ARQUITECTURAS DE SOFTWARE

GUÍA DE ESTUDIO

ELABORADA POR:
ERIKA CAMACHO
FABIO CARDESO
GABRIEL NUÑEZ

REVISADA POR:
PROF. MARIA A. PEREZ DE OVALLES
PROF. ANNA GRIMÁN
PROF. LUIS E. MENDOZA

ABRIL – 2004

ESQUEMA DE CONTENIDO

INTRODUCCIÓN
1. CALIDAD DEL SOFTWARE
2. ARQUITECTURA DE SOFTWARE
2.1 IMPORTANCIA DE LA ARQUITECTURA DE SOFWTARE
2.2 Componentes, conectores y relaciones
3. CALIDAD ARQUITECTÓNICA
3.1 Atributos de Calidad
3.2 Modelos de Calidad
3.2.1 Modelo de Mc Call
3.2.2 Modelo de Dromey
3.2.3 Modelo FURPS
3.2.4 Modelo ISO/IEC 9126
3.2.5 ISO/IEC 9126 adaptado para arquitecturas de software
3.3 Relación entre Arquitectura de Software y Atributos de Calidad
4. ESTILOS Y PATRONES
4.1 Estilo Arquitectónico
4.2 Patrón Arquitectónico
4.3 Patrón de Diseño
5. VISTAS ARQUITECTÓNICAS
5.1 Comparación de Vistas Arquitectónicas
6. NOTACIONES
7. LENGUAJES DE DESCRIPCIÓN ARQUITECTÓNICA
7.1 Conceptos y características de los lenguajes de descripción arquitectónica
7.2 Ventajas del uso de lenguajes de descripción arquitectónica
7.3 Diferencias entre los lenguajes de descripción arquitectónica y otros lenguajes
8. EVALUACIÓN DE ARQUITECTURAS DE SOFTWARE
9. TÉCNICAS DE EVALUACIÓN DE ARQUITECTURAS DE SOFTWARE
9.1 Evaluación basada en escenarios
9.1.1 Utility Tree
9.1.2 Perfiles (*Profiles*)
9.2 Evaluación basada en simulación
9.3 Evaluación basada en modelos matemáticos
9.4 Evaluación basada en experiencia
10. MÉTODOS DE EVALUACIÓN DE ARQUITECTURAS DE SOFTWARE

10.1 Software Architecture Analysis Method (SAAM)
10.2 Architecture Trade-off Analysis Method (ATAM)
10.3 Active Reviews for Intermediate Designs (ARID)
10.4 Modelo de Negociación WinWin
10.5 Cost-Benefit Analysis Method (CBAM)
10.6 Método Diseño y Uso de Arquitecturas de Software propuesto por Bosch (2000)
10.7 Método de comparación de arquitecturas basada en el modelo ISO/IEC 9126 adaptado para arquitecturas de software
10.8 Comparación entre métodos de evaluación
11. HERRAMIENTAS DE ANÁLISIS, DISEÑO Y EVALUACIÓN DE ARQUITECTURAS DE SOFTWARE

Referencias

1. INTRODUCCIÓN

Kazman (1996) plantea que la necesidad del diseño y el análisis de las arquitecturas de software ha llevado al deseo de la creación de herramientas CASE para soportar el proceso de desarrollo, y que la herramienta debería, entre otras cosas, permitir documentar la arquitectura, hacer uso de artefactos previos, servir de ayuda en la exploración de arquitecturas alternativas, y soportar métricas arquitectónicas. Para Kazman (1996), la arquitectura de software es una forma de representar sistemas complejos mediante el uso de la abstracción. Sin embargo, una herramienta como la que se plantea no sólo debe cumplir con los objetivos del diseño, sino que también debería ayudar a garantizar que el sistema construido se corresponda con la arquitectura planteada, mediante un proceso de análisis arquitectónico sistemático. En líneas generales, el planteamiento de Kazman (1996) está relacionado con la necesidad de construir herramientas que permitan hacer del diseño y el análisis de las arquitecturas de software, una actividad más confiable y mejor documentada.

La arquitectura de software es importante como disciplina debido a que los sistemas de software crecen de forma tal que resulta muy complicado que sean diseñados, especificados y entendidos por un solo individuo. Uno de los aspectos que motivan el estudio en este campo es el factor humano, en términos de aspectos como inspecciones de diseño, comunicación a alto nivel entre los miembros del equipo de desarrollo, reutilización de componentes y comparación a alto nivel de diseños alternativos (Kazman, 1996).

El proceso de recolección, mantenimiento y validación de la información arquitectónica es tedioso y altamente propenso a errores. Estos son precisamente los candidatos a ser cubiertos por una herramienta (Kazman, 1996). El control de revisión, análisis de dependencias y proceso de pruebas son sólo algunos ejemplos de herramientas que automatizan exitosamente las tareas que se repiten constantemente durante el desarrollo.

Bredemeyer et al. (2002) proponen que los requerimientos arquitectónicos son necesarios para guiar las actividades de estructuración de un sistema de software. En el proceso de desarrollo se pueden aplicar diversos enfoques para garantizar el cumplimiento de los requerimientos arquitectónicos, así como la evaluación de las alternativas presentadas. La evaluación provee indicadores que permiten, en las fases tempranas, la oportunidad de resolver problemas que pueden presentarse a nivel arquitectónico. Resulta interesante combinar este planteamiento con el presentado por Kazman (1996), con la intención de ampliar la gama de requerimientos que deben ser considerados en la herramienta para evaluación de arquitecturas de software.

Independientemente de la metodología implementada, la intención es obtener una arquitectura con la documentación necesaria, y asegurar que el sistema cumple con los servicios y la funcionalidad que espera el usuario, además de los atributos de calidad asociados que deben cumplirse, y que dirigen las decisiones al momento de la construcción de la arquitectura del sistema (Bredemeyer et al., 2002).

En esta Guía de Estudio se compilas los conceptos relacionados con la temática de Evaluación de Arquitecturas de Software.

1. Calidad de Software

En un mundo cada vez más globalizado, donde cada día desaparecen las barreras comerciales y culturales, la calidad aparece como una necesidad, pues la calidad permite competir con mayores posibilidades de éxito. A pesar de que la programación de sistemas no había sido concebida desde la perspectiva de la calidad, la calidad en productos de software ha tenido un auge importante en la sociedad informatizada de hoy (Azuma, 1999).

La Calidad de Software para Pressman (2002) es "la concordancia con los requisitos funcionales y de rendimiento establecidos con los estándares de desarrollo explícitamente documentados y con las características implícitas que se espera de todo software desarrollado de forma profesional". No obstante la ISO/IEC (*Intenational Standart Organitation* u Organización Internacional de Estándares en español) define a la calidad de software como la totalidad de rasgos y atributos de un producto de software que le apoyan en su capacidad de satisfacer sus necesidades explícitas o implícitas (ISO/IEC 9126, 1998).

Lo más interesante en estas dos definiciones de la Calidad de Software, es la necesidad de que un software de calidad debe satisfacer los requerimientos dados por el usuario. Ahora bien, la IEEE, citado por (Barbacci et al, 1995) afirma que la calidad de un software es el grado en el cual el software posee una combinación deseada de factores.

Pero sin un proceso sistemático que permita garantizar la calidad dentro de un proceso de desarrollo de software no tendría ninguna importancia hablar de Calidad de Software; por eso es vital tratar el tema de Aseguramiento de la Calidad de Software, que es este proceso.

1.1 Aseguramiento de la Calidad de Software

Denominado en la mayoría de las empresas como QA o *Quality Assurance*, permite elaborar actividades sistemáticas que se necesitan para lograr la calidad en el producto, que en este caso es un software (Pressman, 2002). Obviamente esta planificación debe hacerse antes del desarrollo del software.

También Pressman (2002) afirma que el Aseguramiento de la Calidad puede tener las siguientes actividades: evaluaciones en las etapas del desarrollo, auditorias y revisiones, estándares que se aplicarían al proyecto, mecanismos de medida (métricas), métodos y herramientas de análisis, diseño, programación y prueba, y documentación y control de software.

Esto, sin duda alguna, reviste una gran importancia en el proceso de elaboración del software ya que permite proponer y desarrollar las

herramientas necesarias para garantizar la calidad. Por eso estas actividades deben hacerse en la etapa inicial del proceso. El Aseguramiento de la Calidad propicia ciertos atributos y características que influyen significativamente en la Calidad de Software.

2. Arquitectura de Software

Actualmente en la literatura (Bass et al., 1998; Kazman et al., 2001; Hofmeister et al., 2000; Lane, 1990; Buschman et al., 1996; Booch et al., 1999; Abowd, 1995), 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, y por ende, en un ambiente de evaluación de arquitectura de software.

2.1. Importancia de la Arquitectura de Software

La necesidad del manejo de la arquitectura de un sistema de software nace con los sistemas de mediana o gran envergadura, que se proponen como solución para un problema determinado. En la medida que los sistemas de software crecen en complejidad, bien sea por número de requerimientos o por el impacto de los mismos, se hace necesario establecer medios para el manejo de esta complejidad (Hofmeister et al., 1996). En general, la técnica es descomponer el sistema en piezas que agrupan aspectos específicos del mismo, producto de un proceso de abstracción (Bass et al., 1998) y que al organizarse de cierta manera constituyen la base de la solución de un problema en particular.

De aquí que la mayoría de los autores (Bass et al., 1998; Kazman et al., 1998; Hofmeister et al., 1995; Lane, 1990; Buschman et al., 1996; Booch et al., 1999; Abowd, 1995) coinciden en que una arquitectura de software define la *estructura del sistema*. Esta estructura se constituye de *componentes* -módulos o piezas de código- que nacen de la noción de abstracción, cumpliendo funciones específicas, e interactuando entre sí con un comportamiento definido (Bass et al., 1998; Hayes-Roth, 1995; Hofmeister et al., 2000; Buschman et al., 1996; Booch et al., 1999; Abowd, 95). Los componentes se organizan de acuerdo a ciertos criterios, que representan decisiones de diseño. En este sentido, hay autores que plantean que la arquitectura de software incluye *justificaciones* referentes a la organización y el tipo de componentes, garantizando que la configuración resultante satisface los requerimientos del sistema (Boehm et al., 1995).

De esta manera, la arquitectura de software puede ser vista como la estructura del sistema en función de la definición de los componentes y sus interacciones (Bass et al., 1998). La práctica ha demostrado que resulta importante extender el concepto considerando los *requerimientos* y *restricciones* del sistema (Boehm et al., 1995; Lane, 1990), junto a un *argumento* que justifique que la estructura definida satisface los requerimientos, dándole un sentido más amplio a la definición del término.

La arquitectura de software puede considerarse entonces como el "puente" entre los *requerimientos* del sistema y la implementación (Hofmeister et al., 2000). Las actividades que culminan en la definición de la arquitectura pueden ubicarse en las fases tempranas del ciclo de desarrollo del sistema: luego del análisis de los requerimientos y el análisis de riesgos, y justo antes del diseño detallado. Desde esta perspectiva, la arquitectura constituye un *artefacto* de la actividad de diseño (Hofmeister et al., 2000), que servirá de *medio de comunicación* entre los miembros del equipo de desarrollo, los clientes y usuarios finales, dado que contempla los aspectos que interesan a cada uno (Kazman et al., 2001). Además, pasa a ser la base del diseño del sistema a desarrollar, razón por la cual en la literatura la arquitectura es considerada como plan de diseño del sistema (Hofmeister et al., 2000), debido a que es usada como guía para el resto de las tareas del desarrollo.

De igual manera, serán de particular importancia las *propiedades no funcionales* del sistema de software, pues influyen notoriamente en la calidad del mismo. Estas propiedades tienen un gran impacto en el desarrollo y mantenimiento del sistema, su operabilidad y el uso que éste haga de los recursos (Buschman et al., 1996). Entre las propiedades no funcionales más importantes se encuentran: modificabilidad, eficiencia, mantenibilidad, interoperabilidad, confiabilidad, reusabilidad y facilidad de ejecución de pruebas (Kazman et al., 2001). Bass et al. (1998) proponen que el término "requerimiento no funcional" es disfuncional, debido a que implica que tal requerimiento no existe, o que es una especie de requerimiento que puede ser especificado independientemente del comportamiento del sistema. En este sentido, Bass indica que debe hacerse referencia a atributos de calidad, en lugar de propiedades no funcionales.

Puede observarse que al hablar de arquitectura de software, se hace alusión a la especificación de la estructura del sistema, entendida como la organización de componentes y relaciones entre ellos; los requerimientos que debe satisfacer el sistema y las restricciones a las que está sujeto, así como las propiedades no funcionales del sistema y su impacto sobre la calidad del mismo; las reglas y decisiones de diseño que gobiernan esta estructura y los argumentos que justifican las decisiones tomadas.

Habiendo aclarado el alcance que puede tomar el término arquitectura de software, resulta de gran interés introducir formalmente otros términos que resultan pilares fundamentales dentro del contexto de arquitectura, dado que en torno a ellos gira gran parte del estudio que hasta el momento se ha realizado sobre el tema. Tal es el caso de los componentes, los conectores y las relaciones.

2.2. Componentes, conectores y relaciones

Se entiende por *componentes* los bloques de construcción que conforman las partes de un sistema de software. A nivel de lenguajes de programación, pueden ser representados como módulos, clases, objetos o un conjunto de funciones relacionadas

(Buschman et al., 1996). La noción de componente puede llegar a ser muy amplia: el término puede ser utilizado para especificar un conjunto de componentes.

Se distinguen tres tipos de componentes (Perry y Wolf, 1992), denominados también *elementos,* que son:

- ✓ *Elementos de Datos:* contienen la información que será transformada.
- ✓ *Elementos de Proceso*: transforman los elementos de datos.
- ✓ *Elementos de Conexión:* llamados también *conectores,* que bien pueden ser elementos de datos o de proceso, y mantienen unidas las diferentes piezas de la arquitectura.

Una *relación* es la conexión entre los componentes (Buschman et al., 1996). Puede definirse también como una abstracción de la forma en que los componentes interactúan en el sistema a través de los elementos de conexión. Es importante distinguir que una relación se concreta mediante conectores.

Según Bass et al. (1998), en virtud de que está conformado por componentes y relaciones entre ellos, todo sistema, por muy simple que sea, tiene asociada una arquitectura. Sin embargo, no es necesariamente cierto que esta arquitectura es conocida por todos los involucrados en el desarrollo del mismo. Esto hace evidente la diferencia entre la arquitectura del sistema y su descripción. Esta particularidad propone la importancia de la representación de una arquitectura.

Además de los componentes y conectores, Kazman et al. (2001) contemplan las propiedades externamente visibles que comprenden los componentes del software, y las relaciones entre estos. En este sentido, las propiedades externamente visibles hacen referencia a servicios que los componentes proveen, características de desempeño, manejo de fallas, uso de recursos compartidos, etc. En relación a los componentes definidos por la arquitectura de un sistema de software, se tiene la información referente a las interacciones, que son propias de la arquitectura, y que permiten, a nivel de diseño, tomar las decisiones necesarias durante la construcción de un sistema de software.

Kazman et al. (2001) presentan la arquitectura de software como el resultado de decisiones tempranas de diseño, necesarias antes de la construcción del sistema. Según Bass et al. (1998), uno de los aspectos importantes de una arquitectura de software es que, por ser un artefacto de diseño, direcciona atributos de calidad asociados al sistema. Kazman et al. (2001) proponen que las arquitecturas facilitan o inhiben estos atributos. Es por ello que se propone el estudio de los atributos de calidad asociados a la arquitectura de un sistema de software, y cuál es su impacto sobre el mismo.

3. CALIDAD ARQUITECTÓNICA

Barbacci et al. (1995) establecen que el desarrollo de formas sistemáticas para relacionar atributos de calidad de un sistema a su arquitectura provee una base para la toma de decisiones objetivas sobre acuerdos de diseño y permite a los ingenieros realizar predicciones razonablemente exactas sobre los atributos del sistema que son libres de prejuicios y asunciones no triviales. El objetivo de fondo es lograr la

habilidad de evaluar cuantitativamente y llegar a acuerdos entre múltiples atributos de calidad para alcanzar un mejor sistema de forma global.

3.1. Atributos de Calidad

Según Barbacci et al. (1995) la *calidad de software* se define como el grado en el cual el software posee una combinación deseada de atributos. Tales atributos son requerimientos adicionales del sistema (Kazman et al., 2001), que hacen referencia a características que éste debe satisfacer, diferentes a los requerimientos funcionales. Estas características o atributos se conocen con el nombre de *atributos de calidad*, los cuales se definen como las propiedades de un servicio que presta el sistema a sus usuarios (Barbacci et al. 1995).

A grandes rasgos, Bass et al. (1998) establece una clasificación de los atributos de calidad en dos categorías:

- ✓ *Observables vía ejecución*: aquellos atributos que se determinan del comportamiento del sistema en tiempo de ejecución. La descripción de algunos de estos atributos se presenta en la tabla 1.
- ✓ *No observables vía ejecución*: aquellos atributos que se establecen durante el desarrollo del sistema. La descripción de algunos de estos atributos se presenta en la tabla 2.

Atributo de Calidad	**Descripción**
Disponibilidad *(Availability)*	Es la medida de disponibilidad del sistema para el uso (Barbacci et al., 1995).
Confidencialidad *(Confidentiality)*	Es la ausencia de acceso no autorizado a la información (Barbacci et al., 1995).
Funcionalidad *(Functionality)*	Habilidad del sistema para realizar el trabajo para el cual fue concebido (Kazman et al., 2001).
Desempeño *(Performance)*	Es el grado en el cual un sistema o componente cumple con sus funciones designadas, dentro de ciertas restricciones dadas, como velocidad, exactitud o uso de memoria. (IEEE 610.12). Según Smith (1993), el desempeño de un sistema se refiere a aspectos temporales del comportamiento del mismo. Se refiere a capacidad de respuesta, ya sea el tiempo requerido para responder a aspectos específicos o el número de eventos procesados en un intervalo de tiempo. Según Bass et al. (1998), se refiere además a la cantidad de comunicación e interacción existente entre los componentes del sistema.
Confiabilidad *(Reliability)*	Es la medida de la habilidad de un sistema a mantenerse operativo a lo largo del tiempo (Barbacci et al., 1995).
Seguridad externa *(Safety)*	Ausencia de consecuencias catastróficas en el ambiente. Es la medida de ausencia de errores que generan pérdidas de información (Barbacci et al., 1995).
Seguridad interna *(Security)*	Es la medida de la habilidad del sistema para resistir a intentos de uso no autorizados y negación del servicio, mientras se sirve a usuarios legítimos (Kazman et al., 2001).

