Universidad de las Ciencias Informáticas
“Facultad 3”

“Modelo para la ayuda en la toma de decisiones relativas a estilos y patrones arquitectónicos”

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

Autores: Dainerys Díaz Pastrana

Joisel Pérez Pérez

Tutores: Msc. Karina Pérez Teruel

Ing. Dayana Caridad Tejera Hernández

Co-tutor: José Raúl Rodríguez Galera

Junio de 2008

DECLARACIÓN DE AUTORÍA

Nosotros, Dainerys Díaz Pastrana y Joisel Pérez Pérez, declaramos ser autores de la presente tesis y reconocemos a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

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

_____ _____

Dainerys Díaz Pastrana Joisel Pérez Pérez

_____ _____

Karina Pérez Teruel Dayana Caridad Tejera Hernández

"En lo tocante a la ciencia, la autoridad de un militar no es superior al humilde razonamiento de una sola persona."

Galileo Galilei

AGRADECIMIENTOS

A mis padres, mi abuela, mi tía Teresa y a toda mi familia, por su apoyo y confianza siempre.

A mi novio, por todo su cariño, comprensión y ayuda.

A mi compañero de tesis, por su constancia, asistencia y excelente amistad, desde hace cinco años.

A mis amigos, Maylen, Yuly, Halena, Osain, Rafael, Saidel, Norlys, Ailin, Lula, Jorge, Dismey, Merian, que viejos o nuevos amigos, siempre han sido los mejores.

A mis compañeritos de grupo, que siempre me han ayudado.

A todo el que de una u otra manera me ayudó, a todos los amigos que aquí encontré.

GRACIAS!!!

DEDICATORIA

A mi papá Luis, mi mamá Teresa, mi abuela Victoria y mi hermanita Dalylah, todo ha sido para ustedes...

AGRADECIMIENTOS

Por estar presente en todo momento, por las largas noches de trabajo aún en la distancia, por bajar a la realidad mis razonamientos y por aceptar estar al lado de mis constantes solicitudes, por existir, a mi amiga y compañera de Tesis Dainerys Díaz Pastrana.

A mis padres José Lorenzo y Nilda Pérez y a mi hermano Javier por todo el apoyo que siempre me han dado, por su paciencia, dedicación y amor, gracias a ellos y a la revolución pude llegar aquí. Los quiero mucho.

A toda mi familia que nunca cejó en darme aliento para llegar.

Al Comandante en Jefe Fidel Castro Ruz por su ejemplo de constancia y lucha, que siempre seguiré.

A todos los compañeros del CIMEX (Vicky, Lili, Selva, Miguel, Jorge, Lázaro, Luis y Yuri) que me dieron su ayuda incondicional y estuvieron siempre al tanto apoyándome.

A todos mis compañeros de la UCI (Lianet, Eilys, Raymond, Juan Carlos, Osnier, Jorge Luis, Dina, Jazmín, Eliecer, Aneli, etc.), que nunca me fallaron cuando realmente les necesitaba.

A mis hermanos y hermanas del grupo 3305 Iosmel, Anabel, Osmar, Dania, con quienes compartí muchas dudas, tristezas y alegrías, espero seguirlos viendo.

A mi pareja por su paciencia e infinita comprensión.

A las Decanas Yvonne y Saida, a los vice-decanos Pedro y Adrian, a la Profesora Dariela, y a todos los profesores de la Facultad 3, por abogar siempre por el mejoramiento humano, y encontrar soluciones a cualquier dificultad.

A mis tutores Karina, Dayana y José Raúl, por compartir sus conocimientos y permitirme aprender, fueron, al decir de José de La Luz y Caballero, educadores, y verdaderos amigos.

A todos los especialistas que dedicaron su tiempo para validar este trabajo, y a los compañeros de la UCI Teresa, Laritza, César y Pérez Lazo, que me brindaron su mano en la culminación de esta investigación.

En fin, son muchos los que contribuyeron con un granito de arena, a todos, mis más sinceros agradecimientos por estar.

Joisel

DEDICATORIA

A mis padres y a mi hermano Javier, por haber sido mi sustento diario en todo momento, siempre estuvieron allí, gracias por todo, a ustedes y por ustedes.

A esta Revolución y al Comandante en Jefe, nosotros los pinos nuevos, seguiremos con su obra.

Joisel

RESUMEN

En el proceso de desarrollo y diseño de software los atributos de calidad juegan un papel importante, ya que las decisiones de arquitectura de estos sistemas se generan en base a los mismos. Estas decisiones están influenciadas por algunos elementos de diseño, los estilos y patrones arquitectónicos. Se hace difícil para los arquitectos de software definir cuál estilo o patrón arquitectónico se debe emplear en el diseño de la arquitectura, pues resulta complejo aceptar todos los atributos de calidad para un sistema, debido a que cualquier cambio en la arquitectura para mejorar un atributo afectará negativamente a otro. Esto conduce a la no satisfacción de los atributos de calidad especificados por el cliente. Para ello, en la siguiente investigación se desarrolló un modelo para la toma de decisiones relativas a estilos y patrones arquitectónicos basado en los atributos de calidad definidos para el producto de software a desarrollar. El modelo consta de tres pasos fundamentales: primero, el análisis de la relación e influencia entre atributos de calidad; segundo, la ponderación de los atributos de calidad y tercero, la definición de reglas para seleccionar estilo(s) y patrón(es) arquitectónico(s) para potenciar determinados atributos de calidad. El modelo propuesto se validó utilizando el método Criterio de Especialistas.

PALABRAS CLAVES

Atributos de calidad, arquitectura de software, estilos arquitectónicos, patrones arquitectónicos, modelo.

TABLA DE CONTENIDOS

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

INTRODUCCIÓN

Cuba es un país donde el desarrollo de software es aún incipiente. Es por ello que una de las tareas propuestas por el Gobierno Cubano es desarrollar la Industria del Software, lo que ha traído consigo la concepción de estrategias como la creación de la Universidad de las Ciencias Informáticas (UCI) con el fin de elevar la producción y calidad del software cubano.

En la UCI, como parte del proceso de producción de software y de las necesidades actuales que tiene para el logro de sus objetivos, se demanda la construcción de grandes y complejos sistemas de software que requieren de la combinación de diferentes tecnologías y plataformas de hardware y software para alcanzar un funcionamiento acorde con dichas necesidades. Lo anterior exige poner especial atención y cuidado al diseño de la arquitectura bajo la cual estará soportado el funcionamiento de sus sistemas.

El diseño de la arquitectura de software es una de las etapas fundamentales y, en muchos casos, la más importante en el desarrollo de software, pues es aquí donde se aportan todos los conocimientos, creatividad y experiencia de los desarrolladores para crear la mejor propuesta de solución que se dará al cliente.

A lo largo de este proceso de desarrollo y diseño, los atributos de calidad juegan un papel importante, pues en base a estos se generan las decisiones de diseño y argumentos que los justifican. Dado que la arquitectura de software inhibe o facilita los atributos de calidad, resulta de particular interés analizar la influencia de ciertos elementos de diseño utilizados para la definición de la misma, determinando sus características. Estos elementos de diseño son los estilos arquitectónicos y los patrones arquitectónicos.

No solo los requisitos funcionales de un sistema de software son importantes para el diseño arquitectónico. También atributos de calidad deben ser tenidos en cuenta para determinar la estructura y el comportamiento que tendría un sistema. Hacer un adecuado balance entre la mantenibilidad, la interoperabilidad, la portabilidad, la eficiencia, la seguridad, la disponibilidad y la reusabilidad, entre otros atributos, es esencial para obtener un sistema que cumpla las expectativas de las distintas partes interesadas (1).

Lo antes expresado constituye una **dificultad** pues resulta muy difícil aceptar todos los atributos de calidad para un sistema debido a que cualquier cambio en la arquitectura para mejorar un atributo, en general estará afectando negativamente a otro. Cada interesado en el sistema (usuario, cliente, desarrollador, arquitecto, etc.) estará preocupado por algún atributo en específico, pero es imposible alcanzar todos, como ya se dijo. Es ahí cuando se dice que entran en conflicto, ya que es complejo definir cual estilo o patrón arquitectónico se debe emplear en el diseño de la arquitectura. Esto influye negativamente en el cumplimiento de los atributos de calidad especificados por el cliente, ya que la selección del estilo o patrón se realiza, casi siempre, basada en conocimiento empírico por parte de los arquitectos y analistas del proyecto, provocando errores en la concepción de la arquitectura base del software a desarrollar.

Por tanto, es de gran importancia la creación de un modelo que le permita a los arquitectos, primero, valorar cuál(es) atributo(s) de calidad son esenciales y cuáles importantes para el software que va a desarrollarse, en dependencia de las características planteadas por el cliente y, segundo, seleccionar cuáles son los estilos y patrones arquitectónicos más adecuados en correspondencia con estos atributos para diseñar la arquitectura de software.

De ahí que el **problema científico** en cuestión sea:
Los errores en la toma de decisiones relativas a la arquitectura de software provocan la no satisfacción de los atributos de calidad establecidos para el producto a obtener.

Se tiene entonces como **objeto de estudio** la arquitectura de software y como **campo de acción** los modelos para la toma de decisiones referentes a estilos y patrones arquitectónicos.

El **objetivo general** de esta investigación es desarrollar un modelo para la toma de decisiones relativas a estilos y patrones arquitectónicos basado en los atributos de calidad definidos para el producto de software a desarrollar.

Se tienen los siguientes **objetivos específicos**:

1. Estudiar los principales conceptos y elementos relacionados con los modelos para la toma de decisiones relativas a estilos y patrones arquitectónicos.
2. Confeccionar un modelo para la toma de decisiones relativas a estilos y patrones arquitectónicos.
3. Validar el modelo obtenido.

Se formula entonces la **hipótesis**:
Si se desarrolla un modelo para la toma de decisiones relativa a estilos y patrones arquitectónicos se influirá positivamente en la satisfacción de un mayor número de atributos de calidad establecidos para el producto a obtener.

Para dar cumplimiento a cada uno de los objetivos trazados se realizarán las siguientes **tareas de investigación**:

- Sistematización del estudio del estado del arte.
- Clasificación de los atributos de calidad y de los estilos y patrones de arquitectura a considerar en la investigación.
- Estudio de las relaciones e influencias entre los atributos de calidad.
- Establecimiento de los principales parámetros de los atributos de calidad.
- Confección de las reglas a seguir para la elección de estilos y patrones arquitectónicos.
- Aplicación del método Criterio de Especialistas para la validación de la propuesta.

Se utilizaron varios **métodos de trabajo científico**, el teórico y el lógico. Dentro del método teórico, se empleó el *histórico,* que permite estudiar el proceso en cuestión y obtener un conocimiento histórico de su desarrollo y comportamiento. Del método lógico se utilizó el *hipotético - deductivo* ya que se infieren conclusiones a partir de la hipótesis y se comprueba.

La tesis está conformada por dos capítulos. En el primero se realiza un estudio sobre los atributos de calidad, la arquitectura y las relaciones entre estos dos elementos. Se profundiza, además, en los estilos y patrones arquitectónicos, algunos métodos basados en la arquitectura y una herramienta para el diseño de la misma.

En el capítulo dos se propone un modelo para la ayuda en la toma de decisiones relativas a estilos y patrones arquitectónicos, destacando ciertos elementos a tener en cuenta sobre los atributos de calidad y elaborando reglas que sugieren qué estilos y patrones seleccionar ante determinadas situaciones arquitectónicas. Se valida, además, el modelo propuesto utilizando el método Criterio de Especialistas.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

En este capítulo se realizará un estudio sobre los modelos o métodos que existen, relativos a la selección de estilos y patrones a emplear en el diseño y desarrollo de la arquitectura, así como una panorámica referente a la arquitectura y atributos de calidad de software, o sea, se definen aspectos importantes de la teoría en que se basa la solución del problema facilitando de esta manera la comprensión del mismo.

1.1. Atributos de calidad.

Los usuarios, naturalmente, se enfocan en especificar los requisitos funcionales, o de comportamiento, pero hay más en el éxito del software que entregar la funcionalidad correcta simplemente. Los usuarios también tienen expectativas sobre cuán bien el producto trabajará. Las características que caen en esta categoría incluyen cuán fácil de usar es el software, con qué rapidez da respuesta a una solicitud del usuario, cuán a menudo falla y cómo se ocupa de condiciones inesperadas. Tales características son conocidas como atributos o factores de calidad (2).

Los atributos de calidad son un subconjunto de los requisitos no funcionales, un tipo de requisito no funcional que describe una cualidad o propiedad de un sistema. Estos dicen, además, hasta que punto un producto de software demuestra características deseadas y no lo que el producto hace (2).

Los atributos de calidad constituyen la base para la evaluación de una arquitectura, pero por si solos no son suficientes para juzgar la adecuabilidad de esta. Generalmente, los atributos son escritos de la siguiente manera (2):

- “El sistema debe ser robusto.”
- “El sistema debe ser modificable.”
- “El sistema debe ser seguro.”
- “El sistema debe tener un rendimiento aceptable.”

Cada una de las frases anteriores está sujeta a diferentes interpretaciones y malos entendidos. Lo que uno puede considerar robusto, otro puede considerarlo apenas aceptable. El punto es que los atributos de calidad no son cantidades absolutas, existen en un contexto con metas específicas. Por ejemplo, un

sistema es modificable (o no) con respecto a un tipo de cambio específico, es seguro (o no) con respecto a una amenaza determinada (3).

1.1.1 Clasificaciones de los atributos de calidad:

En primer lugar se debe acotar el contexto al que se hace referencia, ya que la calidad en el software se puede entender como calidad de proceso o de producto. La calidad de proceso ha sido objeto de mucho interés en las últimas décadas y su desarrollo ha concienciado a los profesionales de la necesidad de aplicar la calidad a otros aspectos del software. Esta investigación se centra específicamente en la calidad del producto software y, dentro de ella, en el producto derivado de la arquitectura (4).

Varias docenas de características de los productos pueden ser llamadas atributos de calidad aunque la mayoría de los proyectos deben considerar cuidadosamente sólo un puñado de ellos. Si los desarrolladores conocen cuáles de estas características son las cruciales para el éxito de los proyectos, entonces podrán seleccionar la arquitectura para alcanzar los objetivos de calidad especificados. Los atributos de calidad han sido clasificados, de acuerdo a varios criterios o esquemas, en modelos.

Una forma de clasificar los atributos es distinguir aquellas características del software que son discernibles en tiempo de ejecución de aquellas que no lo son. Otro enfoque consiste en separar las características visibles que son fundamentalmente importantes para los usuarios de las que son fundamentalmente importantes para el personal técnico. Este último punto de vista contribuye indirectamente a la satisfacción del cliente, al hacer el producto más fácil de cambiar, corregir, verificar y migrar a nuevas plataformas (2).

Los modelos que se presentarán a continuación tienen una estructura por lo general en niveles. Son los que han ganado mayor popularidad en la comunidad. En el nivel más alto se encuentran los *factores* de calidad, que representan la calidad desde el punto de vista del usuario. Son las características que componen la calidad. También se les llama atributos de calidad externos (5).

Cada uno de los factores se descompone en un conjunto de *criterios* de calidad. Son atributos que, cuando están presentes, contribuyen al aspecto de la calidad que el factor asociado representa. Se trata

de una visión de la calidad desde el punto de vista del producto de software. También se les llama atributos de calidad internos (5).

El modelo de McCall fue el primero en ser presentado en 1977 y se originó motivado por US Air Force y DoD (Deparment of Defense). Se focaliza en el producto final, identificando atributos claves desde el punto de vista del usuario que, como ya se dijo, son denominados factores de calidad. Estos últimos son demasiado abstractos para ser medidos directamente, por lo que, por cada uno de ellos se introducen atributos de bajo nivel, los criterios de calidad, reflejando así la creencia de McCall de que el atributo interno tiene un efecto directo en el atributo externo correspondiente.

McCall organiza los factores de calidad en tres ejes, o sea, propone tres perspectivas para agruparlos (6):

- *Operación del producto* (características de operación): Requiere que pueda ser comprendida rápidamente, operada de manera eficiente y que los resultados sean aquellos requeridos por el usuario.

- *Revisión del producto* (habilidad para ser cambiado): Está relacionada con la corrección de errores y la adaptación de los sistemas. Esto es importante porque es generalmente considerada como la parte más costosa en el desarrollo de software.

- *Transición del producto* (adaptabilidad al nuevo ambiente): Puede que no sea muy importante en todas las aplicaciones. Sin embargo, la orientación a procesamiento distribuido y el rápido cambio en el hardware es probable que incremente su importancia.

La *operación del producto* incluye los siguientes aspectos o factores de calidad**:**

Corrección (correctitud)**:** Hasta qué punto un programa cumple sus especificaciones y satisface los objetivos del usuario. Por ejemplo, si un programa debe ser capaz de sumar dos números y en lugar de sumar los multiplica, es un programa incorrecto. Es quizás el factor más importante, aunque puede no servir de nada sin los demás factores. (El grado en el que el producto cumple con su especificación).

Fiabilidad (confiabilidad)**:** Hasta dónde se espera que un programa lleve a cabo su función pretendida con la exactitud requerida. Hasta qué punto se puede confiar en el funcionamiento sin errores del programa. Por ejemplo, si el programa anterior suma dos números, pero en un 25% de los casos el resultado no es correcto, es poco fiable. (La habilidad del producto de responder ante situaciones no esperadas).

Eficiencia: Cantidad de código y de recursos informáticos (CPU, memoria) que precisa un programa para desempeñar su función. Un programa que suma dos números y necesita 2 MB de memoria para funcionar, o que tarda 2 horas en dar una respuesta, es poco eficiente. (El uso de los recursos tales como tiempo de ejecución y memoria de ejecución).

Integridad: Hasta qué punto se controlan los accesos ilegales a programas o datos. Un programa que permite el acceso de personas no autorizadas a ciertos datos es poco íntegro. (Protección del programa y sus datos).

Usabilidad (Facilidad de manejo)**:** Esfuerzo necesario para aprender, operar, preparar los datos de entrada e interpretar las salidas (resultados) de un programa. (Facilidad de operación del producto por parte de los usuarios).

La *revisión del producto* incluye los siguientes factores de calidad:

Mantenibilidad (Facilidad de mantenimiento)**:** Esfuerzo o costo necesario para localizar y corregir defectos del programa. (Esfuerzo requerido para localizar y corregir fallas).

Flexibilidad: Esfuerzo o costo de modificación del producto cuando cambian sus especificaciones. (Facilidad de realizar cambios).

Facilidad de prueba (testeabilidad)**:** Esfuerzo o costo de probar un programa para comprobar que satisface sus requisitos. Por ejemplo, si un programa requiere desarrollar una simulación completa de un sistema para poder probar que funciona bien, es un programa difícil de probar. (Facilidad para realizar pruebas, para asegurarse que el producto no tiene errores y cumple con la especificación).

La *transición del producto* incluye los siguientes factores de calidad:

Portabilidad: El coste de transportar o migrar un producto de una configuración hardware o entorno operativo a otro. (Esfuerzo requerido para transferir entre distintos ambientes de operación).

