UNIVERSIDAD DE COSTA RICA

SEDE GUANACASTE, RECINTO LIBERIA

BACHILLERATO EN INFORMÁTICA EMPRESARIAL

IF-7100 INGENIERÍA DE SOFTWARE

GRUPO 001

ARQUITECTURA DE SOFTWARE

ESTUDIANTES:

GRIJALBA ORTEGA ANTHONY	B73453
MELÉNDEZ CONTRERAS ANA	B74642
MORENO CHAVARRIA ISABELA	B75235

PROFESOR: LIC IVAN ALONSO

LIBERIA, MIÉRCOLES 18 DE MAYO

Índice

Introducción **3**

Objetivos **4**

Desarrollo **5**

Arquitectura de software 5
Conceptos básicos y ciclo de desarrollo 7
Beneficios de la arquitectura 9
El rol del arquitecto 9
Arquitectura versus Diseño 12
Estilos Arquitectónicos 14
Estilo modulo 14
Estilo componente y conector (C&C) 15
Estilo de asignación 15
Diagrama de contexto 16
Vistas arquitectonicas 4+1 17
Arquitectura Física 18
Arquitectura Lógica 21
Capas lógicas de implementación 22
Arquitectura de software en metodologías ágiles 25

Conclusiones **27**

Bibliografía **28**

Introducción

Durante los últimos años la tecnología se ha incrementado exponencialmente creando nuevas herramientas para el servicio del sector industrial, consumo y herramientas para facilitar y mejorar la calidad de vida del ser humano, lo cual conlleva de igual manera un crecimiento exponencial en el desarrollo de software.

Este crecimiento provoca que los desarrolladores de software deban tener más cuidado y mayor conocimiento de lo implica desarrollar, por ende se deben conocer las diferentes fases por las que pasa un software. No obstante existen diferentes herramientas y metodologías de las cuales los ingenieros en software hacen uso, ya que facilitan el proceso del desarrollo de sistemas.

Dicho lo anterior es de vital importancia tener un conocimiento claro de cada una de las fases que se atraviesa durante el desarrollo de sistemas, por lo tanto, en esta investigación se abordará un tema de suma importancia como lo es la arquitectura de software como un pilar fundamental en el prototipo y puesta en marcha de un sistema, además se hablará de las partes que la conforman y diferentes estilos arquitectónicos que se pueden implementar para el desarrollo de software.

Objetivos

Objetivo General

Investigar en qué consiste la Arquitectura de Software, cuáles son las partes que la conforman y su importancia en el desarrollo de sistemas.

Objetivo Específicos

- Definir qué es Arquitectura de software
- Mencionar las actividades que se realizan en el proceso de arquitectura
- Describir el Ciclo de desarrollo de la arquitectura
- Señalar los beneficios del uso de la arquitectura de software
- Indicar cual es el rol del arquitecto
- Explicar la diferencia entre arquitectura y diseño de software
- Definir qué son los estilos de arquitectura
- Mostrar diferentes tipos de estilos de arquitectura: modulo, componente y conector, estilo de alocación.
- Explicar que es un Diagrama de Contexto
- Describir qué son las vistas 4+1
- Explicar la arquitectura Fisica y Logica
- Definir las capas lógicas de implementación

Desarrollo

Arquitectura de software

La arquitectura de software es una industria muy relevante en la actualidad la cual no siempre ha sido tomada en serio ni ha recibido el debido respeto que se merece. En el mundo del desarrollo de software muchas veces nos encontramos con sistemas complejos, sin embargo, si bien es cierto que no todo el software debe ser estrictamente complejo, si resulta necesario que tenga una base sólida de tal forma que se facilite el mantenimiento y el futuro desarrollo del mismo.

El mantenimiento es un aspecto importante, ya que si bien es posible desarrollar un software en unas cuantas semanas o meses es muy probable que necesite ser mantenido durante varios años, ya sea a través de funcionalidades nuevas o bien la corrección de problemas. El crecimiento también es esencial, ya que cualquier software que no se haya ampliado o modificado tiende a quedar inutilizable durante un período de tiempo relativamente corto.

En tanto que el software comienza a crecer y se torna más complejo, es importante que disponga de una forma bien definida con el fin de que podamos entenderlo como un todo. Por otro lado, si examinamos cada una de sus partes por separado lo más usual es que no podamos explicar exactamente cómo funciona y por qué existe.

Por tanto, de acuerdo con el Software Engineering Institute (SEI) la arquitectura de software se define como la estructura o estructuras del sistema que comprenden elementos de software, las propiedades visibles externamente de esos elementos y las relaciones entre ellos. Dentro de este concepto la definición de los elementos es algo ambigua, ya que puede referirse a varias entidades relacionadas con el sistema, estas pueden encontrarse en el tiempo de ejecución en forma de objetos o subprocesos, entidades lógicas en tiempo de desarrollo en referencia a las clases o componentes, o bien entidades físicas como los nodos o directorio.
En este proceso se realizan diversas actividades, tales como:

- ***Dividir requerimientos:*** Se analizan y organizan los requerimientos del sistema en grupos afines. Por lo general existen múltiples opciones posibles de división y suelen sugerirse diferentes alternativas en esta etapa del proceso.
- ***Identificar subsistemas:*** En este punto se deben identificar los subsistemas que pueden cumplir con los requerimientos ya sea de manera individual o colectiva. Se suelen relacionar los grupos de requerimiento con los

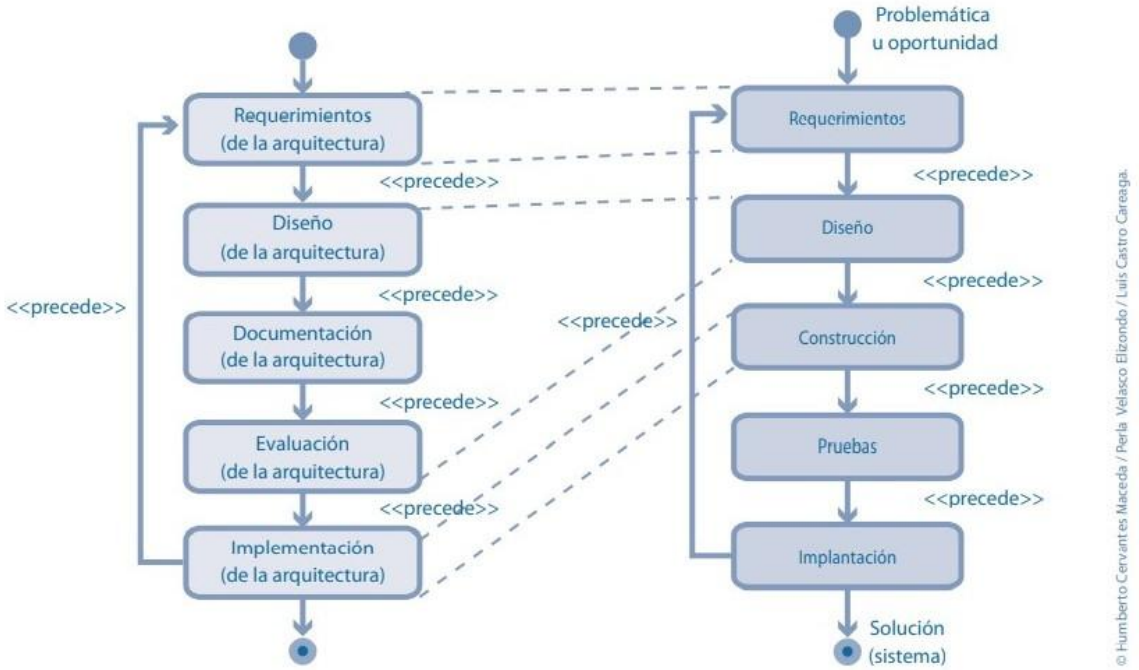
subsistemas, de tal manera que se puede realizar la división de requerimientos y la identificación de subsistemas de manera simultánea, no obstante, el proceso de identificación de los subsistemas puede verse afectado por factores del entorno u organizacionales.

