Unidad 4 / Escenario 7

Lectura fundamental

Arquitectura de *software*

Contenido

Palabras clave: Arquitectura de software, estilos arquitectónicos, conceptos de diseño.

1. Arquitectura de Software

Para comprender qué es la arquitectura de ***software***, empecemos por entender qué es la arquitectura de un sistema, ya que, si bien es cierto que un ***software*** es un tipo de sistema, el concepto de arquitectura aplica para cualquier tipo de sistema. Así, si un sistema es un conjunto ordenado de elemento interrelacionados y que interactúan entre sí, la arquitectura es una descripción de dicho sistema que incluye los elementos, sus propiedades y sus relaciones a través de un lenguaje común mediante el cual los interesados o ***stakeholders*** comprenden el sistema y se comunican para tomar decisiones importantes sobre el mismo. Esta descripción permite:

1. Administrar (reducir) la complejidad del sistema (describir y comprender)
2. Facilitar la comunicación entre todos los interesados en el sistema
3. Apoyar la toma de decisiones

Para entender como lo lograr es necesario comprender los siguientes conceptos:

1.1. Stakeholder o Interesado

De acuerdo con Rozanski y Woods (2012), "Un ***stakeholder*** (interesado) en una arquitectura de ***software*** es una persona, grupo o entidad con un interés o preocupación sobre la realización de la arquitectura". Los ***stakeholders*** tienen distintas responsabilidades durante del desarrollo del ***software,*** entre otras: financiación, desarrollo, pruebas, uso y mantenimiento. Además, los ***stakeholders*** esperan que el ***software*** opere de una forma determinada no sólo en términos de su funcionalidad si no también términos de los atributos de calidad como la confiabilidad o seguridad del sistema.

Los ***stakeholders*** tienen bastante influencia sobre la arquitectura ya que el propósito final de la arquitectura es garantizar que los intereses de todos se están atendiendo. Sin embargo, en algunos casos es posible que los intereses de los distintos ***stakeholders*** entren en conflicto, es responsabilidad de arquitecto producir una arquitectura que satisfaga las necesidades de los involucrados, y en el caso de conflictos, ofrecer una solución que favorezca a la mayoría de ***stakeholdes*** negociando y comunicándoles oportunamente cuando sus necesidades sean restringidas para favorecer los intereses de otros. Ejemplos de ***stakeholders*** son: desarrolladores, usuarios, patrocinadores, clientes, testadores, gerentes, diseñadores yo encargados de otros sistemas con los que interactúa el sistema.

1.2. Concern o preocupación

Son los principales intereses de los ***stakeholders*** en el sistema, y determinan la aceptabilidad del sistema. Las preocupaciones pueden referirse a cualquier aspecto del funcionamiento del sistema, desarrollo o funcionamiento, incluidas cuestiones tales como el rendimiento, fiabilidad, seguridad, distribución y capacidad de evolución. Para lograr la descripción del sistema, se asume entonces que cada uno de los ***stakeholders*** se interesa por partes o aspectos específicos del sistema o ***Stakeholder Ing. Eléctrico;*** difícilmente un ***stakeholder*** se podría fijar en todo el sistema en su detalle y complejidad. Por esta razón, la arquitectura se compone de vistas, que son representaciones de todo el sistema desde la perspectiva de un conjunto relacionado de ***Stakeholder Ing. Eléctrico***. Cada vista es tomada desde un punto de vista

Un punto de vista es la definición de:

- ¿cómo construir y usar una vista?
- ¿qué información debe aparece?
- ¿cuál es el propósito de la vista?
- ¿a quién va dirigida?
- ¿cómo analizar la información?

Una vista puede estar compuesta por uno o varios modelos. Un modelo es una representación de un sistema Los modelos definen los aspectos estáticos o dinámicos del sistema. En escenarios anteriores se ha visto ejemplo de modelos que ahora se integran para conformar la arquitectura del sistema. Estos son: diagramas de clases, casos de uso, diagramas de secuencias, diagramas de estados, entre otros. Rozanski y Woods (2012). Para comprender un poco mejor estos conceptos de arquitectura vamos a ver un ejemplo desde la perspectiva de la disciplina que dio lugar a estos conceptos, es decir de la arquitectura, vista como: la técnica y estilo con los que, se diseña, proyecta y construye un edificio o un monumento. En este caso presentado en la Ilustración 1. Ejemplo de Arquitectura de un sistema, podemos ver que si el sistema se trata de una casa.