Reusabilidad (capacidad de reutilización)**:** Hasta qué punto se puede transferir un módulo o programa del presente sistema a otra aplicación, y con qué esfuerzo. (Facilidad de rehusar el software en diferentes contextos).

Interoperabilidad: El coste y esfuerzo necesario para hacer que el software pueda operar conjuntamente con otros sistemas o aplicaciones de software externos. (Esfuerzo requerido para acoplar el producto con otros sistemas).

Cada uno de estos factores se descompone, a su vez, en criterios. En el modelo de McCall se definen un total de veintitrés criterios. Para dar una noción más clara de los puntos de vista y los factores de calidad se presenta la siguiente tabla y su esquema (Ver Figura 1.):

PUNTO DE VISTA	FACTORES
Operación del producto	- Facilidad de uso - Integridad - Corrección - Fiabilidad - Eficiencia
Revisión del producto	- Facilidad de mantenimiento - Facilidad de prueba - Flexibilidad
Transición del producto	- Facilidad de reutilización - Interoperabilidad - Portabilidad

Tabla 1. Puntos de vista y factores (5)

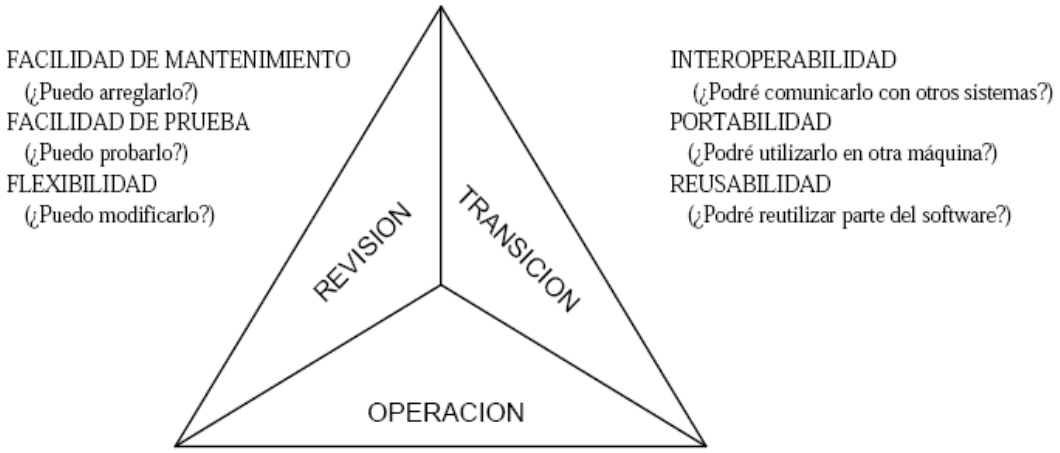


Figura 1. Esquema de puntos de vista y factores (5)

La relación Factores - Criterios que establece el modelo queda plasmada en la siguiente tabla:

Factor	**Criterios**
Usabilidad (Facilidad de uso)	- Facilidad de operación (operabilidad) - Facilidad de Comunicación - Facilidad de Aprendizaje (entrenamiento) - Volumen de E/S Tasa de E/S
Integridad	- Control de Accesos - Auditoria de Accesos
Corrección (correctitud)	- Completitud - Consistencia

	- Trazabilidad
Fiabilidad (confiabilidad)	- Precisión (exactitud) - Consistencia - Tolerancia a fallos - Modularidad - Simplicidad
Eficiencia	- Eficiencia en ejecución (en tiempo) - Eficiencia en almacenamiento (en espacio)
Mantenibilidad (Facilidad de mantenimiento)	- Simplicidad - Consistencia - Concisión - Auto descripción - Modularidad
Testeabilidad (Facilidad de Prueba)	- Simplicidad - Auto descripción - Instrumentación
Flexibilidad	- Auto descripción - Capacidad de expansión - Generalidad - Modularidad
Reusabilidad	- Generalidad - Modularidad - Independencia entre Sistema y Software (independencia del SO) - Independencia del hardware - Auto descripción
Interoperabilidad	- Modularidad - Compatibilidad de comunicaciones - Compatibilidad de datos

Portabilidad	- Auto descripción - Independencia entre Sistema y Software (independencia del SO) - Independencia del hardware - Modularidad

Tabla 2. Relación entre factores - criterios. (5)

Es válido aclarar que con los años el factor de flexibilidad se ha ido fusionando con mantenibilidad. De hecho, en la definición original, dos de los criterios de flexibilidad estaban compartidos con mantenibilidad. También la usabilidad ha cambiado mucho desde la época de McCall. Actualmente incluye aspectos tales como adaptabilidad, aprendizaje y adecuación al contexto. Algunos autores consideran, por ejemplo, que facilidad de aprendizaje es un factor de calidad independiente.

El segundo modelo más conocido es el presentado por Barry Boehm, creador del COCOMO, en 1978. Este modelo introduce características de alto nivel, características de nivel intermedio y características primitivas. Las características de alto nivel, que representan requerimientos generales de uso, pueden ser (6):

- *Utilidad per-se:* Cuan usable, confiable, eficiente es el producto en sí mismo.
- *Mantenibilidad:* Cuan fácil es modificarlo, entenderlo y probarlo.
- *Utilidad general:* Si puede seguir usándose con un cambio de ambiente.

Las características de nivel intermedio representan los factores de calidad de Boehm y son mencionadas a continuación:

- Portabilidad (utilidad general).
- Confiabilidad (utilidad per-se).
- Eficiencia (utilidad per-se).
- Usabilidad o ingeniería humana (utilidad per-se).
- Testeabilidad o prueba (mantenibilidad).
- Facilidad de entendimiento entendibilidad (mantenibilidad).

- Modificabilidad o flexibilidad (mantenibilidad).

Las características primitivas representan a las siguientes características (Ver Figura 2.):

De portabilidad:

- Independencia de dispositivos.
- Auto-contención.

De confiabilidad:

- Auto-contención.
- Exactitud.
- Completitud.
- Consistencia.
- Robustez/Integridad.

De eficiencia:

- Accesibilidad.
- Contabilidad.
- Eficiencia de uso de dispositivos.

De usabilidad:

- Robustez/integridad.
- Accesibilidad.
- Comunicación.

De testeabilidad:

- Comunicación.
- Accesibilidad.
- Auto descripción.
- Estructuración.

De entendibilidad:

- Estructuración.
- Concisión.
- Legibilidad.
- Auto descripción.

De modificabilidad:

Estructuración.

Aumentabilidad.

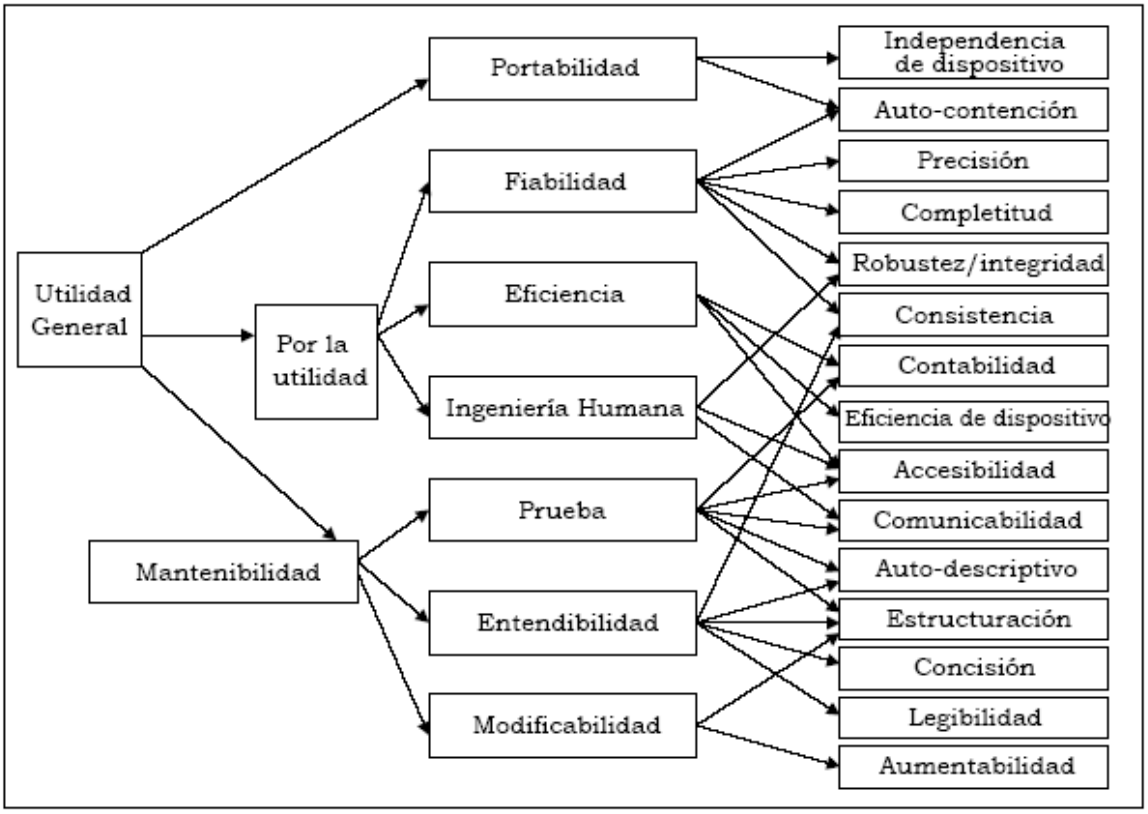


Figura 2. Calidad del producto de software según Boehm. (7)

Aunque ambos modelos parezcan similares, una de las diferencias está en que McCall focaliza en medidas precisas de alto nivel, mientras que Boehm presenta un rango más amplio de características

primarias. Por otro lado, Boehm incluye las necesidades de los usuarios, como lo hace McCall, e incluye además características de rendimiento de hardware que no se encuentran en el modelo de McCall.

Según las clasificaciones de Clements y Bass en 1998, podemos clasificar los atributos de calidad del software en dos grandes grupos: aquellos que pueden comprobarse durante la ejecución del software y aquellos que no pueden comprobarse durante la ejecución. Dentro del primer grupo se encuentran eficiencia, seguridad, disponibilidad, funcionalidad y usabilidad. Dentro del segundo, y no menos importante, están la modificabilidad, portabilidad, reusabilidad, integrabilidad y verificabilidad (1).

Esta clasificación es muy similar a la de McCall y Boehm en cuanto al uso de muchos atributos comunes en ambas, aunque con el avance de la tecnología y los diferentes contextos en los que se ha empleado, algunos han ido deviniendo en el uso de otros términos como es el caso de performance (rendimiento), seguridad (integridad), funcionalidad (corrección), verificabilidad (testeabilidad) e integrabilidad (interoperabilidad). Esta clasificación incluye dos términos nuevos que no existían en las anteriores: la disponibilidad y la fiabilidad del software, y aborda el término de modificabilidad que sería en un nivel más bajo, la asociación de mantenibilidad y flexibilidad definidos por McCall y Boehm.

Si bien esta clasificación, debido al empleo de atributos que también estaban presentes en la clasificación de McCall y Boehm y al uso de términos más acordes con el desarrollo del software, así como la introducción de otros nuevos, constituye un paso de avance, difiere en que no considera el criterio del usuario final como una prioridad a la hora de clasificar los atributos de calidad, cuando el software se realiza por un pedido del mismo.

En 1992, una variante del modelo de McCall fue propuesta como estándar internacional para medición de calidad de software. ISO 9126 Software Product Evaluation: Quality Characteristics and Guidelines for their Use es el nombre formal y está dividida en cuatro partes. La primera, ISO 9126-1, define seis características de calidad principales y veintisiete subcaracterísticas. Las otras tres partes son reportes técnicos (ISO/IEC 9126-2, 3 y 4).

El foco en la calidad cambia durante el ciclo de vida (6). Al principio, durante la recopilación de requisitos y análisis, la calidad es especificada por los requisitos de los usuarios, sobre todo desde el punto de vista

externo. En la fase de diseño e implementación, la calidad externa se traduce en un diseño técnico, confrontándose con el punto de vista de los desarrolladores sobre la calidad interna y complementándose con los requisitos implícitos que el software debe cumplir. La calidad final (la del uso) debe ser apropiada para los usuarios y el contexto de uso.

En ISO 9126 se reconocen seis factores de calidad que se pueden considerar tanto internos como externos, estas características son generales para cualquier tipo de programa informático o software:

Funcionalidad.
Confiabilidad.
Eficiencia.
Usabilidad.
Mantenibilidad.
Portabilidad.

En muchos casos difiere el grado de prioridad que le confieren los desarrolladores y los usuarios finales a estos factores (Ver Tabla 3.).

	Usuario	Desarrollador
funcion.	alta	baja
confiab.	media	media
usabil.	alta	baja
eficiencia	media	media
mantenib.	baja	alta
portab.	media	alta

Tabla 3. Prioridad que dan usuarios y desarrolladores a los atributos de calidad. (6)

En ISO 9126 se reconocen también cuatro factores de calidad de uso:

Eficacia.
Productividad.

Seguridad.

Satisfacción.

A su vez estas características incluyen un conjunto de subcaracterísticas (4):

Subcaracterísticas de funcionalidad

• Adaptabilidad.

• Exactitud.

• Interoperabilidad.

• Seguridad.

Subcaracterísticas de confiabilidad

• Madurez.

• Tolerancia a fallas.

• Recuperabilidad.

Subcaracterísticas de usabilidad

• Comprensibilidad.

• Aprendizaje.

• Operabilidad.

• Atractivo.

Subcaracterísticas de eficiencia

• Comportamiento en el tiempo.

• Utilización de los recursos.

Subcaracterísticas de mantenibilidad

• Analizabilidad.

• Modificabilidad.

• Estabilidad.

• Testeabilidad.

Subcaracterísticas de portabilidad

• Adaptabilidad

• Instalabilidad

• Coexistencia

• Reemplazabilidad

La norma ISO 9126 trata de establecer una definición general de lo que es la calidad del producto software en general basándose en esos atributos de calidad, desde el punto de vista del usuario final, del que lo construye y del uso, de ahí que exista la visión externa, la interna y la de uso. Sin embargo se puede decir que existen subproductos intermedios que dependen de estos atributos de calidad, y que nos pueden dar una previsión de la calidad del producto acabado, como es el caso de las arquitecturas. (Ver Figura 3.)

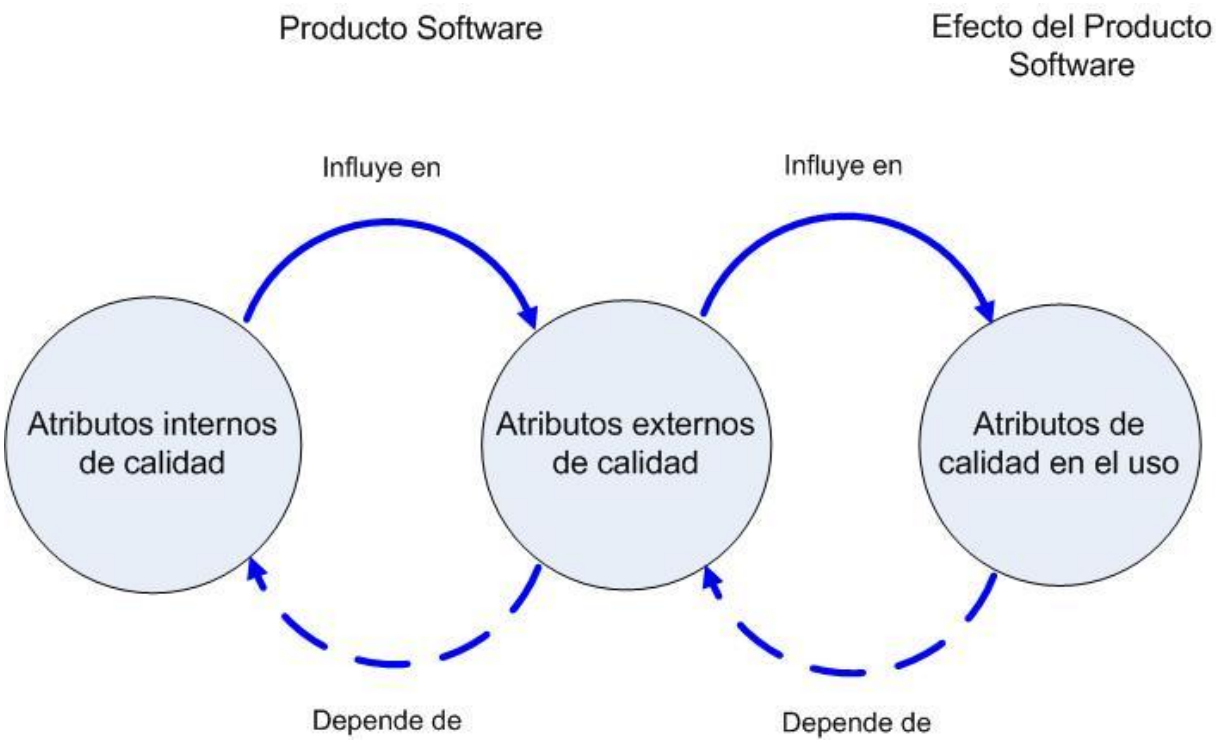


Figura 3. Calidad del Producto en el Ciclo de vida del software. (4)

Se considera que deben separarse los atributos importantes para los usuarios de los que son importantes para el equipo que desarrolla el software, ya que esto indirectamente contribuirá a la satisfacción del cliente. No se consideran por el momento los atributos de calidad en uso porque al definir los atributos importantes para el usuario y el equipo técnico, y mejorándolos, se mejora directamente la calidad observada en atributos de uso del producto software. Además, como es lógico pensar, no todos los atributos antes mencionados son medibles o tienen sentido a la hora de definir el diseño de una arquitectura de software, como es el caso en que se interesa esta investigación. Por tanto, hechas las consideraciones pertinentes y extrayendo los atributos más comunes de las clasificaciones ya mencionadas, considerando además que no existe una calidad perfecta o absoluta sino necesaria y suficiente para un contexto dado, se deriva en la siguiente clasificación (2):

Importante para los Usuarios	Importante para los Desarrolladores
Disponibilidad	Mantenibilidad
Eficiencia	Portabilidad
Flexibilidad	Reusabilidad
Integridad	Testeabilidad
Interoperabilidad	
Fiabilidad	
Robustez	
Usabilidad	

1.1.2. Relación entre arquitectura de software y atributos de calidad.

Para alcanzar un atributo específico es necesario tomar decisiones de diseño arquitectónico que requieren un pequeño conocimiento de funcionalidad. Por ejemplo, el desempeño depende de los procesos del sistema y su ubicación en los procesadores, caminos de comunicación, etc. Por otro lado, al considerar una decisión de arquitectura de software, el arquitecto se pregunta cuál será el impacto de esta sobre

ciertos atributos. Por esta razón, se afirma que cada decisión incorporada en una arquitectura de software puede afectar potencialmente los atributos de calidad.

Cada decisión tiene su origen en preguntas acerca del impacto sobre estos atributos y el arquitecto puede argumentar cómo la decisión tomada permite alcanzar algún objetivo. Con frecuencia, el objetivo es un atributo de calidad en particular, por lo que al tomar decisiones de arquitectura de software que afecten a los atributos de calidad, se tienen dos consecuencias (8):

- Se formula un argumento que explica la razón por la que la decisión tomada permite alcanzar uno o varios atributos de calidad. Esto resulta de gran importancia porque además permite comprender las consecuencias de un cambio de decisión.