- ***Asignar requerimientos a los subsistemas:*** En la teoría este paso suele ser sencillo si la división de los requerimientos es utilizada para identificar los subsistemas, sin embargo, en la práctica no existe una igualdad entre las dos etapas previamente mencionadas. Las limitantes de los subsistemas comerciales pueden significar un cambio en los requerimientos con el fin de ajustarlos a estas restricciones
- ***Especificar las funcionalidades de los subsistemas:*** En este punto deben numerarse las funciones específicas propias de cada subsistema. Esta fase suele verse semejante a la etapa de diseño del sistema, o bien como parte de la especificación de requerimientos si el subsistema es un sistema de software. Además de esto, se deben especificar las relaciones entre los subsistemas.
- ***Definir las interfaces de los subsistemas:*** Se definen las interfaces requeridas por cada uno de los subsistemas. Una vez finalizado este proceso comenzará el desarrollar los subsistemas definidos.

Conceptos básicos y ciclo de desarrollo

Ciclo de desarrollo de la arquitectura

Durante las actividades técnicas para el desarrollo de sistemas, podemos hablar de un ciclo de desarrollo de la arquitectura de software que engloba actividades particulares. Estas se describen a continuación y se integran a las actividades técnicas del desarrollo de sistemas.



Podemos hablar del desarrollo de arquitectura de software en cualquier proyecto de desarrollo e independientemente de la metodología utilizada. Previo a la construcción del sistema dicho desarrollo se divide en las siguientes fases:

1. **Requerimientos:** Esta fase se centra en capturar, documentar y priorizar los requerimientos que afectan a la arquitectura, los atributos de calidad juegan un papel importante en estos requerimientos por lo que se destacan primordialmente en esta etapa. No obstante, los requerimientos funcionales primarios, los casos de uso y las restricciones también son otros requisitos relacionados con la arquitectura. La especificación de los requerimientos sucede durante la etapa de análisis.
2. **Diseño:** La fase de diseño es la etapa principal en relación con la arquitectura y es quizás la más compleja de todas. En esta fase se definen las estructuras que conforman la arquitectura a través de la toma de decisiones de diseño. Dicha estructuración generalmente se realiza con base en dos tipos de

soluciones abstractas probadas conocidas como patrones y tácticas de diseño, al igual que en soluciones concretas tecnológicas como los frameworks.

El diseño realizado debe buscar principalmente cumplir con los requerimientos que afectan a la arquitectura y no solamente integrar diferentes tecnologías solo porque son populares.

3. **Documentación:** Una vez creado el diseño arquitectónico del software es necesario informar a las otras partes interesadas en el sistema, tales como los desarrolladores, los responsables de la implementación, gerentes de proyecto o los propios clientes. A menudo la comunicación exitosa depende de un diseño debidamente documentado; si bien la documentación inicial se realiza durante la etapa de diseño y puede incluir bocetos estructurales o capturas de pantalla de las decisiones de diseño, la documentación formal implica describir su estructura a través de vistas.

   Una vista representa una estructura y generalmente contiene un diagrama junto con información adicional para que el diagrama sea más fácil de entender.
4. **Evaluación:** Puesto que la arquitectura de software juega un papel importante en el desarrollo, con el fin de identificar posibles riesgos o inconvenientes, resulta conveniente evaluar el diseño una vez que se haya documentado. El beneficio de dicha evaluación es que representa una actividad que se puede realizar de forma temprana, incluso antes de codificar, y que el costo de corregir los errores identificados es mucho menor que el costo de corregirlos después que el sistema fue construido.
5. **Implementación:** Una vez que ha sido establecida la arquitectura comienza a construirse el sistema. Durante esta etapa es importante evitar desviaciones del diseño previamente definido por el arquitecto.

Beneficios de la arquitectura

- **Aumenta la calidad del sistema:** La relación entre arquitectura y calidad es directa ya que permite satisfacer los atributos de calidad de un sistema y estos son, a su vez, una de las dos dimensiones principales asociadas con la calidad de los sistemas, siendo la segunda el número de defectos.

- **Mejora el tiempo de entrega de los proyectos:** Algunos de los elementos que se identifican dentro de las estructuras arquitectónicas ayudan directamente a llevar a cabo estimaciones más precisas del tiempo requerido para el desarrollo.Una estructuración adecuada ayuda a asignar el trabajo y facilita el desarrollo en paralelo del sistema por parte de un equipo.

- **Reduce costos del desarrollo:** La reutilización es un factor importante en el momento de hacer un diseño arquitectónico porque ayuda a reducir costos. Un buen diseño contribuye a aminorar la necesidad de volver a hacer el trabajo y facilita el mantenimiento, lo cual también conduce a bajar los gastos.

El rol del arquitecto

La responsabilidad de un arquitecto de software implica las actividades del ciclo de desarrollo, la puede cumplir uno o más individuos (Equipo de arquitectura). El rol que debe desempeñar es el de un líder técnico, que toma decisiones de diseño pertinentes, satisface los drivers arquitectónicos (requerimientos que guían) y los requerimientos del sistema.

Los drivers de la arquitectura incluyen principalmente atributos de calidad. Además de esto, incluyen un subconjunto de los casos de uso que se consideran como primarios. Los casos de uso primarios son aquellos de mayor importancia o de mayor complejidad para el negocio. Por último, las restricciones también son consideradas como drivers arquitecturales.

El hecho de que los drivers sean un subconjunto de todos los requerimientos del sistema puede verse como una ventaja pues es posible comenzar a realizar el diseño de la arquitectura antes de haber terminado de documentar todos los requerimientos. Ciertas metodologías de desarrollo como por ejemplo el Rational Unified Process recomiendan, de hecho, que se siga este enfoque.

Por último, en la actualidad este rol se encuentra presente en diversas compañías de desarrollo de software, cabe recalcar que en el año 2015, el sitio de CNN Money situaba al arquitecto de software como el primero de los cien mejores trabajos en Estados Unidos.

Tipos de requerimientos

La forma en la que se estructura un sistema marca la capacidad para alcanzar los objetivos del negocio, los cuales corresponden a metas que busca alcanzar una organización y que motivan principalmente a la creación de un sistema. Asimismo, para llegar a dichas metas se deben satisfacer las necesidades mediante comportamientos o características propias del negocio.