Tabla 1. Descripción de atributos de calidad observables vía ejecución

Es importante destacar que tener conocimiento de los atributos observables, no necesariamente implica que se satisfacen los atributos no observables vía ejecución. Por ejemplo, un sistema que satisface todos los requerimientos observables puede o no ser costoso de desarrollar, así como también puede o no ser imposible de modificar. De igual manera, un sistema altamente modificable puede o no arrojar resultados correctos.

Atributo de Calidad	Descripción
Configurabilidad *(Configurability)*	Posibilidad que se otorga a un usuario experto a realizar ciertos cambios al sistema (Bosch et al., 1999).
Integrabilidad *(Integrability)*	Es la medida en que trabajan correctamente componentes del sistema que fueron desarrollados separadamente al ser integrados. (Bass et al. 1998)
Integridad *(Integrity)*	Es la ausencia de alteraciones inapropiadas de la información (Barbacci et al., 1995).
Interoperabilidad *(Interoperability)*	Es la medida de la habilidad de que un grupo de partes del sistema trabajen con otro sistema. Es un tipo especial de *integrabilidad* (Bass et al. 1998)
Modificabilidad *(Modifiability)*	Es la habilidad de realizar cambios futuros al sistema. (Bosch et al. 1999).
Mantenibilidad *(Maintainability)*	Es la capacidad de someter a un sistema a reparaciones y evolución (Barbacci et al., 1995). Capacidad de modificar el sistema de manera rápida y a bajo costo (Bosch et al. 1999).
Portabilidad *(Portability)*	Es la habilidad del sistema para ser ejecutado en diferentes ambientes de computación. Estos ambientes pueden ser hardware, software o una combinación de los dos (Kazman et al., 2001).
Reusabilidad *(Reusability)*	Es la capacidad de diseñar un sistema de forma tal que su estructura o parte de sus componentes puedan ser reutilizados en futuras aplicaciones (Bass et al. 1998).
Escalabilidad *(Scalability)*	Es el grado con el que se pueden ampliar el diseño arquitectónico, de datos o procedimental (Pressman, 2002).
Capacidad de Prueba *(Testability)*	Es la medida de la facilidad con la que el software, al ser sometido a una serie de pruebas, puede demostrar sus fallas. Es la probabilidad de que, asumiendo que tiene al menos una falla, el software fallará en su próxima ejecución de prueba (Bass et al. 1998).

Tabla 2. Descripción de atributos de calidad no observables vía ejecución

Bosch (2000) establece que los requerimientos de calidad se ven altamente influenciados por la arquitectura del sistema. Al respecto, Bass et al. (1998) afirman que la calidad del sistema debe ser considerada en todas las fases del diseño, pero los atributos de calidad se manifiestan de maneras distintas a lo largo de estas fases. De esta forma, establecen que la arquitectura determina ciertos atributos de calidad del sistema, pero existen otros atributos que no dependen directamente de la misma. Por ejemplo, la usabilidad de un sistema no está relacionada directamente con la arquitectura del mismo, puesto que los detalles que este atributo envuelve - como el uso de botones o *radio-buttons*, pantallas intuitivas, etc. - se encuentran casi siempre encapsulados en un componente simple.

Independientemente de esto, es importante tener en cuenta que no puede lograrse la satisfacción de ciertos atributos de calidad de manera aislada. Encontrar un atributo de calidad puede tener efectos positivos o negativos sobre otros atributos que, de alguna manera, también se desean alcanzar (Bass et al., 1998).

En su mayoría, los atributos de calidad se pueden organizar y descomponer de maneras diferentes, en lo que se conoce como *modelos de calidad*. Los modelos de calidad de software facilitan el entendimiento del proceso de la ingeniería de software (Pressman, 2002).

3.2. Modelos de Calidad

En la práctica, los modelos calidad resultan de utilidad para la predicción de confiabilidad y en la gerencia de calidad durante el proceso de desarrollo, así como para efectuar la medición del nivel de complejidad de un sistema de software (Kan et al., 1994). Es interesante destacar que la organización y descomposición de los atributos de calidad ha permitido el establecimiento de modelos específicos para efectos de la evaluación de la calidad arquitectónica.

Pressman (2002) indica que los factores que afectan a la calidad del software no cambian, por lo que resulta útil el estudio de los modelos de calidad que han sido propuestos en este sentido desde los años 70. Dado que los factores de calidad presentados para ese entonces siguen siendo válidos, se estudiarán los modelos más importantes propuestos hasta ahora: McCall (1977), Dromey (1996), FURPS (1987), ISO/IEC 9126 (1991) e ISO/IEC 9126 adaptado para arquitecturas de software, propuesto por Losavio et al. (2003).

3.2.1. Modelo de McCall

El modelo de McCall et al. (1977) describe la calidad como un concepto elaborado mediante relaciones jerárquicas entre factores de calidad, en base a criterios y métricas de calidad. Este enfoque es sistemático, y permite cuantificar la calidad a través de las siguientes fases:

- ✓ Determinación de los factores que influyen sobre la calidad del software
- ✓ Identificación de los criterios para juzgar cada factor
- ✓ Definición de las métricas de los criterios y establecimiento de una función de normalización que define la relación entre las métricas de cada criterio y los factores correspondientes
- ✓ Evaluación de las métricas
- ✓ Correlación de las métricas a un conjunto de guías que cualquier equipo de desarrollo podría seguir
- ✓ Desarrollo de las recomendaciones para la colección de métricas

En el modelo de McCall, los factores de calidad se concentran en tres aspectos importantes de un producto de software: características operativas, capacidad de cambios y adaptabilidad a nuevos entornos.

En este modelo, el término *factor de calidad* define características claves que un producto debe exhibir. Los atributos del factor de calidad que define el producto son los nombrados *criterios de calidad*. Las *métricas de calidad* denotan una medida que

puede ser utilizada para cuantificar los criterios. McCall et al. (1977) identifica una serie de criterios, tales como rastreabilidad, simplicidad, capacidad de expansión, etc. Las métricas desarrolladas están relacionadas con los factores de calidad y la relación que se establece se mide en función del grado de cumplimiento de los criterios.

La tabla 3. muestra, para el modelo de McCall, los factores de calidad y sus criterios asociados. En ella se observa que algunos de los criterios son compartidos por más de un factor.

Factor	Criterio
Correctitud	✓ Rastreabilidad ✓ Completitud ✓ Consistencia
Confiabilidad	✓ Consistencia ✓ Exactitud ✓ Tolerancia a fallas
Eficiencia	✓ Eficiencia de ejecución ✓ Eficiencia de almacenamiento
Integridad	✓ Control de acceso ✓ Auditoría de acceso
Usabilidad	✓ Operabilidad ✓ Entrenamiento ✓ Comunicación
Mantenibilidad	✓ Simplicidad ✓ Concreción
Capacidad de Prueba	✓ Simplicidad ✓ Instrumentación ✓ Auto-descriptividad ✓ Modularidad
Flexibilidad	✓ Auto-descriptividad ✓ Capacidad de expansión ✓ Generalidad ✓ Modularidad
Portabilidad	✓ Auto-descriptividad ✓ Independencia del sistema ✓ Independencia de máquina
Reusabilidad	✓ Auto-descriptividad ✓ Generalidad ✓ Modularidad ✓ Independencia del sistema ✓ Independencia de máquina
Interoperabilidad	✓ Modularidad ✓ Similitud de comunicación ✓ Similitud de datos.

Tabla 3. Criterios asociados a los factores de calidad - Modelo de McCall
Fuente: (McCall et al.,1977)

3.2.2. Modelo de Dromey

Dromey (1996) propuso un marco de referencia – o metamodelo - para la construcción de modelos de calidad, basado en cómo las propiedades medibles de un

producto de software pueden afectar los atributos de calidad generales, como por ejemplo, confiabilidad y mantenibilidad. El problema que se plantea es cómo conectar tales propiedades del producto con los atributos de calidad de alto nivel. Para solventar esta situación, Dromey (1996) sugiere el uso de cuatro categorías que implican propiedades de calidad, que son: correctitud, internas, contextuales y descriptivas.

La tabla 4. presenta la relación que establece Dromey (1996) entre las propiedades de calidad del producto y los atributos de calidad de alto nivel.

Propiedades del producto	**Atributos de Calidad**
Correctitud	✓ Funcionalidad ✓ Confiabilidad
Internas	✓ Mantenibilidad ✓ Eficiencia ✓ Confiabilidad
Contextuales	✓ Mantenibilidad ✓ Reusabilidad ✓ Portabilidad ✓ Confiabilidad
Descriptivas	✓ Mantenibilidad ✓ Reusabilidad ✓ Portabilidad ✓ Usabilidad

Tabla 4. Relación entre propiedades del producto y atributos de calidad – Modelo de Dromey
Fuente: (Dromey, 1996)

El proceso de construcción de modelos de calidad propuesto por Dromey (1996) consta de 5 pasos, basados en las propiedades mencionadas. Los pasos del marco de referencia propuesto son:

- ✓ Especificación de los atributos de calidad de alto nivel (por ejemplo, confiabilidad, mantenibilidad)
- ✓ Determinación de los distintos componentes del producto a un apropiado nivel de detalle (por ejemplo, paquetes, subrutinas, declaraciones)
- ✓ Para cada componente, determinación y categorización de sus implicaciones más importantes de calidad
- ✓ Proposición de enlaces que relacionan las propiedades implícitas a los atributos de calidad, o, alternativamente, uso de enlaces de las cuatro categorías de atributos propuestas
- ✓ Iteración sobre los pasos anteriores, utilizando un proceso de evaluación y refinamiento

Para ilustrar sus planteamientos, Dromey (1996) demuestra el uso de su procedimiento para la construcción de un modelo de calidad de implementación, un modelo de calidad de requerimientos, y un modelo de calidad de diseño.

3.2.3. Modelo FURPS

El modelo de McCall ha servido de base para modelos de calidad posteriores, y este es el caso del modelo FURPS, producto del desarrollo de Hewlett-Packard (Grady et al., 1987). En este modelo se desarrollan un conjunto de factores de calidad de software, bajo el acrónimo de FURPS: funcionalidad *(Functionality)*, usabilidad *(Usability)*, confiabilidad *(Reliability)*, desempeño *(Performance)* y capacidad de soporte *(Supportability)*. La tabla 5 presenta la clasificación de los atributos de calidad que se incluyen en el modelo, junto con las características asociadas a cada uno (Pressman, 2002).

Factor de Calidad	Atributos
Funcionalidad	✓ Características y capacidades del programa ✓ Generalidad de las funciones ✓ Seguridad del sistema
Facilidad de uso	✓ Factores humanos ✓ Factores estéticos ✓ Consistencia de la interfaz ✓ Documentación
Confiabilidad	✓ Frecuencia y severidad de las fallas ✓ Exactitud de las salidas ✓ Tiempo medio de fallos ✓ Capacidad de recuperación ante fallas ✓ Capacidad de predicción
Rendimiento	✓ Velocidad del procesamiento ✓ Tiempo de respuesta ✓ Consumo de recursos ✓ Rendimiento efectivo total ✓ Eficacia
Capacidad de Soporte	✓ Extensibilidad ✓ Adaptabilidad ✓ Capacidad de pruebas ✓ Capacidad de configuración ✓ Compatibilidad ✓ Requisitos de instalación

Tabla 5. Atributos de calidad – Modelo FURPS
Fuente: (Pressman, 2002)

El modelo FURPS incluye, además de los factores de calidad y los atributos, restricciones de diseño y requerimientos de implementación, físicos y de interfaz (Grady et al., 1987). Las restricciones de diseño especifican o restringen el diseño del sistema. Los requerimientos de implementación especifican o restringen la codificación o construcción de un sistema (por ejemplo, estándares requeridos, lenguajes, políticas). Por su parte, los requerimientos de interfaz especifican el comportamiento de los elementos externos con los que el sistema debe interactuar. Por último, los requerimientos físicos especifican ciertas propiedades que el sistema debe poseer, en términos de materiales, forma, peso, tamaño (por ejemplo, requisitos de hardware, configuración de red).

3.2.4. Modelo ISO/IEC 9126

El estándar ISO/IEC 9126 ha sido desarrollado en un intento de identificar los atributos clave de calidad para un producto de software (Pressman, 2002). Este estándar es una simplificación del Modelo de McCall (Losavio et al., 2003), e identifica seis características básicas de calidad que pueden estar presentes en cualquier producto de software. El estándar provee una descomposición de las características en subcaracterísticas, que se muestran en la tabla 6.

Característica	Subcaracterística
Funcionalidad	✓ Adecuación ✓ Exactitud ✓ Interoperabilidad ✓ Seguridad
Confiabilidad	✓ Madurez ✓ Tolerancia a fallas ✓ Recuperabilidad
Usabilidad	✓ Entendibilidad ✓ Capacidad de aprendizaje ✓ Operabilidad
Eficiencia	✓ Comportamiento en tiempo ✓ Comportamiento de recursos
Mantenibilidad	✓ Analizabilidad ✓ Modificabilidad ✓ Estabilidad ✓ Capacidad de pruebas
Portabilidad	✓ Adaptabilidad ✓ Instalabilidad ✓ Reemplazabilidad

Tabla 6. Características y subcaracterísticas de calidad – Modelo ISO/IEC 9126
Fuente: (Pressman, 2002)

Es interesante destacar que los factores de calidad que contempla el estándar ISO/IEC 9126 no son necesariamente usados para mediciones directas (Pressman, 2002), pero proveen una valiosa base para medidas indirectas, y una excelente lista para determinar la calidad de un sistema.

3.2.5. ISO/IEC 9126 adaptado para arquitecturas de software

Losavio et al. (2003) proponen una adaptación del modelo ISO/IEC 9126 de calidad de software para efectos de la evaluación de arquitecturas de software. El modelo se basa en los atributos de calidad que se relacionan directamente con la arquitectura: funcionalidad, confiabilidad, eficiencia, mantenibilidad y portabilidad.

Los autores plantean que la característica usabilidad propuesta por el modelo ISO/IEC 9126 puede ser refinada para obtener atributos que se relacionan con los componentes de la interfaz con el usuario. Dado que estos componentes son independientes de la arquitectura, no son considerados en la adaptación del modelo.

La tabla 7 presenta los atributos de calidad planteados por Losavio et al. (2003), que poseen subcaracterísticas asociadas con elementos de tipo arquitectónico.

Característica	Subcaracterística	Elementos de tipo arquitectónico
Funcionalidad	Adecuación	Refinamiento de los diagramas de secuencia
	Exactitud	Identificación de los componentes con las funciones responsables de los cálculos
	Interoperabilidad	Identificación de conectores de comunicación con sistemas externos
	Seguridad	Mecanismos o dispositivos que realizan explícitamente la tarea
Confiabilidad	Tolerancia a fallas	Existencia de mecanismos o dispositivos de software para manejar excepciones
	Recuperabilidad	Existencia de mecanismos o dispositivos de software para reestablecer el nivel de desempeño y recuperar datos
Eficiencia	Desempeño	Componentes involucrados en un flujo de ejecución para una funcionalidad
	Utilización de recursos	Relación de los componentes en términos de espacio y tiempo
Mantenibilidad	Acoplamiento	Interacciones entre componentes
	Modularidad	Número de componentes que dependen de un componente
Portabilidad	Adaptabilidad	Presencia de mecanismos de adaptación
	Instalabilidad	Presencia de mecanismos de instalación
	Coexistencia	Presencia de mecanismos que faciliten la coexistencia
	Reemplazabilidad	Lista de componentes reemplazables para cada componente

Tabla 7. Atributos de calidad planteados por Losavio et al. (2003), que poseen subcaracterísticas asociadas con elementos de tipo arquitectónico.

En la literatura es posible encontrar planteamientos en relación al establecimiento de los atributos de calidad y su relación con la arquitectura de software (Bass et al., 2000). Kazman et al. (2001) establecen que la arquitectura determina atributos de calidad. Bass et al. (2000) explica la relación existente entre estos, conjuntamente con los problemas y beneficios de la documentación de esta relación.

3.3. Relación entre Arquitectura de Software y Atributos de Calidad

Bass et al. (2000) establecen que 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, establecen que al considerar una decisión de arquitectura de software, el arquitecto se pregunta cuál será el impacto de ésta sobre ciertos atributos; por ejemplo, modificabilidad, desempeño, seguridad, usabilidad, etc.

Por esta razón, Bass et al. (2000) afirman 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 afecte a los atributos de calidad, se tienen dos consecuencias:

- ✓ 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.

En su planteamiento, Bass et al. (2000) presentan los problemas que existen en relación a la documentación de la relación entre arquitectura de software y atributos de calidad. De igual forma, establecen que la relación entre la arquitectura de software y los atributos de calidad no se encuentra sistemática y completamente documentada, debido a diversas razones:

- ✓ 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.

A pesar de estas dificultades, Bass et al. (2000) establecen que la relación entre arquitectura de software y atributos de calidad tiene diversos beneficios, que justifican su documentación:

- ✓ Realza en gran medida el proceso de análisis y diseño arquitectónico, puesto que el arquitecto puede reutilizar análisis existentes y determinar acuerdos explícitamente en lugar de hacerlo sobre la marcha. Los arquitectos experimentados hacen esto intuitivamente, basados en su experiencia en codificación. Por ejemplo, durante el análisis, un arquitecto puede reconocer el impacto de la codificación de una estructura en los atributos de calidad.

