Arquitectura de software

Moisés Anchundia, Josselyn Macias, Sergio Vélez, Jefferson Vera, Dayana Villamar

Facultad de Ciencias Informáticas, Carrera de Tecnología de la información
Manta, Ecuador

Moises. e1313196311@live.uleam.edu.ec
Josselyn. e1350696397@live.uleam.edu.ec
Sergio. e1310767015@live.uleam.edu.ec
Jefferson. e1351127624@live.uleam.edu.ec
Dayana.e1314844406@live.uleam.edu.ec

Resumen: La arquitectura de software es un concepto fácil de entender, es un aspecto de diseño que se concentra en algunas características específicas. Antes de elaborar sobre el tema, es conveniente definir el concepto ya que hoy en día el término de arquitectura se usa para referirse a varios aspectos relacionados con las TI. De acuerdo con el Software Engineering Institute (SEI), la Arquitectura de Software se refiere a las estructuras de un sistema, compuestas de elementos con propiedades visibles de forma externa y las relaciones que existen entre ellos. El término "elementos" dentro de la definición del SEI es vago a propósito, pues puede referirse a distintas entidades relacionadas con el sistema. Los elementos pueden ser entidades que existen en tiempo de ejecución (objetos, hilos), entidades lógicas que existen en tiempo de desarrollo (clases, componentes) y entidades físicas (nodos, directorios). La arquitectura de software es de especial importancia ya que la manera en que se estructura un sistema tiene un impacto directo sobre la capacidad de este para satisfacer lo que se conoce como los atributos de calidad del sistema.

Palabras Claves: arquitectura de software, requerimiento, arquitectura, implementación, estructura, componentes.

Abstract: Software architecture is an easy-to-understand concept, it is a design aspect that concentrates on a few specific features. Before elaborating on the subject, it is convenient to define the concept since today the term architecture is used to refer to various aspects related to IT. According to the Software Engineering Institute (SEI), Software Architecture refers to "the structures of a system, composed of elements with externally visible properties and the relationships that exist between them." The term "elements" within the definition of the SEI is vague on purpose, as it can refer to different entities related to the system. Elements can be entities that exist at run time (objects, threads), logical entities that exist at development time (classes, components), and physical entities (nodes, directories). Software architecture is of particular importance as the way a system is structured has a direct impact on the ability of the system to satisfy what are known as the quality attributes of the system.

Keywords: software architecture, requirement, architecture, implementation, structure, components.

I. INTRODUCCION

En los inicios de la ingeniería de software, muchos científicos lucharon por tener formas más simples de realizar su trabajo, ya que las cuestiones más simples, como imprimir un documento, guardar un archivo o compilar el código.

Hoy en día, contamos con herramientas que van indicando en tiempo real si tenemos un error de sintaxis en nuestro código, pero no solo eso, además, son los suficientemente inteligentes para a completar lo que vamos escribiendo, incluso, nos ayudan a detectar posibles errores en tiempo de ejecución.

A medida que las tecnologías son más simples y accesibles para todas las personas del mundo, las aplicaciones se enfrentan a retos que antes no existían, como la concurrencia, la seguridad, la alta disponibilidad, el performance, la usabilidad, la reusabilidad, estabilidad, funcionalidad, modificabilidad, portabilidad, integridad, escalabilidad, etc., etc. Todos estos son conceptos que prácticamente no existían en el pasado, porque las aplicaciones se desarrollaban para una audiencia muy reducida y con altos conocimientos técnicos.

II. DESARROLLO DE CONTENIDOS

A. BREVE HISTORIA DE LA ARQUITECTURA DE SOFTWARE

La comparación entre el software de diseño y arquitectura (civil) primero fue dibujada a finales de 1960, [1] pero el término *arquitectura de software* se convirtió en frecuente sólo en la década de 1990. El campo de la Ciencias de la computación había encontrado problemas relacionados con la complejidad desde su formación. [2] Anteriores problemas de complejidad fueron solucionados por los desarrolladores eligiendo el derecho de estructuras de datos, desarrollo de algoritmos aplicando el concepto de separación de preocupaciones. Aunque el término "arquitectura de software" es relativamente nuevo en la industria, los principios fundamentales del campo se han aplicado esporádicamente por Ingeniería de software pioneros desde mediados de la década de 1980. Primeros intentos de captar y explicar la arquitectura software de un sistema eran impreciso y

desorganizado, a menudo caracterizado por un conjunto de caja y línea de diagramas. [3]

