integrabilidad	Integración de un sistema externo (DA4). El desarrollador requiere integrar alguno de los módulos del sistema con un sistema externo.	3
Integrabilidad	Modificación en la funcionalidad de un sistema externo (DA5). El equipo de desarrollo de un sistema externo realiza una modificación sobre alguno de sus procesos utilizados por el sistema objeto de la investigación.	3

Tabla 4.3: Asignación de valores de ponderación a directrices arquitectónicas definidos por los participantes del proceso de evaluación.

4.4 Fase 3. Definición de directivas de extracción

Una evaluación comparativa, como la que se implementó para este caso es muy común en la práctica. A menudo, con el inicio de un proyecto surge la interrogante acerca de qué arquitectura implementar. Algunos arquitectos de software apuestan por el dominio que tienen ellos sobre la implementación de ciertos patrones, mientras que otros optan por algo más innovador.

Sin embargo, si se quiere tomar una decisión con mayor fundamento, lo correcto es realizar un análisis comparativo entre diversos patrones arquitectónicos con la finalidad de identificar cuál de ellos satisface en mayor grado los objetivos de negocio para los cuales se construirá el sistema.

Para realizar dicho análisis es preciso contar con la documentación de cada una de las arquitecturas que serán sujetas a comparación y es aquí donde surge el problema.

La documentación de la cual un arquitecto puede echar mano para realizar un análisis de este

tipo, por lo general consiste en una serie de bosquejos o diagramas, acompañados de explicaciones textuales acerca de cada componente y sus interacciones, en el mejor de los casos. En ocasiones, ni siquiera existe documentación alguna y lo único que sirve como entrada es el código mismo.

Ante estas situaciones, el arquitecto encargado de realizar el análisis requiere generar la documentación precisa para poder comparar los patrones candidatos. Para ello debe considerar qué características de cada patrón desea comparar, ya que de tales características depende la clase de documentación a generar.

En esta sección se definen dichas características para el caso de estudio en particular.

4.4.1 Entradas

4.4.1.1. Directrices arquitectónicas

Las directrices arquitectónicas representan los objetivos principales a nivel arquitectónico de los sujetos de comparación. Estos objetivos serían la base para determinar qué características de ambos patrones se requiere comparar y por lo tanto modelar o documentar.

4.4.1.2. Documentación existente

Se tomó como documentación base a los modelos (diagramas arquitectónicos) resultantes en la fase de implementación (véase Figuras 3.3 y 4.1) mismos que al tener ya conocimiento acerca de los posibles métodos de evaluación a implementar, fueron construidos cuidando que facilitaran la comparación entre DCI y EDCI para cada directriz arquitectónica.

4.4.2 Salidas

4.4.2.1. Directivas de extracción de vistas

Los atributos de modificabilidad e integrabilidad marcan la pauta para la definición de las directivas de extracción de vistas, ya que, lo que es de interés para esta evaluación es demostrar las características de cada sujeto de comparación en relación a tales atributos.

A partir del trabajo de Bachmann *et al*. [2007] se tomaron las técnicas de modificabilidad como directivas de extracción, ya que representan características deseables de un sistema modificable.

El nivel de modificabilidad de un patrón arquitectónico depende del número de técnicas de modificabilidad implementadas. Por lo tanto se considera más apto para su aplicación en el desarrollo ágil de software, el patrón que mayor número de técnicas tenga implementadas.

A continuación se listan las directivas de extracción para la documentación de los sujetos de comparación.

- Técnicas de diseño para modificabilidad e integrabilidad

    1. Aseguramiento de coherencia semántica. Consiste en reorganizar responsabilidades entre los distintos componentes del sistema con la finalidad de aislar puntos potenciales de impacto a raíz de cambios.
    2. Abstracción de servicios comunes. Al igual que la anterior se relaciona con el incremento de la cohesión, sin embargo esta técnica más allá de aislar los componentes susceptibles a impacto por cambios, los agrupa en funciones similares para ser reutilizadas.
    3. Uso de encapsulamiento. Es un concepto básico del paradigma orientado a objetos y consiste en ocultar la complejidad de los algoritmos mediante el uso de interfaces.
    4. Uso de envolturas. Es un tipo de encapsulación, sin embargo difiere en que una envoltura suele realizar tareas de adaptación e interpretación de los datos que pasan a través de ésta.
    5. Restricción de canales de comunicación. Es la implementación de un tipo de intermediario que restringe los mecanismos de flujo de la información con la finalidad de reducir el número de puntos de interacción entre los componentes de un sistema.
    6. Uso de intermediarios. La finalidad de estos es romper con la dependencia existente entre módulos.
    7. Escalamiento de niveles de abstracción. Consiste en encapsular las implementaciones de algoritmos lo más alto posible en la jerarquía arquitectónica.

4.5 Fase 4. Extracción de vistas e indicadores

Ya que las características a modelar o directivas de extracción han sido definidas, el siguiente paso es documentar los sujetos de comparación para cada una de las directrices arquitectónicas, de manera que sea posible identificar a través de la documentación aquellas que son implementadas por cada patrón.

En el libro *Documenting Software Architectures: Views and Beyond* [Clements *et al.*, 2010] los autores sugieren diversas técnicas para la generación de documentación de arquitecturas de software considerando las diversas circunstancias y objetivos, pero sobre todo las características que se desea distinguir.

Una técnica bastante común es la creación de vistas, las cuales tienen como objetivo representar de manera gráfica la distribución de los componentes, sus interacciones, el flujo de los datos de un componente a otro, la secuencia de ejecución de los procesos del sistema, entre otras características.

Considerando el objetivo de comparar el comportamiento de dos patrones arquitectónicos ante ciertas directrices arquitectónicas, (particularmente de modificabilidad) podemos referir el trabajo de Clements *et al.* [2010], quienes sugieren modelar las arquitecturas utilizando una vista modular de descomposición y dependencias. Este tipo de vista permite distinguir de manera sencilla la distribución de responsabilidades de cada elemento en el patrón, además de las relaciones y dependencias existentes entre ellos. Esto le hace ideal para representar escenarios de modificabilidad.