- ✓ Una vez que el arquitecto entiende el impacto de los componentes arquitectónicos sobre uno o varios atributos de calidad, estaría en capacidad de reemplazar un conjunto de componentes por otro cuando lo considere necesario.
- ✓ Una vez que se codifica la relación entre arquitectura y atributos de calidad, es posible construir un protocolo de pruebas que habilitará la certificación por parte de terceros.

Por todo lo expuesto, se reconoce la importancia de la arquitectura de un sistema de software como la base de diseño de un sistema (Kruchten, 1999), así como también un artefacto que determina atributos de calidad (Kazman et al., 2001).

Ahora bien, es interesante considerar que tanto la arquitectura de un sistema de software como el sistema en sí mismo, se encuentran íntimamente relacionados con su entorno, por lo que es necesario tomarlo en cuenta para efectos del estudio de la calidad y la determinación de los atributos de calidad.

Las características adicionales del sistema o atributos de calidad, están estrechamente relacionadas con el uso que se le dará al sistema propuesto (Kazman et al., 2001), pues es el contexto quien define el comportamiento del mismo, sin dejar de lado la funcionalidad (Barbacci et al., 1997). Por ello es necesario analizar el contexto del sistema, dado que de aquí se deriva mucha información relativa a su calidad.

El sistema y su entorno son compañeros en un contrato en el cual cada uno tiene expectativas y obligaciones (Barbacci et al., 1997). En muchos casos, el nivel de compromiso entre el sistema y su entorno no queda establecido de forma clara con el simple análisis de los requerimientos funcionales de un sistema. En este sentido, Barbacci et al. (1997) proponen la necesidad del análisis de toda la información disponible, tanto del sistema como del contexto.

Barbacci et al. (1997) plantean un esquema general que permite establecer elementos que deben ser tomados en consideración y que vienen dados por distintos tipos de actividades. Por ejemplo, con base en las actividades del sistema y su importancia, para efectos del sistema y su entorno, es posible determinar que, propiedades como el desempeño o la disponibilidad del sistema, deben ser consideradas como atributos de calidad para el diseño de la arquitectura del mismo. De igual forma, Barbacci et al. (1997) proponen el uso de técnicas como los escenarios, listas de chequeo y cuestionarios, como formas cualitativas de determinar el nivel del contrato, elementos de hardware y la arquitectura de software, teniendo como objetivo principal velar por la calidad del sistema.

A lo largo del proceso de diseño y desarrollo, los atributos de calidad juegan un papel importante, pues en base a estos se generan las decisiones de diseño y argumentos que los justifican (Bass et al., 2000). Dado que la arquitectura de software inhibe o facilita los atributos de calidad (Bass et al., 1998), 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*, los *patrones arquitectónicos* y los *patrones de diseño*.

4. ESTILOS Y PATRONES

Bosch (2000) establecen que la imposición de ciertos estilos arquitectónicos mejora o disminuye las posibilidades de satisfacción de ciertos atributos de calidad del sistema. Con esto afirman que cada estilo propicia atributos de calidad, y la decisión de implementar alguno de los existentes depende de los requerimientos de calidad del sistema. De manera similar, plantean el uso de los patrones arquitectónicos y los patrones de diseño para mejorar la calidad del sistema. Al respecto, Buschmann et al. (1996) afirman que un criterio importante del éxito de los patrones - tanto arquitectónicos como de diseño - es la forma en que estos alcanzan de manera satisfactoria los objetivos de la ingeniería de software. Los patrones soportan el desarrollo, mantenimiento y evolución de sistemas complejos y de gran escala.

La diferencia entre *estilos* y *patrones arquitectónicos* no ha sido aclarada. Bengtsson (1999) plantea la existencia de dos grandes vertientes, que surgen de la discusión de los términos. Shaw y Garlan (1996) utilizan indistintamente los términos *estilo arquitectónico* y *patrón arquitectónico*. Por otro lado, Buschmann et al. (1996) establece diferencias sutiles entre ambos conceptos. De cualquier forma, los estilos y los patrones establecen un vocabulario común, y brindan soporte a los ingenieros para conseguir una solución que haya sido aplicada con éxito anteriormente, ante ciertas situaciones de diseño. Además, su aplicación en el diseño de la arquitectura del sistema es determinante para la satisfacción de los requerimientos de calidad.

En vista de la existencia de las dos vertientes, es necesario establecer las posibles diferencias y las razones por las cuales se asume que los términos *estilo arquitectónico* y *patrón arquitectónico* son distintos. De igual manera, se busca resaltar los atributos de calidad propiciados tanto por los estilos como por los patrones arquitectónicos y de diseño.

4.1. Estilo Arquitectónico

Shaw y Garlan (1996) 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.

Buschmann et al. (1996) definen *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.

A simple vista, ambas definiciones parecen expresar la misma idea. La diferencia entre los planteamientos de Shaw y Garlan (1996) y Buschmann et al. (1996) viene dada por la amplitud de la noción de *componente* en cada una de las

definiciones. Buschmann et al. (1996) asume como componentes a subsistemas conformados por otros componentes más sencillos, mientras que Shaw y Garlan utilizan la noción de componente como elementos simples, ya sean de dato o de proceso. En virtud de esto, la diferencia entre ambas definiciones gira en torno al nivel de abstracción, dado que Buschmann et al. (1996) plantean un grado mayor en su concepto de estilo arquitectónico, sugiriendo una estructura genérica para la organización de componentes de ciertas familias de sistemas, independientemente del contexto en que éstas se desarrollen.

La tabla 8 resume los principales estilos arquitectónicos, los atributos de calidad que propician y los atributos que se ven afectados negativamente (atributos en conflicto), de acuerdo a Bass et al. (1998).

Estilo	**Descripción**	**Atributos asociados**	**Atributos en conflicto**
Datos Centralizados	Sistemas en los cuales cierto número de clientes accede y actualiza datos compartidos de un repositorio de manera frecuente.	Integrabilidad Escalabilidad Modificabilidad	Desempeño
Flujo de Datos	El sistema es visto como una serie de transformaciones sobre piezas sucesivas de datos de entrada. El dato ingresa en el sistema, y fluye entre los componentes, de uno en uno, hasta que se le asigne un destino final (salida o repositorio).	Reusabilidad Modificabilidad Mantenibilidad	Desempeño
Máquinas Virtuales	Simulan alguna funcionalidad que no es nativa al hardware o software sobre el que está implementado.	Portabilidad	Desempeño
Llamada y Retorno	El sistema se constituye de un programa principal que tiene el control del sistema y varios subprogramas que se comunican con éste mediante el uso de llamadas.	Modificabilidad Escalabilidad Desempeño	Mantenibilidad Desempeño
Componentes Independientes	Consiste en un número de procesos u objetos independientes que se comunican a través de mensajes.	Modificabilidad Escalabilidad	Desempeño Integrabilidad

Tabla 8. Estilos Arquitectónicos y Atributos de Calidad

Un planteamiento reciente, propuesto por Bass et al. (1999), consiste en los *estilos arquitectónicos basados en atributos* (ABAS), que se establecen como una extensión de la noción de estilo arquitectónico, mediante la asociación de modelos analíticos de atributos de calidad. En este sentido, los autores proponen que estos estilos incluyen un razonamiento cualitativo o cuantitativo, basado en modelos específicos de atributos de calidad. Un *estilo arquitectónico basado en atributos* incluye:

- ✓ La topología de los tipos de componentes y una descripción del patrón de los datos y control de interacción entre ellos, de acuerdo con la definición estándar
- ✓ Un modelo específico de atributos de calidad que provee un método de razonamiento acerca del comportamiento de los tipos de componentes que interactúan en el patrón definido
- ✓ El razonamiento que resulta de la aplicación del modelo específico de atributos de calidad a la interacción de los tipos de componentes

Bass et al. (1999) proponen los estilos arquitectónicos como elementos importantes para el diseño, en tanto estos pueden ser elegidos basándose en el entendimiento de las metas de calidad del sistema en construcción. En este sentido, su planteamiento incluye la extensión del concepto de estilo arquitectónico, incluyendo modelos analíticos de atributos de calidad.

Un estilo arquitectónico basado en atributos (ABAS) consta de cinco partes (Bass et al., 1999), que se muestran en la tabla 9.

Elemento	**Descripción**
Descripción del problema	Describe el problema de diseño que el ABAS pretende resolver, incluyendo el atributo de calidad de interés, el contexto de uso, y requerimientos específicos relevantes al atributo de calidad asociado
Medidas del atributo de calidad	Contiene los aspectos medibles del modelo de atributos de calidad. Incluye una discusión de los eventos que causan que la arquitectura responda o cambie
Estilo Arquitectónico	Descripción del estilo arquitectónico en términos de componentes, conectores, propiedades de los componentes y conexiones, así como patrones de datos y control de interacciones
Parámetros de atributos de calidad	Especificación del estilo arquitectónico en términos de los parámetros del modelo de calidad
Análisis	Descripción de cómo los modelos de atributos de calidad están formalmente relacionados con los elementos del estilo arquitectónico y las conclusiones acerca del comportamiento arquitectónico que se desprende de los modelos

Tabla 9. Partes que conforman un estilo arquitectónico basado en atributos (ABAS)

4.2. Patrón Arquitectónico

Buschmann et al. (1996) define *patrón* como una regla que consta de tres partes, la cual expresa una relación entre un contexto, un problema y una solución. En líneas generales, un patrón sigue el siguiente esquema:

- ✓ *Contexto.* Es una situación de diseño en la que aparece un problema de diseño
- ✓ *Problema.* Es un conjunto de fuerzas que aparecen repetidamente en el contexto
- ✓ *Solución.* Es una configuración que equilibra estas fuerzas. Ésta abarca:
  - Estructura con componentes y relaciones
  - Comportamiento a tiempo de ejecución: aspectos dinámicos de la solución, como la colaboración entre componentes, la comunicación entre ellos, etc.

Partiendo de esta definición, 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.

Así mismo, Buschmann et al. (1996) plantean que los patrones arquitectónicos 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. Propone que 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.

Visto de esta manera, el concepto de *patrón arquitectónico* propuesto por Buschmann et al. (1996) equivale al establecido por Shaw y Garlan (1996) para *estilo arquitectónico*, quienes tratan indistintamente estos dos términos.

Barbacci et al. (1997) hacen la analogía de la construcción de una arquitectura de un sistema complejo como la inclusión de instancias de más de un patrón arquitectónico, compuestos de maneras arbitrarias. La colección de patrones arquitectónicos debe ser estudiada en términos de factores de calidad e intereses, en anticipación a su uso. Esto quiere decir que un patrón puede ser analizado previamente, con la intención de seleccionar el que mejor se adapte a los requerimientos de calidad que debe cumplir el sistema. De manera similar, Barbacci et al. (1997) proponen que debe estudiarse la composición de los patrones, dado que ésta puede dificultar aspectos como el análisis, o poner en conflicto otros atributos de calidad. La tabla 10 presenta algunos patrones arquitectónicos, además de los atributos que propician y los atributos en conflicto, de acuerdo a Buschmann et al. (1996).

Patrón Arquitectónico	**Descripción**	**Atributos asociados**	**Atributos en conflicto**
Layers	Consiste en estructurar aplicaciones que pueden ser descompuestas en grupos de subtareas, las cuales se clasifican de acuerdo a un nivel particular de abstracción.	Reusabilidad Portabilidad Facilidad de Prueba	Desempeño Mantenibilidad
Pipes and Filters	Provee una estructura para los sistemas que procesan un flujo de datos. Cada paso de procesamiento está encapsulado en un componente filtro (*filter*). El dato pasa a través de conexiones (*pipes*), entre filtros adyacentes.	Reusabilidad Mantenibilidad	Desempeño
Blackboard	Aplica para problemas cuya solución utiliza estrategias no determinísticas. Varios subsistemas ensamblan su conocimiento para construir una posible solución parcial ó aproximada.	Modificabilidad Mantenibilidad Reusabilidad Integridad	Desempeño Facilidad de Prueba
Broker	Puede ser usado para estructurar sistemas de software distribuido con componentes desacoplados que interactúan por invocaciones a servicios remotos. Un componente *broker* es responsable de coordinar la comunicación, como el reenvío de solicitudes, así como también la transmisión de resultados y excepciones.	Modificabilidad Portabilidad Reusabilidad Escalabilidad Interoperabilidad	Desempeño

Model-View-Controler	Divide una aplicación interactiva en tres componentes. El modelo *(model)* contiene la información central y los datos. Las vistas *(view)* despliegan información al usuario. Los controladores *(controlers)* capturan la entrada del usuario. Las vistas y los controladores constituyen la interfaz del usuario.	Funcionalidad Mantenibilidad	Desempeño Portabilidad

Tabla 10 (a) Patrones arquitectónicos y atributos de calidad

Patrón Arquitectónico	Descripción	Atributos asociados	Atributos en conflicto
Presentation-Abstraction-Control	Define una estructura para sistemas de software interactivos de agentes de cooperación organizados de forma jerárquica. Cada agente es responsable de un aspecto específico de la funcionalidad de la aplicación y consiste de tres componentes: presentación, abstracción y control.	Modificabilidad Escalabilidad Integrabilidad	Desempeño Mantenibilidad
Microkernel	Aplica para sistemas de software que deben estar en capacidad de adaptar los requerimientos de cambio del sistema. Separa un núcleo funcional mínimo del resto de la funcionalidad y de partes específicas pertenecientes al cliente.	Portabilidad Escalabilidad Confiablidad Disponibilidad	Desempeño
Reflection	Provee un mecanismo para sistemas cuya estructura y comportamiento cambia dinámicamente. Soporta la modificación de aspectos fundamentales como estructuras tipo y mecanismos de llamadas a funciones.	Modificabilidad	Desempeño

Tabla 10 (b). Patrones arquitectónicos y atributos de calidad (Continuación)

Con la intención de hacer una comparación clara entre estilo arquitectónico y patrón arquitectónico, la tabla 11 presenta las diferencias entre estos conceptos, construida a partir del planteamiento de Buschmann et al. (1996).

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 11. Diferencias entre estilo arquitectónico y patrón arquitectónico.

4.3. Patrón de Diseño

Un *patrón de diseño* provee un esquema para refinar los subsistemas o componentes de un sistema de software, o las relaciones entre ellos. Describe la estructura comúnmente recurrente de los componentes en comunicación, que resuelve un problema general de diseño en un contexto particular (Buschman et al., 1996).

Son menores en escala que los patrones arquitectónicos, y tienden a ser independientes de los lenguajes y paradigmas de programación. Su aplicación no tiene efectos en la estructura fundamental del sistema, pero sí sobre la de un subsistema (Buschman et al., 1996), debido a que especifica a un mayor nivel de detalle, sin llegar a la implementación, el comportamiento de los componentes del subsistema. La tabla 12 presenta algunos patrones de diseño, junto a los atributos de calidad que propician y los atributos que entran en conflicto con la aplicación del patrón, según Buschmann et al. (1996).

Patrón de Diseño	Descripción	Atributos asociados	Atributos en conflicto
Whole-Part	Ayuda a constituir una colección de objetos que juntos conforman una unidad semántica.	Reusabilidad Modificabilidad	Desempeño
Master-Slave	Un componente maestro *(master)* distribuye el trabajo a los componentes esclavos *(slaves)*. El componente maestro calcula un resultado final a partir de los resultados arrojados por los componentes esclavos.	Escalabilidad Desempeño	Portabilidad
Proxy	Los clientes asociados a un componente se comunican con un representante de éste, en lugar del componente en sí mismo.	Desempeño Reusabilidad	Desempeño
Command Procesor	Separa las solicitudes de un servicio de su ejecución. Maneja las solicitudes como objetos separados, programa sus ejecuciones y provee servicios adicionales como el almacenamiento de los objetos solicitados, para permitir que el usuario pueda deshacer alguna solicitud.	Funcionalidad Modificabilidad Facilidad de Prueba	Desempeño
View Handler	Ayuda a manejar todas las vistas que provee un sistema de software. Permite a los clientes abrir, manipular y eliminar vistas. También coordina dependencias entre vistas y organiza su actualización.	Escalabilidad Modificabilidad	Desempeño
Forwarder-Receiver	Provee una comunicación transparente entre procesos de un sistema de software con un modelo de interacción punto a punto *(peer to peer).*	Mantenibilidad Modificabilidad Desempeño	Configurabilidad
Client-Dispatcher-Server	Introduce una capa intermedia entre clientes y servidores, es el componente despachador *(dispatcher)*. Provee una ubicación transparente por medio de un nombre de servicio, y esconde los detalles del establecimiento de una conexión de comunicación entre clientes y servidores.	Configurabilidad Portabilidad Escalabilidad Disponibilidad	Desempeño Modificabilidad
Publisher-Subscriber	Ayuda a mantener sincronizados los componentes en cooperación. Para ello, habilita una vía de propagación de cambios: un editor (*publisher*) notifica a los suscriptores (*suscribers*) sobre los cambios en su estado.	Escalabilidad	Desempeño

Tabla 12. Patrones de diseño y atributos de calidad.

La figura 1.1 presenta la relación de abstracción existente entre los conceptos de estilo arquitectónico, patrón arquitectónico y patrón de diseño. En ella se representa el planteamiento de Buschmann et al. (1996), que propone el desarrollo de arquitecturas de software como un sistema de patrones, y distintos niveles de abstracción.