- Surgen preguntas sobre el impacto de una decisión sobre otros atributos, que a menudo se responden en el contexto de otras decisiones.

La relación entre la arquitectura de software y los atributos de calidad no se encuentra sistemática y completamente documentada, debido a diversas razones (8):

- Existen atributos de calidad que, luego de ser estudiados durante años, poseen definiciones generalmente aceptadas. Sin embargo, existen algunos que carecen de definición, lo que inhibe el proceso de exploración de la relación en estudio.

- Los atributos no están aislados ni son independientes entre sí. Muchos atributos conforman un subconjunto de otro, es decir, se definen en función de otros atributos que lo contienen. Por ejemplo, el atributo disponibilidad puede ser un atributo por sí mismo; sin embargo, puede ser subconjunto de usabilidad y seguridad.

- El análisis de atributos no se presta a estandarizaciones, puesto que existen diferentes patrones con distintos niveles de profundidad, y resulta complicado establecer cuáles patrones se pueden utilizar para analizar calidad.

- Las técnicas de análisis son específicas para un atributo en particular. Por esta razón es difícil comprender la interacción entre varios análisis de atributos específicos.

1.2. Arquitectura de software.

La arquitectura de software es un área de investigación y práctica dentro de la Ingeniería de Software. En particular, la arquitectura de sistemas grandes ha sido objeto de un interés creciente durante la pasada década (8).

1.2.1. Definición.

El concepto de arquitectura se usa de forma amplia y en campos muy distintos, por lo que su significado es algo difuso. Actualmente, en la literatura es posible encontrar numerosas definiciones del término arquitectura de software, cada una con planteamientos diversos. Se hace evidente que su conceptualización sigue todavía en discusión, puesto que no es posible referirse a un diccionario en busca de un significado y tampoco existe un estándar que pueda ser tomado como marco de referencia.

Sin embargo, al hacer un análisis detallado de cada uno de los conceptos disponibles, resulta interesante la existencia de ideas comunes entre los mismos, sin observarse planteamientos contradictorios, sino más bien complementarios. La intención primordial del análisis no es concluir ni proponer un concepto que englobe todas las ideas planteadas hasta el momento, sino establecer aquellos elementos que no deben perderse de vista al momento de introducirse en el contexto de las arquitecturas de software.

No es novedad que ninguna definición de la arquitectura de software es respaldada unánimemente por la totalidad de los arquitectos. El número de definiciones circulantes alcanza un orden de tres dígitos, amenazando llegar a cuatro. De hecho, existen grandes compilaciones de definiciones alternativas o contrapuestas, como la colección que se encuentra en el SEI a la que cada quien puede agregar la suya (9).

En general, las definiciones entremezclan despreocupadamente tres elementos: primero, el trabajo dinámico de estipulación de la arquitectura dentro del proceso de ingeniería o el diseño (su lugar en el

ciclo de vida), segundo, la configuración o topología estática de sistemas de software contemplada desde un elevado nivel de abstracción y tercero, la caracterización de la disciplina que se ocupa de uno de esos dos asuntos, o de ambos (9).

Algunos autores plantean sobre la arquitectura de software (10):

– "…*nivel del diseño del software donde se definen la estructura y propiedades globales del sistema".*
(Garlan y Perry, 1995)

– "…*se centra en aquellos aspectos del diseño y desarrollo que no pueden tratarse de forma adecuada dentro de los módulos que forman el sistema*."
(Shaw y Garlan, 1996)

En resumen:

La arquitectura de software de un programa o sistema de computación es la estructura o las estructuras del sistema, que contienen componentes de software, las propiedades externamente visibles de dichos componentes y las relaciones entre ellos.

1.2.2. Importancia

Aunque todavía no se ha constituido un repositorio uniformizado de estudios de casos en base al cual se pueda extraer una conclusión responsable, la arquitectura de software ha resultado instrumental en un número respetable de escenarios reduciendo costos, evitando errores, encontrando fallas, implementando sistemas de misión crítica, etc (9).

Aún cuando se señalen dificultades ocasionales, nadie duda de la necesidad de una visión arquitectónica. Según Barry Boehm:

"Si un proyecto no ha logrado una arquitectura del sistema, incluyendo su justificación, el proyecto no debe empezar el desarrollo en gran escala. Si se especifica la arquitectura como un elemento a entregar, se la puede usar a lo largo de los procesos de desarrollo y mantenimiento"

Sintetizando, las virtudes de la arquitectura de software son (9):

Comunicación mutua: La arquitectura de software representa un alto nivel de abstracción común que la mayoría de los participantes, si no todos, pueden usar como base para crear entendimiento mutuo, formar consenso y comunicarse entre sí. En sus mejores expresiones, la descripción arquitectónica expone las restricciones de alto nivel sobre el diseño del sistema, así como la justificación de decisiones arquitectónicas fundamentales.

Decisiones tempranas de diseño: La arquitectura de software representa la encarnación de las decisiones de diseño más tempranas sobre un sistema, y esos vínculos tempranos tienen un peso fuera de toda proporción en su gravedad individual con respecto al desarrollo restante del sistema, su servicio en el despliegue y su vida de mantenimiento. La arquitectura representa una puerta de peaje: el desarrollo no puede proseguir hasta que los participantes involucrados aprueben su diseño.

Restricciones constructivas: La arquitectura de software indica los componentes y las dependencias entre ellos. Por ejemplo, una vista en capas de una arquitectura documenta típicamente los límites de abstracción entre las partes, identificando las principales interfaces y estableciendo las formas en que unas partes pueden interactuar con otras.

Reutilización, o abstracción transferible de un sistema: La arquitectura de software encarna un modelo relativamente pequeño, intelectualmente tratable, de la forma en que un sistema se estructura y sus componentes se entienden entre sí; este modelo es transferible a través de sistemas; en particular, se puede aplicar a otros sistemas que exhiben requerimientos parecidos y puede promover reutilización en gran escala. El diseño arquitectónico soporta reutilización de grandes componentes o incluso de frameworks en el que se pueden integrar componentes.

Evolución: La arquitectura de software expone las dimensiones a lo largo de las cuales puede esperarse que evolucione un sistema. Haciendo explícitas estas “paredes” perdurables, quienes mantienen un sistema pueden comprender mejor las ramificaciones de los cambios y estimar con mayor precisión los costos de las modificaciones. Esas delimitaciones ayudan también a establecer mecanismos de conexión que permiten manejar requerimientos cambiantes de interoperabilidad, prototipado y reutilización.

Análisis: Las descripciones arquitectónicas aportan nuevas oportunidades para el análisis, incluyendo verificaciones de consistencia del sistema, conformidad con las restricciones impuestas por un estilo, conformidad con atributos de calidad, análisis de dependencias y análisis específicos de dominio y negocios.

Administración: La experiencia demuestra que los proyectos exitosos consideran una arquitectura viable como un logro clave del proceso de desarrollo industrial. La evaluación crítica de una arquitectura conduce típicamente a una comprensión más clara de los requerimientos, las estrategias de implementación y los riegos potenciales.

1.3. Estilos y patrones

Un estilo es un concepto descriptivo que define una forma de articulación u organización. El conjunto de los estilos cataloga las formas básicas posibles de estructuras de software, mientras que las formas complejas se articulan mediante composición de los estilos fundamentales (11).

Los patrones de software, por su parte, no son más que un conjunto de soluciones a problemas habituales. Una definición más formal podría ser: “Un patrón es una solución a un problema particular, aceptada como correcta, a la que se ha dado un nombre y que puede ser aplicada en otros contextos”.

Un patrón captura la experiencia y conocimiento de expertos, quienes han producido soluciones exitosas a problemas, a fin que esas soluciones queden a disposición de personas con menos experiencia. Sin embargo, los patrones no proveen siempre las soluciones definitivas, algunas veces, los usuarios de patrones deben tener creatividad para utilizar, instanciar o implementar un patrón (11).

En el ámbito de la Ingeniería de Software actualmente los patrones pueden aplicarse a nivel del: análisis de requisito, el diseño de la arquitectura, el diseño detallado, la interacción con el usuario y el código. Con lo que puede establecerse la siguiente clasificación:

- Patrones de análisis.
- Patrones de arquitectura.
- Patrones de diseño.
- Patrones de Interacción o Interfaz.

1.3.1. Estilos y patrones de Arquitectura.

Shaw y Garlan definen estilo arquitectónico como una familia de sistemas de software en términos de un patrón de organización estructural, que define un vocabulario de componentes y tipos de conectores y un conjunto de restricciones de cómo pueden ser combinadas. Para muchos estilos puede existir uno o más modelos semánticos que especifiquen cómo determinar las propiedades generales del sistema partiendo de las propiedades de sus partes (8).

Buschmann define estilo arquitectónico como una familia de sistemas de software en términos de su organización estructural. Expresa componentes y las relaciones entre estos, con las restricciones de su aplicación y la composición asociada, así como también las reglas para su construcción. Así mismo, se considera como un tipo particular de estructura fundamental para un sistema de software, conjuntamente con un método asociado que especifica cómo construirlo. Éste incluye información acerca de cuándo usar la arquitectura que describe, sus invariantes y especializaciones, así como las consecuencias de su aplicación (8).

David Garlan, Andrew Kompanek, Ralph Melton y Robert Monroe definen el estilo como una entidad consistente en cuatro elementos: primero, un vocabulario de elementos de diseño: componentes y conectores tales como tuberías, filtros, clientes, servidores, parsers, bases de datos, etcétera; segundo, reglas de diseño o restricciones que determinan las composiciones permitidas de esos elementos; tercero, una interpretación semántica que proporciona significados precisos a las composiciones y cuarto, análisis susceptibles de practicarse sobre los sistemas construidos en un estilo, por ejemplo, análisis de

disponibilidad para estilos basados en procesamiento en tiempo real, o detección de abrazos mortales para modelos cliente-servidor (12).

En resumen se puede definir estilo arquitectónico, como un conjunto de reglas de diseño que identifican las clases de componentes y conectores que se pueden utilizar para componer en sistema o subsistema, junto con las restricciones locales o globales de la forma en que la composición se lleva a cabo.

Buschmann propone los patrones arquitectónicos como descripción de un problema particular y recurrente de diseño, que aparece en contextos de diseño específico, y presenta un esquema genérico demostrado con éxito para su solución. El esquema de solución se especifica mediante la descripción de los componentes que la constituyen, sus responsabilidades y desarrollos, así como también la forma como estos colaboran entre sí. Expresan el esquema de organización estructural fundamental para sistemas de software. Provee un conjunto de subsistemas predefinidos, especifica sus responsabilidades e incluye reglas y pautas para la organización de las relaciones entre ellos. Son plantillas para arquitecturas de software concretas, que especifican las propiedades estructurales de una aplicación (con amplitud de todo el sistema) y tienen un impacto en la arquitectura de subsistemas. La selección de un patrón arquitectónico es, por lo tanto, una decisión fundamental de diseño en el desarrollo de un sistema de software (12).

Por tanto los patrones arquitectónicos expresan esquemas de organización estructural fundamentales para los sistemas de software. Estos se ocupan de cuestiones que están más cerca del diseño, la práctica, la implementación, el proceso, el refinamiento, el código. Los patrones arquitectónicos, se han materializado con referencia a lenguajes y paradigmas también específicos de desarrollo.

Con la intención de hacer una comparación clara entre estilo arquitectónico y patrón arquitectónico, la Tabla 4. presenta las diferencias entre estos conceptos.

Estilo Arquitectónico	Patrón Arquitectónico
Sólo describe el esqueleto estructural y general *para aplicaciones*	Existen en varios rangos de escala, comenzando con patrones que definen la estructura básica de *una* aplicación
Son independientes del contexto al que puedan ser aplicados	Partiendo de la definición de *patrón*, requieren de la especificación de un contexto del problema
Cada estilo es independiente de los otros	Depende de patrones más pequeños que contiene, patrones con los que interactúa, o de patrones que lo contengan
Expresan técnicas de diseño desde una perspectiva que es independiente de la situación actual de diseño	Expresa un problema recurrente de diseño muy específico, y presenta una solución para él, desde el punto de vista del contexto en el que se presenta
Son una categorización de sistemas	Son soluciones generales a problemas comunes

Tabla 4. Diferencias entre estilo arquitectónico y patrón arquitectónico. (12).

1.3.1.1. Clasificación de estilos y patrones arquitectónicos

En las definiciones consideradas, y a pesar de algunas excepciones, como las enumeraciones de Taylor y Medvidovic o Mehta y Medvidovic, se percibe una unanimidad que no suele encontrarse con frecuencia en otros territorios de la arquitectura de software. La idea de estilo arquitectónico ha sido, en rigor, uno de los conceptos mejor consensuados de toda la profesión, quizá por el hecho de ser también uno de los más simples. Pero aunque posee un núcleo invariante, las discrepancias comienzan a manifestarse cuando se trata, primero, de enumerar todos los estilos existentes y segundo, de suministrar la articulación matricial de lo que (pensándolo bien) constituye, a nivel arquitectónico, una ambiciosa clasificación de todos los tipos de aplicaciones posibles. Predeciblemente, la segunda disyuntiva demostrará ser más aguda y rebelde que la primera, porque, como lo estableció Georg Cantor, hay más clases de cosas que cosas, aún cuando las cosas sean infinitas. A la larga, las enumeraciones de estilos de grano más fino que se publicaron históricamente contienen todos los ejemplares de las colecciones de grano más grueso, pero nadie ordenó el mundo dos veces de la misma manera (12).

Para dar un ejemplo que se sitúa en el límite del escándalo, el estilo cliente-servidor ha sido clasificado ya sea como variante del estilo basado en datos por Shaw, como estilo de flujo de datos, como arquitectura distribuida por Garlan, como estilo jerárquico, como miembro del estilo de máquinas virtuales, como estilo orientado a objetos por Richard Upchurch, como estilo de llamada-y-retorno por Buckman o como estilo independiente. Mientras las clasificaciones jerárquicas más complejas incluyen como máximo seis clases básicas de estilos, la comunidad de los arquitectos se las ha ingeniado para que un ejemplar pertenezca alternativamente a ocho. Los grandes frameworks (TOGAF, RM-ODP, 4+1, IEEE) homologan los estilos como concepto, pero ninguno se atreve a brindar una taxonomía exhaustiva de referencia. Hay, entonces, más clasificaciones divergentes de estilos que estilos de arquitectura, cosa notable para números tan pequeños, pero susceptible de esperarse en razón de la diversidad de puntos de vista.

En un estudio comparativo de los estilos, Mary Shaw considera los siguientes, mezclando referencias a las mismas entidades a veces en términos de “arquitecturas”, otras invocando “modelos de diseño” (12):

1. Arquitecturas orientadas a objeto.
2. Arquitecturas basadas en estados.
3. Arquitecturas de flujo de datos: Arquitecturas de control de realimentación.
4. Arquitecturas de tiempo real.
5. Modelo de diseño de descomposición funcional.
6. Modelo de diseño orientado por eventos.
7. Modelo de diseño de control de procesos.
8. Modelo de diseño de tabla de decisión.
9. Modelo de diseño de estructura de datos.

El mismo año, Mary Shaw, junto con David Garlan [Garlan 1994], propone una taxonomía diferente, en la que se entremezclan lo que antes llamaba “arquitecturas” con los “modelos de diseño” (12):

1. Tubería-filtros.
2. Organización de abstracción de datos y orientación a objetos.
3. Invocación implícita, basada en eventos.
4. Sistemas en capas.

5. Repositorios.
6. Intérpretes orientados por tablas.
7. Procesos distribuidos, ya sea en función de la topología (anillo, estrella, etc.) o de los protocolos entre procesos (por ejemplo, algoritmo de pulsación o heartbeat). Una forma particular de proceso distribuido es, por ejemplo, la arquitectura cliente-servidor.
8. Organizaciones programa principal / subrutina.
9. Arquitecturas de software específicas de dominio.
10. Sistemas de transición de estado.
11. Sistemas de procesos de control.
12. Estilos heterogéneos.

De particular interés es el catálogo de “patrones arquitectónicos”, que es como el influyente grupo de Buschmann denomina a entidades que, con un empaquetado un poco distinto, no son otra cosa que los estilos. Efectivamente, esos patrones “expresan esquemas de organización estructural fundamentales para los sistemas de software. Proporcionan un conjunto de subsistemas predefinidos, especifican sus responsabilidades e incluyen guías y lineamientos para organizar las relaciones entre ellos”. En la hoy familiar clasificación de POSA, Buschmann, Meunier, Rohnert, Sommerlad y Stal enumeran estos patrones (12):

1. Del fango a la estructura:

- Capas.
- Tubería-filtros.
- Pizarra.

2. Sistemas distribuidos:

- Broker (p. ej. CORBA, DCOM, la World Wide Web).

3. Sistemas interactivos:

- Modelo-Vista-Controlador (MVC).
- Presentation-Abstraction-Control.

4. Sistemas adaptables:

- Reflection (meta nivel que hace al software consciente de sí mismo).
- Microkernel (núcleo de funcionalidad mínima).

En Software Architecture in Practice, un texto fundamental de Bass, Clements y Kazman se proporciona una sistematización de clases de estilo en cinco grupos:

1. Flujo de datos (movimiento de datos, sin control del receptor de lo que viene “corriente arriba”):

- Proceso secuencial por lotes.
- Red de flujo de datos.
- Tubería-filtros.

2. Llamado y retorno (estilo dominado por orden de computación, usualmente con un solo thread de control):

- Programa principal / Subrutinas.
- Tipos de dato abstracto.
- Objetos
- Cliente-servidor basado en llamadas.
- Sistemas en capas.

3. Componentes independientes (dominado por patrones de comunicación entre procesos independientes, casi siempre concurrentes):

- Sistemas orientados por eventos.

- Procesos de comunicación.

4. Centrados en datos (dominado por un almacenamiento central complejo, manipulado por computaciones independientes):

- Repositorio.
- Pizarra.

5. Máquina virtual (caracterizado por la traducción de una instrucción en alguna otra):

- Intérprete.

Es interesante el hecho que se proporcionen sólo cinco clases abarcativas. Mientras que en los inicios de la arquitectura de software se alentaba la idea de que todas las estructuras posibles en diseño de software serían susceptibles de reducirse a una media docena de estilos básicos, lo que en realidad sucedió fue que en los comienzos del siglo XXI se alcanza una fase barroca y las enumeraciones de estilos se tornan más detalladas y exhaustivas. Considerando solamente los estilos que contemplan alguna forma de distribución o topología de red, Roy Fielding establece la siguiente taxonomía (12):

1. Estilos de flujo de datos:
1.1. Tubería-filtros.
1.2. Tubería-filtros uniforme.

2. Estilos de replicación:
2.1. Repositorio replicado.
2.2. Cache.

3. Estilos jerárquicos:
3.1. Cliente-servidor.
3.2. Sistemas en capas y Cliente-servidor en capas.
3.3. Cliente-Servidor sin estado.

