Reporte Técnico

Proceso de evaluación para arquitecturas de software usadas en el sector empresarial

Omar Salvador Gómez Gómez

Versión 2.0
Julio 2005

Histórico de revisiones

Versión	Fecha	Descripción	Autor
0.1	06.07.05	Primer borrador.	OG
0.2	10.07.05	Corrección de sección de introducción y trabajo previo.	OG
0.3	11.07.05	Corrección de trabajo previo y propuesta de solución.	OG
0.4	12.07.05	Corrección de propuesta de solución y problemática, se agregaron las referencias en la sección 7.	OG
0.5	13.07.05	Corrección de glosario de términos, y revisión del documento.	OG
0.6	14.07.05	Se cambio el formato de las referencias.	OG
1.0	15.07.05	Versión 1.0	OG
1.1	16.07.05	Se agregaron las secciones resumen, agradecimientos, lista de figuras y lista de tablas.	OG
1.2	17.07.05	Ajuste de imágenes y encabezado del documento.	OG
1.3	19.07.05	Corrección ortográfica al documento.	OG
2.0	20.07.05	Versión final del documento	OG

Tabla de Contenido

Lista de figuras

Lista de tablas

Resumen

El éxito o fracaso de un sistema de software esta determinado por su arquitectura. La única manera de saber si una arquitectura de software es exitosa es evaluándola. El propósito de evaluar arquitecturas a sistemas de software es analizar la arquitectura para identificar riesgos potenciales y para verificar que los requerimientos de calidad hayan sido contemplados en el diseño de la arquitectura. En este documento se presenta un proceso para evaluación de arquitecturas de software usadas en el sector empresarial llamado PEASSE. PEASSE identifica patrones de diseño para posteriormente evaluar arquitecturas de software haciendo uso de métodos de evaluación existentes.

Palabras clave: Evaluación de arquitecturas de software, patrones de diseño, ALMA, SNA, PEASSE, DFSS, IDOV.

Agradecimientos

Este trabajo fue parcialmente financiado por el Consejo de Ciencia y Tecnología del Estado de Guanajuato (CONCYTEG) a través del Proyecto Número GTO-2002-C01-5333 titulado Promoviendo calidad en la industria de software: Recursos humanos, investigación, servicios".

Al Doctor Cuauhtémoc Lemus Olalde por sus valuables recomendaciones dadas para la elaboración de este documento.

1 Introducción

En esta sección se presentan una serie de conceptos e ideas que serán usadas en el presente documento así como también se presenta la estructura de este documento.

El proceso de evaluación de arquitecturas de software que se propone en este documento esta basado en la metodología DFSS. DFSS es una metodología que tiene sus orígenes en seis sigma, la cual es usada principalmente en el área de manufactura por muchas empresas a nivel mundial.

DFSS ayuda a mejorar la habilidad de conocer mejor los requerimientos del cliente, así como reducir la complejidad y variabilidad de los procesos en una empresa. Esta mejora se logra a través de una serie de técnicas y herramientas estadísticas que son parte de seis sigma. DFSS es una metodología que es usada por empresas para diseñar o rediseñar productos, procesos o servicios. En este documento se hace una adecuación para usar la metodología DFSS en el proceso de evaluación de arquitecturas de software.

De acuerdo a Bass, Clements y Kazman [06] la arquitectura de software de un programa o sistema computacional es la estructura o estructuras del sistema, las cuales constan de: Elementos de software (que en lo sucesivo llamaremos componentes), propiedades visibles externamente entre estos componentes y las relaciones entre ellos.

Dependiendo la manera de cómo los elementos de una arquitectura de software son relacionados, Shaw y Garlan identificaron varios estilos arquitectónicos. Un estilo arquitectónico se define como un vocabulario de componentes y tipos de componentes, así como un conjunto de restricciones sobre como estos componentes pueden ser combinados. Algunos de estos estilos arquitectónicos son: Procesamiento secuencial, tuberías y filtros, sistemas orientados a objetos, sistemas en capas, sistemas basados en eventos, bases de datos, entre otros [07].

Es posible que una arquitectura de software este compuesta por varios estilos arquitectónicos, como es el caso de la mayoría de los sistemas de información modernos que utilizan las empresas hoy en día. Los estilos arquitectónicos que han sido identificados en este dominio de aplicación son una mezcla de sistemas compuestos por capas que utilizan bases de datos y están construidos por pequeñas unidades de programas llamadas objetos.

Debido a la complejidad de los sistemas de información actuales, no es suficiente que una arquitectura de software este compuesta por uno o más estilos arquitectónicos. Es necesario que los componentes que conforman una arquitectura estén relacionados unos a otros de manera que éstos implementen uno o más patrones de diseño. Un patrón de diseño describe una estructura comúnmente recurrente de comunicación entre componentes que resuelve un problema de diseño general dentro de un contexto en particular [08]. Por lo que un patrón de diseño proporciona un esquema para refinar los componentes dentro de una arquitectura o para refinar las relaciones entre ellos.

Un *Framework* es un conjunto de clases que cooperan entre sí para hacer un diseño reutilizable de un sistema de software. Un *Framework* puede ser configurable para una aplicación en particular creando subclases específicas de las clases abstractas que componen el *Framework*. Un *Framework* captura las decisiones de diseño que son comunes para un dominio de aplicación, por lo que hace énfasis en reutilizar diseño en vez de reutilizar el código [08].

La relación que existe entre estilos arquitectónicos, patrones de diseño y *Frameworks* es la siguiente: Los estilos arquitectónicos pueden contener patrones de diseño y los patrones de diseño pueden ser implementados en un *Framework* específico. Un ejemplo de esta relación la podemos

encontrar en arquitecturas de software usadas en el sector empresarial. Este tipo de arquitecturas están formadas por el estilo arquitectónico de capas. Una o varias de estas capas contienen patrones de diseño. Para el caso de la capa de presentación a menudo se utiliza el patrón de diseño MVC, para el caso de la capa de lógica de negocios es posible usar el patrón de diseño *Facade*. Finalmente estos patrones de diseño pueden ser implementados en algún *Framework*. Para el caso del patrón de diseño MVC se puede utilizar el *Framework Struts* [09], mientras que para el patrón de diseño *Facade* se puede utilizar el *Framework Spring* [10].

Una vista es una representación de un conjunto coherente de elementos arquitectónicos. Las vistas consisten de una representación de un conjunto de elementos y las relaciones entre ellos; muestran una perspectiva de la arquitectura de software. Bass Clements y Kazman proponen tres grupos de vistas, los cuales son: Estructuras de módulos, estructuras de asignación, estructuras de conectores y componentes [06].

Una vez descritos los estilos arquitectónicos, los patrones de diseño y las vistas que conforman una arquitectura de software, a continuación se hablará acerca de los atributos de calidad que pueden existir en las arquitecturas de software. Cualquier arquitectura de software definida de manera apropiada promueve que se satisfagan atributos de calidad.

Un atributo de calidad es un requerimiento no funcional de un sistema de software por ejemplo: Confiabilidad, facilidad de modificación, rendimiento, facilidad de uso, entre otros. De acuerdo al estándar [11], un atributo de calidad representa el grado en el cual un software posee una combinación deseable de estos atributos. Jan Bosch [12] define dos categorías para agruparlos, las cuales son: Atributos de desarrollo y atributos de operación. Los atributos de desarrollo son relevantes desde una perspectiva de ingeniería de software, estos atributos pueden ser: Facilidad de mantenimiento, flexibilidad, facilidad de modificación, entre otros. Los atributos operacionales son cualidades del sistema en operación por ejemplo: El rendimiento del sistema, la confiabilidad, robustez, tolerancia a fallas, seguridad.

Las evaluaciones a una arquitectura de software pueden ser realizadas para varios propósitos, por ejemplo: Para evaluar riesgos en el sistema, predecir el costo de mantenimiento del sistema, realizar comparaciones con otras arquitecturas, entre otros propósitos. El objetivo común de la mayoría de los métodos de evaluación es evaluar la arquitectura de software para facilitar o inhibir el que se cumplan los atributos de calidad requeridos por alguna arquitectura en particular utilizando métodos o procedimientos de evaluación ya definidos [13].

El objetivo del presente documento se centra en definir un proceso para evaluar arquitecturas de software usadas en el sector empresarial, así como la elaboración de un *Checklist* para evaluar si un software que es desarrollado con una tecnología especifica es capaz de cumplir con los atributos de calidad definidos en la arquitectura.

El resto del documento está organizado de las siguiente manera: En la sección dos se describe la problemática a resolver, en la sección tres se presenta el trabajo previo, en la sección cuatro se describe la solución propuesta, en la sección cinco se presentan las conclusiones y trabajo fututo, en la sección seis se muestra el glosario de términos usado en este documento, la sección siete contiene las referencias usadas en el presente documento y la sección ocho se describe el *Checklist* de evaluación.