La definición técnica del requerimiento corresponde a una especificación que describe alguna funcionalidad, atributo o factor de calidad de un sistema de software. Puede describir también algún aspecto que restringe la forma en que se construye ese sistema.

Requerimientos del negocio

Los requerimientos de negocio deben identificarse al inicio del desarrollo de un sistema, porque de estos nace la motivación por la cual se planea la creación de un software. De igual forma, permite plasmar la visión y alcance de la organización para llevar a cabo el proyecto.

Requerimientos de usuario y requerimientos funcionales

Los requerimientos de usuario y los requerimientos funcionales especifican aspectos de carácter funcional sobre los servicios que pueden realizar los usuarios a través del sistema. Los requerimientos de usuario especifican servicios que por lo habitual dan soporte a procesos de negocio que los usuarios podrán llevar a cabo mediante el sistema, por ejemplo: “Comprar boleto (de autobús) en línea”. Los funcionales describen detalles finos de diseño y/o implementación relacionados a los requerimientos del usuario, por ejemplo: “El origen y destino del viaje deben ser elegidos a partir de un combobox”.

Atributos de calidad

Los atributos de calidad especifican características útiles para establecer criterios sobre la calidad del sistema. Actualmente no existe una lista única de atributos de calidad relevantes para diseñar la arquitectura de software. Sin embargo, existen estándares para la evaluación de la calidad, como el ISO/IEC 9126 (International Standard Organization, 2001) pueden ser útiles para el arquitecto durante la identificación y la especificación de este tipo de drivers arquitectónicos.

›**Figura 2-4**. Atributos de calidad considerados por el estándar ISO/IEC. 9126.

Restricciones

Las restricciones describen aspectos que limitan el proceso de desarrollo del sistema. Para facilitar su manejo se distinguen dos subclases:

- Restricciones técnicas: Se refiere a solicitudes expresas sobre temas como métodos de diseño, implementación, productos de hardware o lenguajes de programación, etc. Siendo productos de software provistos por terceros.

- Restricciones administrativas: Se refiere a aspectos relacionados con el costo y tiempo de desarrollo, así como con el equipo de desarrollo.

Reglas de negocio e interfaces externas

Las reglas de negocio especifican políticas, estándares, prácticas o procedimientos organizacionales y/o gubernamentales que rigen o restringen la forma en que se realizan las actividades o procesos de una organización. No en todos los casos, influyen las reglas de negocio en la creación de un sistema.

Un ejemplo sencillo es que durante el desarrollo de un sistema, exista una regla en donde se especifica el procedimiento que usa la compañía para calcular un descuento sobre el valor de un boleto de autobús, por lo tanto, esta tiene influencia en el establecimiento de los requerimientos funcionales relacionados con la compra de boletos en línea.

Las interfaces externas especifican los detalles sobre las interfaces necesarias para que el sistema pueda comunicarse con componentes externos de software o hardware. Un ejemplo es el pago electrónico en un sitio web para realizar una transacción bancaria.

Arquitectura versus Diseño

En el mundo de la tecnología tanto la arquitectura como el diseño de sistemas suelen estar estrechamente ligados, a tal punto en que en el mundo académico la comunidad difiere significativamente al confundir estos dos términos; no obstante, durante la mención del tema por parte de algunos profesionales en el área se suele interpretar una relación ambigua entre ambos campos.

Según (Blanco, s. f.) la principal diferencia entre ambas áreas se encuentra en el camino al que nos enfrentamos. La arquitectura se trata de estrategia, estructura y propósito, con vista hacia lo abstracto. Por el contrario, el diseño tiene la mirada puesta en la implementación y la práctica y hacia lo concreto. Dicho esto, tanto la arquitectura como el diseño son fundamentales a la hora de construir un software de calidad.

Hasta cierto punto tanto la arquitectura como el diseño tienen el mismo propósito, sin embargo, (Carrascoso Puebla & Chaviano Gómez, 2008) mencionan que la arquitectura de software se enfoca en la estructura del sistema y las conexiones de los componentes, distinguiéndose del diseño de software tradicional como el orientado a objetos, el cual se enfoca más en el modelado de las abstracciones del más bajo nivel, como los algoritmos y los tipos de datos. A medida que se refina la arquitectura de alto nivel sus conectores pueden ir perdiendo protagonismo, extendiéndose a elementos arquitectónicos de nivel inferior, lo cual hace que la arquitectura se transforme en diseño.

Cuando hablamos de arquitectura de software nos referimos a un mayor nivel de abstracción; trata con componente y no con procedimientos; las interacciones entre dichos componentes y no entre las interfaces; las restricciones se hacen sobre los componentes y sus interacciones, no sobre los algoritmos, procedimientos y tipos. En términos de la composición la arquitectura es cruda, el diseño es más bien una buena composición procedimental. En la arquitectura las interacciones entre los componentes se relacionan con un protocolo de alto nivel, mientras que en el diseño implican interacciones de tipo procedimental, por ejemplo a través de mensajes o llamadas a rutinas (Carrascoso Puebla & Chaviano Gómez, 2008).

Según menciona (Rinaldi, s. f.) en su artículo la arquitectura de software es una metáfora relativamente nueva en materia de diseño de software y abarca tanto las metodologías de diseño como las metodologías de análisis. Los arquitectos de software modernos cumplen una combinación de roles, tales como el de analista de sistemas, diseñador de sistemas e ingeniero de software, sin embargo, la arquitectura se trata de algo más que la simple reubicación de funciones, ya que dicho concepto intenta llevar las actividades de análisis y diseño a un marco de diseño más grande y cohesivo. Lo que distingue a la arquitectura de software de la simple combinación de técnicas de análisis y diseño es la integración de diferentes metodologías y modelos.

En términos simples se puede llegar a la conclusión de que la arquitectura de software no es lo mismo que diseño de software, ya que esta se lleva a cabo en una etapa muy temprana del desarrollo de un proyecto con un alto nivel de abstracción, tratando con los componentes del proyecto, las secciones, los conectores, la configuración y las restricciones, lo cuál deja al diseño de software la responsabilidad de detallar elementos arquitectónicos de más bajo nivel pero de una manera más refinados como lo son los procedimientos, los algoritmos y los tipos de datos.

Estilos Arquitectónicos

Los estilos arquitectónicos son un conjunto de decisiones de diseño de arquitectura que se aplican al desarrollo específico y restringen las decisiones de diseño de un sistema. Los diferentes estilos expresan un esquema de organización estructural para sistemas de software; proveen un conjunto de tipos de elementos predefinidos, especifica sus responsabilidades e incluye reglas y guías para organizar las relaciones entre ellos.