4.5.1 Entradas

4.5.1.1. Directivas de extracción

Los resultados obtenidos en la fase anterior, fueron la principal entrada para esta fase.

Las técnicas de modificabilidad propuestas por Bachmann *et al.* [2007] representan las características que requieren ser observables en la documentación a utilizarse como base de la comparación entre EDCI y DCI.

Una vez generadas las vistas (documentación) de cada patrón, para cada directriz arquitectónica

de acuerdo a las directivas de extracción, la comparación entre los sujetos es posible.

4.5.1.2. Documentación existente

Como documentación base se tomaron los diagramas de componentes realizados en la fase de implementación (véase Figuras 3.3 y 4.1) de esta investigación, los cuales fueron ligeramente ajustados y validados en la primera fase de la evaluación.

4.5.2 Salidas

4.5.2.1. Modelado de directrices arquitectónicas mediante vistas

Para el proceso de diseño de EDCI se generaron un conjunto de vistas para las diversas directrices arquitectónicas con la finalidad de representar las mejoras aplicadas en relación a DCI. Esta documentación fue utilizada como base para esta fase del proceso. Únicamente fue necesario asegurar la concordancia de estas vistas, con las directivas de extracción definidas.

En la Tabla 4.4 se muestra una serie de referencias cruzadas a secciones y figuras que muestran para cada directriz arquitectónica las vistas generadas para el proceso de comparación.

Directriz arquitectónica (objetivo de negocio)	**Referencia a vistas generadas**
1. Edición de flujos de autorización (DA1).	Sección 3.6.1, Figuras 3.4 y 3.5
2. Modificación de datos en reportes (DA2).	Sección 3.6.2, Figuras 3.6 y 3.7
3. Integración de módulos externos (DA3).	Sección 3.7.1, Figuras 3.8 y 3.10
4. Integración de un sistema externo (DA4).	Sección 3.7.1, Figuras 3.9 y 3.11
5. Modificación en la funcionalidad de un sistema externo (DA5).	Sección 3.7.2, Figuras 3.12 y 3.13

Tabla 4.4: Referencias a vistas generadas de cada patrón, para cada directriz arquitectónica.

De acuerdo con el título de esta sección, no sólo se espera obtener como salida el conjunto de vistas arquitectónicas para cada una de las directrices. Otro punto importante que se espera como resultado de esta fase del proceso es la definición de indicadores.

Los indicadores permiten evaluar para cada estilo arquitectónico su desempeño en relación a cada una de las directrices arquitectónicas. En este caso, es de especial interés el desempeño de los patrones arquitectónicos en cuanto a modificabilidad e intergabilidad. Por ello, se seleccionó como indicadores a las técnicas de modificabilidad propuestas por Bachmann *et al*. [2007].

El grado en que cada patrón analizado implemente estas técnicas, determinaría su nivel de modificabilidad.

4.6 Fase 5. Asignación de puntuación

Finalmente, para definir la puntuación asignada a cada patrón arquitectónico en relación a las directrices arquitectónicas e indicadores de modificabilidad, se convocó a reunión a los participantes en el proceso de evaluación. En dicha reunión se llevó acabo el siguiente proceso para la asignación de puntuaciones.

Primero, se presentaron nuevamente los sujetos de comparación y la forma en la que estos abordaban cada una de las directrices arquitectónicas. Después, con base en la implementación o no implementación de cada una de las técnicas de diseño seleccionadas como indicadores, se asignó una puntuación de 1 a 5, en donde cada valor representa un valor cualitativo desde muy bajo (valor de 1) a muy alto (muy bajo, bajo, medio, alto y muy alto) de acuerdo al grado de implementación de cada indicador. Esta puntuación fue determinada en consenso entre los participantes.

Mediante el proceso anterior se obtuvieron las puntuaciones de manera individual para cada indicador de modificabilidad.

A continuación se muestran en las Tablas 4.5 y 4.6 las puntuaciones obtenidas por cada patrón arquitectónico en relación a cada indicador de modificabilidad. La última fila de ambas tablas representa la puntuación total obtenida por cada patrón para cada una de las directrices arquitectónicas.

Técnica de modificabilidad	DCI (DA1)	EDCI (DA1)	Justificación	DCI (DA2)	EDCI (DA2)	Justificación
1. Aseguramiento de coherencia semántica.	3	4	EDCI ayuda a mejorar la coherencia semántica, separando el código modificable del código estable.	2	4	El enfoque propuesto EDCI, evita la creación de entidades compuestas o modelos de vistas, habilitando modificaciones en tiempo de ejecución.
2. Abstracción de servicios comunes.	3	4	EDCI abstrae los algoritmos de lógica de autorización a servicios comunes mediante la implementación del patrón *estrategia*.	3	4	La generación de consultas para obtener los resultados de los reportes se encuentra abstraída y parametrizada para EDCI en la capa de acceso a datos.
3. Uso de encapsulamiento.	3	4	EDCI agrega una capa de acceso a datos que encapsula todas las operaciones de interacción con la fuente de datos, liberando al modelo de posibles modificaciones.	3	4	Al igual que en el caso de la directiva anterior, se tiene que mencionar la ausencia de una capa de acceso a datos en el primer patrón arquitectónico y su implementación en el segundo.
4. Uso de envolturas.	0	0	Ninguno de los patrones implementa esta técnica para esta directriz.	0	0	Ninguno de los patrones implementa esta técnica para esta directriz.
5. Restricción de canales de comunicación.	3	4	EDCI añade una restricción de comunicación adicional al agregar una capa de acceso a datos.	3	4	EDCI añade una restricción adicional obligando a que el flujo de datos ocurra única y exclusivamente por medio de la capa de acceso a datos (DAL).
6. Uso de intermediarios.	3	4	La capa de acceso a datos (DAL) en EDCI, funciona como intermediario entre las interacciones y la operación de la base de datos.	3	4	EDCI agrega un intermediario adicional para el acceso a datos.
7. Escalamiento de niveles de abstracción.	4	4	EDCI al implementar estrategias y una capa de acceso a datos agrega un nivel intermedio de abstracción adicional no considerado para este indicador al ubicarse a más bajo nivel.	4	4	En EDCI se agregó una capa exclusiva para la construcción de reportes que también actúa mediante ejecución de contextos, manteniendo el mismo máximo nivel de abstracción que DCI.
Puntuación total	19	24		18	24	

Tabla 4.5: Puntuación asignada a cada sujeto de comparación para cada directriz de modificabilidad respecto a indicadores (técnicas de modificabilidad).