2 Descripción del problema

¿Cómo saben las empresas desarrolladoras de software si eligieron la arquitectura correcta para el software que desarrollan? El éxito o fracaso de un sistema de software esta determinado por su arquitectura. Por ejemplo una arquitectura que no sea fácil de modificar le costará mucho dinero a la empresa el hacer las modificaciones que el cliente requiera, debido a que la empresa no consideró en la arquitectura este requerimiento de calidad. En la actualidad los sistemas de información usados por las empresas son cada vez más complejos por lo que las empresas que desarrollan software deben de elaborar arquitecturas de software capaces de cumplir las necesidades que demandan este tipo de sistemas.

La única manera de asegurar que se esta elaborando correctamente una arquitectura de software es realizando evaluaciones a la arquitectura. Comúnmente se da el caso en que existe más de una opción que cumple con los requerimientos funcionales, pero que deben evaluarse para determinar si cumplen con los requerimientos no funcionales o atributos de calidad, o bien si existe un conflicto que impide contar con la combinación de ellos para la misma arquitectura.

Otro caso se da cuando muchas de las decisiones arquitectónicas para un software que el cliente puede aceptar o rechazar giran entorno al presupuesto acordado para el desarrollo de un sistema; de esta manera, en ciertas ocasiones se debe evaluar una arquitectura de acuerdo a componentes que se pueden sacrificar para reducir costos y para ello, se deben justificar los atributos de calidad que a la vez se están sacrificando. Actualmente en la industria, varias de estas justificaciones se encuentran soportadas de acuerdo a la tecnología y plataforma que se haya seleccionado para el desarrollo del sistema de software, por lo cual también resulta importante considerar este elemento en la propuesta de una arquitectura de software.

El presente documento propone un proceso de evaluación para arquitecturas de software usadas en el sector empresarial con la finalidad de ayudar a mejorar las arquitecturas que son utilizadas por las empresas desarrolladoras de software. Con este proceso de evaluación se asegura que las empresas elijan correctamente la arquitectura para el software que desarrollan.

3 Trabajo previo

En esta sección se presentan los trabajos previos relacionados con la metodología seis sigma, DFSS, vistas arquitectónicas, introducción a evaluaciones de arquitecturas de software, método de evaluación ALMA [20], método de evaluación SNA [21] y patrones de diseño que son utilizados en la propuesta de solución de este documento.

3.1 Seis Sigma y DFSS

En este punto se hablará de seis sigma y DFSS, que son metodologías usadas principalmente en el área de manufactura. El proceso de evaluación de arquitecturas de software propuesto en este documento hace uso de la metodología DFSS.

El concepto de seis sigma (6σ) nace de la industria manufacturera. A mediados de la década de los 80's, Motorola con el principal objetivo de reducir los costos de calidad en sus procesos aplica esta metodología utilizando una serie de técnicas estadísticas [01].

El objetivo de la metodología 6σ es mejorar la calidad en los procesos, productos o servicios existentes que una empresa ofrece o desarrolla. Seis sigma esta basada en la metodología conocida como DMAIC, la cual consta de las fases:

- Definir. Se define el alcance del proyecto así como el problema y los defectos.
- Medir. Se mide el nivel de rendimiento actual de los procesos.
- Analizar. Se analiza el proceso para determinar las causas primarias del problema y de los defectos.
- Mejorar (*Improve*). Se hace una mejora al proceso identificando e implementando soluciones para eliminar las causas primarias.
- Controlar. Se controla y monitorea el rendimiento del proceso mejorado.

Esta metodología es usada cuando un producto, proceso o servicio existe en la empresa pero éste no satisface las necesidades del cliente [02].

Por medio del nivel de sigma, el cual es una medida que indica la cantidad de variación de un proceso, es posible mejorar los procesos de una empresa, logrando así un incremento en la calidad de los productos, procesos o servicios que ofrece. Por ejemplo el nivel promedio de sigma de la mayoría de las empresas es de 3σ, este nivel indica que el número de defectos por millón de oportunidades es cerca de 66,800 defectos [01].

De acuerdo a Harry and Schroeder [05], las empresas que han adoptado los principios y conceptos de la metodología seis sigma, han descubierto que una vez que estas empresas alcanzan niveles de 5σ (233 defectos por millón de oportunidades), la única manera de pasar de este nivel es rediseñar los procesos, productos o servicios que estas empresas ofrecen [03].

Estudios han demostrado que el costo de reparar errores de diseño se incrementa considerablemente a lo largo del ciclo de vida del producto. Si un error en el diseño es detectado durante la fase de manufactura, este error cuesta cientos de veces más el repararlo, que si hubiera sido reparado en la fase de diseño [01].

General Electric consciente de esta situación, fue la primera en rediseñar una línea de productos utilizando diseño para seis sigma (DFSS) [04]. El principal objetivo de DFSS es el diseñar nuevos productos, procesos o servicios con una alta calidad de una manera efectiva y redituable enfocándose en satisfacer los requerimientos del cliente [01].

De acuerdo a Simon [02], el nivel mínimo de sigma esperado por las empresas que utilizan DFSS es de 4.5σ (no mas de un defecto por cada mil oportunidades), sin embargo las empresas pueden alcanzar el nivel 6σ dependiendo del producto, proceso o servicio que se este diseñando.

Así como seis sigma esta basada en la metodología DMAIC, los inicios de DFSS se basaron en la metodología DMADV la cual esta formada por las siguientes fases:

- Definir. Se definen las metas del proyecto y del cliente, existen requerimientos tanto internos como externos.
- Medir. Se mide y se determinan las especificaciones y necesidades del cliente, se pueden utilizar *Benchmarks*.
- Analizar. Se analizan las opciones del proceso para satisfacer las necesidades del cliente.
- Diseñar. Se diseña un proceso detallado para satisfacer las necesidades del cliente.
- Validar. Se valida el rendimiento del diseño y la habilidad para satisfacer las necesidades del cliente.

Una pequeña variación de esta metodología se le conoce como DMADOV (Diseñar, Medir, Analizar, Diseñar, Optimizar y Verificar).

Existen algunas variedades de DFSS las cuales brevemente se describen a continuación:

DCCDI
Definir. Se definen las metas del proyecto.
Cliente. Se elaboran análisis.
Concepto. Las ideas son desarrolladas, revisadas y seleccionadas.
Diseñar. El diseño es realizado para satisfacer las necesidades del cliente.
Implementación. Se realiza la implementación para completar el desarrollo del producto, proceso o servicio y se comercializa.

IDOV.
Identificar. Se identifican las especificaciones del cliente así como las cualidades críticas del cliente.
Diseñar. Se transforman las cualidades críticas del cliente en requerimientos funcionales y en alternativas de solución.
Optimizar. Por medio de herramientas estadísticas y modelos se predice y optimiza el diseño.
Validar. Se asegura que el diseño que se ha desarrollado cumpla con las cualidades críticas del cliente.

DMEDI (Definir, Medir, Explorar, Desarrollar e Implementar), esta metodología es usada por PricewaterhoseCoopers [02].

Cada una de estas metodologías utiliza distintos tipos de herramientas de acuerdo a la fase en que se encuentre. Algunas de las herramientas mas comunes que se utilizan en DFSS son: Función del despliegue de la calidad (QFD), Análisis de Efectos y Modos de Fallas (FMEA), *Benchmarking*, Diseño de experimentos, simulación, optimización estadística, prueba de errores, diseños robustos, ente otras [04].

El proceso de evaluación para arquitecturas de software propuesto en este documento esta basado en la fase de validación del método IDOV. Por lo que se adecuó la fase de validación de IDOV a la evaluación de arquitecturas de software de la siguiente manera: La fase de validación consta de evaluar el diseño y los atributos de calidad de una arquitectura de software, para asegurarse de que cumplan con los requerimientos especificados por el cliente [01].

3.2 Vistas arquitectónicas

Una vez que la arquitectura de software es representada por uno o varios estilos arquitectónicos y por patrones de diseño, es necesario conocer algunos aspectos de la arquitectura por ejemplo: Cómo interactúan los componentes unos con otros, como están interconectados los componentes entre sí, en que tipo de plataforma tanto de software como de hardware van a ejecutarse los componentes, entre otros aspectos. En 1995 fueron publicados dos artículos [14][15] en los cuales se proponen utilizar vistas arquitectónicas para representar diferentes aspectos de una arquitectura de software. Nord [15] propone cuatro vistas:

- Vista conceptual. Describe el sistema en términos de sus principales elementos de diseño y las relaciones entre ellos.
- Vista de módulos. Abarca dos estructuras ortogonales que son: Descomposición modular y de capas.
- Vista de ejecución. En esta vista se describe la estructura dinámica del sistema.
- Vista de código. Describe como el código fuente, archivos binarios y librerías son organizados en un entorno de desarrollo.