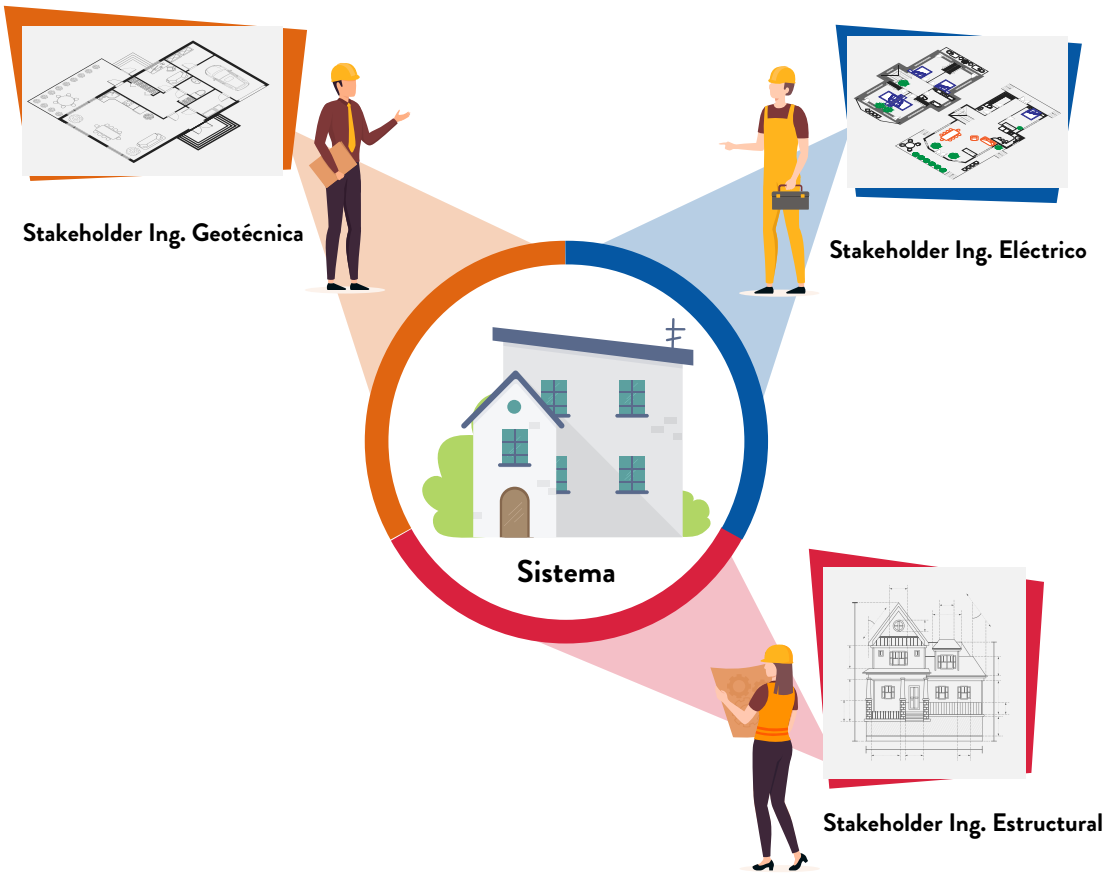

Figura 1. Ejemplo de Arquitectura de un sistema

Fuente: elaboración propia

En este ejemplo, vemos que en la construcción del sistema casa se involucran distintos tipos de profesionales en este caso Ingenieros estructurales, eléctricos y geotécnicos y cada uno observa el sistema o la casa desde la perspectiva de su especialidad generando planos o modelos que se ocupan de sus intereses y están diseñados siguiendo un lenguaje definido. Así el sistema es la casa, los ***stakeholders*** son cada uno de los ingenieros, el punto de vista está definido por su profesión desde la cual se define: 1) cuáles son los intereses de este profesional en el sistema 2) qué modelos o planos usa para describir estos intereses y las decisiones que toma para atender dichos intereses 3) cuál es el lenguaje más apropiado para hacerlo ya que cada uno usa planos distintos que se construyen de una manera estandarizada para que cualquier otro profesional de la misma disciplina pueda interpretarlos. Los modelos son cada uno de los planos que cada ***stakeholder*** realiza. Adicionalmente, las arquitecturas permiten describir estados del sistema en el tiempo:

- La arquitectura actual (AS-IS): es la descripción del sistema en la situación actual
- La arquitectura objetivo (TO-BE): es la descripción del sistema deseado para un futuro definido
- Arquitecturas de transiciones, define situaciones intermedias que permiten llegar a la solución deseada

Entonces, adoptar el enfoque arquitectural para describir el sistema permite y requiere a la vez:

1. Separar los S***takeholder Ing. Eléctrico*** o preocupaciones de acuerdo con el punto de vista de los stakeholders.
2. Definir modelos estáticos que representen la estructura del sistema y modelos dinámicos que representen su comportamiento.
3. Facilitar la comunicación con los grupos de interesados ***stakeholders.***
4. Administrar (reducir) la complejidad del sistema.
5. Planear la implementación de la arquitectura definiendo la arquitectura actual, objetivo y de transición (tantas como sean necesarias

1.3. Definiciones de Arquitectura de Software

Habiendo comprendido los conceptos básicos de arquitectura de un sistema, ahora se verán las definiciones de arquitectura de ***software*** propuestas por distintos autores que son referentes del tema.

1. "El conjunto de estructuras necesarias para razonar sobre el sistema, que comprende elementos de software, relaciones entre ellos y propiedades de ambos" (Bass, 2012).
2. "La arquitectura del software de un programa o sistema informático es la estructura o estructuras del sistema, que comprende elementos de software, las propiedades visibles externamente de esos elementos y las relaciones entre ellos." (Bass, 2012). En esta definición, podemos ver que se hace énfasis en los aspectos visibles de los elementos entre sí, es decir, que en la arquitectura se enfatiza en los aspectos de interacción o relación entre elementos y se omiten los detalles de implementación interna inherente a cada elemento. Adicionalmente, podemos ver que todo sistema tiene una arquitectura, independientemente de que tan bien documentada o conscientemente diseñada y gestionada esté siendo.

3. “La arquitectura se define la organización fundamental de un sistema, conformada por sus componentes, las relaciones entre estos y el medio ambiente, y los principios que rigen su diseño y evolución” ANSI/IEEE Std 1471-2000.

4. Una arquitectura es el conjunto de decisiones importantes sobre la organización de un sistema de software, la selección de los elementos estructurales, las interfaces, mediante las cuales estos elementos se conectan para componer el sistema, y como estos elementos se componen para formar subsistemas y el estilo arquitectónico que guía esta organización” (Rational Unified Process, 1999).

5. “Una arquitectura de sistema de software comprende: 1) Una colección de componentes de software y sistema, conexiones y restricciones, 2) • Una colección de declaraciones de necesidades de los interesados del sistema. 3) • Una justificación que demuestre que los componentes, las conexiones y las restricciones definen un sistema que, si se implementa, satisfaría las necesidades de los interesados del sistema” (Boehm, 1995).

6. Soni, Nord y Hofmesister proponen que la arquitectura de software tiene al menos cuatro perspectivas diferentes: 1) La arquitectura conceptual que describe el sistema en términos de sus principales elementos de diseño y las relaciones entre ellos, 2) La arquitectura de interconexión del módulo abarca dos estructuras ortogonales: descomposición funcional y capas, 3) La arquitectura de ejecución describe la estructura dinámica de un sistema y 4) la arquitectura del código describe cómo se organizan el código fuente, los binarios y las bibliotecas en el entorno de desarrollo.

Estos son solo algunas de las múltiples definiciones que se pueden encontrar del concepto de arquitectura de ***software*** en la literatura relacionada. En la mayoría de estas definiciones se resalta el concepto de elemento arquitectural, a continuación, verás los tipos de elementos que pueden conformar una arquitectura de software.

1.4. Elemento arquitectural

Son las partes que conforman el sistema, en ese caso el software. Estos elementos se clasifican en dos tipos: estáticos y dinámicos.