Técnica de modificabilidad	DCI (DA3/DA4)	EDCI (DA3/DA4)	Justificación	DCI (DA5)	EDCI (DA5)	Justificación
1. Aseguramiento de coherencia semántica.	2	4	La necesidad de definir procesos de interacción a nivel modelo en DCI, es mitigada mediante la localización de posibles cambios en un solo componente.	2	4	La técnica de separación de responsabilidades y localización de cambios por parte de EDCI incrementa el grado de coherencia semántica.
2. Abstracción de servicios comunes.	3	4	La agrupación de métodos consumibles es una abstracción de servicios comunes, ya que estos pueden ser reutilizados por varios procesos.	3	4	Tanto servicios consumibles como servicios a consumir se abstraen en componentes (servicios web) aislados en EDCI.
3. Uso de encapsulamiento.	4	5	La capa de acceso a datos mejora de antemano el nivel de esta técnica y los servicios web como capa de integración, agregan un nivel más de encapsulación.	4	5	La capa de acceso a datos misma que es diseñada acorde a las técnicas recomendadas para directrices anteriores, proporciona los mismos beneficios en esta directriz.
4. Uso de envolturas.	0	3	Mediante la implementación de envolturas en la capa de acceso a datos es posible evitar la propagación de cambios desde el módulo externo hacia el sistema caso de estudio.	0	3	Mediante la implementación de envolturas en la capa de acceso a datos es posible evitar la propagación de cambios desde el sistema externo hacia el caso de estudio.
5. Restricción de canales de comunicación.	3	4	EDCI añade una restricción adicional obligando a que el flujo de datos ocurra única y exclusivamente por medio de la capa de acceso a datos DAL.	3	4	Los canales de comunicación quedan restringidos a los contextos en el caso de DCI y en la capa de acceso a datos (además de en los contextos) en el caso de EDCI.
6. Uso de intermediarios.	3	4	EDCI tiene en la capa de acceso a datos un intermediario adicional.	3	4	En DCI se utilizan los contextos e interacciones como intermediarios, mientras que EDCI agrega la capa de acceso a datos como intermediario adicional.
7. Escalamiento de niveles de abstracción.	4	4	El grado más elevado de abstracción se localiza a nivel de contextos en ambos patrones arquitectónicos.	4	4	También para esta directriz el nivel máximo de abstracción se localiza en los contextos.
Puntuación total.	19	28		19	28	

Tabla 4.6: Puntuación asignada a cada sujeto de comparación para cada directriz de integrabilidad respecto a indicadores (técnicas de modificabilidad).

Para obtener la puntuación a nivel de directrices arquitectónicas, es decir, a un nivel más general, se llevó acabo el siguiente proceso:

1. Para cada directriz arquitectónica, se sumaron las puntuaciones obtenidas respecto a cada indicador obteniendo un total parcial del desempeño de cada patrón arquitectónico. Dicho total parcial se observa en la última fila de las Tablas 4.5 y 4.6.

2. Debido a que no todas las directrices arquitectónicas tienen el mismo nivel de importancia o impacto en el caso de estudio, se les asignó un valor de ponderación asociado a este nivel de impacto.

3. Dicho valor de ponderación es multiplicado por el total parcial de las puntuaciones obtenidas por cada patrón respecto a cada una de las directrices.

4. Finalmente como resultado, se obtuvo un total absoluto, el cual representa la suma de las puntuaciones obtenidas, considerando el grado de importancia para el negocio de cada directriz arquitectónica.

La Tabla 4.7 muestra para cada directriz arquitectónica la sumatoria de las puntuaciones obtenidas por cada patrón respecto a indicadores, misma que ha sido multiplicada por un valor de ponderación dando como resultado una puntuación final considerando el nivel de impacto de cada directriz arquitectónica.

Directriz arquitectónica	Valor de ponderación	DCI	EDCI	DCI (ponderación)	EDCI (ponderación)
1. Edición de flujos de autorización (DA1).	5	19	24	95	120
2. Modificación de datos en reportes (DA2).	3	18	24	54	72
3. Integración de un módulo externo (DA3).	5	19	28	95	140
4. Integración de un sistema externo (DA4).	3	19	28	57	84
5. Modificación en la funcionalidad de un sistema externo (DA5).	3	19	28	57	84

Tabla 4.7: Puntuaciones ponderadas obtenidas por directriz arquitectónica considerando el impacto de cada una de ellas.

Una vez obtenidas las calificaciones respecto a indicadores y directrices, sólo queda comprobar si los criterios de satisfacción de las directrices (respuesta/tiempo de respuesta) efectivamente se cumplen.

Para esto, se realizó otra reunión pero esta vez con el equipo de desarrollo (cuatro integrantes) que participaría en la construcción del proyecto. En esta reunión se les solicitó a los asistentes que tomando en cuenta cada uno de los patrones arquitectónicos comparados, estimaran un tiempo de respuesta (en horas-hombre) para cada directriz.