Kruchten [14] por el contrario propone las siguientes vistas:

- Vista lógica. Describe el diseño de los objetos que son modelados cuando un método de diseño orientado a objetos es usado. En esta vista también se puede modelar diagramas de entidad relación.
- Vista de proceso. Describe el diseño de los aspectos de concurrencia y sincronización del sistema.
- Vista física. Describe la relación de los componentes de software que serán ejecutados en el hardware y refleja aspectos de distribución.
- Vista de desarrollo. Describe la organización estática de los componentes de software en un entorno de desarrollo.
- Escenarios. Los cuales muestran cómo los componentes de las cuatro vistas anteriores trabajan en conjunto de manera transparente.

Aunque en la actualidad no hay un estándar que defina las vistas que debe tener una arquitectura de software, es posible usar como referencia las vistas antes mencionadas.

3.3 Introducción a evaluaciones de arquitecturas software

¿Cómo saber si una arquitectura promueve o alcanza algún atributo de calidad, por ejemplo que la arquitectura sea fácil de modificar, que sea segura, que sea fácil de intercambiar a varias plataformas tanto de hardware como de software? La única forma de saberlo es realizando evaluaciones a la arquitectura de software. Los primeros esfuerzos realizados para evaluar arquitecturas de software son de un trabajo realizado por Parnas y Weiss [16] en el año de 1985. En este trabajo Parnas y Weiss expresan uno de los principios fundamentales que hay detrás de una evaluación: Las revisiones de diseño de software que son indirectas y no estructuradas no funcionan.

La motivación de realizar este trabajo fue por la observación de los enfoques que había en esa época para revisar diseños. En esos enfoques no era clara la forma de realizar la revisión, se usaban técnicas *ad hoc* y no repetibles, es decir no existían procesos definidos de revisión y evaluación. Parnas y Weiss proponen una solución a este problema usar revisiones de diseño activas. Estas revisiones consisten en elaborar una serie de cuestionarios especializados en donde cada uno de los cuestionarios es diseñado para encontrar diferentes tipos de errores en el diseño. Como cada revisión requiere de un tipo de experto, se identifican a los diferentes tipos de expertos encargados de realizar las revisiones.

Desde los trabajos realizados por Parnas y Weiss a la fecha, varios métodos de evaluación de arquitecturas de software han sido desarrollados para evaluar problemáticas relacionadas con los atributos de calidad de una arquitectura de software. Algunos de estos métodos de evaluación son: SAAM [17], ATAM [18], PASA [19], ALMA [20], SNA [21] entre varios mas. Por ejemplo SAAM esta principalmente enfocado a identificar riesgos potenciales en la arquitectura, también puede ser usado para evaluar la facilidad de modificación. ATAM proporciona en principio la manera de entender una capacidad arquitectónica del sistema con respecto a varios atributos de calidad identificando y analizando puntos sensitivos y *Trade-off*s entre atributos. ALMA se especializa en predecir el atributo de calidad facilidad de modificación y tiene tres posibles objetivos: Evaluar riegos, predecir el costo de mantenimiento y comparación entre arquitecturas de software. PASA esta orientado a evaluar arquitecturas de software con respecto al atributo de calidad rendimiento. SNA evalúa el grado de supervivencia ante la presencia de ataques y fallas en un sistema de software.

3.3.1 Fases de evaluación

La evaluación clásica de una arquitectura de software ocurre cuando la arquitectura ha sido especificada pero aun no ha comenzado la implementación. Existen dos variaciones de la evaluación clásica: Evaluaciones tempranas y evaluaciones tardías.

Evaluaciones tempranas. La evaluación no necesita esperar hasta que la arquitectura este completamente especificada. La evaluación puede ser usada en cualquier fase del proceso de creación de la arquitectura para examinar aquellas decisiones arquitectónicas hechas y seleccionar entre las opciones arquitectónicas pendientes [22].

Evaluaciones tardías. Son una forma de evaluar arquitecturas existentes. Esta evaluación toma lugar cuando la arquitectura ya existe y la implementación ha sido completada. Esto ocurre cuando una organización hereda algún sistema legado y necesita ser integrado con algún sistema existente. Por medio de técnicas de ingeniaría inversa se reconstruye la arquitectura de software del sistema legado. La evaluación a este nivel ayuda a los nuevos propietarios a entender el sistema legado permitiéndoles conocer si el sistema cumple con los atributos de calidad requeridos [22].

A continuación en la figura 3.1 se muestra de manera gráfica los métodos de evaluación de arquitecturas de software más utilizados en la actualidad y en que fase de evaluación pueden ser aplicados.

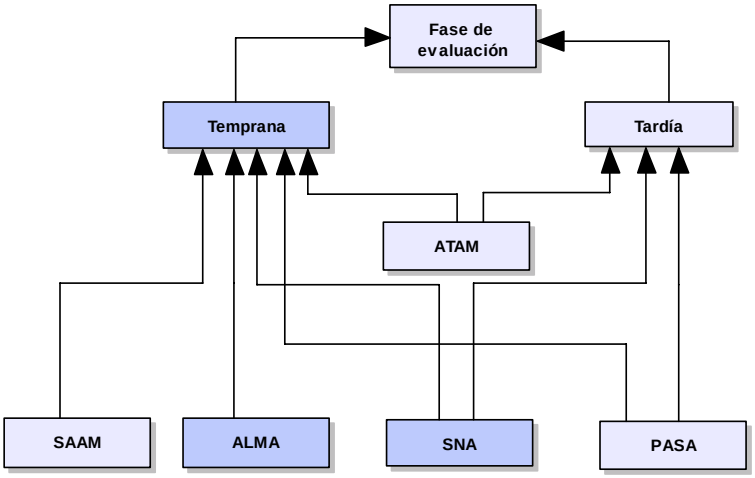

Figura 3.1 Fases de evaluación en las que pueden ser aplicados los métodos de evaluación.

De acuerdo a Clements, Kazman y Klein [22] dos reglas de oro para realizar evaluaciones a arquitecturas de software son:

1. Realizar una evaluación cuando los equipos de desarrollo inician a tomar decisiones que dependen sobre la arquitectura; y
2. Cuando el costo de no tomar esas decisiones podrían pesar más que el costo de mantener una evaluación.

El proceso de evaluación que se propone en este documento utiliza fases tempranas para evaluar arquitecturas de software. Sin embargo este proceso deja abierta la posibilidad de evaluar arquitecturas de software en fases tardías debido a la flexibilidad que tiene este proceso en utilizar otros métodos de evaluación.

3.3.2 Técnicas de evaluación

Las técnicas de evaluación existentes pueden clasificarse en dos grupos: Cualitativas y cuantitativas. Las técnicas de evaluación cualitativas incluyen escenarios, cuestionarios y *Checklists* mientras que las cuantitativas incluyen: Simulaciones, prototipos y modelos matemáticos. En la figura 3.2 se muestra de manera gráfica estas técnicas [12].

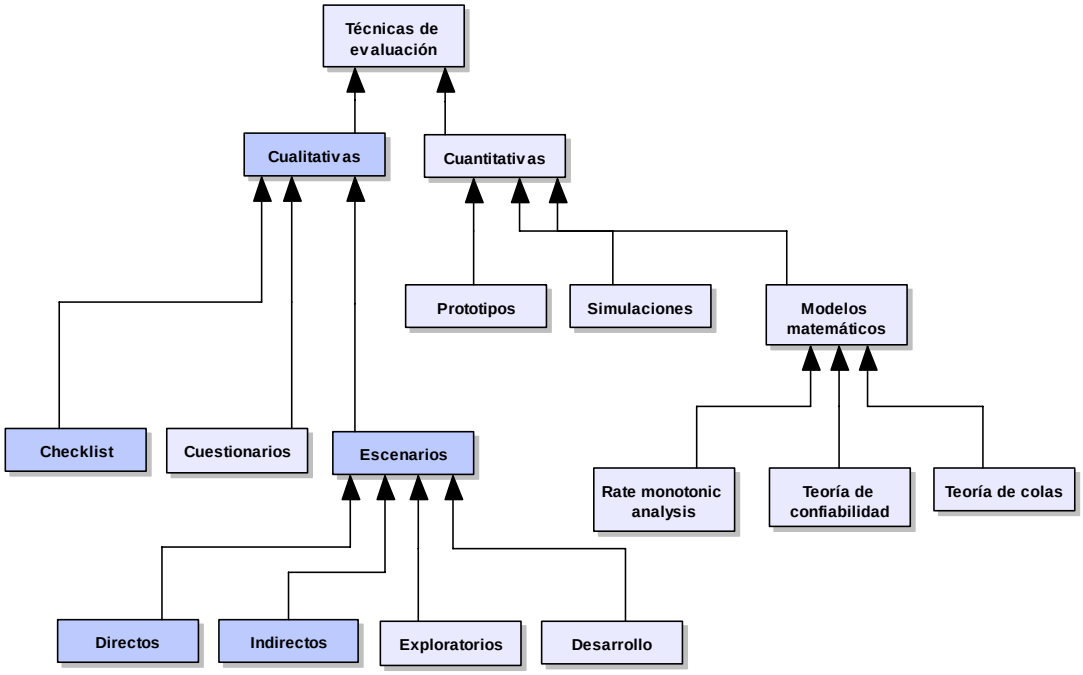

Figura 3.2 Técnicas de evaluación para arquitecturas de software.

