Estudio comparativo para la facturación electrónica entre arquitecturas monolíticas y orientadas a microservicios con un enfoque en las normas de calidad ISO 25010

Torres Erazo, Ángel Mauricio

Departamento de Ciencias de la Computación

Carrera de Ingeniería de Sistemas e Informática

Trabajo de titulación, previo la obtención del título de Ingeniero en Sistemas e Informática

Ing. Galárraga Hurtado, Juan Fernando

28 de agosto del 2020

URKUND

Urkund Analysis Result

Analysed Document: TorresErazoAngelMauricio.pdf (D78431974)
Submitted: 9/1/2020 5:04:00 AM
Submitted By: amtorres3@espe.edu.ec
Significance: 4 %

Sources included in the report:

TESIS ROBERTO CARLOS BAUTISTA RAMOS.doc (D30203883)
http://www.scielo.org.co/scielo.php?script=sci_arttext&pid=S1900-38032017000100236
https://martinfowler.com/articles/microservices.html
https://iso25000.com/index.php/normas-iso-25000/iso-25010?limit=3&start=6
https://www.scrumguides.org/docs/scrumguide/v2016/2016-Scrum-Guide-Spanish.pdf
https://www.scrum.org/resources/blog/que-es-scrum
https://www.valledelcauca.gov.co/loader.php?lServicio=Tools2&lTipo=viewpdf&id=27962
https://docplayer.es/amp/162699635-Universidad-nacional-agraria-de-la-selva.html

Instances where selected sources appear:

13

.....

Galárraga Hurtado, Juan Fernando

C. C. 171146481-6

DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

CARRERA DE INGENIERÍA DE SISTEMAS E INFORMÁTICA

CERTIFICACIÓN

Certifico que el trabajo de titulación, "**Estudio comparativo para la facturación electrónica entre arquitecturas monolíticas y orientadas a microservicios con un enfoque en las normas de calidad ISO 25010**" fue realizado por el señor **Torres Erazo, Ángel Mauricio** el cual ha sido revisado y analizado en su totalidad por la herramienta de verificación de similitud de contenido; por lo tanto cumple con los requisitos legales, teóricos, científicos, técnicos y metodológicos establecidos por la Universidad de las Fuerzas Armadas ESPE, razón por la cual me permito acreditar y autorizar para que lo sustente públicamente.

Sangolquí, 1 de septiembre del 2020

.....

Galárraga Hurtado, Juan Fernando

C. C. 171146481-6

ESPE
UNIVERSIDAD DE LAS FUERZAS ARMADAS
INNOVACIÓN PARA LA EXCELENCIA

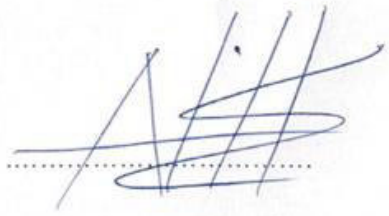
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

CARRERA DE INGENIERÍA DE SISTEMAS E INFORMÁTICA

RESPONSABILIDAD DE AUTORÍA

Yo, **Torres Erazo, Ángel Mauricio**, con cédula de ciudadanía n° 171730029-5, declaro que el contenido, ideas y criterios del trabajo de titulación: **Estudio comparativo para la facturación electrónica entre arquitecturas monolíticas y orientadas a microservicios con un enfoque en las normas de calidad ISO 25010** es de mi autoría y responsabilidad, cumpliendo con los requisitos legales, teóricos, científicos, técnicos, y metodológicos establecidos por la Universidad de las Fuerzas Armadas ESPE, respetando los derechos intelectuales de terceros y referenciando las citas bibliográficas.

Sangolquí, 1 de septiembre del 2020



Torres Erazo, Ángel Mauricio

C.C.: 171730029-5

DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

CARRERA DE INGENIERÍA DE SISTEMAS E INFORMÁTICA

AUTORIZACIÓN DE PUBLICACIÓN

Yo, **Torres Erazo, Ángel Mauricio,** con cédula de ciudadanía n°171730029-5, autorizo a la Universidad de las Fuerzas Armadas ESPE publicar el trabajo de titulación: **Estudio comparativo para la facturación electrónica entre arquitecturas monolíticas y orientadas a microservicios con un enfoque en las normas de calidad ISO 25010** en el Repositorio Institucional, cuyo contenido, ideas y criterios son de mi responsabilidad.

Sangolquí, 1 de septiembre del 2020

.....

Torres Erazo, Ángel Mauricio

C.C.:171730029-5

Dedicatoria

Quiero dedicar este esfuerzo a mi abuelita Elvia María Piedad Suárez, quien me apoyo durante mi formación universitaria; ella más que nadie supo creer en mi desde el principio.

¡Gracias Mamita Pía!

Su nieto, Angelito.

Agradecimiento

Agradecer a mis padres que con su ejemplo supieron inculcarme el valor y la constancia; y a no darme por vencido y a creer que la familia es un elemento fundamental en la formación de toda persona. A mis hermanos quienes me han permitido conocerles en las distintas etapas de su vida cada uno a su forma conviviendo y aprendiendo uno del otro. A todos mis tíos quien con sus ejemplos me han permitido tomar lo mejor de cada uno de ellos. Y en especial a mi esposa quien me ha brindado todo su apoyo durante los meses de realización del presente proyecto, sin duda eres mi fuente de motivación y de amor en mi vida.

Por ultimo quiero agradecer al Ing. Fernando Galárraga, por guiarme y aconsejarme en el desarrollo del presente trabajo.

Gracias a todos.

Mauricio.