Beneficios del uso de estilos arquitectónicos:

- Permite seleccionar una solución entendible y probada a ciertos problemas, definiendo los principios organizativos del sistema.
- Al basar la arquitectura en estilos que son conocidos las personas entienden más fácilmente las características importantes de la misma

Algunos estilos de arquitectura son los siguientes:

1. **Estilo módulo**

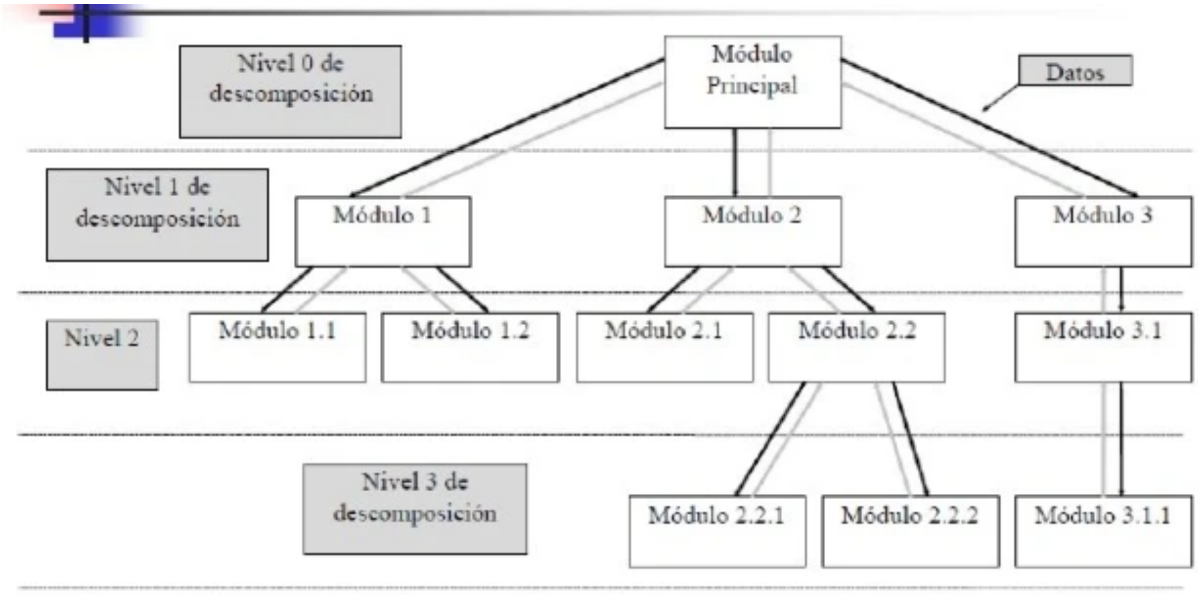
Un estilo de módulo es un tipo de estilo que introduce un conjunto específico de módulos y especifica reglas sobre cómo los elementos de esos tipos se pueden combinar.

Los módulos son los elementos principales de los estilos de módulo, dicho esto es importante saber que un módulo es una unidad de implementación que proporciona un conjunto coherente de responsabilidades. Un módulo puede tomar la forma de una clase, una colección de clases, una capa, un aspecto o cualquier descomposición de la unidad de ejecución. Cada módulo tiene una colección de propiedades que se le asignan. Estas propiedades están destinadas a expresar la información importante asociada con el módulo, así como las restricciones del módulo.

Algunos estilos de módulo según Clements et (2011) son los siguientes:

- **El estilo de descomposición:** Es utilizado para mostrar la estructura de módulos y submódulos (es decir, relaciones de contención entre módulos).
- **El estilo usos:** Se utiliza para indicar relaciones de dependencia funcional entre módulos
- **El estilo de generalización:** con este estilo se indican relaciones de especialización entre módulos.
- **El estilo en capas:** utilizado para describir la relación de uso permitido en una forma restringida entre grupos de módulos llamados capas

- **El estilo del modelo de datos:** utilizado para mostrar las relaciones entre los datos entidades



Representación de una descomposición modular. Tomado de ittgweb (2016)

2. **Estilo componente y conector (C&C)**

Un estilo componente y conector es un tipo de estilo que introduce un conjunto específico de componente y con tipos de conectores y especificaciones cumple las reglas sobre cómo elementos de esos tipos se pueden combinar. Este estilo típicamente también está asociado con un modelo computacional que prescribe cómo flujo de datos y control a través de sistemas diseñados en ese estilo.

Según Clements, las vistas de componentes y conectores se usan comúnmente para:

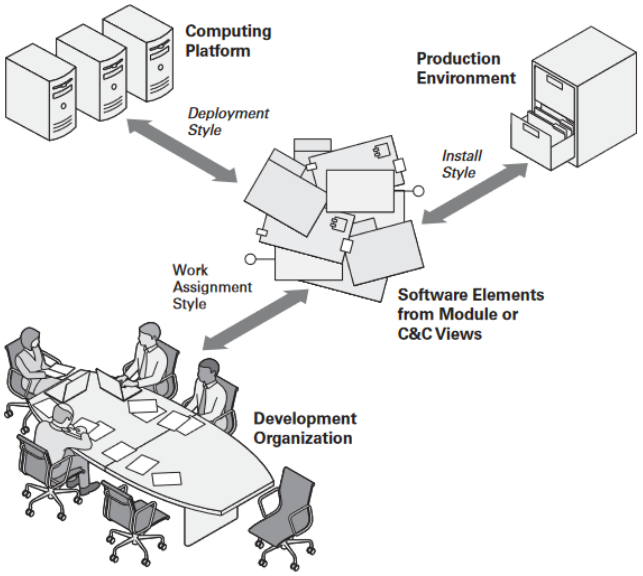
- ➢ Mostrar a los desarrolladores y otras partes interesadas cómo funciona el sistema.
- ➢ Guiar el desarrollo especificando la estructura y el comportamiento del tiempo de ejecución elementos
- ➢ Ayudar a arquitectos y otros a razonar sobre la calidad del sistema de tiempo de ejecución atributos, como el rendimiento, la fiabilidad y la disponibilidad

3. **Estilo de asignación**

Un estilo de asignación es un tipo de estilo que describe el mapeo de unidades de software a elementos de un entorno en que se encuentra el software desarrollado o ejecutado.

Por otra parte Less, Clements y Kazman (2012) mencionan que los estilos de asignación lo que hacen es definir cómo los elementos de C&C o estructuras de módulos se asignan a cosas que no son software como por ejemplo: hardware, equipos y sistemas de archivos. De la misma manera mencionan que el estilo de asignación incluye las siguientes estructuras o estilos :

- **Estructura de despliegue:** describe el mapeo entre los componentes y conectores del software y el hardware de la plataforma informática en la que se ejecuta el software. En otras palabras muestra cómo se asigna el software a los elementos de procesamiento y comunicación del hardware. La estructura de despliegue se puede utilizar para razonar sobre rendimiento, integridad de datos, seguridad y disponibilidad.
- **Estructura de instalación:** muestra cómo los elementos de software (generalmente módulos) son mapeados a las estructuras de archivos en el desarrollo, integración o configuración del sistema de ambientes de control.
- **Estructura de asignación de trabajo:** describe la asignación entre los módulos del software y las personas, equipos o unidades de trabajo encargadas del desarrollo de esos módulos.



Representación de los estilos de asignación. Tomado de Clements et al (2011)

Diagrama de contexto

Un diagrama de contexto de sistema (DCS) es un diagrama de flujo de datos de nivel superior donde se modela el ambiente en el cual el sistema se desenvuelve, en este diagrama se indican elementos externos con los cuales el sistema interactúa. Dichos elementos externos con los cuales el sistema interactúa pueden ser humanos, otros sistemas informáticos u objetos físicos. La interacción del sistema con dichas entidades es el intercambio de información consumida o producida, no es el flujo de materiales de orden físico.