Cuatro técnicas diferentes para evaluar atributos de calidad han sido identificadas: Escenarios, simulaciones, modelos matemáticos y razonamiento basado en experiencia. Por cada atributo de calidad el ingeniero puede seleccionar alguna de estas técnicas de evaluación. A continuación se describe cada una de estas técnicas.

Evaluación basada en escenarios. Para evaluar un atributo de calidad en particular, un conjunto de escenarios es desarrollado los cuales concretizan el significado actual de los requerimientos. Por instancia el requerimiento de facilidad de modificación puede ser especificado por un perfil de cambio que capture cambios típicos a los requerimientos, entonces el perfil puede ser usado para evaluar el esfuerzo requerido de adaptar la arquitectura a una nueva situación. El

atributo de facilidad de modificación puede ser expresado naturalmente a través de escenarios de cambio.

Simulación. La simulación requiere que los componentes principales de la arquitectura estén implementados mientras que los otros componentes son simulados, resultando un sistema ejecutable. Realizar simulaciones es particularmente útil para evaluar atributos de operación, tales como rendimiento, tolerancia a fallas, entre otros.

Modelos matemáticos. Los Modelos matemáticos son una alternativa a la simulación desde que ambos enfoques son principalmente convenientes para evaluar atributos de calidad operacionales. Sin embargo estas dos técnicas pueden ser combinadas. Por ejemplo para evaluar el atributo de calidad rendimiento se pueden utilizar: *Rate monotonic analysis* [23] o teoría de colas [24].

Razonamiento basado en experiencia. Esta técnica se basa en razonamiento basado en experiencias tempranas y argumentación lógica. Los ingenieros de software experimentados a menudo tienen valuables visiones que pueden proporcionar ayuda muy significativa en evitar malas decisiones de diseño. Aunque algunas de estas experiencias a menudo pueden estar basadas en anécdotas la mayoría a menudo se puede justificar usando razonamiento lógico [12].

El proceso de evaluación definido en este documento utiliza escenarios directos e indirectos como técnica de evaluación cualitativa.

3.4 Método de evaluación ALMA *(**Architecture-Level Modifiability Analysis)***

El atributo de calidad que predice ALMA [20] es facilidad de modificación. La facilidad de modificación de un sistema de software es la facilidad con la cual el sistema puede ser modificado para cambiar en su entorno o la facilidad para cambiar los requerimientos al sistema o las especificaciones funcionales del sistema.

ALMA hace uso de escenarios como su principal técnica de evaluación, en estos escenarios se especifican los posibles cambios que puede sufrir la arquitectura de software a evaluar. ALMA es un método de evaluación orientado a metas. La definición de la meta es la actividad más importante de este método [25]. La meta específica de este método se centra en evaluar el atributo de calidad facilidad de modificación. Las metas a evaluar para este atributo de calidad pueden ser:

- Predicción del costo de mantenimiento. Estimar el esfuerzo requerido para satisfacer escenarios de cambio.
- Evaluación de riesgos. Identificar los tipos de cambios para los cuales la arquitectura de software es inflexible.
- Selección de arquitectura de software. Comparar dos o más arquitecturas de software candidatas para seleccionar la mejor.

El método ALMA es utilizado antes de implementar la arquitectura de software por lo que es un método que se encuentra dentro de los métodos de evaluación temprana.

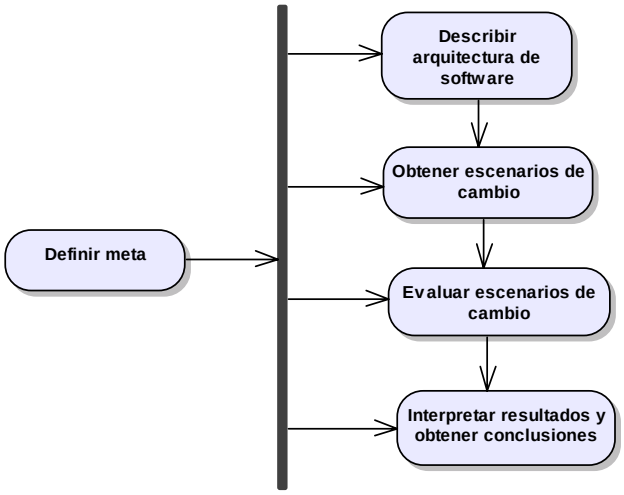

Figura 3.3 Pasos del método ALMA.

Como se muestra en la figura 3.3, ALMA tiene una estructura que consiste de los siguientes cinco pasos:

1. Establecer la meta. Se determina el objetivo del análisis.
2. Describir la arquitectura de software. Dar una descripción relevante de las partes de la arquitectura de software
3. Obtener escenarios de cambio. Encontrar el conjunto de escenarios relevantes.
4. Evaluar escenarios. Determinar el efecto del conjunto de escenarios.
5. Interpretar resultados. Realizar conclusiones de los resultados del análisis.

A continuación se describe más detalladamente cada uno de estos pasos.

Paso 1. Establecer la meta.

La primera actividad en ALMA esta enfocada con la determinación de la meta del análisis la cual puede ser alguna de las siguientes: Predecir el costo de mantenimiento, evaluación de riesgos o selección de una arquitectura de software.

Paso 2. Describir la arquitectura.

Recolectar la información disponible de la arquitectura de software. Esta información le permite al analista evaluar los escenarios. La evaluación de escenarios consta de dos pasos: Analizar el impacto de los escenarios y expresar su impacto. El análisis de impacto se realiza para identificar los componentes de la arquitectura que son afectados por un escenario de cambio. El efecto del escenario es expresado usando alguna escala de medición, dependiendo sobre la meta del análisis de facilidad de modificación.

Para realizar el análisis de impacto, las vistas arquitectónicas que se requieren deben de proporcionar: La descomposición del sistema en componentes, las relaciones entre componentes y las relaciones del entorno del sistema. Las vistas que pueden ser usadas para realizar este análisis son: Vista conceptual y vista de módulos [15] o la vista lógica [14]. La información que es proporcionada por estas vistas es crucial para el análisis de impacto ya que se puede determinar si las modificaciones a un componente afectarán a otros componentes.

Paso 3. Obtener escenarios de cambio.

La obtención de escenarios de cambio es el proceso de encontrar y seleccionar los escenarios de cambio a ser usados en el paso de evaluación. La obtención de escenarios de cambio involucra tales actividades como identificar usuarios para entrevistarlos, documentar los escenarios que resulten de las entrevistas, etc. Es recomendable no seguir obteniendo escenarios de cambio cuando: Ya se hayan considerado los principales escenarios de cambio y cuando sé empiecen a repetir.

Dependiendo sobre la meta del análisis de evaluación se deben escoger los escenarios de cambio por ejemplo:

- Si la meta es estimar el esfuerzo de mantenimiento, se recomienda seleccionar aquellos escenarios que corresponden a los cambios que tienen una alta probabilidad de que ocurran durante la vida operacional del sistema.
- Si la meta es sobre evaluación de riesgos, es recomendable seleccionar los escenarios que exponen riesgos al ser cambiados los componentes.
- Si la meta es comparar diferentes arquitecturas, es recomendable usar cualquiera de los dos puntos anteriores y concentrarse sobre los principales escenarios que se diferencian entre las arquitecturas seleccionadas.

Paso 4. Evaluar escenarios.

El siguiente paso es evaluar los efectos de los escenarios de cambio sobre la arquitectura. En este paso, el analista coopera con los arquitectos y diseñadores. Junto con ellos el analista determina el impacto de cada uno de los escenarios de cambio y expresa los resultados en una manera conveniente para la meta del análisis. Aquí es donde se realiza el análisis de impacto de la arquitectura. En general el análisis de impacto consiste de los siguientes pasos:

1. Identificar los componentes afectados.
2. Determinar el efecto sobre los componentes.
3. Determinar el efecto de cambio en los demás componentes.

El primer paso es determinar los componentes que necesitan ser modificados para implementar el escenario de cambio. En el segundo paso se identifican las funciones de los componentes que son afectados por esos cambios. Cabe señalar que los cambios pueden propagarse sobre los limites del sistema por lo que los cambios en el entorno pueden impactar al sistema o los cambios al sistema pueden afectar el entorno. Por lo que es necesario considerar los sistemas en el entorno y sus interfaces. El último paso es determinar los efectos del cambio en otros componentes, en este paso los arquitectos y diseñadores determinan si los cambios a un componente afectan a otros componentes.

Finalmente se deben de expresar los resultados del análisis de impacto ya sea cuantitativamente, usando medidas cuantitativas o cualitativamente, describiendo los cambios necesarios.

Paso 5. Interpretar resultados.