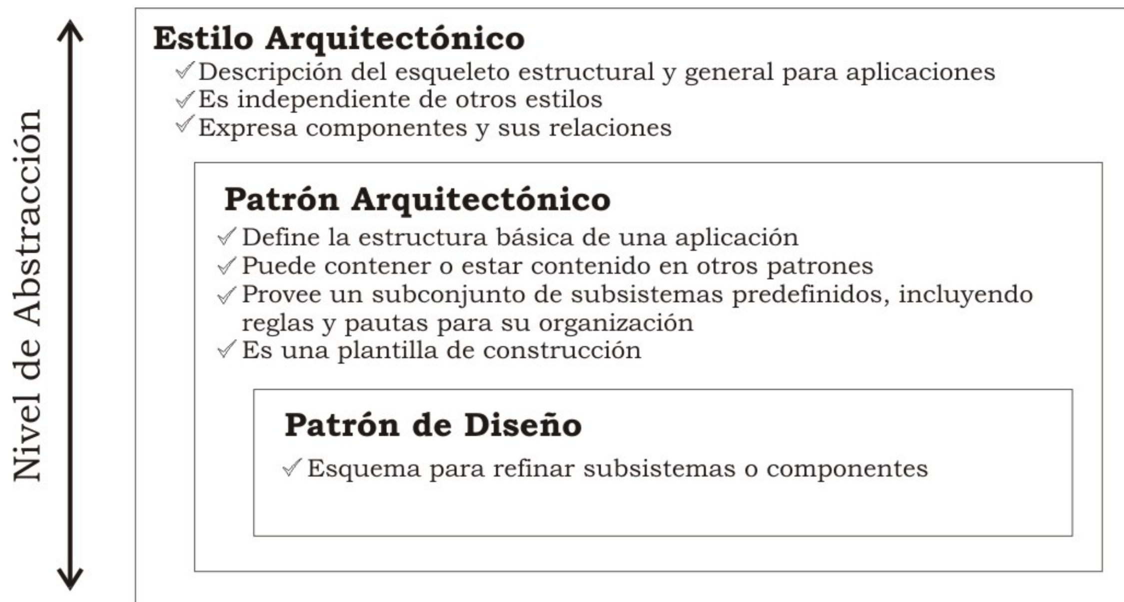

Figura 1.1. Relación de abstracción entre estilos y patrones

Los estilos y patrones ayudan al arquitecto a definir la composición y el comportamiento del sistema de software, y una combinación adecuada de ellos permite alcanzar los requerimientos de calidad.

Ahora bien, la organización del sistema de software debe estar disponible para todos los involucrados en el desarrollo del sistema, ya que establece un mecanismo de comunicación entre los mismos. Tal objetivo se logra mediante la representación de la arquitectura en formas distintas, obteniendo así una descripción de la misma de forma tal que puede ser entendida y analizada por todos los involucrados, con miras a obtener un sistema de calidad. Estas descripciones pueden establecerse a través de las *vistas arquitectónicas*, las *notaciones* como UML y los *lenguajes de descripción arquitectónica* (Bengtsson, 1999).

Las vistas arquitectónicas, a través de distintos niveles de abstracción, resaltan diversos aspectos que conciernen a los involucrados en el desarrollo. Resulta interesante analizar estas perspectivas de una arquitectura, y la forma en que éstas ayudan a todos los involucrados en el proceso de desarrollo a razonar sobre los atributos de calidad del sistema.

5. Vistas Arquitectónicas

De acuerdo al nivel de responsabilidad dentro del desarrollo de un sistema y la relación que se establezca con el mismo, son muchas las partes involucradas e interesadas en la arquitectura de software (Kruchten, 1999), a saber:

- ✓ El *analista del sistema*, quien la utiliza para organizar y expresar claramente los requerimientos y entender las restricciones de tecnología y los riesgos.
- ✓ *Usuarios finales y clientes*, que necesitan conocer el sistema que están adquiriendo.
- ✓ El *gerente de proyecto*, que la utiliza para organizar el equipo y planificar el desarrollo.
- ✓ Los *diseñadores*, que lo utilizan para entender los principios subyacentes y localizar los límites de su propio diseño.
- ✓ Otras *organizaciones desarrolladoras* (si el sistema es abierto), que la utilizan para entender cómo interactuar con el sistema.
- ✓ Las *compañías subcontratadas*, que la utilizan para entender los límites de su sección de desarrollo.
- ✓ Los *arquitectos*, quienes velan por la evolución del sistema y la reutilización de componentes.

Todas estas personas deben comunicarse de manera efectiva para discutir y razonar acerca de la arquitectura, y así alcanzar las metas del desarrollo. En virtud de esto, Kruchten (1999) plantea que debe tenerse una representación del sistema que todos puedan comprender.

Una única representación de la arquitectura del sistema resultaría demasiado compleja y poco útil para todos los involucrados, pues contendría mucha información irrelevante para la mayoría de estos. Es por ello que se plantea la necesidad de representaciones que contengan únicamente elementos que resultan de importancia para cierto grupo de involucrados.

Buschmann et al. (1996) establece que una *vista arquitectónica* representa un aspecto parcial de una arquitectura de software, que muestra propiedades específicas del sistema. Bass et al. (1998), haciendo uso indistinto de los términos estructura y vista, proponen que las *estructuras arquitectónicas* pueden definirse agrupando los componentes y conectores de acuerdo a la funcionalidad del sistema, sincronización y comunicación de procesos, distribución física, propiedades estáticas, propiedades dinámicas y propiedades de ejecución, entre otras.

Por su parte, Kruchten (1999) define una *vista arquitectónica* como una descripción simplificada o abstracción de un sistema desde una perspectiva específica, que cubre intereses particulares y omite entidades no relevantes a esta perspectiva. Para la definición de una vista, Kruchten propone la identificación de ciertos elementos, que se mencionan a continuación:

- ✓ Punto de vista: involucrados e intereses de los mismos
- ✓ Elementos que serán capturados y representados en la vista y las relaciones entre estos
- ✓ Principios para organizar la vista
- ✓ Forma en que se relacionan los elementos de una vista con otras vistas
- ✓ Proceso a ser utilizado para la creación de la vista

Kazman et al. (2001), Bass et al. (1998), Hofmeister et al. (2000) y Kruchten (1999), proponen, en función de las características del sistema o del proceso de desarrollo del mismo, distintas vistas arquitectónicas. Es importante resaltar que las vistas propuestas no son independientes entre sí, puesto que son perspectivas distintas de un mismo sistema (Kruchten, 1999). Por tal motivo, las vistas

arquitectónicas deben estar coordinadas, de manera tal que al realizar cambios, estos se vean correctamente reflejados en las vistas afectadas, garantizando consistencia entre las mismas.

Ante la diversidad de planteamientos sobre las distintas perspectivas de un mismo sistema, resulta interesante establecer comparaciones entre los mismos, puesto que, en algunos casos, hacen referencia a un mismo tipo de perspectiva bajo nombres de vistas distintos, o por el contrario, bajo el mismo nombre expresan perspectivas diferentes. De igual forma, hay vistas que contemplan varias perspectivas, así como también varias vistas pueden crear una única perspectiva (Kazman et al., 2001).

5.1. Comparación de Vistas Arquitectónicas

De acuerdo al análisis realizado, la tabla 13 presenta las similitudes observadas entre las vistas propuestas por Kazman et al. (2001), Bass et al. (1998), Hofmeister et al. (2000) y Kruchten (1999), en función de las perspectivas que éstas ofrecen. Para cada perspectiva se presenta el nombre de la vista planteado por cada uno de los autores analizados. Adicionalmente, se hace referencia a los interesados en cada una de las vistas y las propiedades no funcionales de la arquitectura que maneja cada perspectiva.

Para Bass et al. (1998) cada estructura (o vista arquitectónica) es una abstracción de acuerdo a criterios diferentes que pueden usar su propia notación y definir de forma independiente el significado de los componentes, relaciones, argumentos, principios y pautas.

Por su parte, Hofmeister et al. (1996) proponen cuatro vistas arquitectónicas tomando como punto de partida el estudio de sistemas implementados en la vida real.

Kazman et al. (2001) describen las vistas arquitectónicas distinguiendo los componentes y las relaciones entre estos dentro de cada una, planteando que a menudo es útil combinar la información de dos o más vistas. Por ejemplo, la distribución de los procesos en los componentes de hardware del sistema puede obtenerse de la combinación de la vista de concurrencia y la vista física. De igual manera, podría observarse las diferentes funcionalidades del sistema en los módulos de código de la vista de código, etc. En este sentido, es válido el planteamiento acerca de que no existe un conjunto “canónico” de vistas, de forma tal que es posible seleccionar o crear vistas que contengan información importante acerca de una arquitectura en particular (Kazman et al., 2001).

Así como las vistas arquitectónicas ofrecen una descripción de una arquitectura, existen notaciones que, mediante un conjunto definido de elementos y formas de representación, permiten de igual manera establecer la descripción de una arquitectura de software. Este es el caso del *Unified Modeling Language (UML).*

Perspectiva	Kazman, et al. (2001)	Kruchten (1999)	Hofmeister, et al. (2000)	Bass et al. (1998)	Parte Interesada	Atributo de Calidad
Abstracción de requerimientos funcionales del sistema	Vista Funcional	Vista Lógica	Vista Conceptual	Vista Conceptual o Lógica	Cliente Usuario final Analista	Modificabilidad Reusabilidad Dependencia Seguridad Externa
Creación de procesos e hilos de ejecución, comunicación entre ellos y recursos compartidos.	Vista de Concurrencia	Vista de Proceso	Vista de Ejecución	Vista de Procesos o Coordinación + Vista de Llamadas	Arquitectos Desarrolladores Equipo de Pruebas Mantenimiento	Desempeño Disponibilidad
Organización de los elementos Arquitectónicas implementados.	Vista de Desarrollo	Vista de Implantación	Vista de Código	Vista Física + Vista de Módulos	Programadores Mantenimiento Gerentes de Configuración Gerentes de Desarrollo	Mantenibilidad Modificabilidad Capacidad de Prueba
Distribución de procesos en la plataforma	Vista Física + Vista de Concurrencia	Vista de Desarrollo	Vista de Módulos y Vista de Ejecución	Vista de Flujo de Control	Arquitectos Desarrolladores Equipo de Pruebas Mantenimiento Ing. Hardware	Desempeño Escalabilidad Disponibilidad Seguridad Interna
Escenarios y casos de uso	-	Vista de Casos de Uso	Vista Conceptual	Vista de Usos	Cliente Usuario final Analista	Reusabilidad Disponibilidad Modificabilidad
Especificación abstracta de clases, objetos, funciones y procedimientos.	Vista de Código	-	-	Vista de Clases + Vista de Flujo de Datos	Diseñadores Desarrolladores	Modificabilidad Portabilidad Mantenibilidad

NOTA: el signo “+” indica combinación de vistas.

Tabla 13. Comparación de vistas arquitectónicas en función de las perspectivas del sistema

6. NOTACIONES

Unified Modeling Language (UML) ha conseguido un rol importante en el proceso de desarrollo de software en la actualidad (Booch et al., 1999). La unificación del método de diseño y las notaciones, ha ampliado, entre otras cosas, el mercado de herramientas CASE que soportan el proceso de diseño de arquitecturas de software. UML ofrece soporte para clases, clases abstractas, relaciones, comportamiento por interacción, empaquetamiento, entre otros. Estos elementos se pueden representar mediante nueve tipos de diagramas, que son: de clases, de objetos, de casos de uso, de secuencia, de colaboración, de estados, de actividades, de componentes y de desarrollo.

Bengtsson (1999) presenta las características generales de UML, y las razones por las que resulta interesante su aplicación para efectos de la representación de una arquitectura de software. Establece que en UML existe soporte para algunos de los conceptos asociados a las arquitecturas de software, como los componentes, los paquetes, las librerías y la colaboración. UML permite la descripción de componentes en la arquitectura de software en dos niveles; se puede especificar sólo el nombre del componente o especificar las clases o interfaces que implementan estos.

De igual forma, UML provee una notación para la descripción de la proyección de los componentes de software en el hardware. Esto corresponde a la vista física del modelo 4+1 (Kruchten, 1999). La proyección de los componentes de software permite a los ingenieros de software hacer mejores estimaciones cuando se intenta medir la calidad del sistema implementado, lo cual contribuye a la búsqueda de la mejor solución para un sistema específico. Esta notación puede ser extendida con mayor nivel de detalle y los componentes pueden ser conectados entre sí con el uso de las bondades del lenguaje UML.

La colaboración entre componentes se puede representar mediante conjuntos de clases, interfaces y otros elementos que interactúan entre sí para proveer servicios que van más allá de la funcionalidad de cada uno de ellos por separado. La colaboración posee un aspecto estructural, esto es, el diagrama de clases de los elementos involucrados en la interacción y un diagrama de comportamiento. UML también permite que las colaboraciones posean relaciones entre sí.

Por último, los patrones y frameworks están también soportados por UML, mediante el uso combinado de paquetes, componentes y colaboraciones, entre otros. Booch et al. (1999) proponen de forma detallada todos los aspectos que hacen de UML un lenguaje conveniente para la representación de las arquitecturas de software.

Sin embargo, el problema de la representación de las arquitecturas de software encuentra, aún con un lenguaje como UML, limitaciones de representación (Rausch, 2001). En este sentido, Rausch (2001) propone un lenguaje de especificación de arquitecturas de software basado en UML y OCL (Object Constraint Language).

El planteamiento de Rausch (2001) se basa en que el comportamiento de una arquitectura no se limita a la comunicación que existe entre sus componentes, sino que toda la estructura, la creación y destrucción de instancias y tipos de datos, debe documentarse, y es también punto de discusión del diseño arquitectónico.

El modelo de Rausch (2001) plantea la inclusión de los componentes básicos de una arquitectura: componentes, interfaces, conexiones, atributos, valores de los atributos y mensajes enviados a las interfaces. Todas las instancias nombradas representan las unidades operacionales de la arquitectura, y es necesario especificar su comportamiento. En este sentido, Rausch (2001) propone que deben considerarse tres categorías del comportamiento del sistema: *comportamiento estructural*, que captura los cambios del sistema, incluyendo la creación y destrucción de instancias; *evaluaciones de variables*, que representan el espacio de variables globales y locales; y *la comunicación de los componentes*, que describe la interacción entre estos.

En su estudio, Rausch (2001) propone que es posible hacer la descripción de los elementos básicos de la arquitectura mediante el uso de UML, e indica que la especificación de la estructura del sistema no es suficiente para obtener un modelo completo del mismo. Por esto, para la descripción del comportamiento de los componentes de la arquitectura, propone el uso de OCL. Según su estudio, este lenguaje ofrece facilidades que no provee UML, por lo que considera que el uso de ambos es una posible solución para solventar los problemas de descripción de las arquitecturas de software.

De forma similar, para Kazman et al. (2001), el lenguaje UML no resuelve satisfactoriamente todos los problemas que implica la descripción de una arquitectura de software. Por ejemplo, no soporta el refinamiento sucesivo de los diseños como de las abstracciones arquitectónicas a cualquier nivel de refinamiento jerárquico. Kazman et al. (2001) proponen que UML es un lenguaje abstracto, cuyos diseños no son fáciles de comprender por personas que no posean conocimientos del lenguaje.

En virtud de la importancia de la descripción de la arquitectura de software, Kazman et al. (2001) plantea los *lenguajes de descripción arquitectónica* como elementos que, aunque presentan algunas desventajas, merecen atención, puesto que pueden considerarse como una solución alternativa al problema planteado.

7. Lenguajes de Descripción Arquitectónica

El problema inherente en la mayoría de los desarrollos de software es la naturaleza abstracta de un programa de computación. A diferencia de otros productos de distintas áreas de la ingeniería (autos, casas, aviones, etc.), el software no es tangible, no posee una forma natural de visualización y no hay solución perfecta al problema planteado. Actualmente, la forma más exacta de descripción del sistema es el código fuente o el código compilado. De aquí que el problema de la descripción de una arquitectura de software es encontrar una técnica que cumpla con los propósitos del desarrollo de software; en otras palabras, la comunicación entre las partes interesadas, la evaluación y la implementación (Bengtsson, 1999).

Hasta la fecha, las arquitecturas de software han sido representadas por esquemas simples de cajas y líneas (Bass et al., 1998; Begtsson, 1999), en los que la naturaleza de los componentes, sus propiedades, la semántica de las conexiones y el comportamiento del sistema como un todo, se define de manera muy pobre. En este sentido, Bass et al. (1998) proponen que aunque este tipo de representación ofrece una imagen intuitiva de la construcción del sistema, existen muchas preguntas que no pueden responderse con este tipo de representación. Entre ellas se encuentran, por ejemplo, la esencia de los componentes, cuál es su tipo, qué hacen, cómo se

comportan, de qué otros componentes dependen y de qué manera, así como qué significan las conexiones y qué mecanismos de comunicación están involucrados, entre otras. Los *lenguajes de descripción arquitectónica (Architecture Description Languages, ADL)* surgen como posible solución ante inquietudes de este estilo (Bass et al., 1998; Clements, 1996).

7.1. Conceptos y características de los lenguajes de descripción arquitectónica

Bass et al. (1998) proponen que el establecimiento de una notación estándar para la representación de arquitecturas, a través de un lenguaje de descripción arquitectónica, permite mejorar la comunicación entre el autor y el lector, logrando tener un medio de entendimiento común, ahorrando tiempo en indagar el significado de los gráficos que representan la arquitectura y sus componentes. El lenguaje de descripción arquitectónica sirve de soporte para el análisis y las decisiones tempranas de diseño, y sería factible la construcción de herramientas que asistan en el proceso de desarrollo. Además, este tipo de lenguaje provee un mecanismo para la construcción de la arquitectura como artefacto, transferible a otros sistemas, de manera tal que pueda ser tomada como marco de referencia o como punto de partida para el resto de las tareas del proceso de desarrollo.

Los *ADL* resultan de un enfoque lingüístico para el problema de la representación formal de una arquitectura (Bass et al., 1998), y por ende, son usados para describir una arquitectura de software (Clements, 1996). Este tipo de lenguaje puede ser descriptivo formal o semi-formal, un lenguaje gráfico, o incluir ambos (Bengtsson, 1999), y sus características vienen dadas por los requerimientos que implica.

En la literatura (Shaw, 1994; Luckham, 1993; Bass et al., 1998) es posible encontrar múltiples conjuntos de requerimientos que intentan definir qué debe hacer un lenguaje de este tipo.