Considerando la complejidad de respuesta para cada directriz respecto a cada patrón, cada desarrollador estimó el tiempo que le tomaría responder a cada una de las directrices. Una vez que cada desarrollador realizó su estimación, cada uno expuso sus razones de porqué asignó tal número de horas. De esta manera, comparando diversos puntos de vista se fueron ajustando las estimaciones hasta llegar a un consenso.

Los resultados finales se muestran en la Tabla 4.8 y están dados en horas-hombre, ya que se considera el caso en el que un sólo desarrollador respondería a la directriz en caso de presentarse.

Directriz arquitectónica	DCI	EDCI	Comentarios
1. Edición de flujos de autorización (DA1)	16	10	Considerando el caso en el que se tiene que modificar o agregar un flujo de autorización aplicable a dos entidades del modelo de dominio. En DCI, sería necesario agregar una interacción para cada entidad, mientras que en EDCI el algoritmo de autorización podría definirse de una única vez, a manera de estrategia.
2. Modificación de datos en reportes (DA2)	8	0	Para este caso en DCI se tendría que agregar un nuevo atributo en la entidad correspondiente del modelo al igual que en el enlace de datos para que el atributo sea llenado. En el caso de EDCI, al tener consultas parametrizadas, la carga de los atributos sería meramente dinámica, es decir no requeriría ni siquiera de recompilación del código fuente.
3. Integración de módulos externos (DA3)	24	16	El tiempo de respuesta para esta directriz dependería del tamaño del módulo que se quiere integrar. En el caso simple de un catálogo con altas, bajas y cambios, en DCI tendríamos que agregar el modelo correspondiente y el resto de los componentes de la arquitectura. Con EDCI basta con crear el servicio web que proveerá el enlace al módulo externo y consumirlo.
4. Integración de un sistema externo (DA4)	24	16	Para esta directriz se consideró el mismo caso que para la anterior.
5. Modificación en la funcionalidad de un sistema externo (DA5)	8	4	Considerando el caso en el que el tipo de un dato provisto por un sistema externo se modifica, en DCI tendría que modificarse la entidad correspondiente del modelo para ese tipo de dato, al igual que la lógica de negocio donde dicho dato pueda generar un error. En el caso de EDCI las modificaciones quedarían localizadas en la capa de acceso a datos, la que actuaría como envoltura y sería la encargada de hacer la transformación de los datos y evitar que el cambio se propague.
Total	80	46	

Tabla 4.8: Estimaciones de tiempo de respuesta para cada directriz arquitectónica, definidas por el equipo de desarrollo de GRP en horas-hombre.

4.7 Fase 6. Resumen

Después de aplicar el método SACAM para evaluar las mejoras brindadas por EDCI, las puntuaciones obtenidas pueden ser presentadas desde tres diferentes perspectivas: directrices arquitectónicas, indicadores de modificabilidad (técnicas) y tiempos de respuesta para cada directriz.

En la Figura 4.3 se presenta un resumen de las puntuaciones obtenidas por cada sujeto de comparación, en relación a las diferentes directrices arquitectónicas consideradas como criterios de comparación. La presentación de los resultados desde esta perspectiva, ofrece una visión general de las mejoras obtenidas.

En general, EDCI ofrece una mejora en la totalidad de las directrices arquitectónicas analizadas,

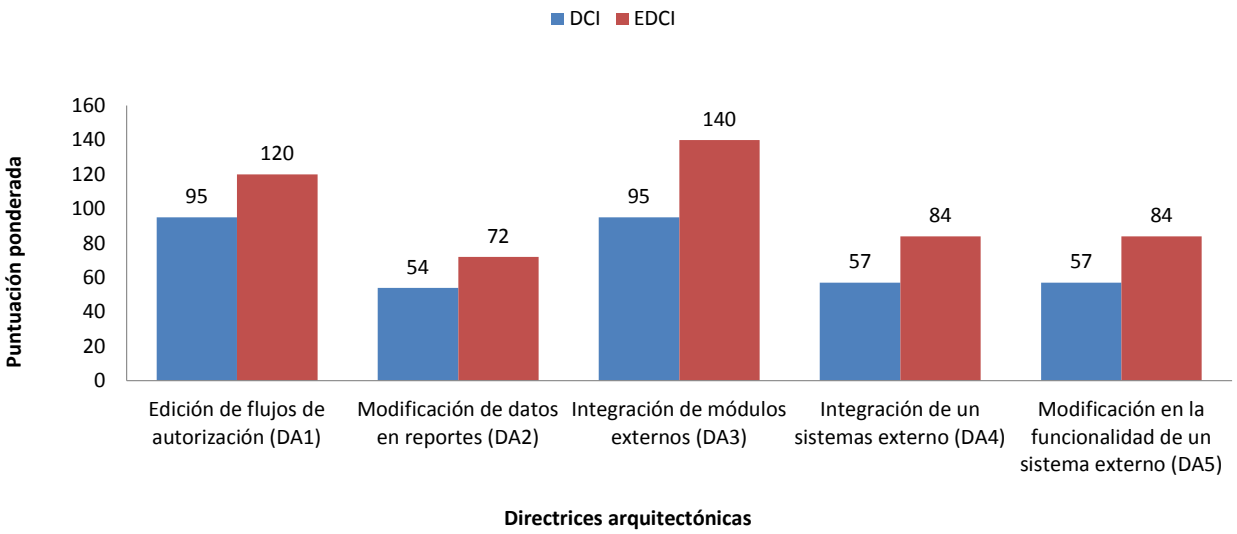

Figura 4.3: Resultados de la comparación desde una perspectiva de directrices arquitectónicas

especialmente para DA3, DA4 y DA5, las cuales están relacionadas con el atributo de integrabilidad. La mejora obtenida respecto a las dos primeras directrices aunque importante, es menor.