Índice de Contenidos

Índice de Tablas

Índice de Figuras

Resumen

El presente proyecto de investigación realiza un estudio comparativo entre las arquitecturas monolíticas y de microservicios, tomando en consideración la aplicación de las normas ISO/IEC 25000 entorno a la característica de la mantenibilidad, proponiendo como caso de estudio el producto software de tipo web facturación electrónica. De acuerdo a estadísticas presentadas por el Standish Group alrededor del 70% de los proyectos de software fracasan; siendo justamente una de las principales causas de este fracaso, relacionada con la mantenibilidad del proyecto. Proponiendo la adopción de arquitecturas que permitan ser exitosas y adicionalmente contribuyan a disminuir el fracaso y los altos costos de mantenimiento, adicionalmente habilitando el cambio del entorno de la organización, contribuyendo a adaptarse a la arquitectura más adecuada para sus requerimientos. El estudio ha tomado las principales subcaracterísticas de las normas ISO 25000 específicos del enfoque de la mantenibilidad apartado 25010, proponiendo valores obtenidos del producto software y apoyándose en el uso de herramientas de obtención de métricas de software a nivel de código; que permiten precisar los valores de los objetos de estudio. Para los resultados presentados deben considerarse las motivaciones y la realidad de las organizaciones y empresas que están involucradas en el desarrollo de software.

PALABRAS CLAVE:

- **ARQUITECTURAS DE SOFTWARE**
- **MICROSERVICIO**
- **MONOLITO**
- **NORMAS ISO 25000**
- **FACTURACIÓN ELECTRÓNICA**

Abstract

This research project carries out a comparative study between monolithic and microservice architectures, taking into consideration the application of ISO / IEC 25000 standards around the characteristic of maintainability, proposing as a case study the web-type software product electronic billing. According to statistics presented by the Standish Group, around 70% of software projects fail; being precisely one of the main causes of this failure, related to the maintainability of the project. Proposing the adoption of architectures that allow being successful and additionally contribute to reducing failure and high maintenance costs, additionally enabling the change of the organization's environment helping to adapt to the most appropriate architecture for its requirements. The study has taken the main sub-characteristics of the ISO 25000 standards specific to the maintainability approach section 25010, proposing values obtained from the software product and relying on the use of tools for obtaining software metrics at the code level; that allow specifying the values of the objects of study. For the results presented, the motivations and reality of the organizations and companies that are involved in software development must be considered.

KEYWORDS:

- **SOFTWARE ARCHITECTURES**
- **MICROSERVICE**
- **MONOLITH**
- **INTERNACIONAL ORGANIZATIONAL FOR STANDARDIZATION 25000**
- **ELECTRONIC BILLING**

Capítulo I

Introducción

Antecedentes

Actualmente la industria de software presenta grandes avances en cada una de las aristas que conforman esta rama de la computación, agilidad a los cambios, demandan un proceso de mejora continua y de innovación en las empresas.

La gestión de los procesos de desarrollo de software en conjunto con la evolución y requerimientos del mercado, requieren de diseños y estructuras flexibles, adaptables a la realidad de las empresas y de alta confiabilidad e integridad en su información.

La experiencia de estudios previos y de casos de éxito anteriores nos permiten obtener resultados adaptables a las plataformas web implementadas bajo el concepto de software como servicio (Software as a Service), brindando seguridad y confiabilidad en la implementación de soluciones de software conforme a la arquitectura de estudio.

La arquitectura de software nos permite visualizar los contextos en los cuales se desempeñan las soluciones de software, especialmente desde un punto de vista organizativo y estructural, el término "Arquitectura de microservicios" ha surgido en los última década para describir una forma particular de diseñar aplicaciones de software como conjuntos de servicios desplegables de forma independiente. (Fowler, Martin Fowler Blog, 2016)

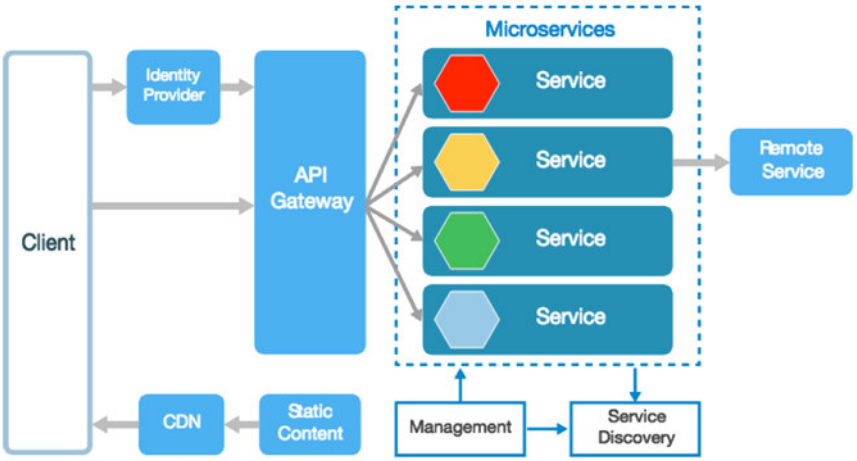
Los microservicios son servicios relativamente pequeños y autónomos que se implementan de forma independiente, con un único propósito claramente definido. (Fowler & Lewis, Microservices, 2014)

Proponen aplicaciones en forma de descomposición vertical a través en un subconjunto de funcionalidades de negocio independientes cada servicio puede ser desarrollado, implementado y probado de forma independientemente por diferentes equipos de desarrollo y utilizando diferentes pilas de tecnología.