3.4. Cliente-servidor con caché en cliente.
3.5. Cliente-servidor con caché en cliente y servidor sin estado.
3.6. Sesión remota.
3.7. Acceso a datos remoto.

4. Estilos de código móvil:
4.1. Máquina virtual.
4.2. Evaluación remota.
4.3. Código a demanda.
4.4. Código a demanda en capas.
4.5. Agente móvil.

5. Estilos peer-to-peer:
5.1. Integración basada en eventos.
5.2. C2.
5.3. Objetos distribuidos.
5.4. Objetos distribuidos brokered.

6. Transferencia de estado representacional (REST).

Cuando las clasificaciones de estilos se tornan copiosas, daría la impresión que algunos subestilos se introducen en el cuadro por ser combinatoriamente posibles, y no tanto porque existan importantes implementaciones de referencia, o porque sea técnicamente necesario. En un proyecto que se ha desarrollado en China hacia el año 2001, se ha podido encontrar una clasificación que si bien no pretende ser totalizadora, agrupa los estilos de manera peculiar, agregando una clase no prevista por Bass, Clements y Kazman, llamando a las restantes de la misma manera, pero practicando enroques posicionales de algunos ejemplares (12):

1. Sistemas de flujo de datos, incluyendo:
- Secuencial por lotes.
- Tubería y filtro.

2. Sistemas de invocación y retorno (call-and-return), incluyendo:
- Programa principal y sub-rutina.
- Sistemas orientados a objeto.
- Niveles jerárquicos.

3. Componentes independientes, incluyendo:
- Procesos comunicantes.
- Sistemas basados en eventos.

4. Máquinas virtuales, incluyendo:
- Intérpretes.
- Sistemas basados en reglas.
- Cliente-servidor.

5. Sistemas centrados en datos, incluyendo:
- Bases de datos.
- Sistemas de hipertexto.
- Pizarras.

6. Paradigmas de procesos de control.

En un documento anónimo compilado por Richard Upchurch, del Departamento de Ciencias de la Computación e Información de la Universidad de Massachusetts en Dartmouth se proporciona una "lista de posibles estilos arquitectónicos", incluyendo (12):

1. Programa principal y subrutinas.
2. Estilos jerárquicos.
3. Orientado a objetos (cliente-servidor).
4. Procesos de comunicación.
5. Tubería y filtros.

6. Intérpretes.
7. Sistemas de bases de datos transaccionales.
8. Sistemas de pizarra.
9. Software bus.
10. Sistemas expertos.
11. Paso de mensajes.
12. Amo-esclavo.
13. Asincrónico / sincrónico paralelo.
14. Sistemas de tiempo real.
15. Arquitecturas específicas de dominio.
16. Sistemas en red, distribuidos.
17. Arquitecturas heterogéneas.

Esta lista no realiza ningún tipo de diferenciación ni clasificación entre estilos y patrones. De hecho, puede que denote la transición hacia las listas enciclopédicas, que por lo común suelen ser poco cuidadosas en cuanto a mantener en claro los criterios de articulación de la taxonomía. En 2001, David Garlan, uno de los fundadores del campo, proporciona una lista más articulada (12):

1. Flujo de datos:
- Secuencial en lotes.
- Red de flujo de datos (tuberías y filtros).
- Bucle de control cerrado.

2. Llamada y Retorno:

- Programa principal / subrutinas.
- Ocultamiento de información (ADT, objeto, cliente/servidor elemental).

3. Procesos interactivos:
- Procesos comunicantes.
- Sistemas de eventos (invocación implícita, eventos puros).

4. Repositorio Orientado a Datos:
- Bases de datos transaccionales (cliente/servidor genuino).
- Pizarra.
- Compilador moderno.

5. Datos Compartidos:
- Documentos compuestos.
- Hipertexto.
- Fortran COMMON.
- Procesos LW

6. Jerárquicos:
- En capas (intérpretes).

Al lado de las propuestas meta-estilísticas que se han visto, puede apreciarse que toda vez que surge la pregunta de qué hacer con los estilos, de inmediato aparece una respuesta que apunta para el lado de las prácticas y los patrones; no obstante aunque todavía no se ha consensuado una taxonomía unificada de estilos, ellos ayudarán, sin duda, a establecer y organizar los contextos en que se implementen los patrones (12).

Los estilos y patrones que se mencionarán a continuación no aspiran a ser todos los que se han propuesto, sino apenas los más representativos y vigentes, y que se tomarán como guía para esta investigación. Como se ha visto, la agrupación de estilos y patrones es susceptible de realizarse de múltiples formas, conforme a los criterios que se apliquen en la constitución de los ejemplares. No se ha de rendir cuentas aquí por la congruencia de la clasificación (nadie ha hecho lo propio con las suyas), porque cada vez que se la ha revisado en modo outline, se cedió a la tentación de cambiar su estructura, la cual en la opinión de los autores de esta investigación, seguirá sufriendo metamorfosis con el tiempo:

1. Estilo de Flujos de Datos:

- Tuberías y Filtros (Pipes and Filters).

2. Estilo de Llamada y Retorno:
   - Arquitectura Orientada a Objetos.
   - Arquitectura en Capas (Arquitectura en 3 capas).
   - Arquitectura Modelo-Vista-Controlador.
   - Arquitecturas basadas en Componentes.

3. Estilo Centrado en Datos:
   - Arquitectura de Pizarra o Repositorios.

4. Estilos Peer-to-Peer:
   - Arquitecturas basadas en eventos (Invocación Implícita).
   - Arquitectura Cliente/Servidor.
   - Arquitecturas Orientadas a Servicios.

1.4. Métodos basados en la Arquitectura

Cuanto más temprano se encuentre un problema en un proyecto de software, mejor. El costo de arreglar un error durante las fases de requerimientos o diseño, es mucho menor al costo de arreglar ese mismo error en la fase de verificación. Dado que la arquitectura es un producto temprano de la fase de diseño, esta tiene un profundo efecto en el sistema y en el proyecto. Es mejor cambiar la arquitectura antes que otros artefactos, que están basados en ella, se estabilicen (3).

Realizar una evaluación de la arquitectura es la manera más económica de evitar desastres. Existe toda una inmensa disciplina especializada en evaluación de arquitecturas, dividida en escuelas y metodologías que han llegado a ser tantas que debieron clasificarse en tipos (evaluación basada en escenarios, en cuestionarios, en listas de verificación, en simulación, en métricas); pero esta técnica está lejos de estar integrada a conceptos descriptivos y abstractos como los que dominan el campo estructural, y sólo de tarde en tarde se vinculan los métodos de evaluación con las variables de diseño teorizadas como estilos o puestas en práctica como patrones (12)

En la literatura existen varios métodos de evaluación para verificar desde múltiples atributos de calidad hasta algunos en específico. Ejemplos de estos métodos de evaluación son ATAM y SAAM, que serán tratados en detalle a continuación.

1.4.1. ATAM

El Architecture Tradeoff Analysis Method (ATAM) obtiene su nombre no solo porque nos dice cuan bien una arquitectura particular satisface las metas de calidad, sino que también provee ideas de cómo esas metas de calidad interactúan entre ellas, como realizan concesiones mutuas entre ellas.

Tener un método estructurado, permite hacer el análisis repetible y ayuda a asegurar que las preguntas acerca de una arquitectura serán contestadas en forma temprana, cuando es relativamente económico corregir problemas (3).

El ATAM también puede ser utilizado para analizar sistemas legados. Esta necesidad nace cuando el sistema legado necesita ser modificado, integrado con otro sistema, entre otras necesidades. Aplicar el ATAM incrementa el entendimiento de los atributos de calidad del sistema legado.

La parte principal del ATAM consiste de nueve pasos. Estos pasos se dividen cuatro grupos (3):

- Presentación, donde la información es intercambiada.
- Investigación y análisis, donde se valoran los atributos claves de calidad requeridos, uno a uno con las propuestas arquitectónicas.
- Pruebas, donde se revisan los resultados obtenidos contra las necesidades relevantes de los stakeholders.
- Informes, donde se presentan los resultados del ATAM.

Los pasos del ATAM se distribuyen en el tiempo. El ATAM esta compuesto por cuatro fases, que corresponden a los intervalos de tiempo de mayor actividad.

La fase 0 es de preparación, en la cual el equipo de evaluación es creado y se forma una sociedad entre la organización evaluadora y la organización cuya arquitectura será evaluada. Las fases 1 y 2, las fases de evaluación del ATAM, comprenden los nueve pasos vistos hasta ahora. La fase 1 esta centrada en la arquitectura, y se concentra en obtener y analizar la información arquitectónica. La fase 2 esta centrada en los stakeholders, y se concentra en obtener los puntos de vista de los de los stakeholders y verificar los resultados de la fase 1. En la fase 3 se realiza el reporte final, y se continua con las acciones (si hay) planeadas, y la organización evaluadora actualiza sus archivos e incorpora la experiencia adquirida.

1.4.2. SAAM

Software Architecture Analysis Method (SAAM) fue el primer método de evaluación basado en escenarios que surgió. El foco de este método es la modificabilidad.

Los creadores de SAAM idearon un método para evaluar, por medio de escenarios, los diferentes atributos de calidad que las arquitecturas de software demandaban. En la práctica SAAM ha demostrado ser útil para evaluar muchos atributos de calidad rápidamente, como portabilidad, modificabilidad, extensibilidad, integrabilidad, así como el cubrimiento funcional que tiene la arquitectura sobre los requerimientos del sistema. El método también puede ser utilizado para evaluar aspectos más ligados con la arquitectura como performance o confiabilidad. Sin embargo, el método ATAM explora estos aspectos con más profundidad (3).

SAAM puede ser utilizado para evaluar una o múltiples arquitecturas. Si se comparan dos o más se culmina el análisis con una tabla indicando las fortalezas y debilidades de cada una en cada escenario. Si se evalúa una sola, se culmina con un reporte señalando los componentes computacionales donde la arquitectura no alcanza el nivel requerido. En ningún caso se emite un valor absoluto acerca de la “calidad arquitectónica”.

SAAM utiliza el agrupamiento de escenarios como criterio para evaluar la arquitectura. Esto significa que si se agrupa un conjunto de escenarios por ser similares y luego se observa que son equivalentes, la agrupación ha sido exitosa, porque significa que la funcionalidad del sistema ha sido modularizada

adecuadamente. Por el contrario, si la agrupación de estos escenarios similares afecta diferentes componentes (no son equivalentes), la arquitectura posiblemente debe ser corregida.

El método de evaluación consiste en seis pasos (3):

Paso 1. Desarrollo de escenarios: La meta principal es capturar las principales actividades que el sistema debe soportar.
Paso 2. Descripción de la Arquitectura: En este paso se presentan las arquitecturas candidatas.
Paso 3. Clasificación de escenarios: En este paso debemos clasificar los escenarios.
Paso 4. Evaluación de escenarios: Se deben listar los cambios necesarios en la arquitectura para soportarlo, y el costo de llevarlos a cabo debe ser estimado.
Paso 5. Interacción de escenarios: Debemos determinar las interacciones de escenarios sobre cada componente de cada arquitectura.
Paso 6. Evaluación general: Un peso es asignado a cada escenario en términos de su influencia para que el sistema sea exitoso. El peso puede ser elegido de acuerdo a los objetivos del negocio, costos, riesgos, etc.

Las principales fortalezas del método son (3):

- Los interesados entienden en profundidad la arquitectura o arquitecturas que se analizarán.
- En algunos casos, luego de la sesión de evaluación de SAAM, la documentación relacionada con arquitectura de software es mejorada.
- Aumenta la comunicación entre los interesados.
- Con respecto a la modificabilidad, podemos decir que, los escenarios correspondientes a futuros cambios pueden ser evaluados en la arquitectura o arquitecturas candidatas, logrando estudiar e identificar las áreas potencialmente complejas. El esfuerzo y costo de los cambios mencionados pueden ser estimados con anticipación.

Las debilidades del método son las siguientes (3):

- El proceso de generación de escenarios está basado en la visión de los interesados. En general, existe un muy pequeño esfuerzo para imaginar los escenarios indirectos.
- SAAM no provee una métrica clara sobre la calidad de la arquitectura analizada.
- La descripción de la arquitectura puede resultar confusa a la hora de realizar comparaciones si no se adoptan una notación y un nivel de detalle comunes.
- El equipo de evaluación confía en la experiencia de los arquitectos para proponer arquitecturas candidatas. En general, este equipo no tiene conocimiento sobre el conjunto completo de requerimientos y los objetivos del negocio.

1.5. Herramientas para el diseño de la arquitectura.

El Instituto de Ingeniería de Software (SEI, por sus siglas en inglés) ha estado trabajando durante varios años en el área de métodos, herramientas, y tecnologías para apoyar el desarrollo de arquitecturas de software de calidad superior. Se han estado proporcionando métodos para ayudar a arquitectos a recoger los requisitos arquitectónicamente significativos, diseñar la arquitectura usando patrones y tácticas, evaluar y documentar la arquitectura y reconstruirla desde el código.

Muchos de los sistemas de software actuales toman la arquitectura como tema central o base conceptual. Esta se diseña y mantiene como un recurso vital para el proyecto y la organización que lo construye. Para mantener la veracidad de esta afirmación, se han diseñado herramientas que ayudarán al arquitecto a identificar y guardar registros, preocupaciones y metas del negocio, a guardar y rastrear los requisitos arquitectónicamente significativos y los atributos de calidad, usando los escenarios de atributos de calidad. Estas herramientas para el diseño de la arquitectura también ayudan a rastrear los requisitos para diseñar decisiones que los satisfagan.

1.5.1. Asistente de software para el diseño de la arquitectura: ArchE.

Arquitectura de Expertos, ArchE (por sus siglas en inglés), es un asistente elaborado por el SEI para el diseño de la arquitectura. Es una herramienta diseñada para proporcionar la información útil sobre una arquitectura actual al arquitecto, para encontrar una buena solución a un problema dado.

ArchE usa tres tipos diferentes de entrada: los atributos de calidad para el sistema que se está diseñando, las características que el sistema debe soportar y cualquier restricción de diseño como la utilización de un diseño de legado (13).

ArchE resume un conjunto de responsabilidades que el sistema computará. Los atributos de calidad también implican ciertas responsabilidades. Construye una representación de las responsabilidades y las dependencias entre ellas. Un tipo de dependencia es que una responsabilidad puede necesitar ser computada anterior a otra o que una responsabilidad puede descomponerse en varias.

ArchE usa la información adquirida y los escenarios de atributos de calidad para crear una arquitectura inicial que consiste en módulos, unidades de concurrencia y sus relaciones con las responsabilidades. Entonces muestra la arquitectura inicial, para ver cuan bien satisface los atributos de calidad, y una serie de sugerencias para las mejoras a la arquitectura (basadas en el uso de tácticas arquitectónicas). El arquitecto selecciona una opción, ArchE lo aplica a la arquitectura y calcula cuan bien la arquitectura revisada satisface los atributos de calidad (14).

Una vez seleccionada la arquitectura, ArchE manda una descripción de esta en un archivo XML que puede importarse en Rational Rose u otras herramientas de diseño más detalladas.

ArchE sugiere opciones de diseño alternativas que beneficiarán al diseño arquitectónico desde el punto de vista de los atributos de calidad y permitirá a diseñadores y arquitectos explorar estas alternativas teniendo en cuenta su impacto en los mismos. No obstante ArchE es actualmente un prototipo experimental y el SEI aún no lo distribuye.

1.6. Conclusiones

El desarrollo de software trae aparejado el uso de técnicas, métodos, herramientas y modelos que facilitan a las organizaciones encargadas de las tecnologías de la información generar productos que cumplan las expectativas del cliente e incluso las rebasen, obteniendo de esta manera productos con calidad. Si durante el desarrollo del software, al final de cada fase se obtiene un producto de calidad, entonces, el

producto final tendrá la calidad requerida. La arquitectura de un software es uno de los artefactos resultantes más importantes en este proceso. Por tanto, se hace necesario garantizar una buena arquitectura basada en el cumplimiento de los atributos de calidad. El estudio de varias herramientas para la evaluación de distintas arquitecturas, de las clasificaciones de los atributos de calidad y de los estilos y patrones arquitectónicos, así como la interrelación entre ellos, permitió conocer que la información existente sobre cuáles estilos o patrones serían los más indicados a usar en dependencia de los atributos de calidad especificados por el cliente, no es suficiente y no resuelve el dilema presentado a la hora de seleccionar una arquitectura candidata, evidenciando la necesidad e importancia de la creación de un modelo que ayude a los arquitectos en esta toma de decisiones.

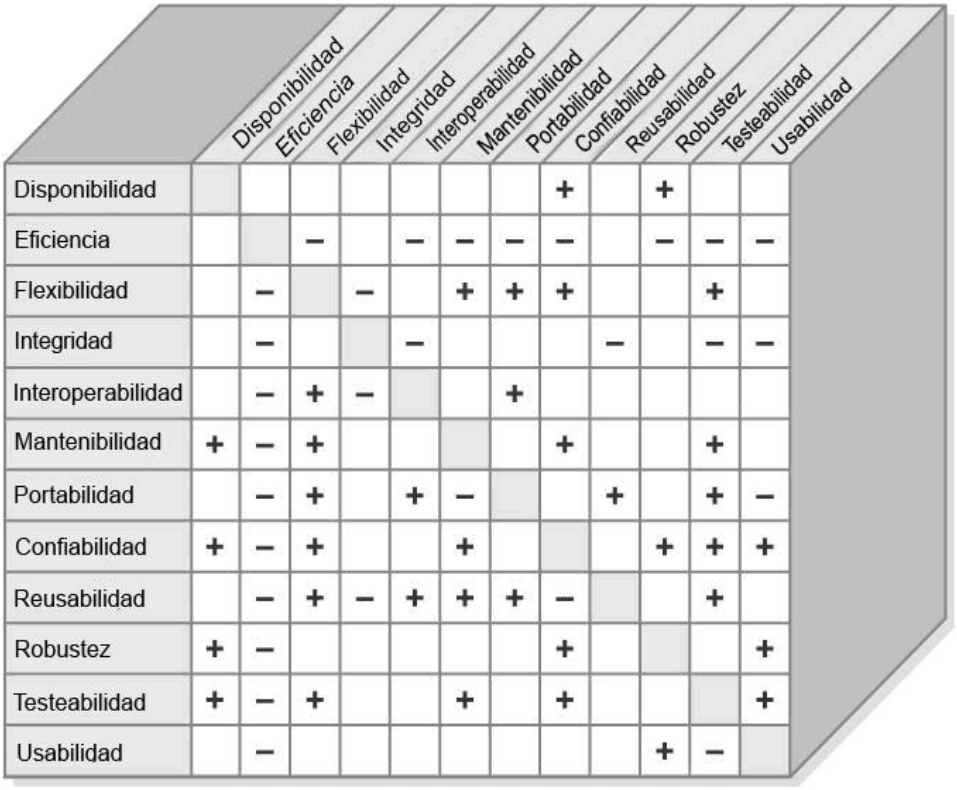
CAPÍTULO 2: DEFINICIÓN DEL MODELO

En este capítulo se realizará el modelo para la ayuda en la toma de decisiones relativas a estilos y patrones arquitectónicos. Este modelo se basará en tres pasos fundamentales. Primero, el análisis de la relación e influencia de un atributo de calidad sobre otros; segundo, la definición de medidas para ponderar los atributos de calidad, y así conocer cuales son los más importantes a la hora de proponer una arquitectura, y por último, la exposición de una serie de reglas para seleccionar un patrón o estilo arquitectónico en dependencia de los atributos de calidad ya ponderados en el paso anterior.