Una vez finalizada la evaluación de los escenarios de cambio, es necesario interpretar los resultados. La interpretación de los resultados dependerá sobre la meta del análisis de evaluación y sobre los requerimientos del sistema.

En este punto se ha presentado el método de evaluación ALMA, que es un método para analizar la facilidad de modificación de una arquitectura de software. Este método esta basado en el uso de escenarios. ALMA consta de cinco pasos principales: Definición de la meta, descripción de la arquitectura de software, obtención de escenarios de cambio, evaluación de los escenarios de cambio e interpretación de los resultados obtenidos por la evaluación.

ALMA distingue las siguientes metas que pueden ser usadas en el análisis de la facilidad de modificación de una arquitectura de software, estas metas son: Predicción del costo de mantenimiento, evaluación de riesgos y comparación de arquitecturas de software.

3.5 Método SNA *(**Survivable Network Analysis**)*

EL método de evaluación SNA [21] permite sistemáticamente evaluar las propiedades de supervivencia de un sistema propuesto, existente o evaluar la supervivencia de un sistema cuando se realicen modificaciones a sistemas existentes. Esta evaluación se realiza a nivel de la arquitectura de software.

La supervivencia es la capacidad de un sistema para cumplir con su misión a tiempo en presencia de ataques, fallas o accidentes. La supervivencia es evaluada en sistemas distribuidos que se encuentran ilimitados en un entorno de red sin un control central o sin políticas de seguridad unificadas. Internet es un ejemplo de un entorno de red sin límites con muchas aplicaciones de red cliente-servidor [26].

Una de las características principales de los sistemas con capacidades de supervivencia es que los servicios esenciales del sistema se mantienen en funcionamiento aun cuando se encuentran en presencia de ataques, fallas o accidentes. Estos servicios esenciales pueden ser: niveles específicos de integridad, confidencialidad, rendimiento, entre otros.

La capacidad de mantener en funcionamiento los servicios esenciales del sistema deben de mantenerse incluso aun si una porción del sistema ha sido Inhabilitada.

Un sistema que tiene capacidad de supervivencia debe de exhibir tres propiedades:

- Resistencia. Es la capacidad del sistema para repeler ataques.
- Reconocimiento. Es la capacidad del sistema para detectar ataques y evaluar el daño hecho.
- Recuperación. Es la capacidad de mantener en funcionamiento los servicios esenciales del sistema aun en presencia de ataques.

SNA puede ser utilizado antes de implementar la arquitectura de software o una vez que el sistema ha sido implementado, por lo que este método puede ser usado tanto en evaluaciones tempranas como tardías.

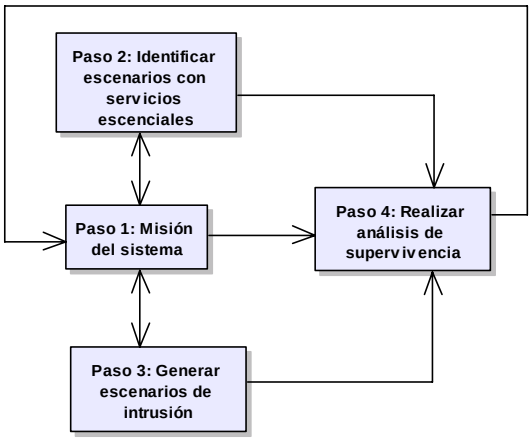

Figura 3.4 Pasos del método SNA.

Como se muestra en la figura 3.4, SNA tiene una estructura que consiste de los siguientes cuatro pasos [27]:

1. Obtener la misión del sistema y de la arquitectura.
2. Identificar escenarios con los servicios esenciales.
3. Generar escenarios de intrusión.
4. Realizar análisis de supervivencia.

Paso 1. Obtener la misión del sistema y de la arquitectura.

En este paso se obtiene la definición del sistema, la misión de los objetivos del sistema así como requerimientos funcionales. Se discute el entorno de uso del sistema en términos de sus capacidades y de la ubicación de los usuarios del sistema así como de los volúmenes de manejo de transacciones del sistema. Se revisan los riesgos del sistema en términos de tipos de condiciones adversas que pueden ser encontradas. Se obtiene la arquitectura del sistema.

Paso 2. Identificar escenarios con los servicios esenciales.

Se seleccionan los servicios esenciales y los activos del sistema. Los servicios esenciales y los activos son aquellas capacidades críticas para cumplir con la misión del negocio del sistema la cual debe ser mantenida bajo condiciones adversas. Se definen escenarios de uso los cuales invocan a los servicios esenciales del sistema. Los escenarios de uso son compuestos de una serie de pasos sucesivos requeridos por los usuarios para invocar a los servicios y acceder a los activos como es el caso de la información. Finalmente los escenarios de uso son rastreados a través de la arquitectura de software para identificar los componentes esenciales que participan en proporcionar los servicios esenciales y los activos.

Paso 3. Generar escenarios de intrusión.

Se selecciona un conjunto representativo de posibles ataques basados en el entorno de operación del sistema. Se definen los escenarios de intrusión y se rastrean a través de la arquitectura de software para identificar componentes que comprometan la supervivencia del sistema logrando así el acceso al sistema y dañarlo.

Paso 4. Realizar análisis de supervivencia.

En este paso se realiza el análisis de supervivencia. Se identifican los escenarios de intrusión que contienen los componentes que son ambos esenciales y que comprometen la supervivencia del sistema. Por cada componente identificado se analiza en términos de las capacidades de resistencia, reconocimiento y recuperación. Estos análisis se colocan en un mapa de supervivencia que contiene por cada escenario de intrusión las estrategias actuales y recomendadas de resistencia, reconocimiento y recuperación. El mapa de supervivencia proporciona retroalimentación a la arquitectura de software original y a los requerimientos del sistema [21].

En este punto se ha presentado el método de evaluación SNA, que es un método para analizar la supervivencia de un sistema en condiciones adversas. Este método esta basado en el uso de escenarios. SNA consta de cuatro principales pasos: Obtener la misión del sistema y de la arquitectura, identificar escenarios con los servicios esenciales, generar escenarios de intrusión y realizar análisis de supervivencia.

3.6 Patrones de diseño

En este punto se presentan algunos patrones de diseño que han sido identificados en las arquitecturas de software usadas en el sector empresarial. El usar apropiadamente patrones de diseño en arquitecturas de software asegura que se cumplan atributos de calidad tales como: Facilidad de modificación, seguridad, rendimiento entre otros. A continuación se describen los

patrones de diseño: *Facade* [08], *Singleton* [08], Modelo Vista Controlador [28] y *Data Transfer Object* [29].

3.6.1 *Facade*

Intención. Proporcionar una interfaz unificada para un conjunto de interfaces en un subsistema. *Facade* define una interfaz de alto nivel haciendo fácil de usar los subsistemas.

Motivación. Estructurar un sistema en subsistemas ayuda a reducir la complejidad. Una meta común de diseño es minimizar la comunicación y las dependencias entre subsistemas. Una forma de alcanzar esta meta es introducir un objeto *Facade* que proporcione una sencilla y simple interfaz que contenga las operaciones generales de los subsistemas.

Aplicabilidad. Usar el patrón de diseño *Facade* cuando:

- Se quiera proporcionar una interfaz para un subsistema complejo. *Facade* puede proporcionar una vista por de facto del subsistema que es suficiente para la mayoría de los clientes.
- Para desacoplar el subsistema de sus clientes y otros subsistemas, logrando así independencia y portabilidad.
- Se quiera agregar capas a los subsistemas. Usar el patrón de diseño *Facade* para definir un punto de entrada a cada nivel de los subsistemas.

3.6.2 *Singleton*

Intención. Asegurarse de que una clase solamente tenga una instancia, y proporcione un punto de acceso global a esta instancia.

Motivación. Es importante para algunas clases tener exactamente una instancia. ¿Cómo asegurarse de que una clase tenga solamente una instancia y que la instancia sea fácilmente accesible? Se asegura haciendo que la clase sea responsable de manejar solamente una instancia.

Aplicabilidad. Usar el patrón *Singleton* cuando:

- Deba haber exactamente una instancia de una clase, y esta instancia debe ser accesible a los clientes desde un punto de acceso bien conocido.

3.6.3 Modelo Vista Controlador (MVC)

El patrón arquitectónico Modelo Vista Controlador, divide una aplicación interactiva en tres componentes. El modelo contiene la funcionalidad principal y los datos. La vista muestra la información al usuario. El controlador maneja las entradas del usuario. La vista y el controlador juntos forman parte de la interfaz de usuario. Un mecanismo de propagación de cambios asegura la consistencia entre la interfaz de usuario y el modelo.
Contexto. Aplicaciones interactivas con interfaces humano computadora flexibles.

Estructura.

El componente modelo contiene la funcionalidad principal de la aplicación. Este encapsula los datos apropiados y exporta funciones que realizan procesamiento específico de la aplicación. Los controladores llaman a esas funciones por medio de la vista. El modelo también proporciona las funciones necesarias para acceder a los datos los cuales son usados por la vista.