Arquitectura de software como concepto tiene su origen en la investigación de Edsger Dijkstra en 1968 y David Parnas en la década de 1970. Estos científicos hicieron hincapié en que la estructura de un sistema de software es importante y la estructura correcta es crítica. Durante la década de 1990 hubo un esfuerzo concertado para definir y codificar aspectos fundamentales de la disciplina, con la investigación de trabajo concentrándose en estilos arquitectónicos (patrones), lenguajes de descripción de arquitectura, y documentación de arquitectura. [4] Instituciones de investigación han desempeñado un papel destacado en la promoción de arquitectura de software como disciplina. Por ejemplo, Mary Shaw y David Garlan de Carnegie Mellon escribió un libro titulado *Arquitectura de software: Perspectivas en una disciplina emergente* en 1996, que promovió conceptos de arquitectura de software como componentes, conectores y estilos.

B. Arquitectura del software

Arquitectura de software se refiere a las estructuras fundamentales de un sistema de software, la disciplina de la creación de tales estructuras y la documentación de estas estructuras. Estas estructuras son necesarias para razonar sobre el sistema de software. Cada estructura se compone de elementos de software, las relaciones entre ellos y las propiedades de elementos y relaciones, [5] junto con razón de ser para la introducción y configuración de cada elemento. La *arquitectura* de un software de sistema es una metáfora, análoga a la arquitectura de un edificio. [6]

La arquitectura de software es sobre tomar decisiones estructurales fundamentales que son costosos de cambiar una vez implementado. Opciones de arquitectura de software, también llamados decisiones arquitectónicas, incluyen opciones estructurales específicas de posibilidades en el diseño de software. Por ejemplo, los sistemas que controlaban la lanzadera de espacio vehículo de lanzamiento tenía el requisito de ser muy rápido y confiable. Por lo tanto, una adecuada computación en tiempo real tendría que ser elegido. Además, para satisfacer la necesidad de fiabilidad la elección podría realizarse para tener múltiples redundantes e independientemente producido copias del programa y ejecutar estas copias en independiente hardware y contrastar resultados.

La documentación de arquitectura de software facilita la comunicación entre partes interesadas, captura las decisiones sobre el diseño de la arquitectura y permite la reutilización de componentes de diseño entre los proyectos. [7]

C. Objetivo y restricciones de la arquitectura

Los requerimientos y restricciones de relevancia a considera para la definición de la arquitectura son:

Todas las funciones deben estar disponibles por cualquiera de los dos navegadores de web comercialmente disponibles.

Toda la interpretación y las exigencias recopiladas, como lo estipulado en el Documento de Visión deben ser tenidas en cuenta cuando la arquitectura está siendo desarrollada.

D. OBJETIVOS:

- Cliente-servidor:

Las arquitecturas cliente-servidor son comúnmente utilizadas en sistemas distribuidos y se identifican por la separación de objetivos o funcionalidades en dos o más nodos en donde algunos nodos toman el rol de servidor y otros el rol del cliente.

- Arquitectura Basada en Componentes:

El objetivo de este patrón es descomponer la aplicación en componentes funcionalmente reutilizables que exponen una interfaz de comunicación bien definida.

- Arquitectura en Capas:

Su objetivo principal es agrupar componentes que tienen funcionalidad similar en distintas capas que son apiladas verticalmente una encima de la otra.

- Arquitectura Orientada a Objetos:

Su objetivo es interactuar y mantener su propio estado local y ofrecer operaciones sobre dicho estado

- Arquitectura Orientada a Servicios:

Su objetivo es desarrollar sistemas distribuidos en la que los componentes del sistema son servicios independientes y se ejecutan en computadoras distribuidas

- Modelo Vista Controlador:

El objetivo de esta es manejar los datos del sistema y sus operaciones, la vista define y gestiona como se presentan esos datos al usuario y el controlador.

- Fachada de Aplicación

El objetivo de esta es proporcionar una interfaz unificada para un conjunto de interfaces de un sistema, haciendo que los subsistemas sean más fáciles de utiliza. [8]

E. RESTRICCIONES

Las restricciones de la arquitectura imponen condiciones sobre la arquitectura que normalmente no son negociables, limitan el rango de alternativos de decisión del arquitecto, muchas veces hace la vida más fácil para el arquitecto en otras lo complica, se pueden clasificar según la naturaleza:

- Negocio:

La tecnología debe ejecutarse como un complemento para MS BizTalk, ya que queremos venderlo a Microsoft.