De acuerdo con lo que dice Clements et al (2011) El diagrama de contexto define el límite entre un sistema (o parte de un sistema bajo consideración) y su ambiente, al mostrar las entidades con las cuales interactúa y los flujos principales de información entre el sistema y las entidades. Es importante tomar en cuenta que aquí "contexto" significa un entorno en el que el parte del sistema interactúa.

Además Clements menciona que un diagrama de contexto de nivel superior es un diagrama de contexto en el que el alcance es todo el sistema y comenta que los diagramas de contexto muestran lo siguiente:

- Una representación del sistema o parte del sistema cuya arquitectura está siendo documentada.
- Entidades externas.
- Relaciones con las entidades externas que tiene el sistema.

Hay que tomar en cuenta que las entidades externas se muestran fuera del símbolo distinguido del sistema que se describe; las relaciones son expresadas en el vocabulario de la categoría de la vista contenedora.

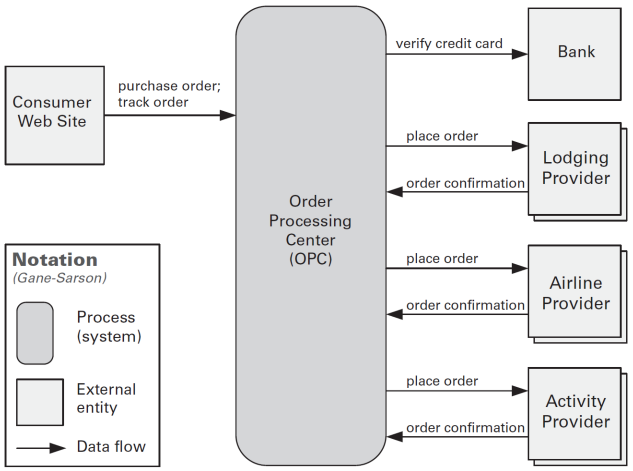


Diagrama de Contexto de C&C. Tomado de Clements et al (2011)

Vistas arquitectonicas 4+1

El modelo 4+1 describe la arquitectura del software usando cinco vistas concurrentes.

- ***Vista lógica:*** Describe el modelo de objetos del diseño cuando se usa un método de diseño orientado a objetos. Para diseñar una aplicación muy orientada a los datos, se puede usar un enfoque alternativo para desarrollar algún otro tipo de vista lógica, tal como diagramas de entidad-relación.

  Esta vista da soporte al análisis y especificación de los requerimientos funcionales, es decir, lo que el sistema necesita proporcionar al usuario desde una perspectiva de servicios. Según menciona (Anónimo, 2016) el sistema se descompone en un conjunto de abstracciones principales que se derivan del dominio del problema a modo de objetos o clases. Los diagramas de clases, de comunicación y de secuencia son regularmente utilizados para representar esta vista en diagramas UML.

- ***Vista de procesos:*** Según el artículo («3. Modelo de Vistas de Arquitectura 4 1 (Vista de Proceso) Wiki», 2020) la vista de procesos se relaciona con algunos requisitos no funcionales que son importantes como características que debe poseer un sistema o aplicación de software. Entre algunos de los requisitos no funcionales que se consideran son el rendimiento, disponibilidad, integridad, concurrencia y la tolerancia a fallos. El objetivo principal de esta vista es mostrar los procesos del sistema y cómo se comunican. También muestra los hilos de control que están realizando las operaciones definidas en la vista anterior. Para la representación de la vista de procesos en UML se utiliza un diagrama de actividad.

- **Vista física:** También conocido como el punto de vista de implementación, describe el mapeo del software en el hardware y refleja los aspectos de distribución. En otras palabras esta vista representa una visión del sistema desde una perspectiva de un ingeniero de sistemas.

Según menciona (Anónimo, 2016) la vista física se centra en los requisitos no funcionales como lo son la disponibilidad del sistema, la tolerancia a fallos, la ejecución y escalabilidad. Además presenta como los procesos, objetos, entre otros, corresponden a los

nodos de proceso: componentes (nodos de proceso), conectores (LAN, WAN, bus) y contenedores (subsistemas físicos).

La manera de representar esta vista en forma de diagramas UML es a través de diagramas de implementación.

- ***Vista de desarrollo:*** describe la organización estática del software en su ambiente de desarrollo. La vista de desarrollo muestra el sistema desde el punto de vista del programador y se encarga de la gestión del software.

De acuerdo con (Anónimo, 2016) la vista de desarrollo se centra en la organización de módulos de software en su entorno de desarrollo. El software está empaquetado en piezas pequeñas, los subsistemas están organizados jerárquicamente en capas y cada una de ellas proporciona una interfaz clara a las capas superiores.

Según menciona (Anónimo, 2016) la vista de desarrollo contiene requisitos internos que se relacionan con la facilidad de desarrollo, gestión de software, evaluación de costos, planificación, seguimiento del progreso del proyecto, portabilidad, seguridad y restricciones que son impuestas por las herramientas o el lenguaje de programación. Dicha organización del software se apoya en lo que son los diagramas de módulos o de subsistemas que muestran tanto las relaciones de exportación como las de importación.

La notación más utilizada de esta vista es a través de UML mediante los diagramas de componentes y de paquetes.

Arquitectura Física

Cuando hablamos de arquitectura física nos referimos también a la arquitectura de hardware, lo cual abarca principalmente a servidores, routers, entre otros, así como también al medio de comunicación entre estos. Según (Proyecto Especial, s. f.) la arquitectura física es una disposición de elementos físicos que proporciona la solución de diseño para un producto, servicio o empresa, y está destinado a satisfacer los elementos de la arquitectura lógica y los requisitos del sistema. En otras palabras, es la manera de especificar los dispositivos en los que se van a ejecutar los elementos funcionales definidos en la arquitectura lógica.

De acuerdo con (RENIEC, 2015) las capacidades de cada equipo, el número de equipos, las configuraciones y la distribución de los mismos influye directamente en el

rendimiento y la disponibilidad del servicio. Sin embargo, estos aspectos no son necesariamente mejorados con un equipo moderno de última generación.

La arquitectura física se divide en dos tipos:

- **Cliente-Servidor:** Es el más conocido y usualmente se compone de dos actores: el proveedor y el consumidor. En esta arquitectura existe un servidor principal al cual se conectan múltiples clientes con el fin de hacer uso de los recursos necesarios para el funcionamiento del programa, en relación a esto, (Blancarte, s. f.) menciona que el cliente solo es una capa para representar los datos y se detonan acciones para modificar el estado del servidor, mientras que el servidor es el que hace todo el trabajo pesado.

  En esta arquitectura tanto el cliente como el servidor se desarrollan como dos aplicaciones distintas, por lo que cada una se puede desarrollar de manera independiente, lo que da como resultado dos aplicaciones separadas que se pueden construir en diferentes tecnologías pero siempre respetando el mismo protocolo definido para establecer la comunicación.