Los componentes vista presentan la información al usuario. Diferentes vistas presentan la información del modelo de diferentes maneras. Cada vista define un procedimiento de actualización que es activado por un mecanismo de propagación de cambios. Cuando el

procedimiento de actualización es llamado, una vista regresa los valores actuales a ser mostrados del componente modelo y los pone en la pantalla.

El componente controlador acepta las entradas del usuario como eventos. El cómo estos eventos son entregados al controlador depende de la plataforma de la interfaz del usuario. Los eventos son trasladados en peticiones para el componente modelo o para el componente vista asociado.

3.6.4 *Data Transfer Object (DTO)*

En cualquier aplicación distribuida hay dos razones del porque un cliente interactúa con un servidor. La primera razón es para que el cliente lea algunos datos del servidor con la finalidad de mostrar al usuario los datos. La segunda razón es para cambiar algunos datos sobre el servidor ya sea creando, actualizando o borrando los datos.

Cuando en un sistema se requiera intercambiar cantidades grandes de datos, este requerimiento se puede llevar a cabo leyendo muchos parámetros en una llamada de un método por ejemplo cuando se actualizan datos al servidor o haciendo muchas llamadas al servidor para regresar los datos por ejemplo cuando un cliente necesita leer datos del servidor. Esta forma de leer y modificar datos requiere de cantidades grandes de parámetros por lo que se realizan demasiadas llamadas al servidor logrando así una degradación en el rendimiento del sistema.

Los *Data Transfer Objects* pueden ser usados para minimizar las llamadas entre el cliente y el servidor logrando así un incremento en el rendimiento del sistema. Los DTOs son usados tanto para leer datos como para actualizar datos en un sistema distribuido. Cuando un cliente necesita actualizar algunos datos en el servidor, éste puede crear un DTO para encapsular toda la información que el servidor necesita ya sea para agregar, actualizar o borrar datos. Usualmente los DTOs son enviados a objetos que implementan el patrón de diseño *Facade* para su procesamiento. Cuando un cliente necesita leer algunos datos del servidor, el cliente obtiene todos los datos necesarios en una sola llamada ya que los datos vienen encapsulados en la forma de un *Data Transfer Object* [29].

Aunque en la literatura se puede encontrar una gran variedad de patrones [8][28][29][30] usados en distintos dominios de aplicación, en el presente documento solamente se mencionan cuatro patrones de diseño: *Facade*, *Singleton*, Modelo Vista Controlador y *Data Transfer Object*. Ya que estos patrones son utilizados en el dominio de aplicación del sector empresarial.

En esta sección se han presentado los trabajos previos de la metodología seis sigma y DFSS, vistas arquitectónicas, introducción a evaluaciones de arquitecturas de software, método de evaluación ALMA, método de evaluación SNA y patrones de diseño, estos trabajos previos son utilizados en el presente documento para la elaboración de la propuesta de solución.

4 Propuesta de solución

En esta sección se propone el proceso llamado PEASSE (Proceso de Evaluación de Arquitecturas de Software en el Sector Empresarial). Como su nombre lo dice el objetivo de este proceso es el evaluar arquitecturas de software que sean usadas en el sector empresarial. PEASSE es un proceso propuesto para que sea utilizado en la fase de validación de la metodología DFSS conocida como IDOV. En la figura 4.1 se muestra el diagrama con algunas metodologías de DFSS entre ellas IDOV, también se muestran las fases que componen la metodología IDOV, en particular el diagrama muestra la adecuación hecha en la fase de validación para utilizar el proceso PEASSE.

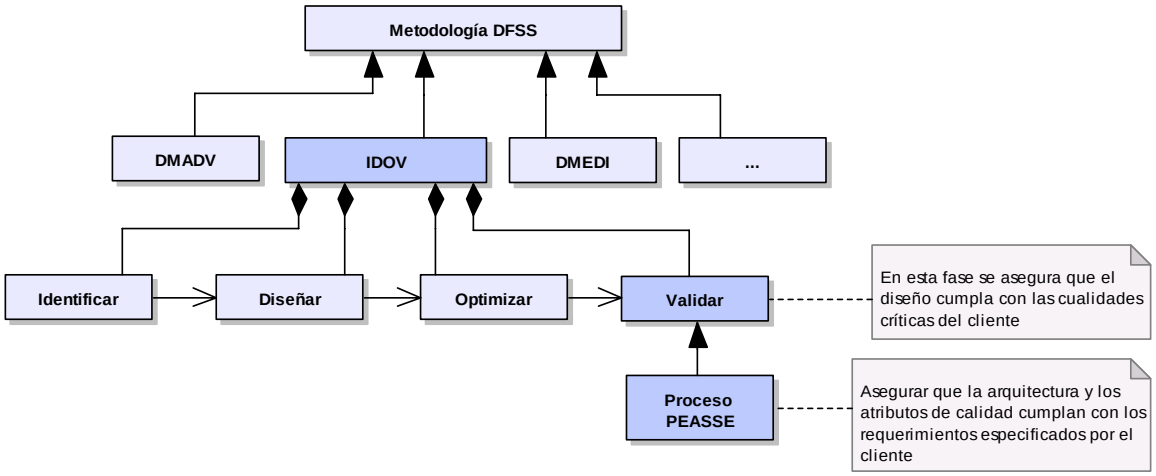

Figura 4.1 Fase validar de la metodología IDOV en la que se propone utilizar el proceso PEASSE.

PEASSE hace uso de la identificación de patrones de diseño para asegurar que se logren conectar o relacionar de manera más transparente atributos de calidad tales como: Facilidad de modificación, rendimiento, seguridad, entre otros. Además de la identificación de patrones de diseño, PEASSE utiliza dos métodos de evaluación: ALMA y SNA dependiendo el atributo de calidad a evaluar.

Sin embargo este proceso de evaluación es lo suficientemente flexible, ya que permite hacer uso de otros métodos de evaluación de arquitecturas de software. Al igual que ALMA, PEASSE es un proceso de evaluación orientado a metas. PEASSE esta diseñado para realizar la evaluación de una arquitectura de software antes de que la arquitectura se encuentre implementada, por lo que es un proceso que se encuentra dentro de los métodos de evaluación temprana. A continuación se muestra la figura 4.2 que representa el proceso PEASSE.

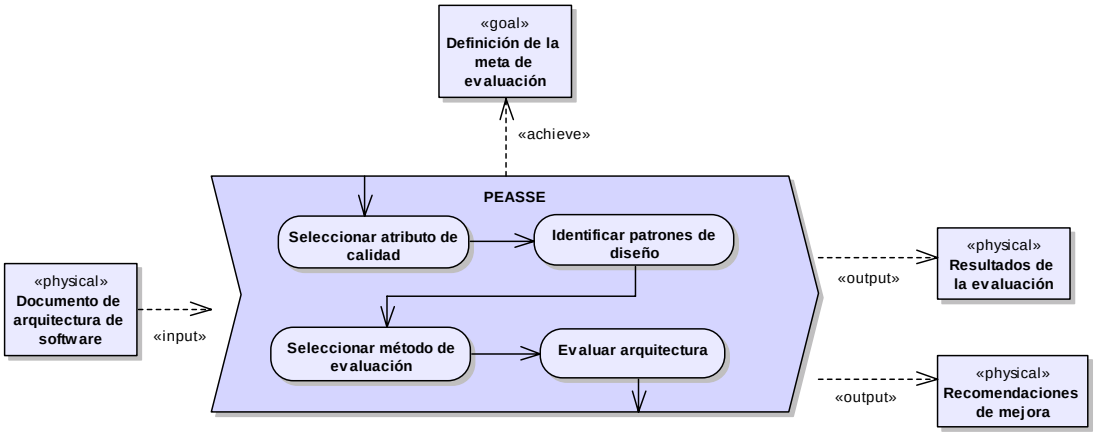

Figura 4.2 Proceso PEASSE.

La entrada requerida para iniciar con el proceso de evaluación PEASSE es tener listo el documento de arquitectura de software, el siguiente paso es definir la meta de la evaluación, una vez establecida la meta se realiza el proceso de evaluación obteniendo como salida de este proceso los resultados de la evaluación y las recomendaciones de mejora a la arquitectura de software.

PEASSE tiene una estructura que consiste de los siguientes seis pasos que se basan en DFSS, y de manera particular en IDOV:

1. Definir meta de la evaluación.
2. Seleccionar atributo de calidad en base a la meta.
3. Identificar patrones de diseño.
4. Seleccionar método de evaluación.
5. Evaluar arquitectura.
6. Preparar reporte de evaluación.

En la siguiente figura 4.3 se muestra de manera grafica los pasos y la secuencia que constituyen al método PEASSE.