2.1. Análisis de las relaciones e influencias entre lo atributos de calidad.

Diferentes partes del producto necesitan diferentes combinaciones de atributos de calidad. La eficiencia puede ser crítica para ciertos componentes, como se verá más adelante, mientras la usabilidad es primordial para otros. Las características de calidad que se aplican a todo el producto se diferencian de aquellas que son específicas de ciertos componentes, ciertas clases de usuarios, o de uso, en ciertas situaciones particulares (2).

Algunas combinaciones de atributos llevan a tomar determinadas decisiones arquitectónicas. Usuarios y desarrolladores deben decidir cuáles atributos son más importantes que otros, respetando esas prioridades cuando se toman decisiones. La Figura 4. muestra algunas interrelaciones típicas entre los atributos de calidad.



	Disponibilidad	Eficiencia	Flexibilidad	Integridad	Interoperabilidad	Mantenibilidad	Portabilidad	Confiabilidad	Reusabilidad	Robustez	Testeabilidad	Usabilidad
Disponibilidad							+		+			
Eficiencia			–		–	–	–	–		–	–	–
Flexibilidad		–		–		+	+	+			+	
Integridad		–			–				–		–	–
Interoperabilidad		–	+	–			+					
Mantenibilidad	+	–	+					+			+	
Portabilidad		–	+		+	–			+		+	–
Confiabilidad	+	–	+			+				+	+	+
Reusabilidad		–	+	–	+	+	+	–			+	
Robustez	+	–						+				+
Testeabilidad	+	–	+			+		+				+
Usabilidad		–								+	–	

Figura 4. Relaciones positivas y negativas entre los atributos de calidad seleccionados. (2).

El signo negativo en una celda significa que incrementar el atributo en esa fila afecta negativamente el atributo en la columna. Lo contrario sucede con el signo positivo. Aumentar el atributo de la fila correspondiente, tendría entonces un efecto positivo sobre el atributo de la columna. Una celda en blanco indica que el atributo de la fila tiene poco impacto en el atributo de la columna. Por ejemplo, los enfoques de diseño que aumentan el componente de software portabilidad, hacen el software más flexible, más fácil de conectarse a otros componentes de software, más fácil de reutilizar y más fácil de probar. La eficiencia tiene un impacto negativo en la mayoría de los otros atributos. Si se escribe el más rápido y denso código que se pueda, usando trucos de codificación y confiando en la ejecución de los efectos secundarios, es muy probable que sea difícil de mantener y mejorar, además no será fácil de portar a otras plataformas. Del mismo modo, los sistemas que optimizan la facilidad de uso o que están diseñados para ser flexibles, reutilizables e interoperables con otros componentes de software o de hardware, a menudo penan de incurrir en el rendimiento. Se deben balancear las posibles reducciones de rendimiento contra los

beneficios de la solución propuesta para asegurarse de que se están asumiendo compromisos razonables (2).

La matriz de la Figura 4. no es simétrica, ya que el efecto que tiene el incremento del atributo A sobre el atributo B, no es necesariamente el mismo efecto que el incremento de B tendrá sobre A. Por ejemplo, la figura muestra que incrementar la flexibilidad no tendría efecto a destacar sobre la reusabilidad. En caso contrario, aumentando reusabilidad, se beneficia la flexibilidad del sistema en cuestión.

Para alcanzar el equilibrio óptimo de las características del producto, se debe identificar, especificar, y dar prioridad a los atributos de calidad pertinentes durante la obtención de requisitos. Para definir los atributos de calidad importantes para un proyecto, sería ventajoso tener en cuenta estas contradicciones reflejadas en la Figura 4. para evitar hacer compromisos con objetivos contradictorios. Por ejemplo: No espere maximizar la usabilidad si el software debe correr en múltiples plataformas (portabilidad).

Es difícil de probar por completo la integridad de los requisitos de los sistemas de alta seguridad. Reutilizar componentes genéricos o interconexiones con otras aplicaciones podrían poner en peligro los mecanismos de seguridad.

Una aplicación que cumpla con los requisitos funcionales y esté disponible todo el tiempo cada vez que sea necesario, va a ser confiable para los clientes que la utilizan. Pero si además, el código de esta aplicación es poco robusto, entonces traerá consigo que la integridad de la misma disminuya y por tanto estará vulnerable a cualquier ataque, disminuyendo sus probabilidades de disponibilidad.

Como es usual reprimiendo las expectativas del sistema o definiendo las contradicciones entre los atributos, hace que sea imposible para los desarrolladores satisfacer plenamente estos últimos.

2.2. Método para ponderación de atributos de calidad.

El primer paso para ponderar los atributos de calidad, sería asignarle a cada uno de sus parámetros correspondientes pesos relativos. Estos dependerían de las necesidades del cliente o los desarrolladores, según el atributo y sistema que se trate. La asignación de estos pesos relativos permitirá a los arquitectos

centrar sus esfuerzos en los atributos de calidad más importantes, o incluso, en sus características más relevantes.

Los pesos pueden expresarse de la forma alto / medio / bajo. A la hora de calcular, se utilizaría la escala de tipo cardinal en el rango de 1-9, donde de 1-3 corresponda a la categoría baja, mediana pertenezca de 4-6 y de 7-9 se refiera a peso de tipo alto.

A continuación se muestran los parámetros establecidos para cada atributo de calidad. Cada uno de estos tendrá un peso específico según la escala utilizada. El peso relativo del atributo de calidad se halla calculando el promedio de los pesos de sus parámetros respectivos.

Eficiencia:

- Tiempo estimado para completar una tarea.
- Tiempo de respuesta del sistema.
- Frecuencia de errores provocados por el usuario.
- El sistema debe recibir los servicios de sus componentes en el transcurso de un tiempo indicado.
- El sistema controla que ningún componente se quede sin recursos cuando los necesita.
- Cantidad de tareas que pueden ser efectuadas de forma exitosa por el sistema en un período de tiempo dado.

Mantenibilidad:

- Es posible verificar el estado de los componentes del sistema.
- El sistema debe facilitar la sustitución/adaptación de un componente.
- Tiempo empleado desde la petición del usuario de soporte y mantenimiento de sistema, hasta la resolución del problema.
- El sistema debe tener un mecanismo para actualizarse obteniendo las mejoras de su nueva versión.
- Mitigar las consecuencias causadas por los efectos del mantenimiento.
- Seguimiento y soporte posterior a la implantación.

Portabilidad:

- Los componentes pueden instalarse en distintos entornos satisfactoriamente.
- Los componentes manejan adecuadamente recursos compartidos del sistema.
- Facilidad del sistema para adherirse a los estándares, convenciones, etc.
- El sistema brinda facilidades para adaptarse a distintas situaciones.
- El sistema es instalable de forma sencilla (consumo de tiempo).
- Nivel de complejidad de configuración.

Fiabilidad:

- Los componentes respetan un estándar de fiabilidad.
- La comunicación entre los componentes no viola los estándares de fiabilidad.
- Los componentes del sistema manejan entradas de datos incorrectas.
- -Todas las operaciones ejecutadas por los componentes se realizan correctamente bajo condiciones adversas.
- Los componentes del sistema no fallan bajo ciertas condiciones especificadas.
- Ante problemas con el ambiente, un subconjunto determinado de los componentes puede continuar prestando sus servicios.

Disponibilidad

- El sistema posee salvas de respaldo.
- El sistema posee mecanismos para tomar medidas por su cuenta, en caso de eventos anormales (reiniciarse, etc.)
- El sistema maneja fallas bajo control para evitar fallas críticas y serias (destrucción de base de datos, sistema colapsado e inoperable).
- Tiempo* que puede estar disponible el sistema, sin ataques de intrusos, ni virus al sistema, sólo ejecutando todas las operaciones del usuario.
- Tiempo* de reparación del sistema cada vez que no está disponible. (Tener en cuenta el nivel de severidad de la rotura, destrucción de base de datos, pérdida de múltiples transacciones de una sola transacción destrucción temporal de información).

* En este caso se refiere solo al tiempo que brinda el software en el desempeño de las tareas, se excluye el mantenimiento hecho por el trabajo humano.

- Tiempo* que el sistema permanece no disponible cuando una falla ocurre. (Tener en cuenta el nivel de severidad de la rotura, destrucción de base de datos, pérdida de múltiples transacciones de una sola transacción destrucción temporal de información).

Flexibilidad

- El sistema brinda facilidades para modificarlo.
- El sistema posee mecanismos para adaptarse a nuevas situaciones.
- El sistema presenta facilidades para la migración entre sistemas operativos.
- El sistema da posibilidades para utilizar información proveniente de sistemas legados, en caso de ser necesario.
- El sistema permite parametrizar su funcionamiento basado en características de rendimiento.
- De todos los módulos que el sistema posee permite eliminar módulos, sin afectar el comportamiento de los otros.

Integridad:

- El sistema detecta la actuación de un intruso e impide acceso a los componentes que manejen información sensible
- El sistema asegura que los componentes no pierdan datos ante un ataque (interno o externo).
- El sistema necesita declaración de login con tipos de acceso, y contraseñas con mecanismos de encriptación adecuados.
- El sistema posee algún sistema de seguridad contra virus.
- El sistema tiene mecanismos para detectar diferentes tipos de operaciones ilegales (ataques, etc.) contra el acceso ilícito al sistema.
- El sistema posee módulos para manejar el impacto de eventos de corrupción de datos (reproducción y recuperación de los datos).

Interoperabilidad:

- El sistema recurre frecuentemente a la transferencia de datos a otros software.

* En este caso se refiere solo al tiempo que brinda el software en el desempeño de las tareas, se excluye el mantenimiento hecho por el trabajo humano.

- El intercambio entre los componentes lo realiza manualmente por el usuario o automáticamente por el sistema.
- El sistema funciona correctamente con otro sistema aún cuando el usuario o el mismo software han fallado previamente a la hora de intercambiar información con él, después de sucedido esto, interactúa bien con otro sistema completamente diferente.
- La salida del formato de datos de la interfaz de su sistema coinciden con las regulaciones, estándares y convenciones para la comunicación con otros software.
- Errores por falla de transmisión de datos y advertencias entre sistemas.
- Grado de sincronización entre componentes de software.

Robustez:

- El sistema funciona "razonablemente" bien, incluso ante situaciones no anticipadas en los requisitos funcionales.
- El sistema tiene mecanismos para priorizar ciertas funciones ante situaciones no esperadas.
- El sistema posee mecanismos para la manipulación y el tratamiento de excepciones.
- El sistema posee mecanismos para evitar, comprobar, detectar y eliminar fallas.
- El sistema provee de estrategias para su corrección y recuperación ante errores.
- El sistema proporciona vías para manejar defectos que puedan surgir en su funcionamiento.

Usabilidad:

- Las interfaces del software son consistentes y predecibles, con el mercado que representan y para todos los tipos de usuarios.
- Las interfaces son soportadas por cualquier (navegador, en caso de ser Web), o por el hardware que especificó el cliente para las PC de los usuarios, incluso aun cando las PCs estén ejecutando otros procesos.
- El sistema posee tutoriales, demostraciones, ayuda, a las cuales el usuario puede acceder en cualquier momento.
- Facilidad para que el usuario recupere datos de entrada, información por errores cometidos, y reintentar realizar tareas.

- El sistema brinda facilidad a los usuarios para entender, memorizar y recordar sus mensajes. Existe algún mensaje que pueda causar el retraso del usuario entendiéndolo antes de realizar la próxima acción.
- El sistema provee facilidad para personalizar de forma sencilla funciones a conveniencia del usuario, incluyendo elementos de apariencia.

Reusabilidad:

- Existe independencia entre los componentes de su aplicación.
- Empleo de partes de productos (componentes) software obtenidos en proyectos anteriores.
- Experiencia del equipo de proyecto desarrollando aplicaciones similares.
- Reutilización de parte del software por otros sistemas.
- Desarrollo de la aplicación con el objetivo de ser adaptada a otros menesteres, modificando parte del código en sí.
- Desarrollo de la aplicación parametrizada, de modo que modificando algunos parámetros se utilizará por otros usuarios.

Testeablidad:

- La aplicación posibilita la simplificación de pruebas por usuarios y mantenedores al sistema, haya sido modificado o no.
- Tiempo empleado en pruebas para chequear si la falla reportada fue resuelta o no.
- El sistema permite la testeabilidad de sus componentes de forma independiente.
- El sistema posee herramientas para realizar los test, estas herramientas son reutilizables.
- Las modificaciones al sistema son fáciles de validar.
- Nivel de dependencia entre el buen desempeño del proceso de testeo y el hardware.

Aunque existen muchas consideraciones que habría que tener en cuenta a la hora de darle valores a los atributos de calidad, fuertemente se recomienda, en lo posible, tener en cuenta los diferentes aspectos que se plantean a continuación:

- Rendimiento del CPU de la(s) computadora(s) donde se ejecuta el software, y en caso de existir clientes que usen estos dispositivos, considerarlos también.

- Rendimiento de la red operativa y la comunicación.
- Cargas de estrés del usuario final.
- Sistema funcionando 24 horas, 7 días a la semana (procesamiento periódico)
- La utilización de recursos del sistema.
- Los niveles de interrupción.
- Producción del usuario final bajo presión.
- Distracciones del usuario final.
- Nivel de habilidad del usuario final.
- Usuario finales especialistas o técnicos.
- Grupo limitado de usuarios o usuarios públicos.

2.3. Reglas para la toma de las decisiones referentes a estilos y patrones de arquitectura.

Anteriormente algunos autores habían planteado que ciertos atributos de calidad eran afectados por determinados patrones arquitectónicos. Con base en sus afirmaciones Shaw, Garlan y Bosch; acerca de cómo los estilos arquitectónicos propician determinados atributos de calidad, en la Tabla 5. fueron traducidos los mismos en algunas de las características de calidad que se definieron anteriormente. De este modo, la Tabla 5. resume algunas de las características de calidad que son potenciadas y castigadas por algunos de los patrones arquitectónicos mencionados (15).

Patrón	Características Potenciadas	Características Castigadas
Orientado a Objeto	Mantenibilidad (en general), Confiabilidad (en general), Integridad, Reusabilidad (caso especial de Mantenibilidad)	Eficiencia (en general)
Pipes and Filters	Mantenibilidad (en general), Integridad, Reusabilidad (caso especial de Mantenibilidad)	Eficiencia (en general), Confiabilidad, Mantenibilidad/ Estabilidad

Capas	Mantenibilidad (en general, incluyendo la Estabilidad), Integridad	Eficiencia (en general), Confiabilidad (en general, especialmente Madurez y Tolerancia a Fallas) ,

Tabla 5. Características potenciadas y castigadas por algunos de los patrones arquitectónicos mencionados. (15)

No obstante, a continuación se darán un conjunto de reglas a seguir para conocer cuáles patrones y estilos arquitectónicos se deben aplicar en dependencia de los atributos de calidad priorizados por el arquitecto de software y el cliente:

Flexibilidad

1. Si el sistema es distribuido, puede usarse arquitectura basada en eventos, ya que puede agregarse y/o reemplazarse un componente registrándolo para los eventos del sistema, pues no existe acoplamiento entre componentes.

2. Si el problema debe responder a nuevas situaciones en cortos períodos de tiempo, no es recomendable usar arquitectura orientada a objetos, pues es costoso acomodarse a situaciones u objetos nuevos (Se refiere específicamente a un entorno como el que se puede dar con Inteligencia Artificial).

3. Si el sistema se encuentra dividido en etapas o módulos y requiere de cambios en un módulo, utilizando lógicas de control de flujo, no usar patrón filtros y tuberías pues no maneja con demasiada eficiencia construcciones condicionales, bucles, etc. Agregar un paso suplementario afecta la performance de cada ejecución de la tubería. Por otro lado, considerar que si se usa arquitectura orientada a objetos, ellos para interactuar deben conocer sus identidades, así que cuando se modifica un objeto, debe modificar también todos los objetos que lo invocan.

4. Si se desea agregar una etapa, módulo o servicio al sistema, usar patrón filtros y tuberías dado que los filtros no dependen de otros filtros, cada uno de ellos puede ser reemplazado por uno o más filtros, puede ser eliminado, u otros filtros pueden ser agregados.

5. Si el sistema requiere de un constante cambio del tipo de datos de entrada, se recomienda usar arquitectura orientada a objetos pues al tener relacionados los procedimientos que manipulan los datos con los datos a tratar, cualquier cambio que se realice sobre ellos quedará reflejado automáticamente en cualquier lugar donde estos datos aparezcan.

6. Si el problema se encuentra dividido en etapas o módulos puede planearse la flexibilidad futura intercambiando los pasos del proceso, mediante el uso del patrón tuberías y filtros realizando intercambio de filtros, por su nivel de independencia.

7. Si en el problema los requisitos de interfaz de usuario tienden a cambiar con mayor rapidez que las reglas de negocios y los usuarios pueden preferir distintas opciones de representación, o requerir soporte para nuevos dispositivos como teléfonos celulares o PDAs, puede usarse el patrón modelo-vista-controlador, dado que el modelo no depende de las vistas, agregar nuevas opciones de presentación generalmente no afecta al modelo.

8. Si es posible que la representación de los datos cambie a lo largo del tiempo de vida del programa, entonces los tipos de datos abstractos pueden confinar el cambio en el interior de componentes particulares, por tanto, use el patrón orientado a componentes, por supuesto, sería muy útil si cada componente estuviera elaborado con el patrón n capas.

Eficiencia

1. Si el sistema lo que requiere es abordar el modelo de repositorio cuando el repositorio se encuentra en una máquina remota o en varias, debe usarse la arquitectura cliente/servidor. Nunca debe usarse arquitectura orientada a servicios, esta precisamente permite lo opuesto, es decir, estructuras desacopladas, totalmente descentralizadas, en las cuales se poseen servicios externos

a la plataforma de gestión, o sea donde frente a la flexibilidad se prefiera una centralización de la información, y ya se cuente con los programas y aplicaciones para ello.

2. Si el sistema es distribuido puede usarse arquitectura basada en eventos, pero es recomendable unirla con la arquitectura basada en servicios, porque esta arquitectura sola no permite construir respuestas complejas a funciones de negocios, además un componente no puede utilizar los datos o el estado de otro componente para efectuar su tarea y cuando un componente anuncia un evento, no tiene idea sobre qué otros componentes están interesados en él, ni el orden en que serán invocados, ni el momento en que finalizan lo que tienen que hacer. Pueden surgir problemas de rendimiento global y de manejo de recursos cuando se comparte un repositorio común para coordinar la interacción. También pudiera usarse arquitectura cliente/servidor, aunque se debe tener claro que el tráfico en la red puede congestionarla y provocar un colapso en el sistema para ello deben tomarse estrategias sobre cómo distribuir los datos en la red. En el caso de una organización, por ejemplo, éste puede ser hecho por departamentos, geográficamente, o de otras maneras. Incluso en algunos casos, por razones de confiabilidad o eficiencia, se pueden tener datos replicados, y puede haber actualizaciones simultáneas. Aunque pudiera usarse también solamente la arquitectura orientada a servicios por la posibilidad de adaptarse rápidamente y de forma ágil a los cambios, crecimientos o integraciones debido al desacoplamiento en las relaciones entre servicios.