  La razón principal por la que se separan ambos actores es debido a la necesidad de centralizar la información y separar responsabilidades, ya que según (Blancarte, s. f.) el servidor será la única entidad con acceso a los datos y solamente atenderá a aquellos clientes en los cuales confía, protegiendo de este modo la información y la lógica detrás del procesamiento de los datos, además, el servidor tiene la capacidad de atender a múltiples clientes simultáneamente por lo que suele instalarse en un equipo con muchos recursos, contrario al cliente el cual se instala en equipos con bajos recursos pues este no tendrá que procesar nada y actúa solamente como un observador de los datos y le asigna las tareas pesadas al servidor
- **3-Tier o arquitectura de tres niveles:** De acuerdo con (IBM, 2020) es una arquitectura de software de aplicación que separa las aplicaciones en tres niveles de informática lógica y física: el nivel de presentación o la interfaz de usuario, el nivel de aplicación o donde se procesan los datos, y el nivel de datos donde se almacenan y gestionan los datos asociados con la aplicación.

  La ventaja principal de esta arquitectura es que cada nivel es ejecutado en su propia infraestructura, por lo que cada nivel puede ser desarrollado de manera simultánea por un conjunto diferente de desarrolladores y puede

actualizarse o expandirse según sea necesario sin tener que afectar a los otros niveles.

A diferencia de la arquitectura 3-Layer de la arquitectura lógica, el estilo de implementación de la arquitectura 3-Tier describe la separación de la funcionalidad en distintos segmentos, similar a una arquitectura en capas, sin embargo, cada segmento es un nivel físico en una computadora separada. Según (Aarroyo, 2009) esta arquitectura posee al menos 3 capas lógicas separadas, cada una con una funcionalidad específica; cada capa es independiente de la otra con excepción de aquellas que se encuentren por debajo de ella.

Los tres niveles que interactúan en este modelo son los siguientes:

- **Nivel de presentación:** Se refiere a la interfaz de usuario y de comunicación de la aplicación, es donde el usuario final realiza sus interacciones con el sistema. La finalidad principal de este nivel es mostrar información al usuario y recopilar datos de este.
- **Nivel de aplicación:** Es el corazón de la aplicación. En este nivel se lleva a cabo el procesamiento de la información recopilada en el nivel de presentación, en ocasiones junto con otra información a nivel de datos utilizando la lógica empresarial. Este nivel también puede agregar, eliminar o modificar información en el nivel de datos.
- **Nivel de datos:** Es donde la aplicación almacena y maneja la información procesada, este puede ser un sistema de gestión de bases de datos relacional o bien un servidor de bases de datos. En una arquitectura de tres niveles todas las comunicaciones pasan a través del nivel de aplicación, los niveles de datos y presentación no pueden comunicarse directamente.

Arquitectura Lógica

De acuerdo con (RENIEC, 2015) la arquitectura lógica apoya principalmente los requisitos funcionales, en otras palabras, lo que el sistema debe brindar a los usuarios en cuanto a los servicios. El sistema se divide en una serie de abstracciones principales, las cuales en su mayoría se extraen del dominio del problema a manera de objetos o clases de objetos; es aquí donde se aplican los conceptos de abstracción, encapsulamiento y herencia. Dicha división no solamente se hace para mejorar el análisis funcional, sino también para identificar mecanismos y elementos de diseño comunes a varias partes del sistema.

La arquitectura lógica va muy de la mano con la arquitectura física, al combinar las dos y no omitir ningún detalle de una se pueden obtener aplicaciones con un muy alto rendimiento. Un mal diseño en la arquitectura lógica puede poner en peligro todo el esquema físico sin importar qué tan efectivo sea este último.

En este tipo de arquitectura existen dos tipos:

- **3-Layers:** De acuerdo con (Aarroyo, 2009) esta arquitectura se centra en una distribución jerárquica de los roles y responsabilidades proporcionando una separación efectiva de las preocupaciones, donde cada rol indica la manera y el tipo de interacción con las otras capas. Este tipo de arquitectura presenta las mismas capas anteriormente mencionadas en la arquitectura 3-Tier, sin embargo, según (Aarroyo, 2009) presenta algunas características que le diferencian:
    - ➢ Describe una separación de los servicios de tal modo que las interacciones se llevan a cabo habitualmente entre las capas vecinas.
    - ➢ Las capas de la aplicación pueden estar en el mismo ordenador físico o en equipos separados.
    - ➢ Los componentes de cada capa se comunican con los componentes de otras capas por medio de un conjunto bien definido de interfaces
    - ➢ Cada una de las capas asigna sus propias responsabilidades a cada una de las capas debajo de ella.

- **N-Layers:** Esta arquitectura hace referencia a la distribución lógica de cada una de las capas, en otras palabras, indica de qué manera está estructurado el código. Incluye una capa de presentación, una capa de negocios, una capa de acceso a datos, una capa de entidades de negocio y una capa de datos. En esta arquitectura las capas interactúan solamente con cada una de sus capas adyacentes, lo cual permite abstraer las funcionalidades de las capas superiores e inferiores.

Capas lógicas de implementación

En la arquitectura de software el patrón arquitectónico por capas es el patrón de software más común. Según menciona (Novoseltseva, 2021) los patrones de arquitectura en capas son patrones de n niveles donde los componentes están organizados en capas horizontales, esta es la forma tradicional de diseñar la mayoría de los programas de computadora y cuya finalidad es ser independiente, lo cual quiere decir que cada uno de los componentes se encuentran conectados, sin embargo, no son interdependientes.

Una de las características más importantes de este patrón arquitectónico es la separación de responsabilidades entre los componentes, ya que cada uno de los componentes de una capa específica se encargará solamente de la lógica a la cual pertenece dicha capa.

Cada una de las capas de este patrón tiene una función y responsabilidad específica dentro de la aplicación, entre estas podemos mencionar las siguientes:

- ***Presentación:*** También considerado el primer nivel, en esta capa se manejan los eventos del cliente y se pasan los datos al mismo. Considerando que el cliente podría tratarse de una persona u otro sistema, en el caso de un humano (Bertucelli, 2019) menciona que esta capa se encargará de atender los clics u otros eventos en un HTML y de renderizar la información de manera visual. En el caso de que el cliente se trate de otro sistema esta capa se encargará de atender peticiones rest y devolver la información en un formato estructurado (json, xml, entre otros) el cuál sea más fácil de interpretar para el sistema.

  En otras palabras la capa de presentación es la responsable de manejar todas las interfaces de usuario y la lógica de comunicación del navegador, estos componentes son los que permiten la comunicación entre el cliente y la

lógica de negocios de forma segura, ya que en estos en ningún momento tendrán acceso directo a la capa de datos.

- ***Lógica de negocio:*** Segundo nivel de la arquitectura, esta capa contiene todo lo referente a las reglas que se encuentran en el negocio, es decir, todo lo relacionado con los requerimientos funcionales del sistema. Por ejemplo, si se requiere dar de alta a un usuario esta capa debe proveer un método el cuál deba contener y realizar todos los pasos para dar de alta a dicho usuario, ya sea enviar un email, validar el nombre del usuario, entre otros. Entre otras de sus funciones también se puede mencionar el procesamiento de datos, implementación de funciones de negocios, coordinación de usuarios y la gestión de recursos externos como las bases de datos o sistemas heredados.