- Desarrollo:

El sistema debe estar escrito en Java para que podamos utilizar el personal de desarrollo existente.

- Tiempo:

La primera versión de este producto debe entregarse en un plazo de seis meses.

- Costo:

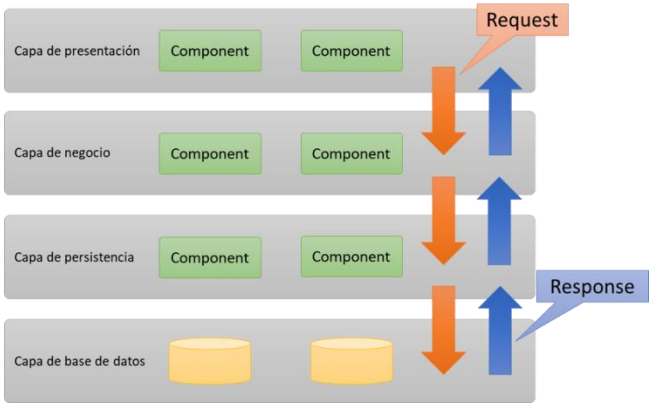
Se debe trabajar en colaboración con MegaHugeTech Corp y obtener más fondos para el desarrollo, por lo que debemos utilizar su tecnología en nuestra aplicación.

F. Arquitectura y esquemas:

Las aplicaciones contienen funcionalidades comunes, entre ellas este almacenamiento en memoria estática, comunicaciones, tratamiento de interfaz, usuarios, errores, transacciones, mantenimiento de la seguridad.

Se hace necesario tener unidades que implementan las diversas funcionalidades que son necesidades comunes a diferentes aplicaciones o a diferentes partes de una misma aplicación.

El esquema es un conjunto de clases relacionadas que brindan, de manera unificada, las funcionalidades necesarias para implementar u servicio específico.



La implementación deal de un servicio exige resolver algunos inconvenientes técnicos a su modelo, esto exactamente implica la utilización de mensajería confiable.

La respuesta del servicio es afectada directamente por aspectos externos como problemas en la red y configuración por citar algunos. Estos deben ser tenidos en cuenta en el diseño, desarrollándose los mecanismos de contingencia que eviten la parálisis de las aplicaciones y servicios que dependen de él.

En la implementación de la arquitectura de software, los esquemas tienen especial importancia cuando se trata de aumentar la reutilización y potenciar la productividad en los ciclos de desarrollo. [9]

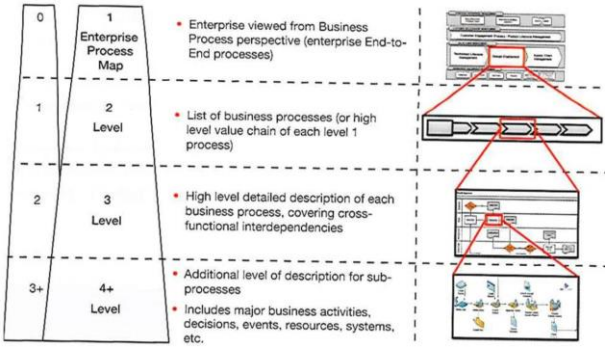
G. *Tipos de arquitectura*

Hay varios tipos arquitectura que ayudan a solucionar varias tipos e problema los cuales estos se pueden ser:

- Por N capas

La arquitectura en capas es una de las más utilizadas, no solo por su simplicidad, sino porque también es utilizada por defecto cuando no estamos seguros de que arquitectura debemos de utilizar para nuestra aplicación.

La arquitectura en capas consta en dividir la aplicación en capas, con la intención de que cada capa tenga un rol muy definido, como podría ser, una capa de presentación (UI), una capa de reglas de negocio (servicios) y una capa de acceso a datos (DAO), sin embargo, este estilo arquitectónico no define cuantas capas debe de tener la aplicación, sino más bien, se centra en la separación de la aplicación en capas (Aplica el principio



Separación de preocupaciones (SoC)) [10].