- **Elementos estáticos:** "Los elementos estáticos de un sistema de software definen las partes del sistema y cuál será su organización. Ese tipo de elemento refleja el sistema durante el diseño y está constituido por elementos de software (Ej., módulos, clases, paquetes, procedimientos o servicios auto-contenidos), elementos de datos (Ej., entidades y tablas de bases de datos, archivos de datos o clases de datos) y elementos de hardware (Ej., los ordenadores en que se van a ejecutar en el sistema u otros tipos de hardware que usará el sistema: routers, switch o impresoras). Los elementos estáticos no consisten sólo en las partes estáticas del sistema, sino también de cómo estos se relacionan entre sí. Las asociaciones, composiciones y otros tipos de relaciones entre elementos de ***software,*** de datos de hardware forman el aspecto estático que compone la arquitectura del sistema." (Arias y Durango, 2016)
- **Elementos dinámicos:** "definen el comportamiento del sistema. Este tipo de elemento se refleja en el sistema durante la ejecución y en éste están incluidos los procesos, los módulos, los protocolos o las clases que realizan el comportamiento. Los elementos dinámicos también describen como el sistema reacciona a los estímulos internos y externos" (Arias y Durango, 2016).

Adicionalmente, en algunas de las definiciones de arquitectura de software se hace énfasis, en la arquitectura debe hacerse cargo de: 1) relacionar los elementos arquitecturales y sus relaciones con la satisfacción de las necesidades de los interesados y 2) el diseño y evolución del sistema, estos dos aspectos se logran a través de las decisiones arquitecturales, a continuación, se define este concepto.

1.5. Decisiones arquitecturales

La estructura del sistema, que se representa por medio de la arquitectura no se define de manera aleatoria, sino que debe tomarse de manera que promueva los objetivos del sistema y promuevan el cumplimiento de los atributos de calidad deseado, para esto el arquitecto debe decidir entre distintas alternativas cómo va a fraccionar el sistema, es decir, qué elementos harán parte del sistema y cómo se relacionarán dichos elementos. Dichas decisiones son llamadas decisiones arquitecturales.

De acuerdo con Arias y Durango (2016) una decisión arquitectural es: "Una elección entre las alternativas de diseño arquitectural. Esa elección se propone para alcanzar uno o más atributos de calidad del sistema, por medio de la(s) estructura(s) arquitecturales que esta envuelve o define" Arias 2016. Además, de acuerdo con Arias y Durango (2016), las decisiones arquitecturales deben cumplir con tres características fundamentales: descripción, objetivos y fundamentación.

- **Descripción:** es la explicación detallada de lo que se haya decidido para el sistema, ya sea generación o modificación de un elemento arquitectural en su estructura o comportamiento, la relación de un elemento con otro, o la decisión de agregar diversos elementos para generar otro elemento que formar un servicio o componente más complejo o cualquier otra decisión que permita la evolución del sistema.
- **Objetivo:** es el propósito que se persigue al implementar dicha decisión, generalmente el objetivo debe estar definido en términos de que objetivos de negocio o atributos de calidad deseados se buscan favorecer. Es relevante tener en cuenta que un objetivo de negocio es favorecer un atributo de calidad del sistema y a su vez restringir otro u otros. Además, para lograr atender los atributos de calidad de un sistema, generalmente se deben implementar varias decisiones arquitecturales así que esto debe quedar muy claro en la definición del objetivo para poder evaluar si el objetivo de la decisión arquitectural logra el propósito esperado y rastrear las decisiones que han tomado en el tiempo para favorecer un atributo de calidad.
- **Fundamentación:** explica o justifica por qué se tomó esta decisión y no otra. Para esto el arquitecto puede apoyarse en estándares, buenas prácticas o patrones definidos por la industria que permite justificar que esta decisión se toma porque hay evidencia previa de que d los mejores resultados en circunstancias similares, o porque al evaluar varias alternativas se puede prever que está es la alternativa que dará mejores resultados.

¿Sabía que...?

de acuerdo con (Pressman, 2010) “La importancia del diseño de la arquitectura se resume, en una palabra: calidad.” El diseño es el mecanismo que permite describir los requerimientos de los participantes para que se reflejen en el producto terminado, se pueda verificar que el sistema cumple con los requerimientos y se pueden introducir cambios evaluando el impacto de éste en los demás elementos del sistema.