  La mayor parte del trabajo de la aplicación se lleva a cabo en esta capa. La capa de lógica de negocios debe ser capaz de manejar sus propias transacciones debido a que múltiples componentes del cliente pueden acceder a los procesos de esta capa de manera simultánea. De acuerdo con (Oracle, s. f.) los servicios de negocio también se pueden crear como servidores independientes como lo pueden ser un servidor de mensajería o un servidor de calendario empresarial.

- ***Acceso a datos:*** Tercer nivel de la arquitectura, a través de esta capa es posible obtener o almacenar los datos que van a ser utilizados por la aplicación. De acuerdo con (IBM, 2021) los servicios en este nivel se encuentran protegidos del acceso directo por parte de los componentes del cliente los cuales están en una red segura, es por ello que la interacción se produce por medio de los procesos del segundo nivel.

  Es importante tener en cuenta que, ya sea en una base de datos o en un archivo, al referirnos a esta capa hablamos estrictamente sobre el acceso a los datos y no sobre cómo se realiza la persistencia de los mismos, por ejemplo, esta capa debería ser capaz de proporcionar una forma de guardar el usuario y otra forma de obtenerlo.

Al llevar a cabo esta separación de responsabilidades en una arquitectura por capas podemos ver que a una capa no le importa de qué manera se implementa la otra capa, por lo tanto, se puede hacer un reemplazo de las interpretaciones de las capas sin preocuparse de que esto afecte a todas las demás. Por ejemplo, se puede tener un método en la capa de acceso a datos el cual guarda la información como un archivo de texto plano, sin embargo, si en el

futuro el mismo método aún existe pero se desea cambiar esta implementación para que se almacene en una base de datos, para la capa de lógica de negocios no importa de qué manera sea almacenada la información.

Subsistemas

El logro principal de la arquitectura de software en este nivel es la identificación de todos los subsistemas principales, los cuales pueden ser grandes: bases de datos, interfaces de usuario, reglas de negocio, intérpretes de comandos, motores de informes, entre otros. La creación de subsistemas solo aplica para sistemas de gran tamaño, puesto que lo ideal es que cada subsistema realice una función específica y sea independiente de todos los demás.

Según el artículo de (Tu Propia Escuela, 2017) la actividad principal de diseño a este punto es decidir cómo dividir el programa en subsistemas principales y definir de qué manera se permite a cada subsistema utilizar a otro subsistema. La división a este nivel generalmente se requiere para proyectos de más de unas pocas semanas. Puede elegirse el mejor enfoque para cada parte de su sistema y utilizar diferentes métodos de diseño dentro de cada subsistema.

Las reglas sobre cómo se comunican los diferentes subsistemas son de especial importancia en este nivel, ya que si todos los subsistemas se pueden comunicar con los demás se pierde completamente la ventaja de separarlos en primer lugar. En el artículo de (Tu Propia Escuela, 2017) se recomienda que cada subsistema sea significativo a la hora de restringir las comunicaciones, ya que si no existen restricciones en la comunicación entre los subsistemas habrá un aumento en la entropía puesto que cada subsistema se comunicará directamente con todos los demás subsistemas, lo que plantea algunos problemas importantes.

En base a lo anterior, resulta aún más difícil si se necesita eliminar un subsistema que tenga muchas conexiones con otros subsistemas. Ante esto, (Tu Propia Escuela, 2017) menciona que es deseable diseñar el sistema de modo que se pueda eliminar el subsistema para usarse en otro lugar sin tener que volver a conectar muchas conexiones. Si se tiene alguna duda con respecto a la comunicación entre los subsistemas es más fácil limitarlas al principio y relajar las restricciones luego, de lo que es relajar las restricciones al inicio y luego limitarlas cuando múltiples llamadas entre subsistemas ya se han codificado entre subsistemas.

En el artículo de (Tu Propia Escuela, 2017) se menciona que es mejor elegir una relación simple entre subsistemas con el fin de establecer conexiones que sean simples de

comprender y mantener. Una de las relaciones más simples, según este autor, es contar con una rutina de llamada entre un subsistema y otro, mientras que una relación más compleja sería que las clases de un subsistema sean heredadas de otro.

Algunas reglas de división de subsistemas según (Anónimo, s.f.) son las siguientes:

- Si los objetos están gráficamente separados deben separarse en distintos subsistemas
- Clientes y servidores deben estar en subsistemas distintos.
- Los objetos que interactúan con el usuario deben estar en un subsistema separado del resto de objetos
- Dos objetos asociados entre sí no tienen que estar necesariamente en el mismo subsistema, pero es aconsejable no cortar demasiadas asociaciones para reducir la dependencia

Como regla general es importante mencionar que un diagrama de nivel de sistema debe ser un gráfico no cíclico, es decir, un sistema no debe contener ninguna relación circular entre sus clases. El diseño a nivel de subsistema será lo que marque la diferencia en programas de gran tamaño.

Arquitectura de software en metodologías ágiles

La naturaleza competitiva del mercado de desarrollo de software, junto con la necesidad que tienen los clientes para reducir el tiempo de comercialización ha obligado a las empresas de desarrollo de software a ser más agresivas en cuanto a sus programas de entrega se refiere. Esto ha llevado a la aparición de metodologías ágiles como es el caso de la metodología Scrum, que de acuerdo con (Mago Vásquez & Alférez Salinas, s. f.) se trata de una metodología para la gestión y desarrollo de software basada en un proceso iterativo e incremental, la cual está basada en sprints iterativos que van de 1 a 4 semanas y se utiliza principalmente para obtener resultados rápidos y la mayor productividad posible.

Dentro de esta metodología se encuentran tres roles fundamentales: Scrum Master, Product Owner y Scrum Team, lo cual evidentemente deja por fuera el rol de un arquitecto de software. Sin embargo, en Scrum es importante definir y asignar responsabilidades, por lo que cada una de las fases son realizadas por equipos y personas competentes y expertos, razón por la cual en sistemas de desarrollo complejos es difícil que una sola persona abarque con la suficiente amplitud técnica las decisiones arquitectónicas del sistema y pueda dar credibilidad a la arquitectura de software.

Por lo tanto, la participación del equipo es altamente importante para la creación y mantenimiento de la arquitectura del proyecto; un equipo de arquitectos de software aislado del Scrum Team puede crear un sistema estructural con un alto riesgo de rechazo. De acuerdo con (Melo, 2018) las responsabilidades del arquitecto de software en un equipo SCRUM están diferenciadas por la fase en la que se encuentren, siendo mayor la necesidad de éste en una fase de preparación inicial del proyecto, antes de iniciar con los sprints.

Conclusiones

En el presente documento se llevó a cabo la explicación a detalle sobre la definición de arquitectura de software junto con los procesos que involucran el desarrollo del mismo, así como cada una de sus fases. De igual forma, se mencionaron los beneficios de utilizar este tipo de tecnología para que la persona que ejerza el rol de arquitecto(a) dentro de un proyecto pueda decidir si es apto o no, según las necesidades de la organización.