Shaw (1994) propone que un lenguaje de descripción arquitectónica debe poseer las siguientes propiedades:

- ✓ Capacidad para representar componentes (primitivos o compuestos), así como sus propiedades, interfaces e implementaciones
- ✓ Capacidad de representar conectores, así como protocolos, propiedades e implementaciones
- ✓ Abstracción y encapsulamiento
- ✓ Tipos y chequeo de tipos
- ✓ Capacidad de integración de herramientas de análisis

Por su parte, Luckham (1993) propone, además, los siguientes requerimientos para un lenguaje de descripción arquitectónica:

- ✓ Integridad comunicacional, es decir, limitación de la comunicación a los componentes conectados entre sí a nivel arquitectónico
- ✓ Capacidad de modelar arquitecturas dinámicas
- ✓ Capacidad para razonar acerca de causalidad y tiempo
- ✓ Soporte para refinamiento jerárquico

- ✓ Capacidad de correspondencia de ciertos comportamientos a arquitecturas posiblemente diferentes

En este mismo sentido, Bass et al. (1998) agregan que un lenguaje de descripción arquitectónica debe proveer:

- ✓ Soporte a las tareas de creación, refinamiento y validación de una arquitectura. Debe englobar reglas acerca de lo que constituye una arquitectura completa o consistente.
- ✓ Debe proveer la capacidad de representar la mayoría de los *patrones arquitectónicos* conocidos (Shaw et al., 1994), directa o indirectamente.
- ✓ Debe tener la capacidad de proveer vistas del sistema que expresen información arquitectónica, pero al mismo tiempo suprimir la implementación de información no arquitectónica.
- ✓ Si el lenguaje puede expresar información que se encuentra a nivel de implementación, debe entonces poseer capacidad para realizar correspondencias con más de una implementación a nivel arquitectónico. En otras palabras, debe soportar la especificación de familias de implementaciones que satisfacen una arquitectura común.
- ✓ Debe soportar capacidad analítica basada en información de nivel arquitectónico, o bien la capacidad para la rápida generación de prototipos de implementación.

7.2. Ventajas del uso de lenguajes de descripción arquitectónica

Bass et al. (1998) presentan una serie de ventajas que proporciona el uso de los lenguajes de descripción arquitectónica en el desarrollo de un sistema de software. En principio, proponen que la descripción inicial del sistema puede ser llevada a cabo de forma textual o gráfica, basada en estilos arquitecturales y tipos de componentes, así como también hacer la descripción de un sistema o subsistema en función de la información que recibe o produce. De igual forma, es posible hacer la descripción del comportamiento y sus elementos asociados, tales como el tipo de eventos que producen, o a los que responden, incluyendo descripciones o documentación de alto nivel.

Otra ventaja que presentan los ADL es la facilidad con la que puede introducirse y mantenerse la información referente al sistema. En este sentido, no sólo es posible efectuar análisis a distintos niveles de detalle, sino que también es posible establecer cambios de tipos sobre los componentes. Así mismo, es posible realizar análisis de desempeño, disponibilidad o seguridad, en tanto el lenguaje de descripción arquitectónica provea la facilidad para ello.

Por último, Bass et al. (1998) indican que los componentes pueden ser refinados en la medida que sea necesario, para distintos tipos de análisis. En cualquier momento un componente puede ser visto conjuntamente con cualquier información que se conozca de él. De igual manera, a partir de las descripciones asociadas a los componentes, se establece la posibilidad de que los mismos puedan ser llevados a nivel de código, o plantillas de código.

Ahora bien, aunque es posible establecer la definición de un lenguaje de descripción arquitectónica y cuáles son las ventajas de su uso, no hay formas claras

de diferenciarlos de otros lenguajes. Sin embargo, Bass et al. (1998) proponen que puede hacerse una distinción en función de cuánta información de tipo arquitectónico es posible representar mediante el lenguaje de descripción arquitectónica. Es necesario aclarar la diferencia que existen entre los ADL y los lenguajes de programación, lenguajes de requerimientos y lenguajes de modelado.

7.3. Diferencias entre los lenguajes de descripción arquitectónica y otros lenguajes

Según Bass et al. (1998) y Clements (1996), las características esenciales que diferencian los ADL de otros lenguajes son:

- ✓ La abstracción que proveen al usuario es de naturaleza arquitectónica
- ✓ La mayoría de las vistas provistas por estos lenguajes contienen información predominantemente arquitectónica. Esto contrasta con los lenguajes de programación o lenguajes de requerimientos, que tienden a mostrar información de otro tipo
- ✓ El análisis provisto por el lenguaje se fundamenta en información de nivel arquitectónico

En principio, los lenguajes de descripción arquitectónica difieren de los *lenguajes de requerimientos* en tanto los últimos describen espacios de problemas, mientras que los primeros tienen sus raíces en el espacio de la solución. En la práctica, los requerimientos suelen dividirse en trozos según el comportamiento para facilitar la representación, y los lenguajes para representar las conductas están generalmente ajustados a la representación de los componentes arquitectónicos, aunque no es el principal objetivo del lenguaje (Clements, 1996).

Por otro parte, los lenguajes de descripción arquitectónica difieren de los *lenguajes de programación* porque los últimos asocian todas las abstracciones arquitectónicas a soluciones específicas, mientras que los lenguajes de descripción arquitectónica intencionalmente suprimen o varían tales asociaciones. En la práctica, la arquitectura está englobada, y es posible recuperarla a partir del código, y muchos lenguajes proveen vistas del sistema a nivel arquitectónico (Clements, 1996).

Así mismo, los lenguajes de descripción arquitectónica difieren de los *lenguajes de modelado* dado que los últimos están más relacionados con el comportamiento del todo, más que el de las partes, mientras que los lenguajes de descripción arquitectónica se concentran en la representación de los componentes. En la práctica, muchos lenguajes de modelado permiten la representación de componentes en cooperación y pueden representar arquitecturas de forma aceptable (Clements, 1996).

Se han propuesto muchos lenguajes de descripción arquitectónica. Rapide (Luckham et al., 1993) y UniCon (Shaw et al., 1994) son ejemplos de ellos. Clements (1996) presenta de forma resumida y comparativa ocho lenguajes de descripción arquitectónica. La tabla 14 presenta los resultados de la comparación entre distintos lenguajes de descripción arquitectónica, como lo son ArTek (Hayes-Roth et al., 1994), CODE (Newton et al., 1992), Demeter (Palsberg et al., 1995), Modechart (Jahanian et al., 1994), PSDL/CAPS (Luqi et al., 1993), Resolve (Edwards et al., 1994), UniCon (Shaw et al., 1994) y Wrigth (Allen et al., 1994), producto del estudio de Clements (1996). La información contenida en la tabla es una comparación entre los lenguajes

de descripción arquitectónica mencionados, hecha en base a criterios basados en las capacidades, características e historia de los mismos.

Símbolo	Significado
S	Sí
N	No
H	Alta Capacidad: el lenguaje provee características específicas para el soporte
M	Capacidad Media: el lenguaje provee características genéricas, y puede obtenerse indirectamente
L	Capacidad Baja: el lenguaje provee poco soporte para la capacidad en cuestión
T	La herramienta desarrollada específicamente para el ADL provee la capacidad
E	Herramientas externas proveen la capacidad
P	El lenguaje provee suficiente información para soportar la capacidad, pero no existe el soporte

Atributos	ARTEK	CODE	DEMETER	MODECHART	PSDL/CAPS	RESOLVE	UNICON	WRIGHT
Aplicabilidad								
Capacidad para representar estilos	M	M	M	M	H	M	M	H
Capacidad para el manejo de tiempo real	M	-	M	H	H	L	H	L
Capacidad para el manejo de S. distribuidos	H	H	H	H	H	M	M	M
Capacidad para el manejo de arq. dinámicas	L	H	H	L	M	L	-	L
Calidad de Definición del Lenguaje								
Completitud de especificación de arquitectura	M	L	H	M	M	L	M	H
Consistencia de la especificación arquitectónica	L	L	M	L	M	H	L	H
Alcance del Lenguaje; usuarios considerados								
Requerimientos	H	L	-	H	H	M	L	L
Diseño detallado / algoritmos	L	M	H	M	H	M	L	L
Código	M	L	H	L	L	H	H	L
Ingeniero de Dominio	S	S	S	N	S	N	S	S
Ingeniero de Aplicación	S	S	S	S	S	S	S	S
Analista de Sistemas	S	N	S	S	S	S	N	S
Historia de Captura de Diseño	-	L	-	L	-	-	-	-
Vistas								
Textual	S	N	S	S	S	S	S	S
Gráfica	S	S	S	S	S	N	S	N
Riqueza de la semántica de la vista	L	M	H	H	H	L	-	L
Referencias cruzadas entre vistas	M	L	L	M	M	L	M	L
Soporte para variabilidad	M	L	H	L	H	H	L	H
Poder expresivo, extensibilidad	H	L	M	M	H	H	M	M
Soporte para creación de arquitectura	T	T	P	T	T	E	T	-
Soporte para validación de arquitectura	T	T	E	T	T	P	T	E
Soporte para refinamiento de arquitectura	T	P	P	T	T	P	P	P
Soporte para análisis de arquitectura	E	-	P	T	T	P	-	-
Soporte para construcción de aplicaciones	-	T	P	-	P	-	-	-
Madurez de la Herramienta								
Disponible en el mercado	N	N	-	N	N	N	N	N
Edad (años)	3	-	10	7	5	-	2	3
Número de sitios en uso	11	-	-	4	12	4	1	1
Soporte al cliente	S	S	N	N	N	S	N	N

Tabla 14. Resultados de la comparación de los ADL.
Fuente: Clements (1996)

8. EVALUACIÓN DE ARQUITECTURAS DE SOFTWARE

Según Kazman et al. (2001), el primer paso para la evaluación de una arquitectura es conocer qué es lo que se quiere evaluar. De esta forma, es posible establecer la base para la evaluación, puesto que la intención es saber qué se puede evaluar y qué no. Resulta interesante el estudio de la evaluación de una arquitectura: si las decisiones que se toman sobre la misma determinan los atributos de calidad del sistema, es entonces posible evaluar las decisiones de tipo arquitectónico con respecto al impacto sobre estos atributos.

La arquitectura de software posee un gran impacto sobre la calidad de un sistema, por lo que es muy importante estar en capacidad de tomar decisiones acertadas sobre ella, en diversos tipos de situaciones, entre las cuales destacan (Bengtsson, 1999):

- ✓ Comparación de alternativas similares
- ✓ Comparación de la arquitectura original y la modificada
- ✓ Comparación de la arquitectura de software con respecto a los requerimientos del sistema
- ✓ Comparación de una arquitectura de software con una propuesta teórica
- ✓ Valoración de una arquitectura en base a escalas específicas

En este sentido, Kazman et al. (2001) proponen que, mediante la arquitectura de software, es posible también determinar la estructura del proyecto de desarrollo del sistema, sobre elementos como el control de configuración, calendarios, control de recursos, metas de desempeño, estructura del equipo de desarrollo y otras actividades que se realizan con la arquitectura del sistema como apoyo principal. En este sentido, la garantía de una arquitectura correcta cumple un papel fundamental en el éxito general del proceso de desarrollo, además del cumplimiento de los atributos de calidad del sistema.

De esta manera, el interés se centra en determinar el momento propicio para efectuar la evaluación de una arquitectura. En este sentido, Kazman et al. (2001) amplían el panorama clásico, indicando las ocasiones en que se hace posible hacer la evaluación de una arquitectura. Su planteamiento establece que es posible realizarla en cualquier momento, pero propone dos variantes que agrupan dos etapas distintas: *temprano* y *tarde*.

Para la primera variante, Kazman et al. (2001) establecen que no es necesario que la arquitectura esté completamente especificada para efectuar la evaluación, y esto abarca desde las fases tempranas de diseño y a lo largo del desarrollo. En este punto, resulta interesante resaltar el planteamiento de Bosch (2000), quien propone que es posible efectuar decisiones sobre la arquitectura a cualquier nivel, puesto que se pueden imponer distintos tipos de cambios arquitectónicos, producto de una evaluación en función de los atributos de calidad esperados. Ahora bien, es necesario destacar que tanto Bosch (2000) como Kazman et al. (2001) establecen que mientras mayor es el nivel de especificación, mejores son los resultados que produce la evaluación.

La segunda variante propuesta por Kazman et al. (2001) consiste en realizar la evaluación de la arquitectura cuando ésta se encuentra establecida y la implementación se ha completado. Este es el caso general que se presenta al momento de la adquisición de un sistema ya desarrollado. Los autores consideran muy útil la evaluación del sistema en este punto, puesto que puede observarse el cumplimiento de los atributos de calidad asociados al sistema, y cómo será su comportamiento general.

Por su parte, Bosch (2000) afirma que la evaluación de una arquitectura de software es una tarea no trivial, puesto que se pretende medir propiedades del sistema en base a especificaciones abstractas, como por ejemplo los diseños arquitectónicos. Por ello, la intención es más bien **la evaluación del potencial de la arquitectura diseñada para alcanzar los atributos de calidad requeridos**. Las mediciones que se realizan sobre una arquitectura de software pueden tener distintos objetivos, dependiendo de la situación en la que se encuentre el arquitecto y la aplicabilidad de las técnicas que emplea. Los objetivos que menciona Bosch (2000) son tres: *cualitativos*, *cuantitativos* y *máximos y mínimos teóricos*.

La medición *cualitativa* se aplica para la comparación entre arquitecturas candidatas y tiene relación con la intención de saber la opción que se adapta mejor a cierto atributo de calidad. Este tipo de medición brinda respuestas afirmativas o negativas, sin mayor nivel de detalle. La medición *cuantitativa* busca la obtención de valores que permitan tomar decisiones en cuanto a los atributos de calidad de una arquitectura de software. El esquema general es la comparación con márgenes establecidos, como lo es el caso de los requerimientos de desempeño, para establecer el grado de cumplimiento de una arquitectura candidata, o tomar decisiones sobre ella. Este enfoque permite establecer comparaciones, pero se ve limitado en tanto no se conozcan los valores teóricos máximos y mínimos de las mediciones con las que se realiza la comparación. Por último, la *medición de máximo y mínimo teórico* contempla los valores teóricos para efectos de la comparación de la medición con los atributos de calidad especificados. El conocimiento de los valores máximos o mínimos permite el establecimiento claro del grado de cumplimiento de los atributos de calidad.

En líneas generales, el planteamiento anterior de Bosch (2000) presenta los objetivos para efectos de la medición de los atributos de calidad. Sin embargo, en ocasiones, la evaluación de una arquitectura de software no produce valores numéricos que permiten la toma de decisiones de manera directa (Kazman, 2001). Ante la posibilidad de efectuar evaluaciones a cualquier nivel del proceso de diseño, con distintos niveles de especificación, Kazman et al. (2001) proponen un esquema general en relación a la evaluación de una arquitectura con respecto a sus atributos de calidad. En este sentido, Kazman y sus colegas afirman que de la evaluación de una arquitectura no se obtienen respuestas del tipo "si - no", "bueno – malo" o "6.75 de 10", sino que explica **cuáles son los puntos de riesgo del diseño evaluado**.

Una de las diferencias principales entre los planteamientos de Bosch (2000) y Kazman et al. (2001) es el enfoque que utilizan para efectos de la evaluación. El método de diseño de arquitecturas planteado por Bosch (2000) tiene como principal característica la evaluación explícita de los atributos de calidad de la arquitectura durante el proceso de diseño de la misma. El autor afirma que el enfoque tradicional en la industria de software es el de implementar el sistema y luego establecer valores para los atributos de calidad del mismo. Este enfoque, según su experiencia, tiene la desventaja de que se destina gran cantidad de recursos y esfuerzo en el desarrollo de un sistema que no satisface los requerimientos de calidad. En este sentido, Bosch

(2000) plantea las técnicas de evaluación: basada en escenarios, basada en simulación, basada en modelos matemáticos y basada en experiencia.

Por su parte, Kazman et al. (2001) proponen que resulta de poco interés la caracterización o medición de atributos de calidad en las fases tempranas del proceso de diseño, dado que estos parámetros son, por lo general, dependientes de la implementación. Su enfoque se orienta hacia la mitigación de riesgos, ubicando dónde un atributo de calidad de interés se ve afectado por decisiones arquitectónicas. En su estudio, Kazman et al. (2001) presentan tres métodos de evaluación de arquitecturas de software, que son Architecture Trade-off Análisis Method (ATAM), Software Architecture Analysis Method (SAAM) y Active Intermediate Designs Review (ARID).

Tanto Bosch (2000) como Kazman et al. (2001) indican la importancia de la especificación exhaustiva de los atributos de calidad como base para efectos de la evaluación de una arquitectura de software. Descripciones tales como *"el sistema debe ser robusto"* o *"el sistema debe exhibir un desempeño aceptable"* resultan ambiguos, puesto que lo que se entiende de ellos puede ser diferente para los distintos involucrados con el sistema (Kazman et al., 2001). El punto es entonces definir los atributos de calidad en función de sus metas y su contexto, y no como cantidades absolutas, según Kazman y sus colegas.

De los planteamientos de evaluación establecidos por Bosch (2000) y Kazman et al. (2001), se tiene que la evaluación de las arquitecturas de software puede ser realizada mediante el uso de diversas técnicas y métodos. En este sentido, resulta interesante estudiar las distintas opciones que existen en la actualidad para llevar a cabo esta tarea.

9. TÉCNICAS DE EVALUACIÓN DE ARQUITECTURAS DE SOFTWARE

Según Bosch (2000), las técnicas utilizadas para la evaluación de atributos de calidad requieren grandes esfuerzos por parte del ingeniero de software para crear especificaciones y predicciones. Estas técnicas requieren información del sistema a desarrollar que no está disponible durante el diseño arquitectónico, sino al principio del diseño detallado del sistema.