En la práctica, la mayoría de las veces este estilo arquitectónico es implementado en 4 capas, presentación, negocio, persistencia y base de datos, sin embargo, es habitual ver que la capa de negocio y persistencia se combinan en una solo capa, sobre todo cuando la lógica de persistencia está incrustada dentro de la capa de negocio [10].

En la arquitectura en capas, todas las capas se colocan de forma horizontal, de tal forma que cada capa solo puede comunicarse con la capa que está inmediatamente por debajo, por lo que, si una capa quiere comunicarse con otras que están mucho más abajo, tendrán que hacerlo mediante la capa que está inmediatamente por debajo. Por ejemplo, si la capa de presentación requiere consultar la base de datos, tendrá que solicitar la información a la capa de negocio, la cual, a su vez, la solicitará a la capa de persistencia, la que, a su vez, la consultará a la base de datos, finalmente, la respuesta retornará en el sentido inverso hasta llegar la capa de presentación.

- Arquitectura de procesos

La arquitectura de procesos tiene el objetivo de garantizar que una empresa automatice sus procesos de negocio [11]. Se puede decir que la arquitectura de procesos es la definición general de todo un sistema de procesos [12].

Al definir la arquitectura de procesos, las empresas hacen las conexiones entre el equipo y la mano de obra disponible, con los recursos y tecnologías que domina, para determinar cómo lograr sus objetivos estratégicos tan eficientemente como sea posible, creando valor en cada etapa de la cadena [13].

H. *La estructura de nivel de procesos se puede dividir en 4 niveles:*

Los requerimientos para el modelado del nivel 1 es que tienen que ser de fácil comprensión y poder interpretarse por personas sin conocimientos en BPMN [14].

El nivel 2 u operacional es la esencia del framework de BPMN ya que abarca toda la lógica del negocio en detalle incluyendo los casos de excepción, identificando las reglas de negocio y la interacción en detalle de los participantes [14].

El nivel 3 o técnico tiene por objetivo automatizar los procesos por medio de software mediante un process engine. Los modelos técnicos bien diseñados pueden ejecutarse directamente en el nivel 3 con un process engine [14].

En nivel 4 es de implementación, posterior a la especificación del nivel 3b es necesario implementar técnicamente en el proceso en una plataforma tradicional [14].

La intención que se tiene al utilizar una buena arquitectura BPM es mejorar la gestión de proceso y, por consiguiente, optimizar los resultados. Además, es necesario que la arquitectura del proceso contenida en la herramienta de BPM utilizada permita una amplia posibilidad de personalización del sistema empleado, para satisfacer las características específicas de cada organización.

I. *Desarrollo Basado en Componentes*

La reutilización de componentes de software es un proceso inspirado en la manera en que se producen y ensamblan

componentes en la ingeniería de sistemas físicos. La aplicación de este concepto al desarrollo de software no es nueva. Las librerías de subrutinas especializadas, comúnmente utilizadas desde la década de los setenta, representan uno de los primeros intentos por reutilizar software. La reutilización de software es un proceso de la Ingeniería de Software que conlleva al uso recurrente de activos de software en la especificación, análisis, diseño, implementación y pruebas de una aplicación o sistema de software [15].

Una de las características más importantes de los componentes es que son reutilizables. Para ello los componentes deben satisfacer como mínimo el siguiente conjunto de características: • identificable: un componente debe tener una identificación clara y consistente que facilite su catalogación y búsqueda en repositorios de componentes.

• accesible sólo a través de su interfaz: el componente debe exponer al público únicamente el conjunto de operaciones que lo caracteriza (interfaz) y ocultar sus detalles de implementación. Esta característica permite que un componente sea reemplazado por otro que implemente la misma interfaz.

• sus servicios son invariantes: las operaciones que ofrece un componente, a través de su interfaz, no deben variar. La implementación de estos servicios puede ser modificada, pero no deben afectar la interfaz.

• documentado: un componente debe tener una documentación adecuada que facilite su búsqueda en repositorios de componentes, evaluación, adaptación a nuevos entornos, integración con otros componentes y acceso a información de soporte[15].

J. MÉTODO WATCH

El método WATCH es un marco metodológico que describe los procesos técnicos, gerenciales y de soporte que deben emplear los equipos de trabajo que tendrán a su cargo el desarrollo de aplicaciones de software empresarial.