3. Si el sistema requiere de adición y mejoramiento de funcionalidades, se recomienda usar la arquitectura orientada a componentes puesto que cada componente puede ser construido y luego mejorado continuamente por un experto u organización, de esta forma la calidad de una aplicación basada en componentes mejorará con el paso del tiempo.

4. Para implementaciones concurrentes usar el patrón tuberías y filtros. En el caso más extremo cada filtro podría correr en un procesador dedicado únicamente para él. Sin embargo, aquellos filtros que solo puedan emitir una vez recibida toda la entrada, retrasarán el funcionamiento del sistema en general; es decir se convierten en cuellos de botella. Para evitar esto tener en cuenta, la descripción arquitectónica para detectar tempranamente estos cuellos de botella y tomar medidas paliativas. Tener en cuenta también que si un filtro no puede producir salida hasta no haber

recibido toda la entrada requerirá un buffer de tamaño arbitrario. El sistema podría entrar en abrazo mortal (deadlock) si los buffers tienen tamaño fijo.

5. Si el problema involucra el traspaso de ricas representaciones de datos, deben evitarse líneas de tubería porque los datos se transmiten en forma completa entre filtros y deben ser transformados para enviar por las tuberías (hay costo de transformación de datos).

6. Si el sistema requiere de poco tiempo de espera, no utilizar el patrón tuberías y filtros pues si un filtro tiene tres tuberías de entradas, tiene que esperar hasta que las tres tuberías no le aporten los datos para poder ejecutarse totalmente. Algo similar ocurre con el uso del patrón n capas, ya que si el número de capas es excesivo, puede ser muy ineficiente, afectando su rendimiento, debido a que la transacción de datos entre los niveles de capas superiores e inferiores puede demandar mucho tiempo.

7. Si las etapas o módulos del sistema por su independencia requieren de funciones para preparar el flujo de datos, no usar el patrón de tuberías y filtros (por la duplicación de funciones), usar en ese caso, la arquitectura orientada a servicios.

8. Si son centrales la comprensión de los datos de la aplicación, su manejo y su representación, considere una arquitectura de repositorio o de tipo de dato abstracto. Si los datos son perdurables, concéntrese en repositorios.

Mantenibilidad

1. Si el sistema es distribuido se recomienda el uso de arquitectura basada en eventos haciendo que procesos de negocios que no están relacionados sean independientes y por tanto más sencillos de mantener. También pudiera utilizarse arquitectura cliente/servidor en el caso en el que, de estar distribuidas las funciones y responsabilidades entre varios ordenadores independientes, es posible reemplazar, reparar, actualizar, o incluso trasladar un servidor, mientras que sus clientes no se verán afectados por ese cambio (o se afectarán mínimamente), siempre y cuando los servidores restantes soporten todas las peticiones de los clientes; aunque la más aconsejable es la

arquitectura orientada a servicios puesto que el cambio en un sistema no afecta a los demás, ni a sus interfaces, gracias al desacoplamiento

2. Si necesita poco esfuerzo para localizar y corregir fallas, emplear patrón tuberías y filtros, puesto que, por la independencia en sus filtros, encontrar los errores es una tarea sencilla. Algo similar ocurre con el patrón MVC que facilita el mantenimiento en caso de errores al estar la estructura y el flujo de la aplicación definidos de forma más clara, sin embargo, no se recomienda su uso porque si el modelo experimenta cambios frecuentes, podría desbordar las vistas con una lluvia de requerimientos de actualización y la cantidad de archivos a mantener y desarrollar se incrementaría considerablemente. También se recomienda el uso del patrón orientado a componentes porque al existir un débil acoplamiento entre componentes, el desarrollador es libre de actualizar y/o agregar componentes según sea necesario, sin afectar otras partes del sistema.

3. Si el problema, por el mercado al cual responde, necesita de constantes cambios y refinamientos en algún nivel, se recomienda el patrón de n capas, pues este patrón facilita la evolución del sistema, ya que los cambios solo deben afectar a la capa donde se encuentre la modificación sin afectar al resto, no obstante considerar que si no se logra la contención del cambio por un mal diseño o un cambio importante de funcionalidad se requerirán cambios que se transmitirán en cascada de un nivel a otro.

4. Si tiene motivos para no vincular receptores de señales con sus originadores, debe considerarse una arquitectura de eventos.

5. Si las tareas están divididas entre productores y consumidores, debe considerarse cliente/servidor.

6. Si las tareas están divididas entre productores y consumidores y constituyen servicios debe considerarse arquitectura orientada a servicios.

7. Si el sistema es basado en desarrollo Web, y es necesario que sea visible en todos los exploradores, puede usarse el patrón n capas siempre teniendo en cuenta a la hora de implementar la capa de presentación que los exploradores actuales no son todos iguales. La

estandarización entre diferentes proveedores ha sido lenta en desarrollarse. Muchas organizaciones son forzadas a escoger uno en lugar de otro, mientras que cada uno ofrece sus propias y distintas ventajas.

8. Si el sistema es basado en aplicaciones distribuidas, se recomienda el uso de la arquitectura orientada a servicios, ya que si falla algún nodo, al no solicitarse todo a un punto específico, el sistema no colapsa, y tendrá acceso al servicio.

Disponibilidad

1. Si su sistema está encaminado a una solución con aplicaciones distribuidas, se recomienda usar arquitectura basada en eventos por las ventajas que presenta sobre todo desde el punto de vista de la capacidad para recuperarse de situaciones adversas de red, de errores de comunicación, caídas del servidor o de periféricos, que ha aumentado espectacularmente. Claro, esta arquitectura debe emplearse si y solo si los componentes no están acoplados. En caso de tener cierto nivel de acoplamiento, es recomendable usar arquitectura orientada a servicios. No obstante, pudiera usarse también arquitectura cliente/servidor ya que las variables que influyen en la disponibilidad del servicio son controlables por el proveedor. Por ejemplo se pueden prevenir los cortes de energía y conectividad del servidor central instalándolo en un buen datacenter con las redundancias necesarias, se pueden controlar los fallos de hardware del servidor central instalando servidores redundantes, o sea, se pueden prevenir las causas de una caída del servicio. Se debe estar consciente de que si no se conocen todas las causas y el servidor central cae, caería todo el servicio o gran parte de los clientes se afectaría. Otra opción podría ser usar la arquitectura orientada a servicios pero mucho cuidado porque la implementación de un bus de servicios empresarial BES, se convierte en un factor critico, si el bus llegase a fallar, la comunicación (integración), de los diferentes servicios y las aplicaciones se perdería, colapsando el sistema basado en servicios.

2. Si se desea manejar independientemente y de forma aislada diversas implementaciones de una “función” específica, en distintas computadoras, se recomienda el uso de la arquitectura basada en eventos, pero debe tenerse en cuenta que los service packs pueden invalidar el modo en que el

agente que está corriendo en la computadora se comunica con el sistema y provocar pérdidas en el servicio. No obstante, con el uso de ciertos componentes permite que la arquitectura soporte bastante bien escenarios desconectados y desconexiones no esperadas.

3. Si se necesita lograr estabilidad en el sistema un buen patrón a utilizar sería el orientado a objetos dado que permite un tratamiento diferenciado de aquellos objetos que permanecen constantes en el tiempo sobre aquellos que cambian con frecuencia, permite aislar las partes del programa que permanecen inalterables en el tiempo.

4. Si el sistema, no necesita del intercambio fluido de mucha información por la red, se recomienda no usar el patrón n capas y cliente/servidor, a no ser que se requiera, tener en cuenta que esta decisión es basada en la cantidad de niveles que se utilicen (Número de servidores) puesto que estas arquitecturas pueden incrementar el tráfico en la red, requiriendo más balance de carga y tolerancia a fallas.

Integridad

1. Si el problema requiere aplicaciones distribuidas, aunque puede usarse arquitectura cliente/servidor, se recomienda fuertemente que se tengan estrategias para el manejo de errores y para mantener la consistencia de los datos. La seguridad en esta arquitectura debe constituir una preocupación importante. Por ejemplo, se deben hacer verificaciones en el cliente y en el servidor. También se puede recurrir a otras técnicas como el encriptamiento. Algo similar ocurre con la arquitectura orientada a servicios, se puede usar, y aunque permite controlar con seguridad los procesos, considerar que resulta mucho más complejo hacerlo con múltiples procesos independientes preparados fácilmente para compartir información que con un reducido número dentro de una aplicación monolítica. Sin una infraestructura adecuada, la gestión de la seguridad de las aplicaciones se convierte habitualmente en una serie de silos que traen consigo un incremento del riesgo de filtraciones de información, un coste en la gestión de seguridad y en el cumplimiento de normativas

2. Si el sistema requiere de protección a datos y transacciones a bases de datos, se recomienda usar patrón n capas, por las mejoras que aporta a la seguridad el uso de capas que respondan a la lógica de negocio y envío/recibo de datos.

3. Si el sistema es basado en aplicaciones distribuidas, puede usar también arquitectura cliente/servidor ya que los accesos, recursos y la integridad de los datos son controlados por el servidor de forma que un programa cliente defectuoso o no autorizado no pueda dañar el sistema, aunque tener en cuenta que pudiera ocurrir y provocar graves daños como pérdida de información, etc.

Interoperabilidad

1. Si el sistema debe ser basado en aplicaciones distribuidas, puede usarse la arquitectura cliente/servidor, porque ésta facilita la integración entre sistemas diferentes y comparte información permitiendo, por ejemplo que las máquinas ya existentes puedan ser utilizadas pero utilizando interfaces mas amigables al usuario. De esta manera, podemos integrar computadoras con sistemas medianos y grandes, sin necesidad de que todos tengan que utilizar el mismo sistema operacional. Aunque hay que tener en cuenta que los clientes y los servidores utilicen el mismo mecanismo (por ejemplo sockets o RPC), lo cual implica que se deben tener mecanismos generales que existan en diferentes plataformas. Aunque se recomienda utilizar la arquitectura orientada a servicios porque al estar basada en estándares abiertos, por ejemplo los web services, estos pueden ser reutilizados por distintas aplicaciones. De esta forma, la información proveniente de múltiples fuentes, programas o tecnologías, e incluso almacenadas en muy diversos formatos o bases de datos, quedan disponibles para múltiples usos, aplicaciones y usuarios facilitando la interoperabilidad entre las soluciones sin que se produzcan problemas de compatibilidad.

2. Si el problema requiere la utilización de un repositorio de datos distribuido, (ejemplo de este tipo de repositorio es el de los bancos donde necesitas de un repositorio de datos distribuido para poder tener sus sucursales en otros países y ciudades), no usar arquitectura en capas.

Robustez

1. Si el sistema se basa en aplicaciones distribuidas puede emplearse arquitectura cliente servidor pero debe tener en cuenta que, por ejemplo, esta arquitectura no tiene la robustez de una red P2P. Cuando un servidor está caído, a no ser que se hayan tomado medidas pertinentes previendo esto, las peticiones de los clientes no pueden ser satisfechas, aunque presenta la ventaja, con respecto a una arquitectura centralizada, de que no es siempre necesario transmitir información gráfica por la red pues esta puede residir en el cliente, lo cual permite aprovechar mejor el ancho de banda de la red. No obstante el servidor es el único eslabón débil en la arquitectura cliente/servidor, debido a que toda la red está construida en torno a él, y deben tomarse todas las medidas para que sea altamente tolerante a los fallos.

2. Si lo que requiere el sistema es disminuir los tiempos de desarrollo y ganar en calidad en el código, debe usarse la arquitectura orientada a componentes, de ser posible con componentes ya testeados en otras aplicaciones de buen rendimiento.

3. Si el sistema requiere de rápido acceso a su base de datos, no usar patrón MVC, pues son necesarias varias llamadas al modelo para actualizar los datos.

4. Si el sistema requiere de la partición de un problema complejo en una secuencia de pasos incrementales, se recomienda el estilo de n capas pues facilita que se pueda descomponer la aplicación en varios niveles de abstracción.

5. Si el problema no admite un buen mapeo en una estructura jerárquica, no debe utilizarse el patrón n capas, incluso cuando un sistema se puede establecer lógicamente en capas, consideraciones de rendimiento pueden requerir acoplamientos específicos entre capas de alto y bajo nivel.

6. Si el sistema requiere de cambios, hay que tener en cuenta que si se utiliza el patrón n capas, los cambios en las capas de bajo nivel tienden a filtrarse hacia las de alto nivel, en especial si se utiliza una modalidad relajada; y esto implicará mayor impacto en costos.

7. Si el problema es "sencillo" no es factible utilizar arquitectura en capas pues aunque ayuda a controlar y encapsular aplicaciones complejas, complica no siempre razonablemente las aplicaciones simples.

8. Como ya se dijo en el apartado de usabilidad, si el sistema requiere soporte de vistas múltiples, puede usarse el patrón MVC, pero es válido aclarar que este patrón introduce nuevos niveles de indirección y por lo tanto aumenta ligeramente la complejidad de la solución, o sea la complejidad va creciendo en algunos casos más rápido que el tamaño de la aplicación.

Usabilidad

1. Si el problema debe realizarse en función de una aplicación distribuida, puede usarse arquitectura cliente/servidor esta arquitectura favorece el uso de interfaces gráficas interactivas, los sistemas construidos bajo esta arquitectura tienen mayor interacción y más intuitiva con el usuario. El uso de interfaces gráficas para el usuario, en esta arquitectura presenta la ventaja por ejemplo con respecto a uno centralizado, de que no es siempre necesario transmitir información gráfica por la red pues esta puede residir en el cliente.

2. Si el sistema requiere de fluida interactividad con el usuario, no debe usarse el patrón filtros y tuberías. Con este patrón las aplicaciones interactivas son complejas (son orientadas al procesamiento secuencial, no interactivo).

3. Si el sistema requiere soporte de vistas múltiples, en dependencia del tipo de usuario (técnicos, especialistas, etc.), use el patrón MCV. Dado que la vista se halla separada del modelo y no hay dependencia directa del modelo con respecto a la vista, la interfaz de usuario puede mostrar múltiples vistas de los mismos datos simultáneamente. Por ejemplo, múltiples páginas de una aplicación de Web pueden utilizar el mismo modelo de objetos, mostrado de maneras diferentes.

Reusabilidad

1. Si el sistema requiere de aplicaciones distribuidas, debe usarse arquitectura orientada a servicios. Esta arquitectura permitirá modificar procesos de negocio con más agilidad y a un menor coste, reutilizando y combinando las aplicaciones con que se cuenta, sin necesidad de programarlas de nuevo por completo ni preocuparse de modificar los sistemas subyacentes que intervienen en ellos.

2. Si el sistema, o parte de él debe ser altamente reusable, tener en cuenta que no todos los sistemas pueden ser estructurados en capas, ni en objetos. Y aún pudiendo ser estructurado en capas, la separación entre una y otra no es trivial. Si hay pocos niveles tenemos el diseño poco organizado. Si hay excesivos niveles el sistema es muy complejo e ineficiente.

3. Si se requiere de reutilización de un mismo nivel en varias aplicaciones, se recomienda usar patrón n capas, si una capa individual posee una abstracción y una interfaz bien definidas, con una buena documentación, la capa puede rehusarse en distintos contextos múltiples. No obstante señalar que a pesar de los costos más altos de no rehusar tal capa, los diseñadores prefieren a menudo volver a escribir esta funcionalidad. Ellos concluyen que la capa existente no encaja con sus propósitos exactamente, puesto que si la usaran tardarían más tiempo en “adaptar” la capa que rehaciéndola.

4. Si el problema puede ser descompuesto en etapas sucesivas, considere el estilo secuencial por lotes o las arquitecturas en tubería.

5. Si el sistema se encuentra dividido en etapas o módulos (no servicios o componentes), donde cada uno no conozca la identidad del módulo de donde proviene el flujo de datos que recibe como entrada y tampoco la identidad del módulo a donde llega el flujo de datos de salida, usar el patrón tuberías y filtros. En el caso en el que cada módulo sea un componente, o un servicio, se recomienda fuertemente usar entonces la arquitectura orientada a componentes o la arquitectura orientada a servicios respectivamente.

6. Si los módulos o etapas son lo más independientes posibles, o sea, en lo particular no comparten estados con otros módulos, usar el patrón tuberías y filtros, ya que La reutilización es muy alta en estos sistemas pues el sistema está dividido, y los filtros no dependen de otros filtros. Aunque en este mismo aspecto la arquitectura orientada a objetos sería otra buena opción a emplear, pues la noción de objeto permite que programas que traten las mismas estructuras de información reutilicen las definiciones de objetos empleadas en otros programas e incluso los procedimientos que los manipulan. De esta forma, el desarrollo de un programa puede llegar a ser una simple combinación de objetos ya definidos donde estos están relacionados de una manera particular.

7. Si su problema representa una lógica que se aleja considerablemente de estereotipos y normas, no usar arquitectura orientada a objetos.

Testeablidad

1. Si debe ir probando el sistema para verificar su correcto funcionamiento y entregarle primeras versiones al cliente, puede usarse la arquitectura orientada a componentes ya que simplifica las pruebas al permitir que sean ejecutadas probando cada uno de los componentes antes de probar el conjunto completo de componentes ensamblados. También se recomienda el patrón tuberías y filtros, y en caso de usar el patrón n capas, tener en cuenta la cantidad de niveles del sistema, ya que resulta un poco difícil programar y probar el software cuando tienen que comunicarse más dispositivos para terminar la transacción de un usuario.

Hay que aclarar que no se observó relación directa entre la arquitectura y la portabilidad y, además, en el caso específico del atributo fiabilidad, su correcto cumplimiento está dado por el nivel de integridad y eficiencia que se alcance al aplicar el patrón arquitectónico.

No obstante, además de estas reglas, se presentan a continuación otras reglas establecidas por Mary Shaw en el año 1995. Es válido aclarar que estas reglas fueron concebidas desde un punto de vista más general, sin tomar como eje principal a los atributos de calidad y considerando la clasificación de patrones arquitectónicos confeccionada por la misma autora (12).

2.3.1. Otras Reglas para la selección de estilos y patrones de arquitectura.

1. Si el problema puede ser descompuesto en etapas sucesivas, si además cada etapa es incremental, de modo que las etapas posteriores pueden comenzar antes que las anteriores finalicen, considere una arquitectura en tubería.

2. Si el problema involucra transformaciones sobre continuos flujos de datos (o sobre flujos muy prolongados), puede considerarse una arquitectura en tuberías.

3. Si se están considerando repositorios y los datos de entrada son ruidosos (baja relación señal-ruido) y el orden de ejecución no puede determinarse a priori, considere una pizarra.

4. Si se están considerando repositorios y el orden de ejecución está determinado por un flujo de requerimientos entrantes y los datos están altamente estructurados, puede aplicarse un sistema de gestión de base de datos.