En vista de que el interés es tomar decisiones de tipo arquitectónico en las fases tempranas del desarrollo, son necesarias técnicas que requieran poca información detallada y puedan conducir a resultados relativamente precisos (Bosch, 2000). Las técnicas existentes en la actualidad para evaluar arquitecturas permiten hacer una evaluación cuantitativa sobre los atributos de calidad a nivel arquitectónico, pero se tienen pocos medios para predecir el máximo (o mínimo) teórico para las arquitecturas de software. Sin embargo, debido al costo de realizar este tipo de evaluación, en muchos casos los arquitectos de software evalúan cualitativamente, para decidir entre las alternativas de diseño (Bosch, 2000). Bosch (2000) propone diferentes técnicas de evaluación de arquitecturas de software, a saber: evaluación basada en escenarios, evaluación basada en simulación, evaluación basada en modelos matemáticos y evaluación basada en experiencia.

9.1. Evaluación basada en escenarios

De acuerdo con Kazman et al. (2001), un *escenario* es una breve descripción de la interacción de alguno de los involucrados en el desarrollo del sistema con éste. Por

ejemplo, un usuario hará la descripción en términos de la ejecución de una tarea; un encargado de mantenimiento hará referencia a cambios que deban realizarse sobre el sistema; un desarrollador se enfocará en el uso de la arquitectura para efectos de su construcción o predicción de su desempeño.

Según Kazman et al. (2001), un escenario consta de tres partes: el estímulo, el contexto y la respuesta. El estímulo es la parte del escenario que explica o describe lo que el involucrado en el desarrollo hace para iniciar la interacción con el sistema. Puede incluir ejecución de tareas, cambios en el sistema, ejecución de pruebas, configuración, etc. El contexto describe qué sucede en el sistema al momento del estímulo. La respuesta describe, a través de la arquitectura, cómo debería responder el sistema ante el estímulo. Este último elemento es el que permite establecer cuál es el atributo de calidad asociado.

Los escenarios proveen un vehículo que permite concretar y entender atributos de calidad. Kazman et al. (2001) y Carriere et al. (2000) coinciden en la importancia del uso de los escenarios, no sólo para efectos de la evaluación de arquitecturas de software. Entre las ventajas de su uso están:

- ✓ Son simples de crear y entender
- ✓ Son poco costosos y no requieren mucho entrenamiento
- ✓ Son efectivos

Actualmente las técnicas basadas en escenarios cuentan con dos instrumentos de evaluación relevantes, a saber: el *Utility Tree* propuesto por Kazman et al. (2001), y los *Profiles,* propuestos por Bosch (2000).

9.1.1. Utility Tree

Según Kazman et al. (2001), un *Utility Tree* es un esquema en forma de árbol que presenta los atributos de calidad de un sistema de software, refinados hasta el establecimiento de escenarios que especifican con suficiente detalle el nivel de prioridad de cada uno.

La intención del uso del Utility Tree es la identificación de los atributos de calidad más importantes para un proyecto particular. No existe un conjunto preestablecido de atributos, sino que son definidos por los involucrados en el desarrollo del sistema al momento de la construcción del árbol.

El Utility Tree contiene como nodo raíz la *utilidad general* del sistema. Los atributos de calidad asociados al mismo conforman el segundo nivel del árbol (Kazman et al., 2001), los cuales se refinan hasta la obtención de un escenario lo suficientemente concreto para ser analizado y otorgarle prioridad a cada atributo considerado.

Cada atributo de calidad perteneciente al árbol contiene una serie de escenarios relacionados, y una escala de importancia y dificultad asociada, que será útil para efectos de la evaluación de la arquitectura.

9.1.2. Perfiles (*Profiles*)

Un perfil (*profile*) es un conjunto de escenarios, generalmente con alguna importancia relativa asociada a cada uno de ellos. El uso de perfiles permite hacer especificaciones más precisas del requerimiento para un atributo de calidad (Bosch, 2000). Los perfiles tienen asociados dos formas de especificación: perfiles completos y perfiles seleccionados.

Los *perfiles completos* definen todos los escenarios relevantes como parte del perfil. Esto permite al ingeniero de software realizar un análisis de la arquitectura para el atributo de calidad estudiado de una manera completa, puesto que incluye todos los posibles casos. Su uso se reduce a sistemas relativamente pequeños y sólo es posible predecir conjuntos de escenarios completos para algunos atributos de calidad (Bosch, 2000).

Los *perfiles seleccionados* se asemejan a la selección de muestras sobre una población en los experimentos estadísticos. Se toma un conjunto de escenarios de forma aleatoria, de acuerdo a algunos requerimientos. La aleatoriedad no es totalmente cierta por limitaciones prácticas, por lo que se fuerza la realización de una selección estructurada de elementos para el conjunto de muestra. Si bien es informal, permite hacer proposiciones científicamente válidas (Bosch, 2000).

Dado que los escenarios se construyen cuidadosamente, Bosch (2000) plantea que puede asumirse que el perfil representa una imagen exacta de la población de escenarios. Para la definición de un perfil, es necesario seguir tres pasos: definición de las categorías del escenario, selección y definición de los escenarios para la categoría y asignación del "peso" a los escenarios.

Bosch (2000) establece que la definición de categorías de escenarios divide la población de escenarios en poblaciones más pequeñas, que cubren aspectos particulares del sistema. La selección y definición de escenarios para cada categoría selecciona un conjunto de escenarios representativo para la subpoblación. Luego, en la asignación del peso a los escenarios, dependiendo del perfil, el peso de un escenario tiene diferentes significados. Se definen escalas que de alguna forma sean cuantificables y puedan ser convertidas a pesos relativos.

Cada atributo de calidad tiene un perfil asociado. Algunos perfiles pueden ser usados para evaluar más de un atributo de calidad. Han sido seleccionados cinco atributos de calidad que son considerados de mayor relevancia para una perspectiva general de ingeniería de software (Bosch, 2000). Tales atributos son: desempeño *(performance)*, mantenibilidad *(maintainability)*, confiabilidad *(reliability)*, seguridad externa *(safety)* y seguridad interna *(security)*. La tabla 15 presenta para cada atributo de calidad, el perfil asociado, la forma en que se definen las categorías, el significado de los "pesos" y posibles métricas de evaluación, de acuerdo al planteamiento de Bosch (2000).

Según Bosch (2000), la técnica de evaluación basada en escenarios es dependiente de manera directa del perfil definido para el atributo de calidad que va a ser evaluado. La efectividad de la técnica es altamente dependiente de la representatividad de los escenarios. El autor propone que existe la evaluación de funcionalidad basada en escenarios, y es utilizada en el diseño orientado a objetos para especificar comportamiento del sistema. La diferencia entre este tipo de

Atributo	Perfil	Categorías	Pesos	Métricas
Mantenibilidad	Perfil de mantenimiento (Maintainance profile)	Se organizan alrededor de las interfaces del sistema (sistema operativo, interfaces con el usuario, interfaces con otros sistemas). Los escenarios de cambio describen modificaciones en los requerimientos	Indican la probabilidad de ocurrencia del cambio de escenario en un período de tiempo	✓ Impacto en término de líneas de código que tienen que cambiarse ✓ Se requiere un estimado de líneas de código de los componentes arquitectónicos.
Desempeño	Perfil de uso (Usage profile)	Descompone los escenarios de uso basado en los tipos de usuario y/o interfaces del sistema	Representan la frecuencia relativa del escenario	✓ Funcionalidad de componentes ✓ Comportamiento del sistema en respuesta a los escenarios de uso en el perfil ✓ Promedio y peor caso de latencia por sincronización y sobrecarga en el sistema
Confiabilidad	Perfil de uso (Usage profile)	Confiabilidad de los componentes, genera la confiabilidad de los escenarios de uso	Indica la robustez del sistema	✓ Datos estimados de confiabilidad del componente ✓ Datos históricos de confiabilidad del componente
Seguridad Interna	Perfil de seguridad (Security profile) Perfil de uso (usage profile)	Basada en todas las interfaces del sistema	Indica la probabilidad de fallas	Depende del aspecto de seguridad a ser evaluado. Por ejemplo, la disponibilidad puede evaluarse en términos del número de veces que se ejecutan operaciones de seguridad.
Seguridad Externa	Perfil de peligro (Hazard profile)	Se organizan de acuerdo a documentos de certificación (sistemas médicos, puntos de interacción del sistema con el mundo real, o componentes críticos del sistema)	Indican la probabilidad de falla u ocurrencia de consecuencias desastrosas.	*Bosch (2000) no establece ejemplos*

Tabla 15. Perfiles, categorías, pesos y métricas asociados a atributos de calidad según Bosch (2000)

evaluación y la evaluación arquitectónica basada en escenarios radica en que la última utiliza los escenarios para evaluar atributos de calidad, en lugar de verificar o describir funcionalidad, además de que se usan otros escenarios para definir otros atributos de calidad.

Es importante destacar que la definición de los casos de uso debe ser independiente del *perfil de uso*, debido a que los casos de uso permiten optimizar el diseño arquitectónico, y pueden resultar una muestra no representativa de la población de escenarios para la evaluación de cierto atributo de calidad (Bosch, 2000).

De acuerdo con Bosch (2000), la evaluación basada en escenarios puede ser empleada para comparar dos arquitecturas y para la evaluación absoluta de una arquitectura. La diferencia está en que la evaluación absoluta requiere mayor cantidad de datos estimados y cuantitativos necesarios para la evaluación.

Bosch (2000) explica que la técnica consiste en dos etapas: *análisis de impacto* y *predición de atributos de calidad*. El *análisis de impacto* toma como entrada el perfil y la arquitectura de software. Para cada escenario del perfil, se evalúa el impacto de la arquitectura y se obtienen los resultados que serán usados en la etapa de *predicción de atributos de calidad*, donde se pronostica el valor del atributo de calidad estudiado de acuerdo a las métricas existentes.

9.2. Evaluación basada en simulación

Bosch (2000) establece que la evaluación basada en simulación utiliza una implementación de alto nivel de la arquitectura de software. El enfoque básico consiste en la implementación de componentes de la arquitectura y la implementación –a cierto nivel de abstracción- del contexto del sistema donde se supone va a ejecutarse. La finalidad es evaluar el comportamiento de la arquitectura bajo diversas circunstancias. Una vez disponibles estas implementaciones, pueden usarse los perfiles respectivos para evaluar los atributos de calidad.

El proceso de evaluación basada en simulación sigue los siguientes pasos (Bosch, 2000):

- ✓ **Definición e implementación del contexto**. Consiste en identificar las interfaces de la arquitectura de software con su contexto, y decidir cómo será simulado el comportamiento del contexto en tales interfaces.
- ✓ **Implementación de los componentes arquitectónicos**. La descripción del diseño arquitectónico debe definir, por lo menos, las interfaces y las conexiones de los componentes, por lo que estas partes pueden ser tomadas directamente de la descripción de diseño. El comportamiento de los componentes en respuesta a eventos sobre sus interfaces puede no ser especificado claramente, aunque generalmente existe un conocimiento común y es necesario que el arquitecto lo interprete, por lo que éste decide el nivel de detalle de la implementación.
- ✓ **Implementación del perfil**. Dependiendo del atributo de calidad que el arquitecto de software intenta evaluar usando simulación, el perfil asociado necesitará ser implementado en el sistema. El arquitecto de software debe ser capaz de activar escenarios individuales, así como también ejecutar un perfil completo usando selección aleatoria, basado en los pesos normalizados de los mismos.

- ✓ **Simulación del sistema e inicio del perfil**. El arquitecto de software ejecutará la simulación y activará escenarios de forma manual o automática, y obtendrá resultados de acuerdo al atributo de calidad que está siendo evaluado.
- ✓ **Predicción de atributos de calidad**. Dependiendo del tipo de simulación y del atributo de calidad evaluado, se puede disponer de cantidades excesivas de datos, que requieren ser condensados. Esto permite hacer conclusiones acerca del comportamiento del sistema.

En el ámbito de las simulaciones, se encuentra la técnica de implementación de prototipos (*prototyping*). Esta técnica implementa una parte de la arquitectura de software y la ejecuta en el contexto del sistema. Es utilizada para evaluar requerimientos de calidad operacional, como desempeño y confiabilidad. Para su uso se necesita mayor información sobre el desarrollo y disponibilidad del hardware, y otras partes que constituyen el contexto del sistema de software. Se obtiene un resultado de evaluación con mayor exactitud (Bosch, 2000).

La exactitud de los resultados de la evaluación depende, a su vez, de la exactitud del perfil utilizado para evaluar el atributo de calidad y de la precisión con la que el contexto del sistema simula las condiciones del mundo real.

En términos de los instrumentos asociados a las técnicas de evaluación basadas en simulación, se encuentran los lenguajes de descripción arquitectónica, descritos en la sección 7, y los modelos de colas.

9.3. Evaluación basada en modelos matemáticos

Bosch (2000) establece que la evaluación basada en modelos matemáticos se utiliza para evaluar atributos de calidad operacionales. Permite una evaluación estática de los modelos de diseño arquitectónico, y se presentan como alternativa a la simulación, dado que evalúan el mismo tipo de atributos. Ambos enfoques pueden ser combinados, utilizando los resultados de uno como entrada para el otro.

El proceso de evaluación basada en modelos matemáticos sigue los siguientes pasos (Bosch, 2000):

- ✓ **Selección y adaptación del modelo matemático**. La mayoría de los centros de investigación orientados a atributos de calidad han desarrollado modelos matemáticos para medir sus atributos de calidad, los cuales tienden a ser muy elaborados y detallados, así como también requieren de cierto tipo de datos y análisis. Parte de estos datos requeridos no están disponibles a nivel de arquitectura, y la técnica requiere mucho esfuerzo para la evaluación arquitectónica, por lo que el arquitecto de software se ve obligado a adaptar el modelo.
- ✓ **Representación de la arquitectura en términos del modelo**. El modelo matemático seleccionado y adaptado no asume necesariamente que el sistema que intenta modelar consiste de componentes y conexiones. Por lo tanto, la arquitectura necesita ser representada en términos del modelo.
- ✓ **Estimación de los datos de entrada requeridos**. El modelo matemático aún cuando ha sido adaptado, requiere datos de entrada que no están incluidos en la definición básica de la arquitectura. Es necesario estimar y deducir estos datos de la especificación de requerimientos y de la arquitectura diseñada.

- ✓ **Predicción de atributos de calidad**. Una vez que la arquitectura es expresada en términos del modelo y se encuentran disponibles todos los datos de ntrada requeridos, el arquitecto está en capacidad de calcular la predicción resultante del atributo de calidad evaluado.

Entre las desventajas que presenta esta técnica se encuentra la inexistencia de modelos matemáticos apropiados para los atributos de calidad relevantes (Bosch, 2000), y el hecho de que el desarrollo de un modelo de simulación completo puede requerir esfuerzos sustanciales.

Entre los instrumentos que se cuentan para las técnicas de evaluación de arquitecturas de software basada en modelos matemáticos, se encuentran las Cadenas de Markov y los *Reliability Block Diagrams*.

9.4. Evaluación basada en experiencia

Bosch (2000) establece que en muchas ocasiones los arquitectos e ingenieros de software otorgan valiosas ideas que resultan de utilidad para la evasión de decisiones erradas de diseño. Aunque todas estas experiencias se basan en evidencia anecdótica; es decir, basada en factores subjetivos como la intuición y la experiencia. Sin embargo, la mayoría de ellas puede ser justificada por una línea lógica de razonamiento, y pueden ser la base de otros enfoques de evaluación (Bosch, 2000).

Existen dos tipos de evaluación basada en experiencia: la *evaluación informal*, que es realizada por los arquitectos de software durante el proceso de diseño, y la realizada por equipos externos de evaluación de arquitecturas.

La tabla 16 presenta de forma resumida los instrumentos utilizados por las diferentes técnicas de evaluación.

Técnica de Evaluación	Instrumento de Evaluación
Basada en Escenarios	✓ Profiles ✓ Utility Tree
Basada en Simulación	✓ Lenguajes de Descripción Arquitectónica (ADL) ✓ Modelos de colas
Basada en Modelos Matemáticos	✓ Cadenas de Markov ✓ *Reliability Block Diagrams*
Basada en Experiencia	✓ Intuición y experiencia ✓ Tradición ✓ Proyectos similares

Tabla 16. Instrumentos asociados a las distintas técnicas de evaluación de arquitecturas de software.

Kazman et al. (2001) proponen que la existencia de un método de análisis de arquitecturas de software hace que el proceso sea repetible, y ayuda a garantizar que las respuestas correctas con relación a la arquitectura pueden hacerse temprano, durante las fases tempranas de diseño. Es en este punto donde los problemas encontrados pueden ser solucionados de una forma relativamente poco costosa. De manera similar, un método de evaluación sirve de guía a los involucrados en el desarrollo del sistema, en la búsqueda de conflictos que puede presentar una arquitectura, y sus soluciones. Por esta razón, resulta conveniente estudiar los métodos de evaluación de arquitecturas de software propuestos hasta el momento.

10. MÉTODOS DE EVALUACIÓN DE ARQUITECTURAS DE SOFTWARE

De acuerdo con Kazman et al. (2001), hasta hace poco no existían métodos de utilidad general para evaluar arquitecturas de software. Si alguno existía, sus enfoques eran incompletos, *ad hoc*, y no repetibles, lo que no brindaba mucha confianza. En virtud de esto, múltiples métodos de evaluación han sido propuestos. A continuación se explican algunos de los más importantes.

10.1. Software Architecture Analysis Method (SAAM)

Según Kazman et al. (2001), el Método de Análisis de Arquitecturas de Software (Software Architecture Analysis Method, SAAM) es el primero que fue ampliamente promulgado y documentado. El método fue originalmente creado para el análisis de la modificabilidad de una arquitectura, pero en la práctica ha demostrado ser muy útil para evaluar de forma rápida distintos atributos de calidad, tales como modificabilidad, portabilidad, escalabilidad e integrabilidad.