Por otro lado, se puso en evidencia las diferencias que existen entre arquitectura de software versus diseño de software, lo cual ha generado bastante confusión entre la comunidad de desarrollo, ya que ambos conceptos son importantes para llevar a cabo un proyecto de Tecnologías de la Información (TI). Igualmente, se destacaron los distintos estilos arquitectónicos con sus respectivas guías y principales ventajas de implementación.

Además, en la arquitectura de software se utiliza el patrón arquitectónico por capas. Y una de las ventajas que tiene este patrón es que existe la separación de responsabilidades entre los componentes, ya que cada uno de los componentes de una capa específica se encarga de resolver la necesidad correspondiente de la capa.

Para finalizar, se manifiesta por medio del escrito qué es la arquitectura física y la solución que proporciona para el diseño de un producto o servicio, y a su vez como la arquitectura lógica mejora el análisis funcional, concluyendo con el entendimiento de que ambos tipos de arquitectura tienen un objetivo en común el cual es obtener un resultado exitoso, acompañado de una documentación detallada de cada una de las etapas que implica el desarrollo de un proyecto con el fin de brindar las soluciones necesarias ante los diferentes problemas de una entidad.

Bibliografía

Codmind. (2021). *Que es la arquitectura de software.* [Archivo de vídeo]. Youtube. Que es la arquitectura de software - YouTube

Clements, P. et al (2011). *Documenting Software Architectures : views and beyond.* USA:Addison Wesley. https://bunker4.zlibcdn.com/dtoken/adbfcabd8b5ce457505133ea11164d16

L. Bass, P. Clement, R. Kazman (2012). *Software Architecture in Practice, 3rd ed,* Addison Wesley,https://bunker2.zlibcdn.com/dtoken/c155cadfe3cb703032d25999d1f3219d

Pressman, R. (2010). *Ingeniería del software Un enfoque práctico.* http://cotana.informatica.edu.bo/downloads/ld-Ingenieria.de.software.enfoque.practico.7ed.Pressman.PDF

L. Bass, P. Clement, R. Kazman, *Software Architecture in Practice, 2nd Edition,* Addison Wesley, 2003. http://elibrary.lt/resursai/Leidiniai/Litfund/Lithfund_leidiniai/IT/Addison%20Wesley%20-%20Software%20Architecture%20In%20Practice%202Nd%20Edition%20Ebook-Lib.pdf

Cervantes Maceda, Humberto, Perla Velasco-Elizondo y Luis Carlos Careaga. *Arquitectura de software. Conceptos y ciclo de desarrollo.* ISBN 978-607-522-456-5. Arquitectura-de-Software-Conceptos-y-Ciclo-de-Desarrollo.pdf (researchgate.net)

Blanco, C. (s. f.). *¿Cuál es la diferencia entre diseño de software y arquitectura de software?* DIFFERBETWEEN. Recuperado 7 de mayo de 2022, de https://es.differbetween.com/article/what_is_the_difference_between_software_design_and_software_architecture#whats_the_difference_between_architecture_and_design

Carrascoso Puebla, Y. A., & Chaviano Gómez, E. (2008, 19 junio). *1.2.3. Diferencia entre Arquitectura de Software y Diseño - ort*. Studylib. Recuperado 7 de mayo de 2022, de https://studylib.es/doc/224441/1.2.3.-diferencia-entre-arquitectura-de-software-y-diseño.%20.%20.

Mago Vásquez, E. R., & Alférez Salinas, G. H. (s. f.). *El Papel de la Arquitectura de Software en Scrum*. SG Buzz. Recuperado 8 de mayo de 2022, de https://sg.com.mx/revista/30/el-papel-la-arquitectura-software-scrum

Melo, F. M. [Fabián Mauricio Melo]. (2018, 1 octubre). *Metodología SCRUM y el rol del arquitecto de software* [Blog]. Asesoftware. https://asesoftware.com/metodologia-scrum-y-el-rol-del-arquitecto-de-software/

Rinaldi, R. (s. f.). *3. Diferencias entre Arquitectura y DiseÃ±o - Warning: TT: undefined function: 32 Diferencias entre*. StuDocu. Recuperado 7 de mayo de 2022, de https://www.studocu.com/co/document/universidad-de-cartagena/ingenieria-de-sistemas/3-diferencias-entre-arquitectura-y-diseno/9779130

Aarroyo, A. (2009, 18 agosto). *¿Es lo mismo arquitectura por capas y N-Tier? – Andoni Arroyo*. Geeks.ms. Recuperado 8 de mayo de 2022, de https://geeks.ms/aarroyo/2009/08/18/es-lo-mismo-arquitectura-por-capas-y-n-tier/

Blancarte, O. (s. f.). *Arquitectura Cliente-Servidor*. Reactive Programming. Recuperado 8 de mayo de 2022, de https://reactiveprogramming.io/blog/es/estilos-arquitectonicos/cliente-servidor

IBM. (2020, 28 octubre). *Arquitectura de tres niveles*. Recuperado 8 de mayo de 2022, de https://www.ibm.com/mx-es/cloud/learn/three-tier-architecture

Proyecto Especial. (s. f.). *¿Cuál es la diferencia entre arquitectura lógica y física?* Recuperado 8 de mayo de 2022, de

https://mgtuts.com/es/other/what-is-the-difference-between-logical-and-physical-architecture.html

Anónimo. (s.f.). [Archivo PDF]. Recuperado de https://b0ve.com/ARCHIVOS/UHU/Scans/ASDM/ASDM%20Resumen%20Digital.pdf

Anónimo. (2016, 13 mayo). *Arquitectura 4-1*. Recuperado de https://datenpdf.com/download/arquitectura-4-1_pdf

Modelo de vistas de arquitectura 4 1 (Vista de Proceso) Wiki. (2020, 30 enero). En *GitHub*. https://github.com/Kevinq181/Proyecto_Arquitectura_Aplicaciones/wiki/3.-Modelo-de-vistas-de-arquitectura-4-1----%28Vista-de-Proceso%29

Bertucelli, M. (2019, 25 noviembre). *Arquitectura de capas*. Somos PNT. Recuperado 12 de mayo de 2022, de https://somospnt.com/blog/118-arquitectura-de-capas

IBM. (2021, 19 junio). *Arquitecturas de tres niveles*. Recuperado 12 de mayo de 2022, de https://www.ibm.com/docs/es/was/9.0.5?topic=overview-three-tier-architectures

Novoseltseva, E. (2021, 9 junio). *5 principales patrones de Arquitectura de Software*. Apiumhub. Recuperado 12 de mayo de 2022, de https://apiumhub.com/es/tech-blog-barcelona/principales-patrones-arquitectura-software/

Oracle. (s. f.). *Dimensión 2: capas lógicas (Visión general técnica de Sun Java Enterprise System 2005Q4)*. Recuperado 12 de mayo de 2022, de https://docs.oracle.com/cd/E19636-01/819-3589/auto24/index.html

Tu Propia Escuela. (2017, 18 septiembre). *Los 5 niveles del Diseño de Software*. Recuperado 12 de mayo de 2022, de https://tupropiaescuela.wordpress.com/2017/09/18/los-5-niveles-del-diseno-de-software/