A continuación, veremos algunos lineamientos que propone Pressman (2010) para determinar que la calidad de un buen diseño:

1. "Debe tener una arquitectura que 1) se haya creado con el empleo de estilos o patrones arquitectónicos reconocibles, 2) esté compuesta de componentes con buenas características de diseño, y 3) se implementen en forma evolutiva, de modo que faciliten la implementación y las pruebas.
2. Debe ser modular, es decir, el ***software*** debe estar dividido de manera lógica en elementos o subsistemas.
3. Debe contener distintas representaciones de datos, arquitectura, interfaces y componentes.
4. Debe conducir a estructuras de datos apropiadas para las clases que se van a implementar y que surjan de patrones reconocibles de datos.
5. Debe llevar a componentes que tengan características funcionales independientes.
6. Debe conducir a interfaces que reduzcan la complejidad de las conexiones entre los componentes y el ambiente externo.
7. Debe obtenerse con el empleo de un método repetible motivado por la información obtenida durante el análisis de los requerimientos del ***software.***
8. Debe representarse con una notación que comunique con eficacia su significado." (Pressman, 2010)

1.6. Estilos arquitectónicos

Los estilos arquitectónicos son descripciones que permiten diferenciar o clasificar las arquitecturas de acuerdo con algunas características. De la misma manera en que en la construcción de edificios se pueden clasificar y diferenciar por estilos arquitectónicos que definen por ejemplo la época o el autor. Al elegir un estilo arquitectónico, éste da las pautas para la construcción del sistema, ya sea un ***software*** o un edificio. En cuanto a sistema de ***software,*** cada estilo arquitectónico describe una categoría de sistemas que incluye:

1. "un conjunto de componentes (como una base de datos o módulos de cómputo) que realizan una función requerida por el sistema,

2. un conjunto de conectores que permiten la "comunicación, coordinación y cooperación" entre los componentes,
3. restricciones que definen cómo se integran los componentes para formar el sistema y
4. modelos semánticos que permiten que un diseñador entienda las propiedades generales del sistema al analizar las propiedades conocidas de sus partes constituyentes" (Pressman, 2010).

Algunas de los estilos arquitectónicos más relevantes son los siguientes:

1.6.1. Arquitecturas centradas en datos

En esta arquitectura hay repositorio central de datos al cual acceden otros componentes que realizan acciones sobre estos datos.

1.6.2. Arquitectura de llamada y retorno

Este estilo se subdivide en los siguientes estilos:

- Arquitecturas de programa principal/subprograma.
- En esta estructura el programa se descompone su funcionalidad, en un programa principal o "main" que invoca otros componentes del programa que a su vez puede invocar a otros para ejecutar la funcionalidad requerida.
- Arquitecturas de llamada de procedimiento remoto.
- Los distintos componentes de la arquitectura están distribuidos en múltiples computadores que están conectadas por en una red. Es decir, que una máquina puede ejecutar código localizado en otra máquina. En este estilo de arquitecturas se clasifican las arquitecturas conocidas como: cliente/servidor.

1.6.3. Arquitecturas orientadas a objetos

Los componentes de un sistema encapsulan datos y las operaciones que deben aplicarse para manipular los datos. La comunicación y coordinación entre los componentes se consigue mediante el paso de mensajes. Los componentes de estilo son los objetos, los cuales se basan en los principios de: encapsulamiento, herencia y polimorfismo.

En la etapa de diseño se definen los siguientes tipos de clases:

- Clases de interfaz de usuario. Definen todas las abstracciones necesarias para la interacción entre el humano y el sistema.
- Clases del dominio de negocios: estas clases corresponden a refinamiento de las clases que se han definido en el análisis, en el diseño se deben definir los atributos y métodos que se deben implementar para algunos elementos del dominio de negocio.
- Clases de proceso. Estas clases administrar y guían la interacción entre las clases del dominio de negocio de acuerdo con los procesos que se ejecutan en el sistema.
- Clases persistentes. Son las clases encargadas del almacenamiento y manipulación de los datos que persisten más allá de la ejecución del ***software.***
- Clases de sistemas. Implantan las funciones de administración y control del ***software*** que permiten que el sistema opere y se comunique dentro de su ambiente de computación y con el mundo exterior.

1.6.4. Arquitecturas en capas

En este estilo se define un número determinado de capas, en cada capa se ejecutan distintos tipos de funciones. Cada capa tiene roles y responsabilidades definidas que se ejecutan de manera secuencial permitiendo la separación de responsabilidades. El rol de cada capa indica el modo y tipo de interacción con otras capas, y la responsabilidad indica la funcionalidad que está siendo desarrollada.