Mientras que para DA3, DA4 y DA5 el porcentaje de mejora es del 47.36 % el porcentaje de mejora para DA1 y DA2 está por debajo del 33.33 %. Estos números permiten ver el hecho de que la mayor oportunidad de mejora para DCI, es en relación a la integrabilidad y que dicha oportunidad de mejora, es bien cubierta por EDCI.

La Figura 4.4 detalla la puntuación obtenida por cada estilo arquitectónico en relación a los diferentes indicadores considerados para la comparación (técnicas de modificabilidad). Este enfoque permite mayor nivel de especificidad para identificar áreas en las cuales un estilo arquitectónico supera al otro.

Desde esta perspectiva se puede apreciar cuales son las fortalezas y debilidades de cada sujeto comparado.

Las mejoras más representativas ofrecidas por EDCI son en relación al aseguramiento de coherencia semántica y el uso de envolturas, en donde se logró un porcentaje de superior al 81.81 %. El uso de envolturas puede ser considerado el mayor aporte de EDCI, ya que gracias a la implementación

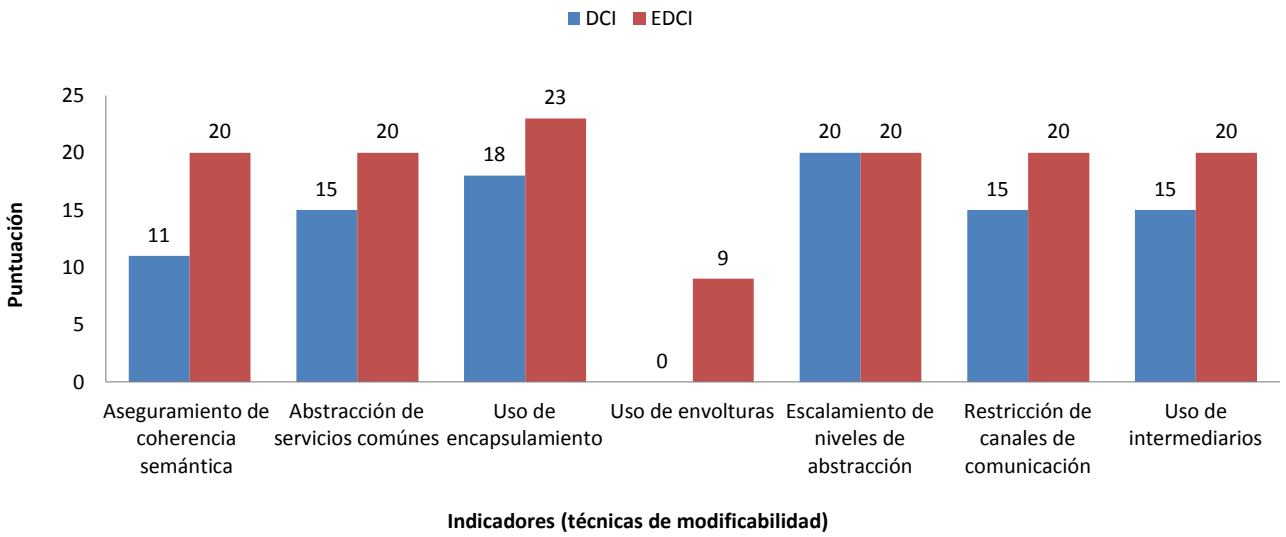

Figura 4.4: Resultados de la comparación desde una perspectiva de indicadores de comparación (técnicas de modificabilidad).

de esta técnica, es posible resolver los problemas de integrabilidad de DCI.

Para la técnica de escalamiento de niveles de abstracción, no se reporta alguna mejora, dado que para ambos patrones, el nivel más elevado de abstracción es el mismo y resulta adecuado. Para los indicadores restantes, el porcentaje de mejora obtenido es del 33.3 %.

Finalmente la Figura 4.5 muestra el tiempo estimado por el equipo de desarrollo, para dar respuesta a la ocurrencia de cada una de las directrices arquitectónicas.

A diferencia de las Gráficas 4.3 y 4.4, en está gráfica se muestra superioridad por parte de DCI. Dado que las barras denotan el tiempo de respuesta de cada patrón ante las diferentes directrices arquitectónicas, el hecho de que EDCI sea inferior es positivo.

De manera global con la implementación de EDCI se logró una reducción de 50.83 %. La mejora más significativa se obtuvo para la directriz arquitectónica DA2, la cuál redujo a cero el tiempo de respuesta, es decir, redujo el tiempo de respuesta al 100 %. Mientras que la mejora de menor impacto se registró para las directrices DA3 y DA4, la cual es del 33.3 %.

Con toda esta información se puede concluir que EDCI es de hecho una versión mejorada de