5. Si el sistema involucra continua acción de control, está embebido en un sistema físico y está sujeto a perturbaciones externas impredecibles de modo que los algoritmos preestablecidos se tornan ineficientes, puede considerarse una arquitectura de bucle cerrado de control.

6. Si se ha diseñado una computación pero no se dispone de una máquina en la que pueda ejecutarla, puede considerarse una arquitectura de intérprete.

7. Si la tarea requiere un alto grado de flexibilidad/configurabilidad, acoplamiento laxo entre tareas y tareas reactivas, pueden aplicarse procesos interactivos.

8. Si las tareas son de naturaleza jerárquica, puede considerarse un worker replicado o un estilo de pulsación (heartbeat).

9. Si tiene sentido que todas las tareas se comuniquen mutuamente en un grafo totalmente conexo, puede considerarse un estilo de token passing.

2.3.2. Atributos de calidad y dos enfoques de Arquitecturas Heterogéneas

Además de todas las reglas mencionadas, es necesario destacar que en la gran mayoría de las aplicaciones no se utiliza un solo patrón o estilo arquitectónico, sino la combinación de varios de ellos (arquitectura heterogénea), buscando satisfacer los atributos de calidad. Generalmente los sistemas reales utilizan estas arquitecturas. Un caso típico es el de un componente de un sistema organizado según un estilo de arquitectura que puede tener una estructura interna desarrollada en otro estilo, por ejemplo (15):

- Un componente de un sistema de tubería UNIX puede estar desarrollado internamente según otro estilo.
- También un conector pipe puede estar implementado por una estructura FIFO.
- Un componente puede usar una mezcla de conectores arquitectónicos.
- Un componente puede acceder a un repositorio con parte de su interfaz, pero interactúa a través de pipes con otros componentes.

Son muchas las combinaciones posibles y mostrarlas todas con los atributos a los cuales benefician y castigan ya se iría de los objetivos del presente trabajo, e incluso podría ser, sin lugar a dudas, tema para otro. No obstante se plantean dos enfoques de la combinación de algunos de ellos, que se considera pueden ser interesantes.

Enfoque 1: Orientado a objeto + Capas + Secuencial Batch + Pipes and Filters.

Enfoque 2: Orientado a objeto + Capas.

Cada uno de los tipos de enfoque descritos anteriormente potencia características de calidad que son mostradas en la Tabla 6.

E.A.	Características propiciadas	Características castigadas
1	Mantenibilidad (caso especial, Reusabilidad). Integridad. Confiabilidad*	Eficiencia, Mantenibilidad. (Estabilidad)
2	Mantenibilidad (caso especial, Reusabilidad) Integridad. Mantenibilidad (Estabilidad). Confiabilidad*	Eficiencia

* Atributos que son potenciados y castigados a la vez.

Tabla 6. Enfoques Arquitectónicos e influencia sobre algunos atributos de calidad. (15).

Mantenibilidad: El enfoque 2 presenta resultados sutilmente superiores al 1. Esto es debido a que la presencia de los estilos Secuencial Batch y tuberías y filtros perjudican la comprobabilidad y la Facilidad de Análisis de la arquitectura. Al mismo tiempo los estilos presentes en ambos enfoques propician la Reusabilidad ya que promueven una alta cohesión y bajo acoplamiento.

En cuanto a la **Integridad**, los Enfoques que incluyeron el estilo Capas propiciaron este atributo de calidad. Éste mismo estilo favoreció la Interoperabilidad con otras aplicaciones al permitir dividir los servicios en niveles. Respecto a la Precisión, en el Enfoque 1 (que incluye Pipes and Filters, y Secuencial Batch) puede propiciar pérdidas de precisión en los resultados.

Confiabilidad: ambos enfoques afectaron positivamente esta característica de calidad. Sin embargo, el Enfoque 2 presenta mejores resultados en cuanto a la madurez. Esto se debe a la presencia de los estilos Secuencial Batch y Tuberías y Filtros que requieren de actualizaciones y procesamientos complejos. Se debe señalar que la Confiabilidad originalmente fue presentada como una característica potenciada/castigada por ambos enfoques. Esto permite suponer básicamente que está siendo propiciado

por uno o más estilos arquitectónicos predominantes en el Enfoque. Es conveniente recordar que en ambos enfoques el único estilo arquitectónico que propicia el atributo confiabilidad es el estilo Orientado a Objetos, los otros estilos lo castigan. Por ello, se deduce que el estilo arquitectónico Orientado a Objetos propicia el atributo Confiabilidad en ambos tipos de enfoques, y permite suponer que es predominante sobre otros estilos.

Usabilidad: aunque esta característica es difícil de evaluar exhaustivamente debido al nivel de abstracción de los estilos, se observa una tendencia a favorecer la misma en aquellos enfoques donde se cuente con el estilo Capas que permita separar la presentación del resto de la aplicación.

Portabilidad: al igual que en el caso anterior, la presencia del estilo Capas puede favorecer la presencia de esta característica, aunque no directamente, tanto en términos de su adaptabilidad como de su coexistencia. No así la instalabilidad.

Eficiencia: dado el conjunto complejo de interrelaciones presentes en todos los estilos, se comprobó que esta característica se puede ver penalizada por ambos enfoques.

Se evidencia entonces que ambos enfoques arquitectónicos propician las mismas características de calidad. Se encontraron evidencias de que el uso de los estilos Secuencial Batch y Tuberías y Filtros pudieran estar perjudicando el logro de ciertas características. Asimismo, se determinó que el estilo Capas favorece el logro de características que no habían sido identificadas originalmente. También se puede deducir que algunos estilos son predominantes sobre otros, al revertir su efecto negativo sobre una característica de calidad, consiguiendo un efecto conjunto positivo para la misma (15).

2.4. Validación del modelo.

La propuesta hecha por esta investigación “Modelo para la ayuda en la toma de decisiones relativas a estilos y patrones arquitectónicos” fue sometida a validación por el método Criterio de Especialistas, para probar así su eficiencia en la aplicación en proyectos productivos y en su uso continuo para el diseño de

arquitecturas basadas en atributos de calidad. Para ello se consultó a cinco especialistas, ingenieros y vinculados a distintos proyectos productivos de la Universidad de Ciencias Informáticas.

El modelo se valoró como una solución importante pues "(…) una deficiencia encontrada en la mayoría de los proyectos de la Universidad es que no utilizan ningún criterio arquitectónico para la selección del patrón a utilizar, y se basan más en la experiencia del arquitecto, que generalmente es poca, o en el uso de una herramienta que lo traiga ya implementado" (Ver Anexo 2.).

Se consideró como "(…) una guía para los arquitectos de software que les permitirá tener en cuenta cada uno de los disímiles elementos a la hora de trazar una arquitectura robusta (…), posibilitando (…) la creación de una arquitectura balanceada teniendo en cuenta los atributos de calidad y la repercusión de ellos entre sí, dando al arquitecto de software una visión amplia del diseño que le ayudará en la toma de decisiones. El modelo ayudará a realizar una evaluación de la arquitectura trazada con antelación a la ejecución del software" (Ver Anexo 1.).

Se dijo además que no se conocen "(…) otras propuestas que ayudaran en este sentido. Es cierto que la mayoría de las veces se diseñan sistemas, utilizan lenguajes y herramientas basados en la intuición y experiencia de las personas con poder de decisión sobre el desarrollo a realizar, muchas veces incluso deciden personas que luego no son los que desarrollan y no son los que se enfrentan a los problemas reales del negocio a informatizar".

Se cree "(…) de vital importancia la existencia de una metodología o guía que apoye a los arquitectos a seleccionar los patrones de arquitectura ideales para la solución que se debe informatizar. Mucha más importancia tiene para la UCI, teniendo en cuenta el ambiente cambiante en que trabajamos y el grado de experiencia del personal que llevan a cabo los desarrollos, personal que conocemos en su gran mayoría son estudiantes de 3ro, 4to y 5to año, liderados muchas veces por profesores que en su gran mayoría son recién graduados (…). Todos, estudiantes y profesores deben aprender sobre la marcha, lo cual puede afectar negativamente el desarrollo de cualquier proyecto. Entonces es importante que existan medios que guíen el proceso de desarrollo, desde las primeras actividades hasta su culminación.

Se piensa, “(...) además, que los pasos propuestos deben ser tenidos en cuenta para la selección de los patrones de diseño de una arquitectura, así como de manera general la metodología que se propone (…)" (Ver Anexo 5.)

La idea de la investigación es entonces “(…) novedosa, compleja y esta alineada a las principales tendencias del mundo en la academia, sobre todo lo relacionado con el uso de técnicas de IA, aplicadas a la toma de decisiones de las herramientas de desarrollo. (…) Se han seguido las principales características de cada uno de los estilos y patrones de arquitectura, para asociarlos a la solución de problemas que tributan en el desarrollo de software a atributos de calidad de la arquitectura. De modo que este trabajo representa una taxonomía guía para ante la necesidad de tomar decisiones arquitectónicas, saber en que momento y según el atributo de calidad que se desee priorizar cual variante arquitectónica usar.” (Ver Anexo 3.)

2.5. Conclusiones

En este capítulo se describió el modelo propuesto para la ayuda en la toma de decisiones referentes a estilos y patrones arquitectónicos. El modelo se basó en 3 pasos fundamentales:

1. Relación de la influencia entre atributos de calidad.

2. Ponderación de los atributos de calidad.

3. Definición de reglas para seleccionar estilo(s) y patrón(es) arquitectónico(s) para potenciar determinados atributos de calidad.

En el primer paso se hizo un análisis de algunas posibles situaciones para determinar la relación existente entre los diferentes atributos de calidad y de esta forma conocer el modo en que incidían unos sobre otros.

En el segundo paso se establecieron una serie de parámetros o aspectos basados en la relación arquitectura-atributos de calidad, para cuantificar los diferentes atributos de calidad según el software requerido por el cliente.

En el tercer paso se definieron un conjunto de reglas basadas en las fortalezas y debilidades de los estilos y patrones arquitectónicos, en función de potenciar o mitigar los atributos de calidad ya definidos previamente.

CONCLUSIONES

Se realizó un estudio de estado del arte sobre los atributos de calidad, la arquitectura de software, los estilos y patrones arquitectónicos y distintos métodos y herramientas para el diseño de la arquitectura. Este estudio sentó las bases para la elaboración de un conjunto de reglas. Estas constituyen un modelo o guía para la selección de los estilos y patrones arquitectónicos y llevan a una mayor satisfacción de los atributos de calidad especificados en el producto a desarrollar.

Con la realización y validación del modelo, se le dieron cumplimiento a los objetivos planteados y se arribó a las siguientes conclusiones:

- La relación establecida entre los atributos de calidad permite determinar la influencia, positiva o negativa, de uno sobre otro, aclarando así el peso que tendrían las decisiones arquitectónicas (selección de estilos y patrones) tomadas sobre dichos atributos.

- La ponderación de los atributos de calidad lleva a que el arquitecto de prioridad al atributo más importante dentro del proyecto a la hora de la selección del estilo y/o patrón a utilizar.

- Los dos factores antes mencionados complementan el modelo, obtenido mediante el enunciado de reglas para seleccionar estilos y patrones de arquitectura.

RECOMENDACIONES

Toda obra humana y sobre todo investigativa, auténtica e inigualable, dista siempre de ser perfecta, debido fundamentalmente al constante cambio en el entorno para el cual fue concebida y producto también a la superación de sus creadores. Esta no es una excepción. Es por ello que se deben tomar en consideración el mejoramiento y la incorporación en un futuro de nuevos elementos que estén acordes con los cambios que se susciten en materia de arquitectura y calidad. Se recomienda además:

- Continuar con la investigación, pues pueden haber quedado elementos a tener en cuenta como por ejemplo incluir el análisis de elementos de arquitectura desde un punto de vista enfocado a procesos.

- Aplicar el modelo en un proyecto productivo, y realizar un estudio más riguroso sobre los patrones y estilos arquitectónicos aplicados o desarrollados en aplicaciones de la Universidad, para establecer entonces una relación entre los atributos que se buscaban conseguir y las arquitecturas ya probadas, sobre la base de la experiencia adquirida en la propia UCI

- Implementar una aplicación en Inteligencia artificial basada en las reglas de esta investigación.

- Hacer de esta investigación una material de consulta para los arquitectos que trabajen en los proyectos.

BIBLIOGRAFÍA

1. **Bastarrica, María Cecilia.** *Atributos de Calidad y Arquitectura del Software.*

2. **Wiegers, Karl E.** *Software Requirements, Second Edition.* 2003.

3. **Mauricio Dávila, Martín Germán, Diego Crutas, Andrés García.** *Evaluación de Arquitecturas de Software.*

4. **Arregui, Juan José Olmedilla.** *Revisión Sistemática de Métricas de Diseño Orientado a Objetos.* 2005.

5. **Leonardo DaVinci, Insituto Tecnológico.** *Gestión, Control y Garantía de la Calidad del Software.* 2006.

6. **Fillottrani, Pablo R.** *Calidad en el Desarrollo de Software: Modelos de calidad de software.* 2007.

7. **M., Luis Eduardo Menzosa.** *Sistemas de Información.*

8. **Erika Camacho, Fabio Cardesco, Gabriel Nuñez.** *Arquitecturas de Software.* 2004.

9. **Reynoso, Carlos Billy.** *Introducción a la Arquitectura de Software.* 2004.

10. **Canal, Carlos.** *Arquitectura, marcos de trabajo y patrones.* 2005.

11. *Metodología para la Gerencia de los Procesos del Negocio sustentada en el uso de patrones.* **Bonillo, Pedro.** s.l. : Journal of Information Systems and Technology Management , 2006, Vol. 3.

12. **Carlos Reynoso, Nicolás Kicillof.** *Estilos y Patrones en la Estrategia de Arquitectura de Microsoft.* 2004.

13. **Felix Bachmann, Len Bass, Phil Bianco.** *Software Architecture Design with Arche.* 2007.

14. Architecture Expert Design Assistant (ArchE). *Software Engineering Institute.* [Online] [Cited: Enero 25, 2008.] http://www.sei.cmu.edu/architecture/products_services/arche.html.

15. **Anna Grimán, María Pérez y Luis Mendoza.** *Estudio de la influencia de mecanismos arquitectónicos en la calidad del software.* 2005.

16. **Young, Ralph R.** *The Requirements Engineering Handbook.* 2004.

17. **Witold Suryn, Blanca Gil.** *ISO/IEC9126–3 internal quality measures: are they still useful?*

18. **Wiegers, Karl E.** *More About Software Requirements: Thorny Issues and Practical Advice.* 2006.

19. —. *Mastering the Requirements Process Second Edition .* 2003.

20. **Welicki, León.** Patrones y Antipatrones: una Introducción - Parte II. *MSDN.* [Online] [Cited: Mayo 10, 2008.] http://www.microsoft.com/spanish/msdn/comunidad/mtj.net/voices/MTJ_3317/default.aspx.

21. **Vega Lebrún Carlos, Rivera Prieto Laura Susana, García Santillán Arturo.** *Mejores prácticas para el Establecimiento y Aseguramiento de la calidad del Software.* 2008.

22. **Suzzane Robertson, James Robertson.** *Mastering the Requirements Process Second Edition* . 2006.

23. *Arquitectura de software: Importancia de su ciclo de vida.* **Suárez, José de Jesús Hernández.** 64, s.l. : Universidad Nacional Autónoma de México, 2007.

24. **Sánchez-Segura, Ana M. Moreno y Maribel.** *Patrones de Usabilidad: Mejora de la Usabilidad del Software desde el momento Arquitectónico.*

25. *Puntos por Función.Una métrica estándar para establecer el tamaño del software*. **Rubio, Sergio Eduardo Durán.** 6, s.l. : Boletín de Política Informática, 2003.

26. **Quiroz, Ricardo Romero.** *Implementaci´on de una arquitectura blackboard para sistemas de hipermedia.* 2003.

27. **Nicolás Passerini, Gustavo Andrés Brey.** *Arquitectura de Proyectos de IT: Tácticas.* 2005.

28. **Muñoz, Coral Calero.** *Métrics del Software: Conceptos básicos, definición y formalización.* 2006.

29. **Medrano, José Miguel Calvo.** *Medida de las subcaracterísticas Capacidad de Análisis y Capacidad de Cambio mediante la norma ISO/IEC 9126.*

30. **Martínez, Nelson Medinilla.** Los patrones no son un peligro. *Departamento de Lenguajes y Sistemas Informáticos e Ingeniería del Software-Unidad Docente de Ingeniería del Software (UDIS).* [Online] 2003. [Cited: Enero 20, 2008.] http://is.ls.fi.upm.es/udis/docencia/proyecto/docs/patrones.html.

31. **Martínez, José Mª. Torralba.** *Reutilización del conocimiento del diseño de software. Consideración en la determinación del precio de oferta al cliente.* 2004.

32. **Lutowski, Rick.** *Software Requirements Encapsulation, Quality, and Reuse.* 2005.

33. **Lucas Besso, Yanina Boccardo, José Girardi.** *Patrón Arquitectural Tuberías y Filtros.* 2006.

34. **Lovelle, Juan Manuel Cueva.** *Calidad del Software.* 1999.

35. **López, G., et al.** *Diseño de aplicaciones con arquitectura orientada a servicios. Modelos de ciclo de vida ad-hoc.*

36. **Lihua Xu, Scott A. Hendrickson, Eric Hettwer, Hadar Ziv, André van der Hoek, and Debra J. Richardson.** *Towards Supporting the Architecture Design Process Through Evaluation of Design Alternatives.*

37. **Lihua Xu, Hadar Ziv, Debra Richardson, Zhixiong Liu.** *Towards Modeling Non-Functional Requirements in Software Architecture.*

38. **Len Bass, Paul Clements, Rick Kazman, Linda Northrop, Amy Zaremski,.** *Recommended Best Industrial Practice for Software Architecture Evaluation.* 1997.

39. **Keefer, Rob.** *Software Architecture An Overview.* 2006.

40. *SOA: Arquitectura de futuro.* **ISV.** s.l. : Editorial MKM, 2008.

41. **ISO.** *ISO 9126 Parte 1: Norma Cubana. Características y sub-características de calidad* . 2003.

42. —. *ISO 9126 Part 4: Quality in use metrics.* 2003.

43. —. *ISO 9126 Part 3: Internal Metrics.* 2003.

44. —. *ISO 9126 Part 2: External Metrics.* 2003.

45. **Ignacio Rivera, Luis Robacio.** *Patrones Arquitectónicos Layers.* 2006.

46. **IBM.** *Servicios IBM de integración de SOA: Conectividad y reutilización.* 2007.

47. **Hernández, Olivia Allende.** *La tecnología orientada a objetos y la ingeniería de software ante la complejidad inherente al software.* 2002.

48. **Gutiérrez, Alfred Kobayashi.** *Aseguramiento de la Calidad de AppWeb.*

49. **Gunnar Övergaard, Karin Palmkvist.** *Use Cases Patterns and Blueprints.* 2004.

50. **Guillen-Drija, Francisca Losavio de ordaz y Christian.** Marco conceptual para un diseño arquitectónico basado en aspectos de calidad. *Revista Universitaria de Investigacion (Sapiens).* [Online] [Cited: 3 22, 2008.] http://www2.scielo.org.ve/scielo.php?script=sci_arttext&pid=S1317-58152006000200009&lng=es&nrm=iso.