Estos pueden ser desarrollados en diferentes lenguajes de programación, pueden escalar independientemente de otros servicios, y se puede implementar en el hardware que mejor se adapte sus necesidades. Además, debido a su tamaño, son más fáciles de mantener y más tolerante a fallos, ya que la falla de un servicio no interrumpirá el sistema completo, lo que podría suceder en un sistema monolítico. (Taibi, Auer, Lenarduzzi, & Felderer, 2019)

Figura 1

Arquitectura General de Microservicios



Nota: En el gráfico podemos evidenciar los componentes que intervienen en la arquitectura el cual ha sido tomado (Wasson, Celarier, 2017)

Software como servicio (SaaS) es un paradigma de entrega de software en el que el software se aloja fuera de las instalaciones y es entregado a través de la web. La forma de pago sigue un modelo de suscripción. (Nitu, 2009) Con la aparición del software como un Servicio (SaaS), las aplicaciones se están alejando de los programas basados en PC basados en la propiedad para ser alojados en los servicios web. (Linlin Wu, 2011)

SaaS ayuda a las organizaciones a evitar gastos de capital y dejar que se centren en su núcleo negocios en lugar de servicios de soporte como TI gestión de infraestructura, mantenimiento de software, etc. (Manish Godse, 2009)

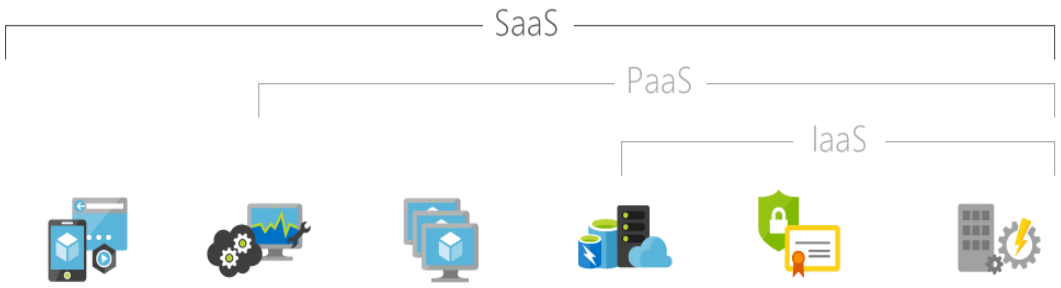
Plataforma como servicio (Platform as a Service): la provisión de una plataforma de desarrollo y entorno que proporciona servicios y almacenamiento, alojados en la nube. (Boniface, y otros, 2010) Proporciona una plataforma como contenedor y un entorno de ejecución en el que los desarrolladores externos implementan y ejecutan sus aplicaciones. (Rodero, 2011) Este nuevo paradigma de desarrollo permite a los propietarios de plataformas aprovechar los beneficios de la creación conjunta de valor y para aprovechar la experiencia y el ingenio externos en una escala sin precedentes (por ejemplo, hay más de 100 000 desarrolladores de iOS. (Tiwana & Konsynski, 2010)

Infraestructura como servicio (Infrastructure as a Service): la provisión de Máquinas "en bruto" (servidores, almacenamiento, redes y otros dispositivos) en la cual los consumidores del servicio instalan su propio software, generalmente como una imagen perteneciente a una máquina virtual. (Boniface, y otros, 2010) Las máquinas virtuales se pueden alquilar a proveedores de IaaS como Amazon EC2 o clústeres virtualizados privados o de propiedad del Proveedor de SaaS.

En ambos casos, la minimización del número de máquinas virtuales generará ahorros. Los ahorros son mayores cuando los proveedores de SaaS utilizan los proveedores de IaaS (Infrastructure as a Service) de terceros ya que no se requiere gasto de capital.

Figura 2

Modelo Representativo de los Paradigmas



Nota: Se puede identificar los principales tipos de computación en la nube tomado de (Microsoft, Azure 2020)

El software como herramienta de utilidad para las organizaciones y empresas al ofrecer la capacidad de satisfacer las necesidades de información, de tal manera que garanticen la implementación de criterios de calidad.

De acuerdo con esta necesidad, diferentes entidades o investigadores han propuesto estrategias modelos, metodologías, guías, incluso normas y estándares de calidad que brindan apoyo al desarrollo y/o uso de un producto software y permiten evaluar si efectivamente tiene un nivel de calidad durante su ciclo de vida, y de esta manera fomentar un ambiente de calidad, con base en la adecuada administración de la información. (CALLEJAS-CUERVO, ALARCÓN-ALDANA, & ÁLVAREZ-CARREÑO, 2017)

Problemática

La arquitectura monolítica actualmente utilizada en la plataforma web Stupendo ha permitido identificar funcionalidades las cuales requieren de especial atención, principalmente en los módulos de recepción y emisión de documentos electrónicos, por lo que existe la necesidad de añadir constantemente nuevas funcionalidades, las cuales son requeridas por sus clientes y usuarios finales. Existen características que requieren que sean mantenibles, estables y eficientes a ejecutarse en entornos de plataformas web (PAS) y de software como servicio (SAS).

Actualmente una de las principales características de la plataforma, es su adopción en el mercado, esto se debe en mayor medida por su capacidad de adaptación e integración a los sistemas y plataformas de administración contable y financieros ya existentes, permitiendo de esta manera a sus clientes reducir los tiempos de implementación e integración.

Esto permite que basados en una arquitectura tradicional como es la monolítica se permita que el software crezca en tamaño y disminuya capacidad de evolutiva afectado al mantenimiento y sostenibilidad de la misma, de esta manera posibilita la facilidad de introducir un fallo en una característica o personalización para un cliente y este afecte a todos los demás.

Esta característica de personalización en base a cada uno de sus clientes se ha convertido en una dificultad para seguir creciendo y evolucionando en funcionalidad y robustez, principalmente por su diseño monolítico.

Razón por lo cual la plataforma crece desmedidamente sin una evolución ordenada, consistente, sustentable y de acuerdo a las necesidades de sus clientes, esto con lleva a una difícil implementación de nuevas características lo que permite que la plataforma sea inestable y propensa a errores.

Justificación

La empresa ESDINAMICO CIA.LTDA. a través de su plataforma SAS Stupendo ha tomado la decisión de brindar a sus clientes una mejor adopción de sus requerimientos y responder de la mejor manera a los cambios solicitados para cada uno de los módulos que intervienen en la facturación electrónica. De la misma manera el mantenimiento del sistema es de vital importancia para la gestión y evolución del producto.

Con la implementación de una arquitectura basada en microservicios se pretende lograr integración desacoplada entre los distintos módulos existentes y por consiguiente permitir nuevas funcionalidades basadas en la necesidad del mercado y de la empresa, además de proporcionar un mantenimiento ágil y optimizado.