Un marco metodológico es un patrón que debe ser instanciado, es decir adaptado cada vez que se use. Cada equipo de trabajo deberá usar el método como un patrón o plantilla metodológica, a partir de la cual dicho equipo debe elaborar el proceso específico de desarrollo de la aplicación que se desea producir.

Se ubica dentro de los métodos disciplinados ya que se centra en los procesos, hace énfasis en los productos y la organización, involucra procesos bien definidos y documentados, requiere de alta formalidad en el proceso de desarrollo, son procesos repetibles, los resultados son predecibles.

Este método incluye, también, una descripción de los procesos de gerencia del proyecto que se aplicarán para garantizar que el proyecto se ejecute en el tiempo previsto, dentro del presupuesto acordado y según los estándares de calidad establecidos [16].

K. CICLO DE VIDA DE UN PROCESO DE DESARROLLO

Es la estructura que contiene los procesos, actividades y tareas relacionadas con el desarrollo y mantenimiento de un producto de software, abarcando la vida completa del sistema, desde la definición de los requisitos hasta la finalización de su uso. [15]

- Comunicación

Este es el momento en el que un cliente solicita un producto de software determinado. Nos contacta para plasmar sus necesidades concretas y presenta su solicitud de desarrollo de software.

- Planificación y análisis

El desarrollo de software comienza con una fase inicial de planificación incluyendo un análisis de requisitos. Nos fijamos en los requisitos que piden los clientes para estudiar cuales están poco claros, incompletos, ambiguos o contradictorios. Se indaga en profundidad y se hacen demostraciones prácticas incluyendo a los usuarios clave. Los requisitos se agrupan en requisitos del usuario, requisitos funcionales y requisitos del sistema. La recolección de todos los requisitos se lleva a cabo: estudiando el software actual

que tengan, entrevistando a usuarios y desarrolladores, consultando bases de datos o mediante cuestionarios.

- Estudio de viabilidad

Después de la recolección de requisitos, se idea un plan para procesar el software. Se analiza que parte del software cubre los requisitos de cada usuario. Se investiga la viabilidad financiera y tecnológica. Se utilizan algoritmos para saber si el proyecto de software es factible o no.

- Análisis del sistema

En este paso el equipo del proyecto asigna recursos y planifica el tiempo de duración del proyecto. Se buscan limitaciones del producto y se identifican los impactos del proyecto sobre toda la organización en su conjunto.

- Diseño

En esta fase ya se comienza a visualizar la solución con la ayuda de las anteriores fases. Se hace un diseño lógico y otro físico. Se crean metadatos, diagramas o pseudocódigos. La duración de esta fase varía de un proyecto a otro.

- Codificación

Esta fase también denominada 'fase de programación' o 'fase de desarrollo' es en la que elige el lenguaje de programación más conveniente, y se desarrollan programas ejecutables y sin errores de manera eficiente. Nuestro enfoque es construir trozos de funcionalidad. Por lo tanto, entregar unidades de funcionalidad concisa. Al final de esta fase se puede obtener un PMV (Producto mínimo viable) o el software completamente desarrollado y listo para implementarse.

- Integración

El Software puede necesitar estar integrado con bibliotecas, bases de datos o con otros programas. Esta fase del SDLC integra el software con las entidades del mundo exterior.

- Pruebas

Esta fase junto con la fase de desarrollo entra en un ciclo continuo hasta que se completan el desarrollo y las pruebas. Probamos, probamos y luego volvemos a probar tanto como sea necesario hasta que la funcionalidad sea del 100%.

Además, se hacen evaluaciones para evitar errores, incluyendo la evaluación de módulos, programas, productos, y finalmente evaluación con el cliente final. Encontrar errores y su arreglarlos a tiempo es la clave para conseguir un software confiable y eficiente.

- *Implementación*

Aquí se instala el software, se evalúa la integración, la adaptabilidad, la portabilidad y se instalan las configuraciones posteriores necesarias.

- Formación

Esta es la fase más interesante, ¡La formación! La adopción del usuario es muy importante y para ello ofrecemos capacitación inicial para cada usuario. Es importante comprobar el nivel de uso, la experiencia de usuario y resolver cualquier dificultad que pueda surgir a la hora de enfrentarse a un nuevo sistema o plataforma.

- Mantenimiento y Funcionamiento

Por último, pero no menos importante el mantenimiento es uno de los elementos clave de éxito de cualquier proyecto. En esta fase se minimizan pequeños errores, se confirma el buen funcionamiento del software, su eficiencia y estabilidad. El proyecto ya está completado y necesitamos monitorear y mantener de forma continua para garantizar que el proyecto siga ejecutándose bien.