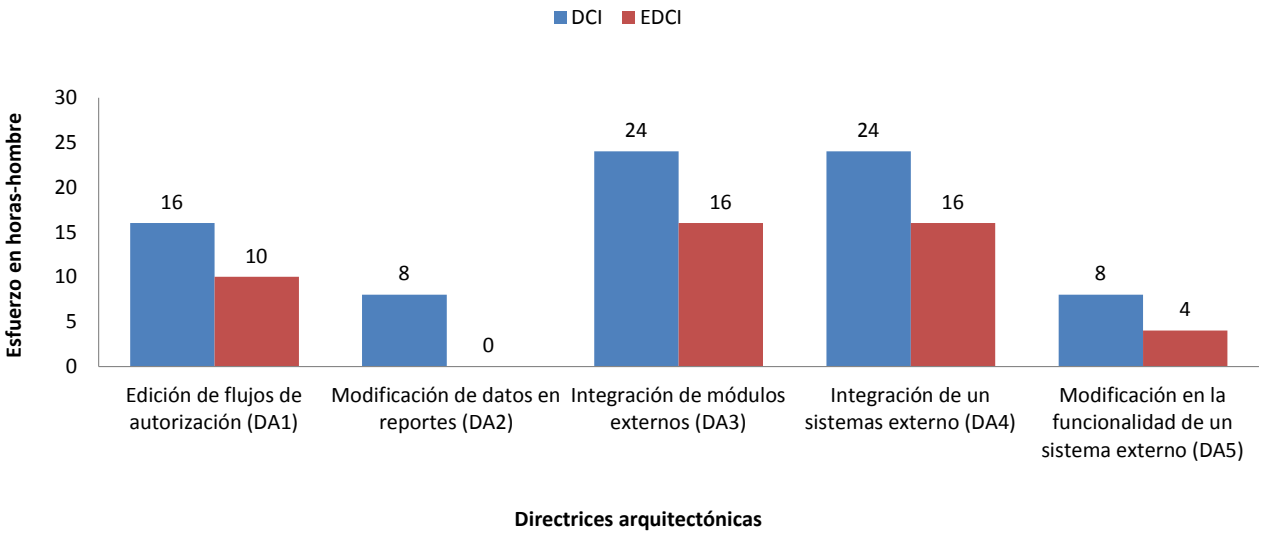

Figura 4.5: Resultados de la estimación de tiempos de respuesta para cada directriz, indicado en horas-hombre.

DCI tanto en lo que corresponde al nivel de modificabilidad e integrabilidad (especialmente para este último atributo), como en relación al tiempo de respuesta del patrón ante cada directriz arquitectónica.

Resumen del capítulo

En este capítulo, se presentó paso a paso el proceso de evaluación del patrón arquitectónico propuesto como resultado de esta investigación. Este proceso consistió básicamente en la comparación directa entre el patrón propuesto y el patrón arquitectónico tomado como base para el diseño del mismo utilizando el método.

Como conclusión del proceso de evaluación, se presentaron los resultados obtenidos desde tres perspectivas diferentes: a) una perspectiva general desde el enfoque de las directrices arquitectónicas seleccionadas. b) una perspectiva detallada, desde las diferentes técnicas de diseño consideradas como indicadores de modificabilidad e integrabilidad y c) una perspectiva que toma en cuenta el tiempo de respuesta estimado para cada una de las directrices en caso de que se presenten.

5

Conclusiones

En este capítulo se presenta un resumen acerca de los productos y resultados de esta investigación, así como del proceso seguido para llegar a ellos. Además, se presentan las conclusiones derivadas de los resultados obtenidos en el Capítulo 4, las contribuciones finales y algunas áreas de oportunidad a manera de trabajo futuro.

5.1 Introducción

El objetivo principal de esta investigación planteado en la Sección 1.3 fue alcanzado de manera satisfactoria. Esto mediante el diseño de un patrón arquitectónico denominado Datos, Contexto e Interacción Mejorados (EDCI). Este patrón arquitectónico consiste en una versión mejorada de un patrón arquitectónico de reciente inclusión al estado del arte, el patrón Datos, Contexto e Interacción (DCI).

A diferencia de DCI, EDCI añade un par de componentes arquitectónicos con la intensión de cubrir escenarios de modificabilidad e integrabilidad (atributos característicos del software ágil).

Primero, EDCI añade una capa de acceso a datos con la intención de romper la dependencia

existente entre la capa de datos y las fuentes de datos propiamente, siguiendo el principio de mantener la capa de datos exenta de posibles modificaciones.

Además de lo anterior EDCI añade una capa de presentación de reportes tabulares con la finalidad de habilitar su definición en tiempo de ejecución y no en tiempo de diseño. Esto es posible mediante la implementación de consultas dinámicas en la propia capa de acceso a datos.

El patrón de diseño *estrategia* es implementado con la intención de centralizar la definición de reglas de flujos de autorización posiblemente compartidas y dotándoles de la habilidad de ser intercambiables.

Finalmente, EDCI agrega un par de capas adicionales (interfaces) para la comunicación con módulos o sistemas externos, de manera que los escenarios de modificabilidad que se presenten en tales módulos o sistemas, queden bien localizados y aislados a un solo componente y de esa manera, prevenir la propagación de efectos a raíz de dichos cambios a otros componentes.

Para derivar en el diseño de EDCI fue necesario ir cubriendo de manera paulatina una serie de objetivos específicos.

Primero, teniendo en cuenta las características principales del software ágil, se realizó un estudio del estado del arte. Esto con el fin de conocer cómo los diversos patrones arquitectónicos orientados a modificabilidad e integrabilidad podían cubrir tales características y cuales eran los mejores en ello.

A partir de dicho estudio se encontró que los patrones más aptos para abordar necesidades de modificabilidad e integrabilidad eran los patrones Broker y Blackboard. Sin embargo, con un mejor desempeño y relativamente de reciente inclusión en el estado del arte se encontró al patrón Datos, Contexto e Interacción (DCI).

Este patrón arquitectónico (DCI) fue tomado como base para el diseño de un patrón que facilitara el desarrollo ágil de software, incrementando el nivel de los atributos de calidad característicos del software ágil, respecto al patrón base.

Para realizar el diseño del patrón arquitectónico propuesto (EDCI), se siguió un reconocido método de diseño de software conocido como Diseño Dirigido por Atributos (ADD)[Wojcik *et al.*, 2006], el cual básicamente consiste en identificar aquellos atributos de calidad prioritarios para el sistema a desarrollar e implementar técnicas de diseño con la finalidad de cubrir los atributos

identificados.

Dado que el método de diseño ADD requiere como principales entradas los requerimientos de un proyecto tanto funcionales como no funcionales, se seleccionó un sistema como caso de estudio. El sistema seleccionado consistió en un proyecto de planificación de recursos actualmente en proceso de desarrollo para CINVESTAV, Tamaulipas. A partir de los requerimientos de este sistema (funcionales, no funcionales y restricciones de diseño) fue posible formular algunas directrices arquitectónicas las cuales rigieron el proceso de diseño y se convirtieron en criterios de evaluación posteriormente.