El método de evaluación SAAM se enfoca en la enumeración de un conjunto de escenarios que representan los cambios probables a los que estará sometido el sistema en el futuro. Como entrada principal, es necesaria alguna forma de descripción de la arquitectura a ser evaluada. De acuerdo con Kazman et al. (2001), las salidas de la evaluación del método SAAM son las siguientes:

- ✓ Una proyección sobre la arquitectura de los escenarios que representan los cambios posibles ante los que puede estar expuesto el sistema
- ✓ Entendimiento de la funcionalidad del sistema, e incluso una comparación de múltiples arquitecturas con respecto al nivel de funcionalidad que cada una soporta sin modificación

Con la aplicación de este método, si el objetivo de la evaluación es una sola arquitectura, se obtienen los lugares en los que la misma puede fallar, en términos de los requerimientos de modificabilidad. Para el caso en el que se cuenta con varias arquitecturas candidatas, el método produce una escala relativa que permite observar qué opción satisface mejor los requerimientos de calidad con la menor cantidad de modificaciones.

La tabla 17 presenta los pasos que contempla el método de evaluación SAAM, con una breve descripción (Kazman et al., 2001).

1. Desarrollo de escenarios	Un escenario es una breve descripción de usos anticipados o deseados del sistema. De igual forma, estos pueden incluir cambios a los que puede estar expuesto el sistema en el futuro.
2. Descripción de la arquitectura	La arquitectura (o las candidatas) debe ser descrita haciendo uso de alguna notación arquitectónica que sea común a todas las partes involucradas en el análisis. Deben incluirse los componentes de datos y conexiones relevantes, así como la descripción del comportamiento general del sistema. El desarrollo de escenarios y la descripción de la arquitectura son usualmente llevados a cabo de forma intercalada, o a través de varias iteraciones.
3. Clasificación y asignación de prioridad de los escenarios	La clasificación de los escenarios puede hacerse en dos clases: directos e indirectos. Un escenario *directo* es el que puede satisfacerse sin la necesidad de modificaciones en la arquitectura. Un escenario *indirecto* es aquel que requiere modificaciones en la arquitectura para poder satisfacerse. Los escenarios indirectos son de especial interés para SAAM, pues son los que permiten medir el grado en el que una arquitectura puede ajustarse a los cambios de evolución que son importantes para los involucrados en el desarrollo.
4. Evaluación individual de los escenarios indirectos	Para cada escenario indirecto, se listan los cambios necesarios sobre la arquitectura, y se calcula su costo. Una modificación sobre la arquitectura significa que debe introducirse un nuevo componente o conector, o que alguno de los existentes requiere cambios en su especificación.
5. Evaluación de la interacción entre escenarios	Cuando dos o más escenarios indirectos proponen cambios sobre un mismo componente, se dice que *interactúan* sobre ese componente. Es necesario evaluar este hecho, puesto que la interacción de componentes semánticamente no relacionados revela que los componentes de la arquitectura efectúan funciones semánticamente distintas. De forma similar, puede verificarse si la arquitectura se encuentra documentada a un nivel correcto de descomposición estructural.
6. Creación de la evaluación global	Debe asignársele un peso a cada escenario, en términos de su importancia relativa al éxito del sistema. Esta asignación de peso suele hacerse con base en las metas del negocio que cada escenario soporta. En el caso de la evaluación de múltiples arquitecturas, la asignación de pesos puede ser utilizada para la determinación de una escala general.

Tabla 17. Pasos contemplados por el método de evaluación SAAM

10.2. Architecture Trade-off Analysis Method (ATAM)

Según Kazman et al. (2001), el Método de Análisis de Acuerdos de Arquitectura (Architecture Trade-off Analysis Method, ATAM) está inspirado en tres áreas distintas: los estilos arquitectónicos, el análisis de atributos de calidad y el método de evaluación SAAM, explicado anteriormente. El nombre del método ATAM surge del hecho de que revela la forma en que una arquitectura específica satisface ciertos atributos de calidad, y provee una visión de cómo los atributos de calidad interactúan con otros; esto es, los tipos de *acuerdos* que se establecen entre ellos.

El método se concentra en la identificación de los estilos arquitectónicos o enfoques arquitectónicos utilizados. Kazman et al. (2001) proponen el término enfoque arquitectónico dado que no todos los arquitectos están familiarizados con el lenguaje de estilos arquitectónicos, aún haciendo uso indirecto de estos. De cualquier forma, estos elementos representan los medios empleados por la arquitectura para alcanzar los atributos de calidad, así como también permiten describir la forma en la que el sistema puede crecer, responder a cambios, e integrarse con otros sistemas, entre otros (Kazman et al., 2001).

El método de evaluación ATAM comprende nueve pasos, agrupados en cuatro fases. La tabla 18 presenta las fases y sus pasos enumerados, junto con su descripción.

Fase 1: Presentación	
1. Presentación del ATAM	El líder de evaluación describe el método a los participantes, trata de establecer las expectativas y responde las preguntas propuestas.
2. Presentación de las metas del negocio	Se realiza la descripción de las metas del negocio que motivan el esfuerzo, y aclara que se persiguen objetivos de tipo arquitectónico.
3. Presentación de la arquitectura	El arquitecto describe la arquitectura, enfocándose en cómo ésta cumple con los objetivos del negocio.
Fase 2: Investigación y análisis	
4. Identificación de los enfoques arquitectónicos	Estos elementos son detectados, pero no analizados.
5. Generación del *Utility Tree*	Se elicitan los atributos de calidad que engloban la "utilidad" del sistema (desempeño, disponibilidad, seguridad, modificabilidad, usabilidad, etc.), especificados en forma de escenarios. Se anotan los estímulos y respuestas, así como se establece la prioridad entre ellos.
6. Análisis de los enfoques arquitectónicos	Con base en los resultados del establecimiento de prioridades del paso anterior, se analizan los elementos del paso 4. En este paso se identifican riesgos arquitectónicos, puntos de sensibilidad y puntos de balance.
Fase 3: Pruebas	
7. Lluvia de ideas y establecimiento de prioridad de escenarios.	Con la colaboración de todos los involucrados, se complementa el conjunto de escenarios.
8. Análisis de los enfoques arquitectónicos	Este paso repite las actividades del paso 6, haciendo uso de los resultados del paso 7. Los escenarios son considerados como casos de prueba para confirmar el análisis realizado hasta el momento.
Fase 4: Reporte	
9. Presentación de los resultados	Basado en la información recolectada a lo largo de la evaluación del ATAM, se presentan los hallazgos a los participantes.

Tabla 18. Pasos del método de evaluación ATAM

10.3. Active Reviews for Intermediate Designs (ARID)

De acuerdo con Kazman et al. (2001) el método ARID es conveniente para realizar la evaluación de diseños parciales en las etapas tempranas del desarrollo. En ocasiones, es necesario saber si un diseño propuesto es conveniente, desde el punto de vista de otras partes de la arquitectura. Según los autores, ARID es un híbrido entre Active Design Review (ADR) y Architecture Trade-Off Method (ATAM), descrito anteriormente.

ADR es utilizado para la evaluación de diseños detallados de unidades del software como los componentes o módulos. Las preguntas giran en torno a la calidad y completitud de la documentación y la suficiencia, el ajuste y la conveniencia de los servicios que provee el diseño propuesto.

Kazman et al. (2001) proponen que tanto ADR como ATAM proveen características útiles para el problema de la evaluación de diseños preliminares, dado que ninguno por sí solo es conveniente. En el caso de ADR, los involucrados reciben documentación detallada y completan cuestionarios, cada uno por separado. En el caso de ATAM, está orientado a la evaluación de toda una arquitectura.

Ante esta situación, y la necesidad de evaluación en las fases tempranas del diseño, Kazman et al. (2001) proponen la utilización de las características que proveen tanto ADR como ATAM por separado. De ADR, resulta conveniente la fidelidad de las respuestas que se obtiene de los involucrados en el desarrollo. Así mismo, la idea del uso de escenarios generados por los involucrados con el sistema es tomada del ATAM. De la combinación de ambas filosofías surge ARID, para efecto de la evaluación temprana de los diseños de una arquitectura de software.

La Tabla 19 presenta los pasos que involucra el método de evaluación ARID, con una breve descripción de cada uno.

En el contexto de los métodos de evaluación, In et al. (2001) proponen un modelo de referencia general para efectos de la toma de decisiones, basado en Cost Benefit Analisys Method (CBAM) y el método de negociación WinWin. Su propuesta pretende ayudar en la selección sistemática entre arquitecturas candidatas, con requerimientos que se negocian entre los involucrados del desarrollo. En este sentido, se presentan los conceptos del modelo de negociación WinWin, CBAM y el marco de referencia propuesto por In et al. (2001).

10.4. Modelo de Negociación WinWin

De acuerdo con In et al. (2001), el modelo WinWin provee un marco de referencia general para identificar y resolver conflictos de requerimientos, mediante la elicitación y negociación de artefactos en función de las condiciones de ganancia. El modelo utiliza la teoría “W”, que pretende que todo involucrado salga ganador. De esta forma, asiste a los involucrados en el desarrollo a identificar y negociar distintos aspectos, reconciliando conflictos entre las opciones de ganancias para todos.

Aunque el modelo propone la resolución de posibles conflictos, no siempre es posible llegar a un acuerdo (In et al., 2001). En este sentido, el método CBAM provee

medios para establecer los balances necesarios, y un marco de referencia para la discusión que puede llevar a una posible solución del problema.

Fase 1: Actividades Previas	
1. Identificación de los encargados de la revisión	Los encargados de la revisión son los ingenieros de software que se espera que usen el diseño, y todos los involucrados en el diseño. En este punto, converge el concepto de *encargado de revisión* de ADR e *involucrado* del ATAM.
2. Preparar el informe De diseño	El diseñador prepara un informe que explica el diseño. Se incluyen ejemplos del uso del mismo para la resolución de problemas reales. Esto permite al facilitador anticipar el tipo de preguntas posibles, así como identificar áreas en las que la presentación puede ser mejorada.
3. Preparar los escenarios base	El diseñador y el facilitador preparan un conjunto de escenarios base. De forma similar a los escenarios del ATAM y el SAAM, se diseñan para ilustrar el concepto de escenario, que pueden o no ser utilizados para efectos de la evaluación.
4. Preparar los materiales	Se reproducen los materiales preparados para ser presentados en la segunda fase. Se establece la reunión, y los involucrados son invitados.
Fase 2: Revisión	
5. Presentación del ARID	Se explica los pasos del ARID a los participantes.
6. Presentación del diseño	El líder del equipo de diseño realiza una presentación, con ejemplos incluidos. Se propone evitar preguntas que conciernen a la implementación o argumentación, así como alternativas de diseño. El objetivo es verificar que el diseño es conveniente.
7. Lluvia de ideas y establecimiento de prioridad de escenarios	Se establece una sesión para la lluvia de ideas sobre los escenarios y el establecimiento de prioridad de escenarios. Los involucrados proponen escenarios a ser usados en el diseño para resolver problemas que esperan encontrar. Luego, los escenarios son sometidos a votación, y se utilizan los que resultan ganadores para hacer pruebas sobre el diseño.
8. Aplicación de los escenarios	Comenzando con el escenario que contó con más votos, el facilitador solicita pseudo-código que utiliza el diseño para proveer el servicio, y el diseñador no debe ayudar en esta tarea. Este paso continúa hasta que ocurra alguno de los siguientes eventos: ✓ Se agota el tiempo destinado a la revisión ✓ Se han estudiado los escenarios de más alta prioridad ✓ El grupo se siente satisfecho con la conclusión alcanzada. Puede suceder que el diseño presentado sea conveniente, con la exitosa aplicación de los escenarios, o por el contrario, no conveniente, cuando el grupo encuentra problemas o deficiencias.
9. Resumen	Al final, el facilitador recuenta la lista de puntos tratados, pide opiniones de los participantes sobre la eficiencia del ejercicio de revisión, y agradece por su participación.

Tabla 19. Pasos del método de evaluación ARID

10.5. Cost-Benefit Analysis Method (CBAM)

De acuerdo con In et al. (2001), el método de análisis de costos y beneficios es un marco de referencia que no toma decisiones por los involucrados en el desarrollo del sistema. Por el contrario, ayuda en la elicitación y documentación de los costos, beneficios e incertidumbre, y provee un proceso de toma de decisiones racional. Uno de los elementos que introduce el método son las llamadas *estrategias arquitectónicas*, que consisten en posibles opciones para la resolución de conflictos entre atributos de calidad presentes en una arquitectura.

El método CBAM abarca los siguientes aspectos (In et al., 2001):

- ✓ Selección de escenarios
- ✓ Evaluación de los beneficios de los atributos de calidad
- ✓ Cuantificación de los beneficios de las estrategias arquitectónicas
- ✓ Cuantificación de los costos de las estrategias arquitectónicas y las implicaciones de calendario
- ✓ Cálculo del nivel de deseabilidad
- ✓ Toma de decisiones

El modelo de referencia para evaluación de arquitecturas de software propuesto por In et al., (2001) comienza con el método WinWin, para efectos de la elicitación de las necesidades de los involucrados y la exploración de las opciones de resolución de conflictos. El método CBAM se propone como solución al esquema sistemático planteado por WinWin, dado que propone la evaluación de costos y beneficios. La tabla 20 presenta los pasos pertenecientes al marco de referencia para evaluación de arquitecturas de software propuesto por In et al. (2001).

1. Selección de escenarios	Cada involucrado en el desarrollo identifica sus condiciones de ganancia. Este paso provee las bases para la identificación de las características ideales del sistema esperadas por los involucrados.
2. Identificación de los conflictos entre atributos de calidad	La lista de condiciones de ganancia es revisada con la intención de identificar conflictos entre atributos de calidad. Los conflictos son categorizados en directos o potenciales.
3. Exploración de las opciones en busca de la resolución de conflictos	Con base en los conflictos detectados en el paso anterior, los involucrados pueden generar opciones de resolución de conflictos, o estrategias arquitectónicas.
4. Medición de los beneficios de los atributos de calidad	Para ayudar en el proceso de toma de decisiones, los involucrados en el desarrollo deben calcular tanto los costos como los beneficios de las estrategias arquitectónicas elaboradas en el paso anterior. Para esto, se calcula una escala de atributos de calidad, donde se asigna un puntaje a cada atributo.
5. Cuantificación de los beneficios	Las escalas establecidas son utilizadas para la evaluación de las estrategias arquitectónicas planteadas en el paso 3. El resultado de la evaluación permite observar el beneficio de cada uno de los cambios arquitectónicos propuestos.

Tabla 20 (a). Pasos contemplados en el marco de referencia para evaluación de arquitecturas de software propuesto por In et al. (2001)

6. Cuantificación de costos e implicaciones de calendario	En este paso se calculan los costos e implicaciones de calendario que aplican para cada una de las estrategias arquitectónicas propuestas en el paso 3. No se propone un modelo específico para la realización de la estimación de costos y tiempo.
7. Cálculo del nivel de Deseabilidad	Este paso contempla una métrica especial, que se refleja como el cociente entre los beneficios y los costos obtenidos. Las estrategias arquitectónicas que resulten con valores mayores se proponen como las más recomendables.
9. Alcanzar un acuerdo	La recomendación general es la documentación del proceso. La acumulación de evidencia permite el establecimiento de un posible consenso, a la hora de la toma de decisiones sobre la arquitectura de un sistema de software.

Tabla 20 (b). Pasos contemplados en el marco de referencia para evaluación de arquitecturas de software propuesto por In et al. (2001) (Continuación)

10.6. Método Diseño y Uso de Arquitecturas de Software propuesto por Bosch (2000)

Bosch (2000) plantea, en su método de diseño de arquitecturas de software, que el proceso de evaluación debe ser visto como una actividad iterativa, que forma parte del proceso de diseño, también iterativo. Una vez que la arquitectura es evaluada, pasa a una fase de transformación, asumiendo que no satisface todos los requerimientos. Luego, la arquitectura transformada es evaluada de nuevo.

El proceso de evaluación propuesto por Bosch (2000) se divide en dos etapas, que son presentadas en la tabla 21.

10.7. Método de comparación de arquitecturas basada en el modelo ISO/IEC 9126 adaptado para arquitecturas de software

Losavio et al. (2003) proponen un método para evaluar y comparar arquitecturas de software candidatas, que hace uso del modelo de especificación de atributos e calidad adaptado del modelo ISO/IEC 9126. Los autores plantean que la especificación de los atributos de calidad haciendo uso de un modelo basado en estándares internacionales ofrece una vista amplia y global de los atributos de calidad, tanto a usuarios como arquitectos del sistema, para efectos de la evaluación. El método contempla siete actividades, que son descritas en la tabla 22.

10.8. Comparación entre métodos de evaluación

La tabla 23 presenta una comparación entre los métodos de evaluación Software Architecture Analysis Method (SAAM), Architecture Trade-off Analysis Method (ATAM), Active Reviews for Intermediate Designs (ARID), Modelo de Negociación WinWin, Cost-Benefit Analysis Method (CBAM), el método de evaluación de arquitecturas propuesto por Bosch (2000) y el método de comparación de arquitecturas basada en el modelo ISO/IEC 9126 adaptado para arquitecturas de software, planteado por Losavio et al. (2003).

Etapa I	
1. Selección de atributos de calidad	Deben seleccionarse aquellos atributos que se consideran cruciales para el éxito del sistema, y cuya satisfacción resulte poco clara a nivel de arquitectura. Resulta necesario porque es poco factible y poco útil evaluar todos los atributos de calidad, dado que requiere una gran cantidad de tiempo.
2. Definición de los perfiles	Para cada atributo de calidad seleccionado, se definen los perfiles respectivos para efectos de la evaluación.
3. Selección de una técnica de evaluación	Para la evaluación de los atributos de calidad dependientes del diseño de la arquitectura se recomienda utilizar la evaluación basada en escenarios, así como también los modelos basados en métricas o modelos matemáticos. Los atributos de calidad operacionales (observables vía ejecución) pueden evaluarse con técnicas de simulación o modelos matemáticos. La selección de la técnica, y la implementación concreta de ésta depende del objetivo y exactitud de la evaluación.
Etapa II	
4. Ejecución de la evaluación	Para cada atributo de calidad, las técnicas arrojan valores cuantitativos.
5. Obtención de resultados	Los resultados se resumen en una tabla que contiene el nivel requerido, el nivel predicho, y un indicador, que puede tener diversos significados: si el atributo se satisface o no, si necesita ser negociado con el cliente, o existencia de alguna relación negativa con otro atributo de calidad. El arquitecto puede decidir acerca de la realización de transformaciones sobre la arquitectura actual, y efectuar una nueva evaluación. Una vez que concluye el proceso de evaluación, con los resultados obtenidos es posible decidir entre la continuación, renegociación o cancelación del proyecto.