Para el caso en el que se decida no implementar la arquitectura basada en microservicios, el producto será cada vez más complicado de mantener y la productividad de los equipos de desarrollo se verá afectada, de tal forma en la cual la rigidez de la misma complique desarrollar nuevos productos complementarios.

Objetivos

Objetivo General

Realizar el estudio comparativo entre las arquitecturas monolíticas y orientadas a microservicios para la facturación electrónica con un enfoque en las normas de calidad ISO 25010.

Objetivos Específicos

i. Analizar las métricas de calidad del software ISO 25010 en el módulo de facturación electrónica basada en la arquitectura monolítica.

ii. Implementar el módulo de facturación electrónica utilizando la arquitectura de microservicios.

iii. Desarrollar un estudio comparativo entre las arquitecturas monolíticas y de microservicios utilizando tecnologías open source en base a las métricas ISO 25010.

Alcance

El trabajo de titulación implementa el módulo de facturación electrónica en una plataforma de software como servicio (SAS) utilizando la arquitectura orientada a microservicios aplicando como referencia las normas ISO 25010 con enfoque en la mantenibilidad del software utilizando el framework Laravel para el lenguaje PHP.

Presentar los resultados de forma clara y en base a la aplicación de estudios anteriores particularmente implementando la arquitectura de microservicios, presentando casos de éxito en plataformas basadas en la nube (Cloud Computing) y de tipo SAS (Software as a Service).

Diseñar los principales diagramas de arquitectura y de componentes principales en el diseño a implementar los cuales corresponden a las funcionalidades de emisión de documentos electrónicos que actualmente presenta la plataforma web Stupendo.

Definición de la Investigación

De acuerdo a Rodríguez (Rodriguez, 2006) la investigación científica es un proceso que, mediante la aplicación del método científico, procura obtener información relevante y fidedigna para comprobar la hipótesis.

El propósito del trabajo de titulación se llevará una investigación aplicada, entendida como la utilización de los conocimientos en la práctica, para aplicarlo en beneficio de los grupos que participa en esos procesos y en la sociedad en general va a ser utilizada para generar un producto (Vargas Cordero, 2009), que nos permite implementar la arquitectura orientada a microservicios en la facturación electrónica.

Según el nivel de conocimiento que se adquiere en la investigación, el presente trabajo de titulación es de tipo Investigación Descriptiva ya que busca relatar el conocimiento adquirido y socializarlo a la comunidad científica (Kirsch, 1992) a través de la generación del estudio comparativo.

Capítulo II

Marco Teórico

Para el presente marco teórico tomamos la definición expuesta por el Instituto Nacional de Estándares y Tecnología de los Estados unidos de Norteamérica el Cloud Computing: “Es un modelo para permitir el acceso (ubicuo, fácil y bajo demanda) a través de una red a un pool compartido de recursos informáticos configurables que pueden ser rápidamente aprovisionados y liberados con un mínimo esfuerzo administrativo y con una mínima interacción con el proveedor de servicios” (Mell & Grance, 2011)

Bajo Demanda

El consumidor del servicio podrá aprovisionar unilateralmente las capacidades informáticas como la hora del servidor y almacenamiento en red sin la intervención humana dispuesta por el proveedor del servicio.

Mecanismos de Acceso

Proveer de mecanismos estándar de acceso a los recursos haciendo énfasis en la utilización de plataformas heterogéneas (dispositivos móviles, laptops, estaciones de trabajo, Tablet).

Proveedores de Recursos

El proveedor de los servicios pone a disposición los recursos informáticos necesarios agrupándolos para servir a varios consumidores, recursos que pueden ser físicos y/o virtuales (almacenamiento, procesamiento, memoria y ancho de banda de red) dinámicamente asignados en función de la demanda del consumidor, teniendo como característica principal que esta disposición de recursos es totalmente transparente.

Capacidad de Elasticidad

La capacidad de aprovisionamiento en función de la demanda del consumidor se recomienda que sea automática y libre de la intervención humana las cuales pueden llegar a ser casi ilimitadas y en cualquier cantidad y en cualquier momento.

Proveedores de Servicio

Otorgar transparencia tanto al consumidor como al proveedor de los servicios disponibles a través del control, monitoreo e información relativa al uso de los recursos.

Modelos de Servicio

Los modelos de servicios nos permiten desarrollar canales de distribución por los cuales se implementa el producto requerido y que adicionalmente presentan características comunes.

Software

El acceso a las aplicaciones soportadas por el proveedor al consumidor las cuales se ejecutan sobre la infraestructura en la nube, a las aplicaciones se pueden acceder a través de distintos dispositivos por medio de una interfaz de cliente ligero. La configuración de las aplicaciones a nivel de usuario queda totalmente disponible para el consumidor, sin embargo, los recursos dispuestos para el acceso a dichas aplicaciones no están disponibles.

Plataforma

Brindar la capacidad de implementar aplicaciones adquiridas o desarrolladas por el consumidor en la nube del proveedor de la infraestructura. El consumidor no controla ni administra la infraestructura de nube subyacente, lo cual incluye los servicios de red, los servidores, los sistemas operativos o el almacenamiento.

Infraestructura

Otorgar al consumidor la capacidad de aprovisionamiento de procesamiento, almacenamiento, redes y otros recursos de software que el consumidor requiera, sin embargo, no está habilitado para administrar o controlar los componentes internos de su infraestructura operativa, así como tampoco a los sistemas de almacenamiento, sistema operativo y aplicaciones desplegadas.

Modelos de Despliegue

Privada

La infraestructura en la nube puede estar dentro o fuera de las empresas y pueden ser o no administradas por la misma o por un tercero, y el uso es de exclusividad de la empresa que sin embargo pueden hacer uso distintos consumidores

Comunitaria

Este tipo de infraestructura está disponible para uso exclusivo de una determinada comunidad de consumidores los cuales compartan los mismos objetivos. Estas infraestructuras pueden ser administradas por la comunidad y operados por una o más organizaciones de la misma.