Si es necesario se dan nuevas formaciones, o se presta documentación sobre como operar y mantener el software en perfecto estado de funcionamiento. Se adaptan entornos del usuario o tecnológicos, dando mantenimiento al software, actualizando el código y configuración.

El software es efectivo cuando se usa de forma apropiada y por eso el mantenimiento y la mejora de los productos de software es crucial para poder corregir defectos que vayan surgiendo o para poder atender a los requisitos del software.

III. Conclusión

La arquitectura de software se está desarrollando y es un guion que dicta estándares técnicos, incluye códigos, herramientas y plataformas, con el cual se forman aplicaciones exitosas. Al construir y poseer una buena arquitectura se pueden identificar riesgos para poder ubicarlos al principio de la atapada de desarrollo siendo de grandes beneficios para el proceso. Los beneficios dentro de una buena arquitectura son más rápidos, económicos, eficientes, seguros,

En conclusión, se detallaron objetivos y restricciones para una buena arquitectura, en donde se detallan cliente servidos, utilizados en sistemas distribuidos, diferentes tipos de arquitecturas, en las restricciones podemos encontrar algunas en negocio, desarrollo, tiempo, costo. Ya que se debe obtener fondos para el desarrollo.

De acuerdo con las arquitecturas y los esquemas se relacionan de manera unificada con las funcionalidades necesarias. Por último se definen un ciclo de vida del proceso de desarrollo donde se contienen actividades y tareas relacionadas.

IV. Referencias

[1] p. Naur y B. Randell, «Ingenieria de Software: Informe de una conferencia patrocinada por el Comite de ciencia de OTAN,» *Garmish*, 7-11 Octubre 1968.

[2] Universidad de Waterloo, «Una muy breve historia de la informatica,» 23 09 2006.

[3] IEEE, «Transactions on software Engineering (2006).,» *"Introduccion a la edicion especial en arquitectura de software"*, 23 09 2006.

[4] G. y. S. (1994), «Introduccion de la arquitectura de software,» 25 09 2006.

[5] P. Clements, F. Bachmann, B. d. Len, D. Garlan, J. Ivers, C. pequeña, P. Merson y R. Nord, «Arquitecturas de Software documentacion,» de *Vistas y mas alla*, Segunda ed., Boston, 2010.

[6] E. D. Perry y a. L. Lobo, «Fundamentos para el estudio de arquitectura de software,» de *Ingenieria de software de ACM SIGSOFT*, 1992, p. 17.

[7] L. Bajo, P. clements y R. Kazman, «Arquitectura de software en la practica,» Tercera ed., Boston: Adisson-Wesley, 2012.

[8] A. A. Segura, «Arquitectura de Software de Referencia para Objetos Inteligentes en Internet de las Cosas,» *Revista Latinoamericana de Ingenieria de Software*, vol. 2, nº 2314-2642, pp. 4-6, 2016.

[9] P. Á. I. I. P. S. Romero, Ciudad de La Habana, Cuba, diciembre del 2005.

[10] C. C. U. Z. B. M. A. R. &. N. J. C. de la Torre Llorente, Guía de Arquitectura N-Capas orientada al Dominio con .NET.4.0, Microsoft, 2010.

[11] A. Velázquez Méndez y A. Maldonado Talamantes, «Arquitectura de pr quitectura de procesos para las instituciones públicas para las instituciones públicas de educación superior de educación superior.,» Cozumel, México, 2005, pp. 109-124.

[12] A. V. Armando Maldonado, «Un Método para definir la Arquitectura de Procesos.,» Association for Information Systems, 2006.

[13] J. P. F. C. V. C. Marisa Perez, «Definición de una Arquitectura de Procesos Utilizando la Metodología BPTrends para la Aplicación del Ciclo de Vida BPM.,» Workshop de Investigadores en Ciencias de la Computación, Buenos Aires, 2017.

[14] B. R. B. H. Jakob Freund, BPMN 2.0 Manual de Referencia y Guía Práctica., Santiago: Camunda, 2017.

[15] «Ungoti,» 30 enero 2018. [En línea]. Available: https://ungoti.com/es/soluciones/desarrollo-de-software/sdlc/.