1.6.5. Arquitecturas orientadas por componentes

Este estilo arquitectural se enfoca en descomponer el sistema en subsistemas que son los componentes funcionales o lógicos. Cada uno de estos componentes tiene una funcionalidad claramente definida y expone una interfaz de comunicación a través de la cual ofrece dicha funcionalidad. Este estilo tiene un nivel mayor de abstracción al estilo orientado por objetos. Los principios que se deben seguir para identificar y diseñar un componente son los siguientes:

1. "Reusabilidad. Cada componente debe estar diseñado para ser utilizado en distintos escenarios por diferentes aplicaciones.
2. Independientes del contexto. Los componentes deben poder operar en distintos contextos, para esto la información que el componente requiere para operar en un determinado contexto debe ser pasadas al componente a través de su interfaz, no deben ser incluidos o en el componente.
3. Extensible. Un componente puede ser extendido desde un componente existente para crear un nuevo comportamiento.
4. Encapsulado. Los componentes exponen interfaces que permiten al programa usar su funcionalidad. Sin revelar detalles internos, detalles del proceso o estado
5. Independiente. Los Componentes están diseñados para tener una dependencia mínima de otros componentes. Por lo tanto, los componentes pueden ser instalados en el ambiente adecuado sin afectar otros componentes o sistemas." (Szyperski, 1996).

1.6.6. Arquitecturas orientadas por servicios

De acuerdo con Marks 2006 SOA es una combinación de: un modelo de negocio, una estrategia de TI y una propuesta arquitectural. SOA como modelo de negocio se basa en la oportunidad de ejecutar algunas funciones del negocio basado en outsourcing. Como estrategia de TI se basa en la idea de proveer servicios de tecnología y activos. Desde el punto de vista técnico una arquitectura orientada a servicios es un paradigma para organizar y utilizar capacidades distribuidas que pueden estar bajo el control y gobierno de diferentes dominios. Este paradigma se basa en la idea de que las entidades (personas y organizaciones) crean capacidades para dar solución a necesidades que enfrentan en la ejecución de sus procesos de negocio. Estas entidades pueden crear capacidades que solucionan las necesidades de otras y viceversa.

Una necesidad es un requerimiento medible que se busca satisfacer, mientras que una capacidad es un recurso que puede ser usado para satisfacer determinada(s) necesidad(es). Sin embargo, no existe necesariamente una correlación uno a uno entre las necesidades y capacidades; de manera que para satisfacer determinada necesidad puede ser necesario combinar múltiples capacidades, mientras que una capacidad puede abarcar más de una necesidad. Esto genera una gran complejidad, el gran aporte que ofrece SOA como propuesta arquitectural es que proporciona un marco bajo es cual es posible combinar las capacidades ofrecidas para hacer frente a las necesidades de un negocio promoviendo la reutilización e interoperabilidad de las capacidades.

2. Conceptos de diseño

En esta sección se definen algunos conceptos que Pressman (2010) considere importantes en el diseño de software, tanto del desarrollo tradicional como del orientado a objeto.

2.1. Abstracción

La abstracción define el nivel de detalle con el que se describe un sistema o alguno de sus componentes. “En el nivel más elevado se enuncia una solución en términos gruesos con el uso del lenguaje del ambiente del problema. En niveles más bajos de abstracción se da la descripción más detallada de la solución.

La terminología orientada al problema se acopla con la que se orienta a la implementación, en un esfuerzo por enunciar la solución. Por último, en el nivel de abstracción más bajo se plantea la solución, de modo que pueda implementarse directamente.

Una abstracción de procedimiento es una secuencia de instrucciones que tienen una función específica y limitada. El nombre de la abstracción de procedimiento implica estas funciones, pero se omiten detalles específicos. Un ejemplo de esto sería la palabra ***abrir***, en el caso de una puerta. *Abrir* implica una secuencia larga de pasos del procedimiento (caminar hacia la puerta, llegar y tomar el picaporte, girar éste y jalar la puerta, retroceder para que la puerta se abra, etcétera).

Una abstracción de datos es un conjunto de éstos con nombre que describe a un objeto de datos. En el contexto de la abstracción de procedimiento abrir, puede definirse una abstracción de datos llamada **puerta**. Como cualquier objeto de datos, la abstracción de datos para puerta agruparía un conjunto de atributos que describirían la **puerta** (tipo, dirección del abatimiento, mecanismo de apertura, peso, dimensiones, etc.). Se concluye que la abstracción de procedimiento abrir usaría información contenida en los atributos de la abstracción de datos **puerta.**" (Pressman, 2010)

2.2. Patrones

Los patrones de diseño proponen una solución a problemas que son comunes en el desarrollo de ***software.***

En otras palabras, brindan una solución ya probada y documentada a problemas de desarrollo de ***software*** que están sujetos a contextos similares. En el siguiente escenario se desarrollará con más detalle qué son los patrones y se verán algunos patrones de diseño que aplican al modelo de clases.