Pública

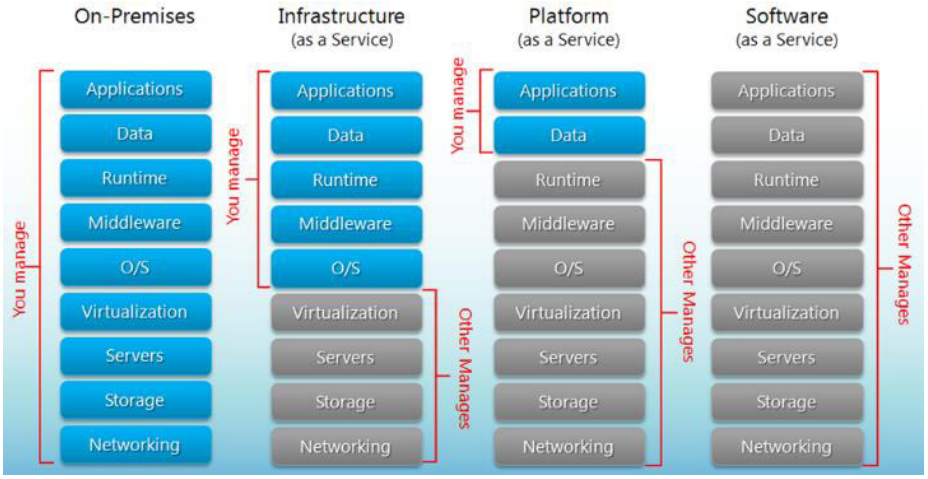
La infraestructura de este tipo de nube se encuentra abierta al público en general, esta puede ser provista y administrada por una organización académica, gubernamental, comercial y/o una combinación de estas.

Híbrida

Es una combinación entre las nubes de infraestructura (públicas, privadas o comunitarias), la característica por la cual se vinculan estas organizaciones es obtener una tecnología estandarizada lo cual permita equilibrar cargas entre nubes.

Figura 3

Modos de Despliegue en la Nube



Nota: Pilas de tecnología las cuales nos permiten adaptar los requerimientos en función de los recursos disponibles y modalidades de despliegue tomado de (http://apprenda.com/library/paas/iaas-paas-saas-explained-compared)

Plataformas de Integración

En la actualidad la necesidad de los mercados cada vez más dinámicos tanto en servicios como en productos, producen interrelaciones las cuales benefician tanto a los consumidores de dichos recursos e información, como a los proveedores que la generan: “aquellas empresas que disponen de herramientas de integración, tanto en la nube como on-premise, estas integran los nuevos procesos de forma sencilla a través de los mecanismos de integración que han normalizado para los distintos casos de uso detectados.

Una vez definidos estos mecanismos, la integración de estos procesos en los flujos productivos de la compañía es sencilla desde el punto de vista tecnológico.

Las herramientas de integración simplifican la incorporación de los nuevos procesos puesto que soportan múltiples interfaces heterogéneos de forma nativa, proporcionan mecanismos de mapping de datos visual, monitorización, seguridad, control de flujo, etc.” (Navarro, 2017)

Servicios Web

Para el consorcio de la World Wide Web (W3C) define los servicios web de la siguiente manera: Un servicio web es un sistema de software diseñado para admitir la interacción interoperable de máquina a máquina a través de una red. Tiene una interfaz descrita en un formato procesable por máquina (específicamente WSDL). Otros sistemas interactúan con el servicio web de la manera prescrita por su descripción usando mensajes SOAP, típicamente transmitidos usando HTTP con una serialización XML junto con otros estándares relacionados con la web. (Booth, y otros, 2014)

Figura 4

Servicios Web

Nota: Interacción de los sistemas de información a través de servicios web tomado de Universidad Veracruzana

Las principales características que presentan los servicios web son:

- Accesibilidad
- Capacidad de descripción propia
- Interoperabilidad

- Localizable

Servicios Rest

Un enfoque no tradicional a los servicios web es la transferencia de estado representacional (REST) ser un conjunto coordinado de restricciones arquitectónicas que intentan minimizar la latencia y la comunicación de redes mientras que, al mismo tiempo, maximiza la independencia y escalabilidad de los componentes de implementación. REST habilita la captura y reúso de interacciones, sustituyendo dinámicamente los componentes y procesando las acciones por medio de intermediarios (Fielding R. T., 2000).

Esta definición en conjunto con la información complementaria requerida para entender los elementos que la componen: Elementos de Datos, Conectores y Componentes. Conforman la arquitectura de software basadas en la red.

REST se utiliza principalmente en el uso de sistemas de hipermedia distribuidos, según Roy: La lógica del diseño detrás de la arquitectura web se puede describir mediante un estilo arquitectónico que consiste en el conjunto de restricciones aplicadas a elementos dentro de la arquitectura (Fielding R. T., 2000).

Este recurso de integración no es un estándar de facto en la industria de software, pero cubre en gran medida las necesidades actuales, en un mundo dominado por la red de redes, ya que este es solo un estilo arquitectónico de software para redes es decir no es un estándar de facto en la industria y es utilizado por sus sólidos cimientos los cuales si son estándares como: HTTP, URL, MIME TYPE, etc.

Los objetivos que busca este estilo son:

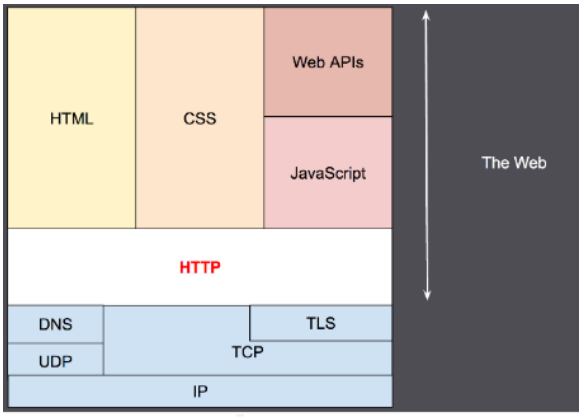
- Escalabilidad de la interacción de los componentes
- Generalidad de interfaces
- Funcionamiento Independiente
- Compatibilidad a través de componentes intermedios

Protocolo HTTP

El Protocolo de transferencia de hipertexto (HTTP) es un protocolo de nivel de aplicación para sistemas de información distribuidos, colaborativos e hipermedia. Es un protocolo genérico, sin estado, que se puede usar para muchas tareas más allá de su uso para hipertexto, como servidores de nombres y sistemas de administración de objetos distribuidos, a través de la extensión de sus métodos de solicitud, códigos de error y encabezados (Berners-Lee, Fielding, & Frystyk, rfc-editor, 1996).