51. **Gastón Coco, Nicolás Passerini, Juan Arias, Gustavo A. Brey.** *Arquitectura de Proyectos de IT-Estilos Arquitectónicos.* 2005.

52. **Ferreiro, Félix González.** *Herramienta de administración para un Centro IP.* 2006.

53. **Fernández, Nelly Condori.** *Un Procedimiento de Medición de Tamaño Funcional para Especificaciones de Requisitos.* 2006.

54. **Dean Leffingwell, Don Widrig.** *Managing Software Requirements: A Use Case Approach, Second Edition.* 2003.

55. **David Garlan, Mary Shaw.** *An Introduction to Software Architecture.* 1994.

56. **Cuesta, Carlos E.** *Arquitecturas de Software.*

57. **Cristiá, Maximiliano.** *Introducción a la Arquitectura de software.* 2007.

58. —. *Catálogo Incompleto de Estilos Arquitectónicos.* 2006.

59. **Cristancho, José Alerto.** *Evaluación de la calidad del software educativo bajo el estándar ISO 9126.*

60. **Christensen, Claudio.** *Presentación, Confiabilidad y Disponibilidad.* 2005.

61. **Casanovas, Joseph.** *Usabilidad y arquitectura del software.* 2004.

62. —. *Usabilidad y arquitectura del software.*

63. **Brey, Gustavo Andrés.** *Arquitectura de Proyectos de IT-Atributos de Calidad.* 2006.

64. **Booch, Grady.** *On Architecture.* 2006.

65. *Calidad en modelos conceptuales: Un Análisis Multidimensional de Modelos Cuantitativos basados en la ISO 9126.* **Beatriz Marín, Nelly Condori-Fernández, Oscar Pastor.** s.l. : Revista de procesos y Métricas de las tecnologías de la Información, 2007, Vol. 4.

66. **Astudillo, Hernán.** *Ingeniería de Software-Arquitectura y Diseño [2].* 2004.

67. —. *Fivw ontological levels to describe and evaluate software architectures.* 2004.

68. **Anna Grimán, María Pérez y Luis Mendoza.** *Estudio de la influencia de mecanismos arquitectónicos en la calidad del software.* 2005.

69. **Andrés flipe Borja, Johnny Gómez Mojica, Raúl Adrés de Villa, Juan Pablo Ramírez Madrid.** *La Importancia de la Arquitectura en el desarrollo de Software de Calidad.* 2005.

70. **Andes, Grupo de Construcción de Software-Facultad de Ingeniería-Universidad de los.** *Evolución de las arquitecturas de Software.*

71. **Anatoli Semenovich Koulinitch, Francisco Espinosa Maceda, Carlos Benavides Martínez***.** *Arquitectura basada en objetos de computación distribuida en la configuración de sistemas distribuidos.*

72. **Amador Durán Toro, Beatriz Bernárdez Jiménez.** *Metodología para la Elicitación de Requisitos de Sistmas Software.* 2.1. 2000.

73. **Aleksander González, Marizé Mijares, Luis E. Mendoza, Anna Grimán, María Pérez.** *Método de Evaluación de Arquitecturas de Software Basadas en Componentes (MECABIC).* 2005.

74. *SOA: Arquitectura de futuro.* **Alba, Julio.** s.l. : Bit, 2008, Vol. 167.

75. **Alain Abran, Rafa Al Qutaish, Jean-Marc Desharnais, Naji Habra.** *An Information Model for Software Quality Measurement with ISO standars.*

76. **Abraham Dávila, Karin Melendez y Luis Flores.** *Determinación de los Requerimientos de Calidad del Producto Software basados en Normas Internacionales.*

77. Patrones para arquitecturas orientadas a eventos y mensajes. *Geeks.ms.* [Online] [Cited: Mayo 6, 2008.] http://geeks.ms/blogs/rcorral/archive/2007/06/05/webcast-patrones-para-arquitecturas-orientadas-a-eventos-y-mensajes.aspx.

78. Modelos avanzados de comunicación de recursos. *Morfeo Project.* [Online] [Cited: 5 12, 2008.] http://forge.morfeo-project.org/wiki/index.php/D_3.2_Documento_de_definici%C3%B3n_de_modelos_avanzados_de_comunicaci%C3%B3n_y_composici%C3%B3n_de_recursos.

79. Interoperabilidad en la Empresa: .NET y J2EE. *MSDN.* [Online] [Cited: 5 9, 2008.] http://www.microsoft.com/spain/interop/developers/dotnetinteroperability.mspx.

80. Estilos y Patrones. *Universidad de Antioquia.* [Online] [Cited: Febrero 4, 2008.] http://siona.udea.edu.co/~aoviedo/Arquitectura%20de%20Software/EstilosyPatrones.htm#_Cómo_definir_el.

81. Diseño preliminar de Arche: Una arquitectura de software de diseño Auxiliar. *Software Engineering Institute.* [Online] [Cited: Enero 25, 2008.] http://64.233.179.104/translate_c?hl=es&u=http://www.sei.cmu.edu/publications/documents/03.reports/03tr021/03tr021.html&prev=/search%3Fq%3Dsoftware%2Barche%252BSEI%26hl%3Des%26lr%3D.

82. Desarrollo de Softwere Basado en Componentes. *MSDN.* [Online] [Cited: Mayo 4, 2008.] http://www.microsoft.com/spanish/msdn/comunidad/mtj.net/voices/MTJ_3985/default.aspx.

83. Arquitecturas cliente/servidor. *Redindustria: Red de Conocimiento en Informática Industrial y Aplicaciones de Gestión en Tiempo Real.* [Online] [Cited: Mayo 6, 2008.] http://redindustria.blogspot.com/2008/03/arquitectruas-clienteservidor.html.

84. Arquitectura de Software. *Universidad de Antioquia*. [Online] [Cited: febrero 4, 2008.] http://siona.udea.edu.co/~aoviedo/Arquitectura%20de%20Software/Arquitectura%20de%20Software.htm#_Modelos_de_Arquitectura.

85. Arquitectura de Aplicaciones de 3 capas . *Desarrollo. NET*. [Online] [Cited: Mayo 2, 2008.] http://www.dotnetjunkies.com/WebLog/desarrollonet/archive/2004/06/17/16855.aspx.

86. Aplicación para el intercambio de archivos basado en una arquitectura PEER-TO-PEER. *Universidad Privada del Valle Bolivia*. [Online] [Cited: Mayo 6, 2008.] http://www.univalle.edu/publicaciones/journal/journal11/pagina05.htm.

87. Análisis de las diferentes variantes de la Arquitectura Cliente/Servidor. *Instituto Nacional de Estadñisitca e Informatica (INEI) de Perú*. [Online] [Cited: mayo 6, 2008.] http://www.inei.gob.pe/web/metodologias/attach/lib616/CAP0308.HTM.

88. A First Look at SEI's Architecture Expert Design Assistant (ArchE). *Real World Software Architecture* . [Online] Enero 24, 2008. C:\Documentacion\Joisel\joisel\Herramientas para evaluar la arquitectura\Versión traducida de http--realworldsa_dotnetdevelopersjournal_com-archerelease.htm.

89. EDA: UNA NUEVA GENERACIÓN DE APLICACIONES. *CIENTEC*. [Online] [Cited: Mayo 7, 2008.] http://www.cientec.com/analisis/eda.asp.

90. Arquitectura de Expertos (Arche) Herramienta. *Software Engineering Institute*. [Online] [Cited: enero 25, 2008.] http://64.233.179.104/translate_c?hl=es&u=http://www.sei.cmu.edu/architecture/arche.html&prev=/search%3Fq%3Dsoftware%2Barche%252BSEI%26hl%3Des%26lr%3D.

ANEXOS

Anexo 1. Criterio del especialista Daynier Ruiz Rodríguez.

Criterio de Especialistas

Validación del Modelo para la ayuda en la toma de decisiones relativas a estilos y patrones arquitectónicos

Nombre del Especialista: Daynier Ruiz Rodríguez

Grado Científico que posee:

Proyecto en el que se encuentra trabajando: Convenio Cuba – Venezuela

Rol que desempeña dentro del Proyecto: Arquitecto Principal

Creo que el Modelo para la ayuda en la toma de decisiones relativas a estilos y patrones arquitectónicos, constituye en si misma una guía para los arquitectos de software que le permitirá tener en cuenta cada uno de los disímiles elementos a la hora de trazar una arquitectura robusta siendo de vital importancia debido a la falta de experiencia de la mayoría de los que juegan este rol dentro de los proyecto de la UCI. Además la propuesta permite crear una arquitectura balanceada teniendo en cuenta los atributos de calidad y la repercusión de ellos entre si, dando al arquitecto de software una visión amplia del diseño que le ayudará en la toma de decisiones. El modelo ayudará a realizar una evaluación de la arquitectura trazada con antelación a la ejecución del software.

Firmado a los 11 **días del mes de Junio de 2008**

"Año 50 de la Revolución"

Anexo 2. Criterio del especialista Jorge Infante Osorio.

Criterio de Especialistas

Validación del Modelo para la ayuda en la toma de decisiones relativas a estilos y patrones arquitectónicos

Nombre del Especialista: Jorge Infante Osorio.

Grado Científico que posee:

Proyecto en el que se encuentra trabajando: Arquitectura Corporativa SOA-PDVSA

Rol que desempeña dentro del Proyecto: Jefe de Equipo de Desarrollo y Gobierno.

El tema propuesto en la tesis es de gran actualidad para el desarrollo productivo de la UCI pues se enfoca en una deficiencia encontrada en la mayoría de los proyectos que no utilizan ningún criterio arquitectónico para la selección del patrón a utilizar, y se basan más en la experiencia del arquitecto, que generalmente es poca, o en el uso de una herramienta que lo traiga ya implementado, como es el caso de MVC.

Con la propuesta que se realiza en esta tesis podremos contar con un modelo que nos permita saber que criterios tener en cuenta a la hora de escoger un patrón arquitectónico basado en las variaciones de los atributos de calidad y el impacto que los mismos han de tener en el funcionamiento de la aplicación. Esto último permitirá garantizar que la arquitectura seleccionada este acorde a las necesidades plasmadas por los clientes y el grupo de desarrollo siempre llegando a un consenso que equilibre las relaciones entre los diferentes atributos de calidad que se plasman en los requerimientos no funcionales de la aplicación. Es a mi entender que esta investigación por su valor investigativo debe de ser concretada en un proyecto productivo durante la selección de su propuesta arquitectónica.

Firmado a los 10 **días del mes de Junio de 2008**

"Año 50 de la Revolución"

Anexo 3. Criterio del especialista René Lazo Ochoa.

Criterio de Especialistas

Validación del Modelo para la ayuda en la toma de decisiones relativas a estilos y patrones arquitectónicos

Nombre del Especialista: René Lazo Ochoa

Grado Científico que posee: Ninguno

Proyecto en el que se encuentra trabajando: ERP País

Rol que desempeña dentro del Proyecto: Arquitecto principal

Los estilos arquitectónicos y los patrones de arquitectura son una de las más provechosas novedades de la ingeniería de software. Su aparición ha posibilitado armar a los desarrolladores de patrones reutilizables, para complejos problemas que se presentan en la estructuración y conformación de los proyectos de software.

La idea del trabajo de tesis que se analiza es novedosa, compleja y esta alineada a las principales tendencias del mundo en la academia, sobre todo lo relacionado con el uso de técnicas de IA, aplicadas a la toma de decisiones de las herramientas de desarrollo. Sobre la solución que se propone, se puede evaluar de aceptable, teniendo en cuenta que se han seguido las principales características de cada uno de los estilos y patrones de arquitectura, para asociarlos a la solución de problemas que tributan en el desarrollo de software a atributos de calidad de la arquitectura. De modo que este trabajo representa una taxonomía guía para ante la necesidad de tomar decisiones arquitectónicas, saber en que momento y según el atributo de calidad que se desee priorizar cual variante arquitectónica usar.

Entre las recomendaciones que se le hacen al trabajo se listan las siguientes:

1. Se recomienda hacer una divisiones entre estilos y patrones de arquitectura
2. Se recomienda definir escenarios arquitectónicos que tipifiquen los problemas, y estos a su vez queden clasificados como lo realiza el trabajo de tesis por atributos de calidad
3. Se recomienda incluir algún mecanismo para la decisión y ponderación de los atributos asociados al patrón, que permitan un proceso de selección más riguroso. Por ejemplo método experto. Considero podría incluirse alguna técnica de IA para que el sistema se comporte como un sistema experto y sea capaz de proponer con niveles de certidumbre ante cada problema a consultar.

Firmado a los 11 **días del mes de Junio de 2008**

"Año 50 de la Revolución"

Anexo 4. Criterio del especialista Luis Alberto Pimentel González.

Criterio de Especialistas

Validación del Modelo para la ayuda en la toma de decisiones relativas a estilos y patrones arquitectónicos

Nombre del Especialista: Luis Alberto Pimentel Glez.

Grado Científico que posee: Ingeniero

Proyecto en el que se encuentra trabajando: SIGEP.

Rol que desempeña dentro del Proyecto: Arquitecto

El documento de tesis, en mi opinión, puede constituir un paso de avance en la organización de las arquitecturas en la universidad. Evidencia la necesidad de analizar detalladamente los requisitos tanto funcionales como no funcionales como paso previo y obligatorio antes de determinar la arquitectura que regirá el desarrollo de un software. El documento se enmarca sobre los estilos arquitectónicos, no obstante considero de gran utilidad que se haga un estudio de la aplicación que se ha dado de los mismos en arquitecturas desarrolladas o adoptadas en la universidad; esto permitirá establecer relaciones entre requisitos y las arquitecturas ya probadas, logrando un carácter práctico ligado a la propia experiencia adquirida en la UCI.

Firmado a los 10 **días del mes de Junio de 2008**

"Año 50 de la Revolución"

Anexo 5. Criterio del especialista Yordanis Tornes Medina.

Criterio de Especialistas

Validación del Modelo para la ayuda en la toma de decisiones relativas a estilos y patrones arquitectónicos

Nombre del Especialista: Yordanis Tornes Medina
Grado Científico que posee: Ingeniero
Proyecto en el que se encuentra trabajando: CTAISC
Rol que desempeña dentro del Proyecto: J' del Proyecto

No conozco propuestas que ayudaran en este sentido. Es cierto que la mayoría de las veces se diseñan sistemas, utilizan lenguajes y herramientas basados en la intuición y experiencia de las personas con poder de decisión sobre el desarrollo a realizar, muchas veces incluso deciden personas que luego no son los que desarrollan y no son los que se enfrentan a los problemas reales del negocio a informatizar.

Si creo de vital importancia la existencia de una metodología o guía que apoye a los arquitectos a seleccionar los patrones de arquitectura ideales para la solución que se debe informatizar. Mucha más importancia tiene para la UCI, teniendo en cuenta el ambiente cambiante en que trabajamos y el grado de experiencia del personal que lleva a cabo los desarrollos, personal que conocemos en su gran mayoría son estudiantes de 3ro, 4to y 5to año, liderados muchas veces por profesores que en su gran mayoría son recién graduados, que si bien puede considerarse que tienen un poco más de experiencia por ser ya graduados, en el campo de la producción de SW para comercializar podemos decir que tampoco tenemos ninguna experiencia. Todos, estudiantes y profesores debemos aprender sobre la marcha, lo cual puede afectar negativamente el desarrollo de cualquier proyecto. Entonces es importante que existan medios que guíen el proceso de desarrollo, desde las primeras actividades hasta su culminación.

El modelo que se propone puede ser efectivo en la elaboración de una buena arquitectura, que claro está, la calidad de la misma depende mucho de los patrones arquitectónicos que se seleccionen para esta. Muchas veces asignamos a un desarrollador el rol de arquitecto porque es muy bueno programando en una plataforma u otra (independientemente de si el producto a desarrollar vaya a ir sobre esa plataforma); un buen arquitecto no es aquel que sea buen programador o conozca bien a fondo la plataforma en la que vaya a estar el Producto, debe conocer además qué es una arquitectura, cómo se diseña, qué cosas se deben tener en cuenta a la hora de definir una arquitectura, entre otros elementos... todos estos elementos no los dominan y muchas veces no conocen los arquitectos que tenemos, debido a la poca experiencia en el desarrollo de SW que tenemos. He aquí dónde se hace necesario tener herramientas que guíen a nuestros arquitectos hacia las buenas prácticas para definir

Anexo 5 (Continuación). Criterio del especialista Yordanis Tornes Medina.

arquitecturas, herramientas que recomienden con basamentos teóricos y prácticos, cuándo y por qué ir por un camino u otro en la definición de una arquitectura.

Pienso además que los pasos propuestos deben ser tenidos en cuenta para la selección de los patrones de diseño de una arquitectura, así como de manera general la metodología que proponen, creo que este trabajo no debe quedarse hasta aquí y siga la investigación, pues pueden haberse quedado muchos elementos a tener en cuenta en todo el proceso que proponen. Por ejemplo, en la ponderación de los atributos de calidad, deben tenerse en cuenta no solo los que más le interesan al cliente, sino también los que informáticamente considere el equipo de desarrollo que deben presentarse, claro está de mutuo acuerdo con el cliente. El cliente no siempre tiene una visión informática del asunto, aunque si siempre, o el 99% de las veces conoce el negocio, no conoce en el mundo de la informática como pudiera lograr un sistema que le permita aprovechar al máximo una línea de producción. Quienes diseñen el sistema deben analizar el qué se debe informatizar, qué se espera y hasta dónde se puede llegar, en función de eso y de mutuo acuerdo con el cliente, potenciar aquellos atributos de calidad que respondan a las interrogantes anteriores.

Firmado a los 10 días del mes de Junio de 2008

"Año 50 de la Revolución"

GLOSARIO

Calidad: grado con el que un sistema, componente o proceso cumple los requerimientos especificados y las necesidades o expectativas del cliente o usuario.

Framework: Estructura de software que está compuesta de componentes personalizables e intercambiables, reusable sobre todo para el desarrollo de software.

Ingeniería de Software: Es la rama de la ingeniería que aplica los principios de la ciencia de la computación y las matemáticas para lograr soluciones costo-efectivas (eficaces en costo o económicas) a los problemas de desarrollo de software, es decir, permite elaborar consistentemente productos correctos, utilizables y costo-efectivos.
Según Pressman, es una tecnología multicapa en la que se pueden identificar los métodos (indican como construir el software), el proceso (es el fundamento de la ingeniería de software, es la unión que mantiene juntas las capas de la tecnología) y las herramientas (soporte automático o semiautomático para el proceso y los métodos).

Escenarios: Son descripciones en lenguaje natural, de situaciones particulares del entorno donde se desarrolla el sistema.

Estilos de grano más fino y grado más grueso: Estilos donde la noción de sus componentes puede variar dependiendo del nivel de detalle desde donde se mire, esto es conocido como Granularidad de componente. Un componente con granularidad gruesa se refiere a que puede estar compuesto por un conjunto de componentes, o ser una aplicación para construir otras aplicaciones o sistemas a gran escala, generalmente abiertos y distribuidos. A medida que se desciende en el nivel de detalle, se dice que un componente es de grano fino.