Una vez concluido el proceso de diseño de EDCI se procedió a evaluarlo mediante comparación en contra del patrón DCI. Para ello se implementó un método de evaluación de arquitecturas de software conocido como Método de Análisis y Comparación de Arquitecturas de Software (SACAM)[Stoermer *et al.*, 2003].

5.2 Conclusión

Los resultados de esta evaluación arrojaron resultados positivos entre los cuales se encuentran los siguientes.

Partiendo desde lo general a lo particular se encontró que EDCI presenta un mejor desempeño ante la totalidad de las directrices arquitectónicas identificadas. En cuanto a las directrices de integrabilidad, el porcentaje de mejora ofrecido por EDCI fue de 47.36 %, mientras que en cuanto a las directrices de modificabilidad el porcentaje de mejora fue como máximo del 33.3 %. Con esto se puede concluir, por un lado, que EDCI efectivamente cumple con el objetivo de la investigación, pero además, pone en evidencia el atributo de calidad débil de DCI (integrabilidad) al ser éste sobre el cual el porcentaje de mejora fue mayor.

Desde una perspectiva más especifica, EDCI ofrece mejoras en seis de siete indicadores de modificabilidad e integrabilidad. Principalmente para los indicadores de mantenimiento de coherencia semántica y el uso de envolturas en donde presenta un porcentaje de mejora superior al 81.81 %, en contraste con el indicador de escalamiento de niveles de abstracción, sobre el cual no se registro porcentaje de mejora. Esto sin embargo, no es del todo negativo, dado que el nivel de abstracción

de DCI es por sí mismo adecuado. Estos resultados, permiten identificar qué indicadores significan áreas de oportunidad para DCI de manera que otros arquitectos puedan también abordarlas, quizás mediante la implementación de técnicas diferentes a las implementadas en este trabajo.

En cuestión del tiempo de respuesta para cada una de las directrices arquitectónicas EDCI también presenta mejoras al reducir el tiempo de respuesta a cada directriz arquitectónica respecto a DCI. La mejora más significativa en este rubro se da sobre la directriz de modificación de datos en reportes. Al permitir la edición de reportes en tiempo de ejecución se evita la necesidad de realizar cambios en tiempo de diseño en caso de que el escenario de la directriz se presente. Por otro lado el mas negativo de los resultados en este aspecto sería la mejora del 33.3 % para la integración de módulos o sistemas externos, sin embargo a pesar de ser la menor de las mejoras obtenidas, sigue siendo una mejora.

A continuación se describen algunos aspectos sobre los cuales podría continuarse con este trabajo de investigación, algunos de ellos con base en áreas de oportunidad del mismo.

5.3 Trabajo futuro

Dado que el patrón EDCI fue diseñado con base en un caso de estudio particular y bajo condiciones controladas muy específicas, el alcance de solución que ofrece sigue siendo limitado, es decir, quizás a pesar de representar mejoras y ventajas al implementarlo en proyectos de tipo GRP, no tenga la misma eficacia en otro tipo de proyectos.

Por lo tanto, una opción de trabajo futuro sería el implementar este patrón en otra clase de proyectos y evaluar su eficacia. Esto a su vez, permitiría descartar cualquier dependencia de los resultados obtenidos con el caso de estudio particular.

Otra opción para trabajo futuro podría ser realizar un estudio exhaustivo con la finalidad de obtener un conjunto de escenarios de modificabilidad e integrabilidad más comunes en la industria del software, definir directrices arquitectónicas a partir de ellos y de allí ajustar el patrón para hacerlo más genérico e incrementar su alcance.

Finalmente, de manera más específica y relacionada con el caso de estudio en particular, este

patrón podría ser sometido a comparación en contra del patrón arquitectónico *Onion* [Palermo, 2008], el cual fue utilizado realmente para el desarrollo del proyecto GRP. Hacer esto tendría la finalidad de detectar ventajas y desventajas de un patrón con respecto al otro y así ajustar el enfoque propuesto (EDCI) para incrementar su eficacia.

A

Planning Poker

En esta sección se presenta el método de Planning Poker, utilizado en la estimación de esfuerzos para la implementación de los requerimientos funcionales y no funcionales del caso de estudio utilizado en esta investigación.

A.1 Planning Poker

La técnica de planning poker, es una técnica ágil de estimación de esfuerzos que se basa en la discusión de diversos puntos de vista acerca del esfuerzo requerido para realizar una tarea (en este caso, para implementar un proceso).

Cada participante en el proceso de estimación recibe un mazo de cartas marcadas con los números: 0, 1, 2, 3, 5, 8, 13, ..., 89, acorde a la sucesión de Fibonacci. Se utiliza esta sucesión como escala de estimación, ya que el espacio entre los elementos de la serie, refleja la incertidumbre existente para la estimación de requerimientos o procesos más grandes.

Un moderador con suficiente conocimiento de los requerimientos (por ejemplo, el rol de *project owner* en SCRUM), explica un requerimiento y responde las dudas de los participantes acerca de

éste. Una vez que el requerimiento es clarificado cada participante selecciona de su mazo de cartas un valor que corresponde al número de horas que estima para implementar el requerimiento, pero sin mostrarlo al resto.

Finalmente, a la indicación del moderador, todos muestran sus estimados. Por lo general en esta parte del proceso suelen presentarse discrepancias de moderadas a muy significativas en las estimaciones. En cualquiera que sea el caso, los participantes con el menor y el mayor estimado, exponen por turnos la razón de su estimación y son discutidas por todo el equipo hasta llegar un consenso y formalizar la estimación fijando un valor determinado [Cohn, 2005].

Referencias

Abrahamson, P., Salo, O., y Ronkainen, J. [2002]. *Agile software development methods - review and analisys* (Inf. Téc. n.º 478). VTT Electronics.