Tabla 21. Etapas contempladas por el método de evaluación de arquitecturas propuesto por Bosch (2000)

Actividades
1. Analizar los requerimientos funcionales y no funcionales principales del sistema, para establecer las metas de calidad 2. Utilizar el modelo de calidad ISO/IEC 9126 adaptado para arquitecturas de software. Algunas métricas pueden definirse con mayor nivel de detalle 3. Presentar las arquitecturas candidatas iniciales 4. Construir la tabla comparativa para las arquitecturas candidatas 5. Establecer prioridades para las subcaracterísticas de calidad tomando en cuenta los requerimientos de calidad del sistema 6. Analizar los resultados obtenidos y resumidos en la tabla, de acuerdo con las prioridades establecidas

Tabla 22. Actividades contempladas en el método de comparación de arquitecturas basado en el modelo ISO/IEC 9126 adaptado para arquitecturas de software
Fuente: Losavio et al. (2003)

	ATAM	SAAM	ARID	WinWin	CBAM	Bosch (2000)	Losavio (2003)
Atributos de Calidad contemplados	✓ Modificabilidad ✓ Seguridad ✓ Confiabilidad ✓ Desempeño	✓ Modificabilidad ✓ Funcionalidad	✓ Conveniencia del diseño evaluado	✓ Funcionalidad	✓ Funcionalidad ✓ Modificabilidad	✓ Seleccionados por el arquitecto, de acuerdo a la importancia sobre el sistema	✓ Funcionalidad ✓ Confiabilidad ✓ Usabilidad ✓ Eficiencia ✓ Mantenibilidad ✓ Portabilidad
Objetos analizados	✓ Estilos arquitectónicos ✓ Documentación ✓ Flujo de datos ✓ Vistas Arquitectónicas	✓ Documentación ✓ Vistas Arquitectónicas	✓ Especificación de los componentes	✓ Conflictos entre requerimientos	✓ Estilos Arquitectónicos ✓ Documentación	✓ Vistas Arquitectónicas ✓ Estilos Arquitectónicos ✓ Patrones Arquitectónicos ✓ Patrones de Diseño ✓ Patrones de Idioma	✓ Especificación de atributos de calidad
Etapas del proyecto en las que se aplica	✓ Luego de que el diseño de la arquitectura ha sido establecido	✓ Luego de que la arquitectura cuenta con funcionalidad ubicada en módulos	✓ A lo largo del diseño de la arquitectura	✓ Luego de que el diseño de la arquitectura ha sido establecido	✓ Luego de que el diseño de la arquitectura ha sido establecido	✓ Luego de que el diseño de la arquitectura ha sido establecido	✓ Luego de que el diseño de la arquitectura ha sido establecido
Enfoques utilizados	✓ Utility Tree y lluvia de ideas para articular los requerimientos de calidad ✓ Análisis arquitectónico que detecta puntos sensibles, puntos de balance y riesgos	✓ Lluvia de ideas para escenarios y articular los requerimientos de calidad ✓ Análisis de los escenarios para verificar funcionalidad o estimar el costo de los cambios	✓ Revisiones de diseños, lluvia de ideas para obtener escenarios	✓ Teoría “W”	✓ Teoría “W” ✓ Análisis de escenarios	✓ Análisis de perfiles (*profiles*)	✓ Análisis y comparación de los resultados obtenidos para las arquitecturas candidatas

Tabla 23. Comparación entre métodos de evaluación.
Fuente: (Kazman et al. 2001)

11. HERRAMIENTAS DE ANÁLISIS, DISEÑO Y EVALUACIÓN DE ARQUITECTURAS DE SOFTWARE

Kazman (1996) plantea que una herramienta de *análisis y diseño* de arquitecturas de software debe cumplir con ciertos requerimientos, entre los que se encuentran:

- ✓ Debe ser capaz de **describir cualquier arquitectura**. En principio, debe permitir la especificación gráfica de arquitecturas, de forma similar a como se lleva a cabo en un equipo de desarrollo. Los esbozos gráficos de la arquitectura se realizan para facilitar el diseño, la exploración y la comunicación. Para lograr el mejor cumplimiento del objetivo, la herramienta deberá hacer uso de los componentes y los conectores que el equipo de desarrollo actualmente usa, más que el uso de un grupo preestablecido por la herramienta.
- ✓ Debe ser capaz de **añadir elementos arquitectónicos de forma recursiva**, y poder establecer asociaciones semánticamente significativas con elementos en otros niveles de abstracción. El cumplimiento de este requerimiento es crucial para soportar el refinamiento iterativo y el análisis de arquitecturas, donde el análisis es significativo a cualquier nivel de refinamiento. Por ejemplo, debería permitir esbozar un nodo en la arquitectura bajo el nombre "base de datos", y permitir preguntas importantes de diseño, o ejecutar análisis de desempeño, sin la necesidad de mayor descomposición del elemento definido. Con el cumplimiento de este requerimiento, la herramienta debería permitir el diseño y análisis arquitectónico en cualquier grado de *resolución*. Por supuesto, a mayor grado de resolución, mejores deben ser las respuestas a la hora de comparar varias alternativas.
- ✓ Debe ser capaz de **determinar la concordancia de interfaces entre componentes**, tanto dentro de la arquitectura que se está diseñando, como con sistemas externos. De igual forma, debe estar en capacidad de determinar la correspondencia de la arquitectura implementada con la arquitectura diseñada.
- ✓ Debe contar con la capacidad de realizar **ingeniería de reverso**.
- ✓ Debe ser capaz de **analizar arquitecturas con respecto a métricas**.
- ✓ Debe **asistir al desarrollador en el diseño**, tanto como una actividad creativa, como una actividad de análisis. En específico, se propone que **debe soportar distintos métodos de diseño**, y los procesos que estos métodos implican.
- ✓ Debe **funcionar como un repositorio** que contenga diseños, trozos de diseño, justificaciones de diseño y escenarios. Como repositorio, debe permitir la búsqueda de una arquitectura, la extracción de subconjuntos significativos de ésta, y también permitir su actualización.
- ✓ Debe proveer la **generación de plantillas de código**, para simplificar la transición del diseño al código, contribuyendo así con la consistencia entre ambos. De igual forma, debe proveer herramientas de modelaje de control de datos y flujo, así como también modelaje de desempeño.

El planteamiento de Kazman (1996) no contempla aspectos de evaluación de las arquitecturas de software diseñadas con una herramienta que cumpla con los requerimientos mencionados. Por otro lado, considerando que la calidad de un sistema de software está determinada en gran parte por su arquitectura, resulta de particular interés extender el alcance de tal herramienta, agregando la capacidad de evaluación del diseño generado.

La intención es entonces resaltar algunos de los elementos que aún pueden añadírsele a una herramienta como la propuesta por Kazman (1996), que derive en un ambiente que tenga otras capacidades no consideradas aún. Entre ellas se encuentran:

- ✓ Inclusión de nuevos elementos de descripción, tales como los ADL, vistas arquitectónicas y distintas notaciones
- ✓ Posibilidad de evaluar la calidad de la arquitectura diseñada en base a los elementos de diseño utilizados
- ✓ Ofrecer la posibilidad de soporte a los distintos métodos y técnicas de evaluación de arquitecturas de software

Para efectos de la evaluación, resulta interesante conocer los distintos tipos de decisión que pueden ser tomados a nivel de diseño, puesto que un ambiente de evaluación y análisis debe estar en capacidad de soportar estos procesos (Kazman, 1996). A nivel arquitectónico, Bredemeyer et al. (2002) establecen que los tópicos de decisión con frecuencia son:

- ✓ Estilos y patrones arquitectónicos
- ✓ Arquitecturas de referencia
- ✓ Responsabilidades asociadas a los componentes
- ✓ Identificación de interconexiones entre componentes
- ✓ Comportamiento dinámico del sistema
- ✓ Interfaces entre componentes y sus responsabilidades
- ✓ Manejo de múltiples configuraciones para sistemas distribuidos o concurrentes

Este proceso de análisis y diseño de arquitecturas de software presenta sobrecargas de recolección, manejo y presentación de la información relevante a ellos. Esto resulta un impedimento sustancial para las organizaciones que quieren adoptar una actitud más madura a su práctica en el diseño de arquitecturas de software. Proveer una herramienta que soporte estas prácticas es un primer paso de ayuda a extender su adopción en la industria (Kazman, 1996).

Es importante que el ambiente esté en capacidad de asistir en el proceso de diseño para efecto de la toma de decisiones, independientemente de la metodología y en el nivel en que se encuentre el proceso de desarrollo (Bredemeyer et al., 2002). Al añadirle la capacidad de apoyo a la evaluación del diseño, con el manejo de las técnicas y métodos existentes hasta el momento, la herramienta proveerá información acerca de los puntos de riesgo en el diseño que se está evaluando. Esto resulta de gran ayuda al arquitecto al momento de tomar decisiones que harán posible la satisfacción de los requerimientos de calidad por parte del diseño en cuestión.

Si bien es cierto que la calidad del sistema depende en gran parte de la implementación, también es cierto que gran parte de ella depende de la arquitectura. De aquí la importancia de la correspondencia entre el diseño y la implementación. Es por ello que el ambiente de análisis, diseño y evaluación de arquitecturas de software se propone como un medio que permitiría la satisfacción de los requerimientos de calidad del sistema establecidos por los involucrados en el desarrollo del mismo.

Referencias

Abowd, G., Allen, R., & Garlan, D. (1995). *Formalizing Style to Understand Descriptions of Software Architecture.* Technical Report. The Software Engineering Institute, Carnegie Mellon University. CMU-CS-95-111. Obtenido el 15-08-2002 de:
http://www-2.cs.cmu.edu/afs/cs.cmu.edu/project/able/ftp/styleformalism-tosem95/styleformalism-tosem95.pdf

Barbacci, M., Klein, M., Longstaff, T., & Weinstock, C. (1995). *Quality Attributes.* Carnegie Mellon University. Technical Report. Obtenido el 27-06-2002 de:
http://www.sei.cmu.edu/publications/documents/ 95.reports/95.tr.021.html

Bass, L., Barbacci, M., Carriere, J., Kazman, R., Klein, M., y Lipson, H. (1999). *Attribute Based Architectural Styles*. Software Engineering Institute, Carnegie Mellon University. Pittsburgh. Obtenido el 10-05-2002 de:
http://www.sei.cmu.edu/pub/documents/ 99.reports/pdf/99tr022.pdf

Bass, L., Clements, P., & Kazman, R. (1998). *Software Architecture in practice*. Addison-Wesley.

Bass, L., Klein, M., & Bachmann, F. (2000). *Quality Attribute Design Primitives*. Software Engineering Institute, Carnegie Mellon University. Obtenido el 30-06-2002 de:
http://www.sei.cmu.edu/publications/documents/ 00.reports/00tn017.html

Bengtsson, P. (1999). *Design and Evaluation of Software Architecture*. University of Karlskrona/Ronneby, Department of Software Engineering and Computer Science, Karlskrona. Obtenido el 13-05-2002 de:
http://www.ipd.hk-r.se/pob/archive/thesis.pdf

Boehm, B., & Abd-Allah, A. (1995). *Reasoning about the Composition of Heterogeneous Architecture.* USC Center for Software Engineering Technical Report. University of Southern California, Los Angeles. Obtenido el 15-08-2002 de:
http://sunset.usc.edu/TechRpts/Papers/Cmps_Reasoning.ps

Booch, G., Rumbaugh, J., & Jacobson, I. (1999). *The UML Modeling Language User Guide*. Adisson-Wesley

Bosch, J. (2000). *Design & Use of Software Architectures*. Addison-Wesley.

Bredemeyer, D., & Malan, R. (2002). *The Visual Architecting Process.* White Paper. Obtenido el 10-05-2002 de:
http://www.bredemeyer.com/pdf_files/ VisualArchitectingProcess.PDF

Brown, A., Carney, D., Clements, P., Meyers, C., Smith, D., Weiderman, N., & Wood, W. (1995) *Assessing the Quality of Large, Software-Intensive Systems: A Case Study.* Proceedings of European Software Engineering Conference-ESEC `95. Sitges, España. Obtenido el 15-08-2002 de:

http://www.sei.cmu.edu/reengineering/pubs/esec95/esec95.pdf

Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., & Stal, M. (1996). *Pattern – Oriented Software Architecture. A System of Patterns.* John Wiley & Sons, Inglaterra.

Carriere, J., Kazman, R., Woods, S. (2000). *Toward a Discipline of Scenario-based Architectural Engineering.* Software Engineering Institute, Carnegie Mellon University. Obtenido el 09-05-2002 de:
http://www.sei.cmu.edu/staff/rkazman/annals-scenario.pdf

Clements, P. (1996). *A survey of Architecture Description Languages*. Software Engineering Institute, Carnegie Mellon University. Obtenido el 16-07-2002 de:
http://www.sei.cmu.edu/publications/articles/survey-adl.html

Dromey, G. (1996). *Cornering the Chimera*. IEEE Software. Vol 13, Nro. 1. Obtenido el 15-08-2002 de:
http://www.computer.org/software/so1996/s1033abs.htm

Grady, R., & Caswell, D. (1987) *Software Metrics: Establishing a company-Wide Program*. Prentice Hall.

Grimán, A. (2000) *Propuesta Metodológica Sistémica para la Gestión del Conocimiento en las Organizaciones*.

Hofmeister, C.; Nord, R.; Soni D. (2000). *Applied Software Architecture*. Addison Wesley.

In, H., Kazman, R., y Olson, D. (2001). *From Requirements Negotiation to Software Architectural Decisions*. Software Engineering Institute, Carnegie Mellon University. Obtenido el 15-08-2002 de:
http://www.cin.ufpe.br/~straw01/epapers/paper13.pdf

ISO/IEC. (1998). *Information Technology – Software Product Quality – Part 1: Quality Model*. Obtenido el 15-08-2002 de:
http://www.usability.serco.com/trump/resources/standards.htm#9126-1

Jain, R. (1991). *The Art of Computer System Performance Analysis. Techniques for Experimental Design, Measurments, Simulation, and Modeling*. John Wiley & Sons.

Johnson, R. (1997). Frameworks = (Components + Patterns). *Communications of the ACM,* 40 (10), 39-42. Obtenido el 07-07-2003 de:
http://www-lifia.info.unlp.edu.ar/poo2000/papers-p7/Frameworks-patterns.pdf

Kan, S., Basili, V., & Shapiro, L. (1994). Software Quality: An overview from the perspective of Total Quality Management. *IBM Systems Journal*, 33 (1), 4-18. Obtenido el 15-08-2002 de:
http://www.research.ibm.com/journal/sj/331/kan.html

Kazman, R. (1996). *Tool Support for Architecture Analysis and Design.* Department of Computer Science, University of Waterloo. Obtenido el 10-05-2002 de:
ftp://ftp.sei.cmu.edu/pub/sati/Papers_and_Abstracts/ISAW-2.ps

Kazman, R., Clements, P., Klein, M. (2001). *Evaluating Software Architectures. Methods and case studies*. Addison Wesley.

Kruchten, P. (1999). *The Rational Unified Process.* Reading, MA: Addison Wesley Longman, Inc.

Lane, T. (1990). *Studying Software Architecture Through Design Spaces and Rules.* Technical report. The Computer Science Department, Carnegie Mellon University. Obtenido el 12-05-2002 de:
http://www.sei.cmu.edu/publications/documents/ 90.reports/90.tr.018.html

Larman, C. (1999). *UML y Patrones: Introducción al análisis y diseño orientado a objetos.* Prentice-Hall Hispanomericana.

Losavio, F., Chirinos, L., Lévy, N., & Ramdane-Cherif, A. (2003). *Quality Characteristics for Software Architecture.* A ser publicado en el JOT 2003.

McCall, J., Richards, P., & Waters, G. (1977). *Factors in Software Quality*. Rome Air Development Center, RADC-TR-77-369.

Perry, D., & Wolf, A. (1992). *Foundations for the Study of Software Architecture.* ACM Sigsoft - Software Engineering Notes, Vol 17, No. 4. Obtenido el 30-05-2002 de:
www.ics.uci.edu/~taylor/ICS221/papers/swa-sen.pdf

Pressman R. (2002) *Ingeniería de Software. Un Enfoque Práctico*. Quinta Edición. Mc Graw Hill.

OTI. (2003) *Eclipse Platform Technical Overview.* Obtenido el 15-04-2003 de:
www.eclipse.org/whitepapers/eclipse-overview.pdf

Pollice, Gary. (2001). *Using the Rational Unified Process for Small Projects: Expanding upon eXtreme Programming.* Rational Software Corporation. Obtenido el 19-10-2002 de:
http:// www.rational.com/media/products/rup/tp183.pdf

Rausch, A., (2001) *Towards a Software Architecture Specification Language based on UML and OCL*. Technische Universitat Munchen. Obtenido el 20-07-2002 de:
http://www4.in.tum.de/~rausch/publications/ 2001/ICSE_UML_Architecture.pdf

Rational Software Corporation. (1998). *“Rational Unified Process: Best Practices for Software Development Teams”*. Obtenido el 09-10-2002 de:
http://www.rational.com/products/rup/whitepapers/rup_bestpractices.pdf

Shaw, M., & Garlan, D. (1996). *Introduction to Software Architectures. New perspectives on an emerging discipline*. Prentice Hall.

Whitten, J., Bentley, L., & Dittman, K. (2001) *Systems Analysis and Design Methods*. 5ta Edición. McGraw-Hill.