Figura 5

Protocolo HTTP en Internet



Nota: Capas del protocolo HTTP en las mismas que son transparentes para el usuario común tomado (https://developer.mozilla.org/es/docs/Web/HTTP/Overview)

El protocolo HTTP es ideal para permitir el intercambio contenidos a través de internet, debido a que no mantiene el estado de anteriores solicitudes realizadas a ese recurso y principalmente el contenido de los mensajes enviados en las peticiones y respuestas son en forma de texto plano. Además, se encuentra basado en el diseño de cliente servidor en el que un cliente (navegador web) envía peticiones a un servidor (servidor web).

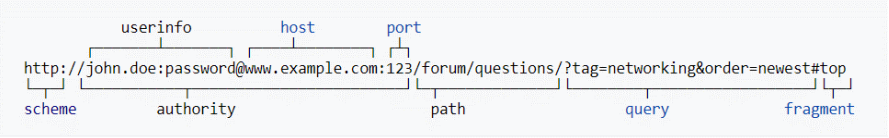
Existen componentes que se presentan de forma transparente en esta arquitectura web basada en capas: routers, módems, switches, etc. Ya que el protocolo HTTP se encuentra en la capa de aplicación este se apoya en las otras capas que contienen a dichos componentes, lo cual permite una mejor transparencia en su utilización.

Identificador Uniforme de Recursos URI

Es una secuencia compacta de caracteres que identifica un recurso abstracto o físico. (Berners-Lee, Fielding, & Masinter, ietf, 2005) y por lo que permite identificar los recursos de una red de forma unívoca.

Figura 6

Representación de URI



Nota: Permite identificar los segmentos involucrados en los identificadores uniformes de recursos tomado de https://en.wikipedia.org/wiki/Uniform_Resource_Identifier

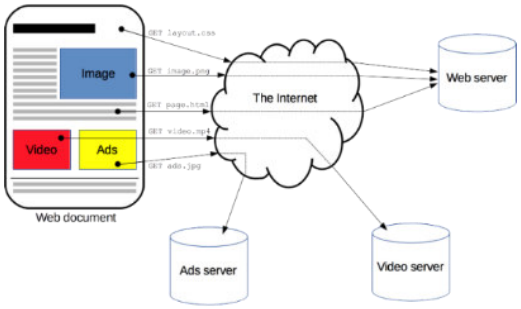
Métodos de Petición

Para indicar la acción a realizar sobre un recurso HTTP define un conjunto de métodos los cuales sirven como solicitud para aplicar al recurso determinado, estos métodos también conocidos como verbos presentan las siguientes características: safe, idempotent, o cacheable. (Fielding & Reschke, 2014)

- GET solicita una representación de un recurso específico. Las peticiones que usan el método GET sólo deben recuperar datos.
- HEAD pide una respuesta idéntica a la de una petición GET, pero sin el cuerpo de la respuesta.
- POST se utiliza para enviar una entidad a un recurso en específico, causando a menudo un cambio en el estado o efectos secundarios en el servidor.
- PUT reemplaza todas las representaciones actuales del recurso de destino con la carga útil de la petición.
- DELETE borra un recurso en específico.
- CONNECT establece un túnel hacia el servidor identificado por el recurso.
- OPTIONS es utilizado para describir las opciones de comunicación para el recurso de destino.
- TRACE realiza una prueba de bucle de retorno de mensaje a lo largo de la ruta al recurso de destino.
- PATCH es utilizado para aplicar modificaciones parciales a un recurso.

Figura 7

Métodos de Petición



Nota: Funcionamiento de los métodos de petición a través de una aplicación o página web. (w3c, 2014)

Códigos de Estado de Respuesta

Representan una respuesta HTTP indicando si una petición de recurso específico se ha completado con éxito. Estos están agrupados en 5 grupos y el rango de valores que pueden tomar:

1. Información (100 - 199)
2. Satisfactorios (200 - 299)
3. Redirecciones (300 - 399)
4. Errores del cliente (400 - 499)
5. Errores del Servidor (500 - 599)

Framework Laravel

Es un marco de trabajo rápido de desarrollo de aplicaciones, eso significa que se enfoca en una curva de aprendizaje superficial (fácil) y minimizando los pasos entre iniciar una nueva aplicación y publicarla. Todas las tareas más comunes en la creación de aplicaciones web, desde interacciones de base de datos hasta autenticación, colas, correo electrónico y almacenamiento en caché son más simples por los componentes que proporciona. (Stauffer, 2019).

Laravel reutiliza y ensambla componentes existentes para proporcionar una capa cohesiva sobre la cual se pueden construir aplicaciones web de una manera más estructurada y pragmática, ha sido diseñado para mejorar la calidad del software al reducir tanto el costo del desarrollo inicial y costos de mantenimiento continuos, así como mejorar la experiencia de trabajar con sus aplicaciones al proporcionar una sintaxis expresiva clara a través de un conjunto básico de funcionalidades permitiendo ahorrar en el tiempo de implementación.