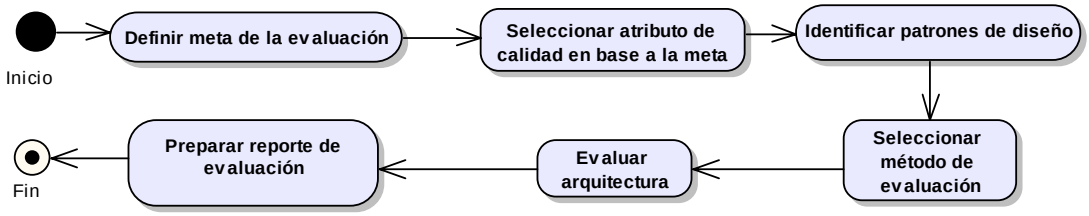

Figura 4.3 Pasos del proceso PEASSE.

A continuación se describe cada uno de los pasos que conforman al proceso PEASSE.

Paso 1. Definir meta de la evaluación.

Una arquitectura de software puede ser evaluada con diferentes metas en mente y desde diferentes perspectivas. Una evaluación puede ser basada sobre si el sistema implementa los requerimientos funcionales especificados o más convenientemente para una evaluación arquitectónica, si la arquitectura de software cumple con atributos de calidad que son representados como requerimientos no funcionales. En este paso se define la meta de la evaluación por ejemplo comparar dos arquitecturas de software con respecto al mismo atributo de calidad para determinar cual arquitectura es mas fácil de modificar.

Paso 2. Seleccionar atributo de calidad en base a la meta.

Una vez que se ha definido la meta de la evaluación a la arquitectura de software, el siguiente paso es seleccionar el atributo de calidad a evaluar. El atributo de calidad a evaluar se encuentra implícitamente en la definición de la meta de evaluación.

Paso 3. Identificar patrones de diseño.

En este paso se identifican los patrones de diseño de la arquitectura de software a evaluar. Para identificar estos patrones es necesario analizar las vistas arquitectónicas que forman parte del documento de arquitectura de software. Las vistas más útiles para identificar patrones de diseño son: Para el caso de Nord [15] la vista conceptual y la vista de módulos, para el caso de Kruchten [14] la vista lógica. Dependiendo del número de patrones de diseño identificados en las vistas arquitectónicas será el grado con el cual los patrones de diseño ayuden a la arquitectura de software a satisfacer atributos de calidad. Es recomendable conocer la aplicabilidad de los patrones de diseño utilizados en el dominio de aplicación del sector empresarial, esto con la finalidad de identificar los atributos de calidad involucrados en cada uno de los patrones de diseño.

Paso 4. Seleccionar método de evaluación.

En este paso se selecciona el método de evaluación a utilizar dependiendo del atributo de calidad seleccionado en el paso número dos. Aunque en el presente documento solamente se definen dos métodos de evaluación los cuales evalúan los atributos de calidad facilidad de modificación y supervivencia, este paso queda flexible para seleccionar algún otro método de evaluación existente. Si se desea utilizar algún otro método es recomendable leer los trabajos de [13][31] ya que en estos trabajos los autores presentan una comparación completa de varios métodos de evaluación de arquitecturas de software.

Paso 5. Evaluar arquitectura.

Una vez que ha sido seleccionado el método de evaluación, se realiza la evaluación a la arquitectura de software siguiendo cada uno de los pasos del método de evaluación seleccionado. Para el caso de evaluar el atributo de calidad facilidad de modificación, en este documento se propone utilizar el método de evaluación ALMA ya que es un método de evaluación que ha sido validado en varios sistemas de software y se encuentra lo suficientemente maduro para utilizarlo [25]. En caso de que el atributo de calidad sea seguridad el cual es parte del atributo de calidad supervivencia, se recomienda utilizar el método SNA, que aunque este método no se encuentre lo suficientemente maduro es posible encontrar varios casos de estudio en la literatura [26][27][32].

A continuación se muestra la tabla 4.1 la cual hace una comparación de varios elementos entre el método de evaluación ALMA y el método de evaluación SNA descritos en este documento.

Método Elemento de comparación	ALMA	SNA
Etapa de maduración	Refinamiento/Desarrollo	Desarrollo
Número de actividades	5	4
Objetivo del método	Predicción del esfuerzo de mantenimiento, análisis del impacto de cambio, comparación entre arquitecturas de software	Analizar resistencia, reconocimiento y recuperación de un sistema
Fase de evaluación	Temprana	Temprana/Tardía
Técnica de evaluación	Escenarios	Escenarios
Atributo de calidad que predice	Facilidad de modificación	Supervivencia
Vistas arquitectónicas usadas	*Nord:* Conceptual, módulos *Kruchten:* Lógica	*Nord:* Conceptual, módulos, ejecución *Kruchten:* Lógica, física
Criterio de entrada	Especificación de la arquitectura de software, requerimientos de calidad	Especificación de la arquitectura de software
Criterio de salida	Conclusiones acerca de los resultados del análisis	Mapa de supervivencia que incluye los análisis de resistencia, reconocimiento y recuperación del sistema
Personas requeridas	Equipos pequeños, normalmente el equipo de desarrollo y el arquitecto de software	Tres o cuatro personas, deben de estar presentes el arquitecto de software y el experto en seguridad
Validación del método	Validado en varios dominios	Solamente casos de estudio

Tabla 4.1 Comparación de los métodos de evaluación ALMA y SNA.

Paso 6. Preparar reporte de evaluación.

En este paso se prepara un reporte el cual contiene los resultados de la evaluación y las recomendaciones. Los resultados se obtienen de diferente manera dependiendo del método de evaluación que se haya utilizado. Por ejemplo para el caso del método ALMA los resultados se preparan de acuerdo a una de las tres metas de evaluación: Predicción del costo de mantenimiento, evaluación de riesgos y comparación de arquitecturas de software. Una vez que se obtienen los resultados de la evaluación la siguiente actividad es escribir las recomendaciones de la evaluación de acuerdo a los patrones de diseño identificados en la arquitectura de software. Por ejemplo, si solamente se encontró el patrón de diseño *Facade* indicar los atributos de calidad involucrados en este patrón de diseño además de indicar los atributos de calidad que no se estén cumpliendo debido a la falta de más patrones de diseño en la arquitectura de software. El reporte con los resultados y recomendaciones es entregado al cliente, diseñador y arquitecto para realizar futuras mejoras a la arquitectura de software.

En esta sección se ha presentado el proceso PEASSE cuya finalidad es evaluar arquitecturas de software que sean utilizadas en el sector empresarial. En este documento PEASSE solamente utiliza dos métodos de evaluación: ALMA y SNA. Sin embargo PEASSE es lo suficientemente flexible de incorporar otros métodos de evaluación para arquitecturas de software. Los pasos principales que conforman a PEASSE son: Definir meta de la evaluación, seleccionar atributo de calidad en base a la meta, identificar patrones de diseño, seleccionar método de evaluación, evaluar arquitectura y preparar reporte de evaluación.

5 Conclusiones y trabajo futuro

PEASSE es un proceso propuesto para que sea utilizado en la fase de validación de la metodología DFSS conocida como IDOV. PEASSE es un proceso de evaluación de arquitecturas de software usadas en el sector empresarial por lo que esta limitado a un dominio de aplicación especifico. PEASSE es un proceso que se concibió para ser usado en fases de evaluación tempranas es decir cuando la arquitectura de software aún no ha sido implementada. Sin embargo es posible utilizar PEASSE para realizar evaluaciones a arquitecturas de software en fases tardías siempre y cuando se hayan aplicado técnicas de ingeniería inversa para reconstruir la arquitectura de software.

Los métodos de evaluación propuestos en este documento están basados en técnicas de evaluación cualitativas como lo son el uso de escenarios, cuestionarios y *Checklists*. Sin embargo una de las fortalezas de PEASSE es que facilita la incorporación de métodos cuantitativos como son: Simulaciones y modelos matemáticos.

Un trabajo futuro es extender PEASSE para que sea utilizado en otros dominios de aplicación, una de las actividades necesarias para lograr este objetivo es la necesidad de identificar estilos arquitectónicos y patrones de diseño usados en un dominio de aplicación específico.

6 Glosario de términos

Término	Descripción
Sigma	Símbolo griego usado para representar la desviación estándar, o la cantidad de variación en un proceso.
DFSS	*Design for six sigma* (Diseño para seis sigma).
Sistema legado	Sistemas que son heredados en las empresas, comúnmente sistemas viejos.
ALMA	*Architecture-Level Modifiability Analysis*.
SNA	*Survivable Network Analysis*.
DTO	*Data Transfer Object*.
PEASSE	Proceso de Evaluación de Arquitecturas de Software en el Sector Empresarial.
Benchmark	Punto de referencia.
Trade-off	Para el contexto de este documento un Trade-off es un equilibrio entre varios atributos de calidad.
Escenario	En el contexto de este documento un escenario es una breve descripción de una simple interacción del usuario con el sistema. De esta manera es posible comprender las relaciones de esta interacción con los componentes de la arquitectura de software.
Checklist	Lista de verificación.
Struts	*Framework* desarrollado en el lenguaje de programación Java utilizado para implementar el patrón arquitectónico MVC, este *Framework* usualmente es usado en el desarrollo de aplicaciones Web.
Spring	*Framework* desarrollado en el lenguaje de programación Java utilizado para implementar de manera optima sistemas de software desarrollados con el estilo arquitectónico de capas.
ADL	*Architecture Description Language*.
OCL	*Object Constraint Language*.