2.3. Modularidad

"El modularidad es la manifestación más común de la división de problemas. El software se divide en componentes con nombres distintos y abordables por separado, en ocasiones llamados ***módulos***, que se integran para satisfacer los requerimientos del problema." (Pressman, 2010)

2.4. Ocultamiento de información

"Este criterio de diseño, propone que, los módulos se especifiquen y diseñen de tal forma que la información contenida en un módulo sea inaccesible para los que no necesiten de ella." (Pressman, 2010). De esta manera, el modularidad se logra definiendo un conjunto de módulos independientes que intercambien sólo aquella información necesaria para lograr la función del ***software***. El uso del ocultamiento de información como criterio de diseño para los sistemas modulares proporciona los máximos beneficios cuando se requiere hacer modificaciones durante las pruebas, y más adelante, al dar mantenimiento al ***software.***

Debido a que la mayoría de los datos y detalles del procedimiento quedan ocultos para otras partes del ***software***, es menos probable que los errores inadvertidos introducidos durante la modificación se propaguen a distintos sitios dentro del ***software.***

2.5. Independencia funcional

Este principio propone que cada módulo resuelva un subconjunto específico de requerimientos y tenga una interfaz sencilla cuando se vea desde otras partes de la estructura del programa. De esta manera las funciones del sistema se subdividen y las interfaces se simplifican. La independencia funcional es una clave para el buen diseño y éste es la clave de la calidad del ***software.*** La independencia se evalúa con el uso de dos criterios cualitativos: la cohesión y el acoplamiento.

- La ***cohesión*** es un indicador de la fortaleza relativa funcional de un módulo, es decir que desarrolla una sola tarea, o varias que están relacionadas. El acoplamiento es una indicación de la interconexión entre módulos en una estructura de ***software***, y depende de la complejidad de la interfaz entre módulos, del grado en el que se entra o se hace referencia a un módulo y de qué datos pasan a través de la interfaz. En el diseño de ***software***, debe buscarse el mínimo acoplamiento posible

Estos conceptos de cohesión y acoplamiento serán retomados en el siguiente escenario al ver patrones de diseño que favorecen que el sistema tenga alta cohesión y bajo acoplamiento.

2.6. Refinamiento

El refinamiento, es el proceso que complementario al de abstracción. Mientras que la abstracción permite ocultar los detalles de bajo nivel. El refinamiento ayuda a revelar estos detalles a medida que se avanza en el diseño. En resumen, la arquitectura de ***software*** es un mecanismo a través del cual se facilita gestionar el sistema y es sobre ésta arquitectura que se pueden elegir los estilos arquitecturales que mejor satisfagan las necesidades de los interesados y se aplican las lecciones aprendidas que se adquieren a través de la experiencia en la gestión de arquitecturas y que se traducen en patrones de diseño.

Referencias

Arias, A., Durango, A. (2016). Ingeniería y Arquitectura del Software. IT Campus Academy.

Bass, L. (2013). Software Architecture in Practice. México: Addison Wesley.

Boehm, B. (1995). Anchoring the Software Process. IEEE Software.

Hofmeister, C., Nord, R.,y Soni, D. (2018). Applied Software Architecture. Addison-Wesley Professional

Pressman, R. (2010). Ingeniería del software. Un enfoque práctico. México, D. F, México: McGraw Hill (pp. 1,26)

Rozanski, N. y Woods E. (2012). Software Systems Architecture: Working With Stakeholders Using Viewpoints and Perspectives. New York: Addison-Wesley Professional

Szyperski, C. (1996). Independently Extensible Systems —Software Engineering Potential and Challenges. En Proc. of the 19th Australasian Computer Science Conference, Melbourne.

INFORMACIÓN TÉCNICA

FACULTAD DE
INGENIERÍA, DISEÑO
E INNOVACIÓN

Módulo: Ingeniería del Software I

Unidad 4: Diseño de Software

Escenario 7: Arquitectura de Software

Autor: Diana Angélica Cruz Ortega

Asesor Pedagógico: Luisa Esperanza Rincón Jiménez

Diseñador Gráfico: Cristian Navarro

Asistente: Ginna Quiroga