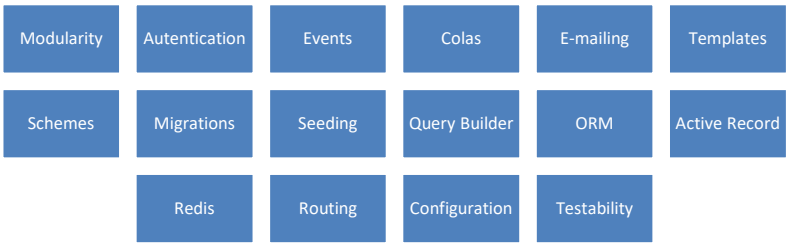
Este marco de trabajo (framework) se enfoca en el paradigma de la convención sobre la configuración: “Es un concepto simple que se utiliza principalmente en la programación. Significa que el entorno en el que trabaja (sistemas, bibliotecas, lenguaje ...) asume muchas situaciones lógicas por defecto, por lo que, si se adapta a ellas en lugar de crear sus propias reglas cada vez, la programación se convierte en una tarea más fácil y productiva”. (Sáez, 2020) Se encuentra relacionado a otros principios como “Valores Predeterminados Sensibles” y “El principio de la mínima sorpresa”. A menudo implica un lenguaje específico de dominio con un conjunto limitado de construcciones o una inversión de control en el que el desarrollador solo puede afectar el comportamiento utilizando un conjunto limitado de hooks (Olsen, Convention Over Configuration, 2007)

Por varios años los desarrolladores de aplicaciones basadas en el lenguaje PHP han implementado sus propios enfoques en tareas comunes tales como el manejo de peticiones y solicitudes HTTP. Esto creo un ruido en la comunidad y no permitía la integración correcta en cada uno de los proyectos en los que se requería reutilizar estas tareas comunes. Por lo que la comunidad alrededor de un framework con el nombre de Symfony decidió asumir la responsabilidad de crear componentes que sean una base bien cimentada para el desarrollo de aplicaciones web robustas.

La visión de Laravel ha sido pragmático al apoyarse sobre componentes aprobados por la comunidad de desarrolladores e incorporarlos a su arquitectura, como por ejemplo los componentes desarrollados por Symfony. En lugar de reinventar la rueda una y otra vez, se adoptado la filosofía de apoyarse principalmente en componentes maduros. A diferencia de frameworks contemporáneos basados en el leguaje PHP, Laravel acoge abiertamente las nuevas características del lenguaje, en beneficio de la simpleza, elegancia y legibilidad del código; en lugar de la retro compatibilidad en versiones anteriores del lenguaje.

Figura 8

Características del Framework Laravel



Nota: Componentes que intervienen en el núcleo de framework tomado de (Laravel Docs, 2018)

Modelo Vista Controlador

La programación del Modelo-Vista-Controlador (MVC) es la aplicación de esta factorización de tres vías, mediante la cual los objetos de diferentes clases se hacen cargo de las operaciones relacionadas con el dominio de la aplicación (el modelo), la visualización del estado de la aplicación (la vista) y la interacción del usuario con el modelo y la vista (el controlador). (Krassner & Pope, 1988)

Esta implementación define tres tipos de objetos claramente identificados de la siguiente manera: El modelo es el objeto de aplicación, la vista es su representación en pantalla y el controlador define el modo en que la interfaz reacciona a la entrada del usuario. Con la finalidad de incrementar la flexibilidad y reutilización MVC desacopla las vistas de los modelos estableciendo entre ellos un protocolo de suscripción/notificación. (Gamma, Helm, Johnson, & Vlissides, 1995)

El MVC (Modelo Vista Controlador) generalmente se trata como un mecanismo único, pero es útil dividirlo en patrones más pequeños. Estos patrones son más fáciles de entender de forma aislada y a menudo se reutilizan en otros contextos, y es más fácil definir con precisión su propósito.

Surgen tres patrones principales: Observador, que describe la relación entre modelos y vistas; Compuesto, que describe la relación entre Vistas y sus subvistas; y Estrategia, que describe la relación entre vistas y controladores. (Rising, 1998)

La amplia utilización de esta técnica de separación de conceptos ha sido en realidad el éxito de muchos frameworks en la actualidad, por lo que se han conformado distintas opiniones si es una implementación de un patrón de arquitectura o de diseño.

Arquitectura Orientada a Microservicios

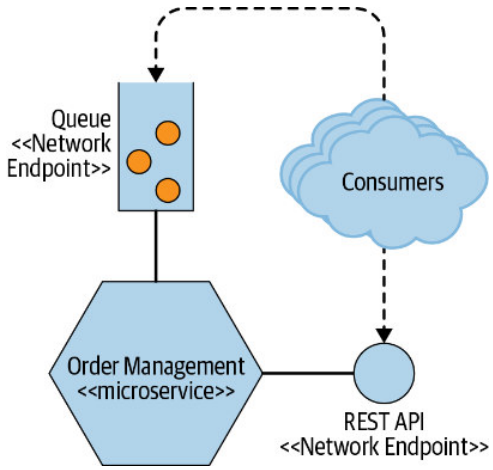
Es un tipo de arquitectura orientada al servicio, aunque con un enfoque sobre cómo se deben trazar los límites del servicio basados en los modelos del dominio del negocio, y con la ayuda de la implementación independiente. Estos servicios son independientes de la tecnología y evitan el uso de bases de datos compartidas como métodos de integración; en cambio, cada microservicio encapsula su propia base de datos cuando es necesario.

De acuerdo a una definición resumida: “los microservicios son pequeños y autónomos” (Newman, Building Microservices, 2015), estos son liberables de forma independiente que se modelan en torno a un dominio empresarial. Un servicio encapsula la funcionalidad y la hace accesible a otros servicios a través de redes. Representan una opción de arquitectura enfocada en brindar alternativas para resolver los problemas que pueda enfrentar. Un microservicio es una entidad simple y aislada con una propuesta concreta, es independiente y funciona con el resto de los microservicios mediante la comunicación a través de un canal acordado.

Los microservicios adoptan el concepto de ocultar la mayor cantidad de información posible dentro de un componente y exponer la menor cantidad posible a través de interfaces externas.