7 Referencias

[01] Antony, Jiju and Ricardo Bañuelas Coronado. "Design for Six Sigma." *IEE Manufacturing Engineering,* vol. 81, no. 1, pp. 119-21, 2002.

[02] Simon, Kerri. *What Is Dfss? and How Does Design for Six Sigma Compare to Dmaic?*. Available from http://software.isixsigma.com/library/content/c020722a.asp.

[03] Bañuelas Coronado, Ricardo and Jiju Antony. "Six Sigma Or Design for Six Sigma?" *The TQM Magazine,* vol. 16, no.4, pp. 250-163, 2004.

[04] Company, UGS ThePLM. *Business Process Initiative: Design for Six Sigma*. Available from http://www.ugs.com/products/nx/docs/wp_nx_six_sigma.pdf.

[05] Mikel Harry and Richard Schroeder. *Six Sigma; the Breakthrough Management Strategy Revolutionizing the World´s Top Corporations*: Currency, 2000.

[06] Len Bass, Paul Clements and Rick Kazman. *Software Architecture in Practice*. Edited by SEI Series In Software Engineering. second ed: Addison Wesley, 2003.

[07] Mary Shaw and David Garlan. *Software Architectures: Perspectives on an Emerging Discipline*: Prentice Hall, 1996.

[08] Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides. *Design Patterns Elements of Reusable Object-Oriented Software*. Edited by Addison-Wesley professional computing series: Addison Wesley, 1995.

[09] Chuck Cavaness. *Programming Jakarta Struts*: O'Reilly, 2002.

[10] Craig Walls and Ryan Breidenbach. *Spring in Action*: Manning Publications, 2005.

[11] IEEE Standard 1061-1992, *Standard for Software Quality Metrics Methodolgy,* New York: Institute of Electrical and Electronics Engineers, 1992.

[12] Bosch, Jan and Peter Molin. "Software Architecture Design: Evaluation and Transformation." Paper presented at the *Proceedings of IEEE Engineering of Computer Based Systems Symposium (ECBS '99)* 1999.

[13] Ali Babar, Muhammad, Liming Zhu and Ross Jeffery. "A Framework for Classifying and Comparing Software Architecture Evaluation Methods." Paper presented at the *Australian Software Engineering Conference*, Australia 2004.

[14] Kruchten, Philippe B. "The 4+1 View Model of Architecture." *IEEE Software,* vol. 12, no. 6, pp. 42-50, 1995.

[15] Soni, Dilip, Robert L. Nord and Christine Hofmeister. "Software Architecture in Industrial Applications." Paper presented at the *Proceedings of the 17th International Conference on Software Engineering*, Washington, USA 1995.

[16] Parnas, David L. and David M. Weiss. "Active Design Reviews: Principles and Practices." Paper presented at the *Proceedings of 18th International Conference on Software Engineering* 1985.

[17] Kazman, Rick, Len Bass, Gregory Abowd and Mike Webb. "Saam: A Method for Analyzing the Properties of Software Architectures." Paper presented at the *Proceedings of the 16th International Conference on Software Engineering*, Sorrento, Italy 1994.

[18] Kazman, Rick, Mark Klein, Mario Barbacci, Thomas Longstaff, Lipson Howard F. and S. Jeromy Carriere. "The Architecture Tradeoff Analysis Method." Paper presented at the *Proceedings of ICECCS*, Monterey, CA 1998.

[19] Williams, Lloyd G. and Connie U. Smith. "Pasa SM: A Method for the Performance Assessment of Software Architecture." Paper presented at the *Proceedings of the 3rd Workshop on Software Performance*, Rome, Italy 2002.

[20] Bengtsson, PerOlof, Nico Lassing and Jan Bosch, Vliet, Hans van. "Architecture-Level Modifiability Analysis (ALMA)." *The Journal of Systems and Software* , vol. 69, pp. 129-147, 2004.

[21] Mead, Nancy R., Robert J. Ellison, Richard C. Linger, Thomas Longstaff and John McHugh. "Survivable Network Analysis Method." CMU/SEI, 2000.

[22] Paul Clements, Rick Kazman and Mark Klein. *Evaluating Software Architectures*. Edited by The SEI Series in Software Engineering: Addison Wesley, 2002.

[23] Klein, Mark. H. *A Practitioners' Handbook for Real-Time Analysis: Guide to Rate Monotonic Analysis for Real-Time Systems*. Boston, MA: Kluwer Academic Publishers, 1993
[24] R. K. Jain. *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measuremnt, Simulation, and Modeling*: Wiley, 1991.
[25] Ali Babar, Muhammad and Ian Gorton. "Comparison of Scenario-Based Software Architecture Evaluation Methods." Paper presented at the *11th Asia-Pacific Software Engineering Conference* 2004.
[26] Ellison, Robert J., Richard C. Linger, Thomas Longstaff and Nancy R. Mead. "Survivable Network System Analysis: A Case Study." *IEEE Software*, vol. 16, no.4, pp. 70-77, 1999.
[27] Ellison, Robert J., David A. Fisher, Richard C. Linger, Lipson Howard F., Thomas Longstaff and Nancy R. Mead. "Survivability: Protecting Your Critical Systems." *IEEE Internet Computing*, vol. 3 no. 6, pp. 55-63, 1999.
[28] Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad and Stal Michael. *Pattern Oriented Software Architecture a System of Patterns*. Edited by Wiley series in software design patterns, vol. 1, John Wiley & Sons Ltd, 1996.
[29] Floyd Marinescu. *EJB Design Patterns Advanced Patterns, Processes, and Idioms*: John Wiley & Sons, Inc., 2002.
[30] Matjaz Juric, Nadia Nashi, Craig Berry, Meeraj Kunnumpurath, John Carnell and Sasha Romanosky. *J2EE Design Patterns Applied*: Per Information, 2002.
[31] Dobrica, Liliana and Eila Niermela. "A Survey on Software Architecture Analysis Methods." *IEEE Transactions on Software Engineering*, pp 638-562, 2002.
[32] Crane, Earl, Hap Huynh, Jeongwoo Ko and Koichi Tominaga. "Western Pennsylvania Hospital Physician Recomendation System SNA Client Report." 2000.

8 Apéndice A. Checklist para evaluación

El siguiente *Checklist* es de utilidad para revisar si en la arquitectura de software existen elementos tales como: Vistas arquitectónicas, métodos de documentación o patrones de diseño. El utilizar patrones de diseño en la arquitectura de software promueve que se satisfagan requerimientos de calidad tales como: Facilidad de modificación, seguridad, rendimiento entre otros.

A continuación se presenta el *Checklist* de evaluación el cual esta formado por cuatro secciones: Identificación de vistas de Nord, identificación de vistas de Kruchten, identificación de métodos de documentación e identificación de patrones.

Propósito.	Verificar si la arquitectura de software a analizar contiene vistas arquitectónicas, métodos de documentación y patrones de diseño.
Instrucciones.	Conforme verifique que se ha cubierto cada elemento de la lista de verificación, marque el recuadro según corresponda.

Elemento a verificar	**Revisado**
Identificación de vistas de Nord	
¿La vista conceptual esta incluida en la arquitectura de software?	
¿La vista de módulos esta incluida en la arquitectura de software?	
¿La vista de ejecución esta incluida en la arquitectura de software?	
¿La vista de código esta incluida en la arquitectura de software?	
Identificación de vistas de Kruchten	
¿La vista lógica esta incluida en la arquitectura de software?	
¿La vista de proceso esta incluida en la arquitectura de software?	
¿La vista física esta incluida en la arquitectura de software?	
¿La vista de desarrollo esta incluida en la arquitectura de software?	
¿La arquitectura de software incluye escenarios que muestran cómo los componentes especificados en las cuatro vistas anteriores trabajan en conjunto?	
Identificación de métodos de documentación	
¿Se utiliza algún ADL como método para documentar la arquitectura de software?	
¿Se utiliza OCL como lenguaje para especificar restricciones y reglas a los componentes de la arquitectura de software?	
¿Se utiliza algún lenguaje de notación grafica para representar las vistas de la arquitectura de software?	
Identificación de patrones	
¿La arquitectura de software esta elaborada usando uno o mas estilos arquitectónicos?	
¿La arquitectura de software implementa el patrón arquitectónico MVC?	
¿La arquitectura de software implementa el patrón de diseño *Facade*?	
¿La arquitectura de software implementa el patrón de diseño *Singleton*?	
¿La arquitectura de software implementa el patrón de diseño DTO?	
¿La arquitectura de software implementa algún otro patrón de diseño?	