Bachmann, F., Bass, L., y Nord, R. [2007]. *Modifiability tactics* (Inf. Téc.). Software Engineering Institute. Descargado de `http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=8299`

Barbacci, M., Ellison, R., Lattanze, A., Stafford, J., Weinstock, C., y Wood, W. [2002]. *Quality attribute workshop, 2nd edition* (Inf. Téc.). Software Engineering Institute. Descargado de `http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=6113`

Bass, L., Clements, P., y Kazman, R. [2003]. *Software architecture in practice* (2nd. ed.). Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc.

Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., ... Sutherland, J. [2001]. *The agile manifesto.* Visitado el 23 de marzo de 2013 de http://agilemanifesto.org. Descargado Visitado el 23 de marzo de 2013, de `http://agilemanifesto.org`

Bengtsson, P., Lassing, N., y Bosch, J. [2002, January]. Architecture-level modifiability analysis. *Journal of Systems and Software*, *69*(1-2), 129–147. Descargado de `http://dx.doi.org/10.1016/S0164-1212(03)00080-3` doi: 10.1016/S0164-1212(03)00080-3

Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., y Stal, M. [1996]. *Pattern-oriented software architecture: a system of patterns*. New York, NY, USA: John Wiley & Sons, Inc.

Clements, P., Bachkmann, F., Bass, L., Garlan, D., Ivers, J., Little, R., ... Stafford, J. [2010]. *Documenting software architectures: Views and beyond* (2nd. ed.). Software Engineering Institute (SEI).

Cohn, M. [2005]. *Agile estimating and planning*. Upper Saddle River, NJ, USA: Prentice Hall PTR.

Coplien, J. O., y Bjrnvig, G. [2010]. *Lean architecture: for agile software development*. Wiley Publishing.

Craig, I. [1995]. *Blackboard systems*. Ablex Publishing Corporation.

Engelmore, R., y Morgan, T. [1988]. *Blackboard systems*. Addison-Wesley.

Folmer, E., Van Gurp, J., y Bosch, J. [2005]. Software architecture analysis of usability. En *Proceedings of the 2004 international conference on engineering human computer interaction and interactive systems* (p. 38-58). Berlin, Heidelberg: Springer-Verlag. Descargado de http://dx.doi.org/10.1007/11431879_3 doi: 10.1007/11431879_3

Ford, G. [2009]. *Sei report on undergraduate software engineering education (2009)* (Inf. Téc.). Software Engineering Institute, Carnegie Mellon University.

Gamma, E., Helm, R., Jhonson, R., y Vlissides, J. [2002]. *Design patterns: Elements of reusable object-oriented software*. Addison-Wesley Professional Computing Series.

IEEE. [1990, December]. IEEE standard glossary of software engineering terminology. *IEEE Std 610.12-1990*, 1–84. Descargado de http://ieeexplore.ieee.org/servlet/opac?punumber=2238 doi: 10.1109/IEEESTD.1990.101064

Jeon, S., Han, M., Lee, E., y Lee, K. [2011]. Quality attribute driven agile development. En *Proceedings of the 2011 ninth international conference on software engineering research, management and applications* (pp. 203–210). Washington, DC, USA: IEEE Computer Society. Descargado de http://dx.doi.org/10.1109/SERA.2011.24 doi: 10.1109/SERA.2011.24

Kazman, R., Bass, L., Abowd, G., y Webb, M. [1994]. Saam: A method for analyzing the properties of software architectures. En *Proceedings of the 16th international conference on software engineering* (pp. 81–90). Los Alamitos, CA, USA: IEEE Computer Society Press.

Lassing, N., Rijsenbrij, D., y Vliet, H. [1993, November]. On software architecture analysis of flexibility, complexity of changes: Size isn´t everithing. En *Proceedings of 2nd nordic software architecture workshop, 1999.* (Vol. 23, pp. 111–122).

Mnkandla, E. [2009, September]. About software engineering frameworks and methodologies. En *Africon, 2009. africon '09.* (pp. 1–5).

Naur, P., y Randell, B. [1968, Octubre, 1968]. Working conference on software engineering. En B. Naur P; Randell (Ed.), *Software engineering: Report of a conference sponsored by nato science committee* (p. 70-77). Garmisch, Alemania.

Nord, R., y Tomayko, E. [2006, March]. Software architecture centric methods and agile development. *IEEE Software*, *23*(2), 47–53. Descargado de http://dx.doi.org/10.1109/MS.2006.54

doi: 10.1109/MS.2006.54

Palermo, J. [2008, July]. *The onion architecture.* Visitado el 8 de noviembre de 2013 de http://jeffreypalermo.com/blog/the-onion-architecture-part-1/. Descargado de `http://jeffreypalermo.com/blog/the-onion-architecture-part-1/`

Reenskaug, T. [1979]. *The model-view-controller (mvc) its past and present* (Inf. Téc.). Xerox PARC.

Reenskaug, T., y Coplien, J. O. [2009, March]. *The dci architecture: A new vision of object-oriented programming.* Visitado el 21 de septiembre de 2013 de http://www.artima.com/articles/dci_vision.html. Descargado de `http://www.artima.com/articles/dci\textunderscorevision.html`

Roy, B., y Graham, N. [2008]. *Methods for evaluating software architecture: A survey* (Inf. Téc.). School of Computing, Queen's University at Kingston.

Runeson, P., y Höst, M. [2009, April]. Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering*, *14*(2), 131–164. Descargado de `http://dx.doi.org/10.1007/s10664-008-9102-8` doi: 10.1007/s10664-008-9102-8

Stoermer, C., Bachmann, F., y Verhoef, C. [2003]. *Sacam: The software architecture comparison analysis method* (Inf. Téc.). Software Engineering Institute.

Vliet, H. [2008]. *Software engineering: Principles and practice* (3rd. ed.). Wiley Publishing.

Wojcik, R., Bachkmann, F., Bass, L., Clements, P., Mersson, P., Nord, R., y Wood, B. [2006]. *Attribute-driven design (add), version 2.0* (Inf. Téc.). Software Engineering Institute.