Figura 9

Exposición de Funcionalidad sobre un REST API y una Cola



Nota: Microservicio que administrar las solicitudes de interacción a través de sus interfaces (Newman, What Are Microservices?, 2019)

La implementación que está oculta para terceros puede cambiarse libremente siempre que las interfaces de red que expone el microservicio no cambien de una manera incompatible con versiones anteriores. Los cambios dentro de un límite de microservicio no deberían afectar a un consumidor ascendente, permitiendo una capacidad de liberación independiente de la funcionalidad.

Por lo cual, (Newman, What Are Microservices?, 2019) manifiesta “esto permite que nuestros microservicios se desarrollen de forma aislada y se liberen bajo demanda. Tener límites de servicio claros y estables que no cambian cuando la implementación interna cambia da como resultado sistemas que tienen un acoplamiento más flexible y una mayor cohesión”.

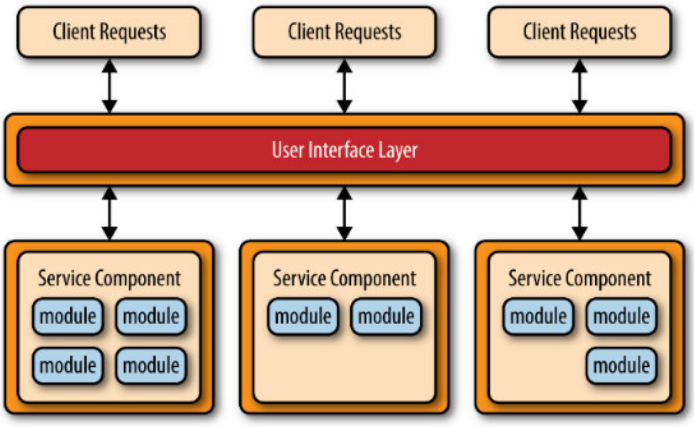
Características Principales

Uno de los objetivos principales que acoge esta arquitectura es la de minimizar los cambios en las interfaces de servicio a través de límites de servicios coherentes y mecanismos de evolución en los contratos de dichos servicios, presentamos las características más representativas que surgen de la aplicación de esta arquitectura.

- Procesos Independientes
- Comunicación sobre API´s
- Alto grado de autonomía
- Pequeños con enfoque específico

Figura 10

Patrón Básico de la Arquitectura de Microservicios



Nota: Identificamos los microservicios que intervienen para satisfacer las solicitudes de los clientes (Richards, 2015)

Ventajas. Las principales ventajas se las describe a partir de la siguiente lista:

- Alinear la organización en función de la arquitectura

- Liberar funcionalidad rápidamente
- Escalar de forma independiente
- Fácil de enfocarse en preocupaciones de seguridad
- Rápida adopción de nuevas tecnologías
- Adoptar de mejor manera la incertidumbre de la era de la digitalización

Desventajas. A través del siguiente listado describimos sus principales desventajas:

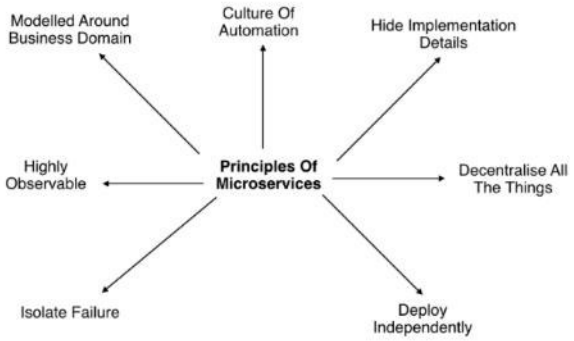
- Distintos enfoques para abordar
- Toma tiempo en implementarlo totalmente
- Pruebas más complejas a realizar
- Herramientas de monitorización más complejas
- Resiliencia requiere la implementación de nuevos patrones
- Existen más maquinas que configurar, coordinar y controlar
- Los sistemas distribuidos con llevan su complejidad

Principios de los Microservicios

La implementación de una arquitectura puede seguir distintos caminos para conseguir su objetivo, y en la arquitectura orientada a microservicios no es la excepción; por lo cual se han determinado una serie de principios a seguir para crear servicios utilizando esta arquitectura.

Figura 11

Principios de los Microservicios



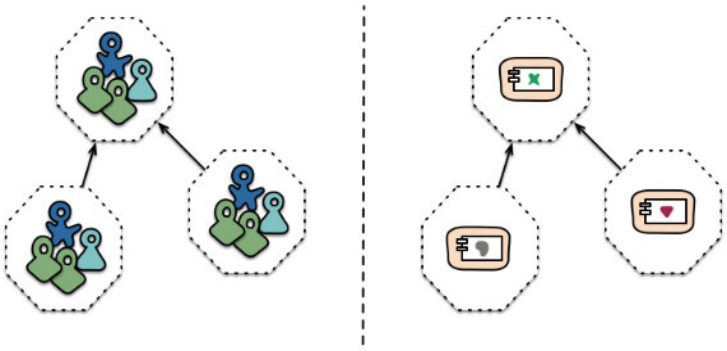
Nota: Principios los cuales nos apoyan para tener una implementación satisfactoria (Newman, Sam Newman & Associates, 2020)

Modelar el Dominio de Negocio. Para modelar servicios organizados en torno a la capacidad empresarial se requiere una implementación amplia de software para esa área de negocios en concreto, lo cual implica: la interfaz de usuario, el almacenamiento persistente y cualquier colaboración externa.

En consecuencia, los equipos son multifuncionales, incluida la gama completa de habilidades necesarias para el desarrollo: experiencia de usuario, base de datos y gestión de proyectos. (Fowler & Lewis, Microservices, 2014)

Figura 12

Límites del Servicio Reforzados por el Equipo



Nota: Identificar estos límites nos permiten modelar la arquitectura para aceptar nuevos cambios a futuro referencia (Fowler & Lewis, Microservices, 2014)

Adicionalmente existen técnicas que permiten una clara identificación en lo que se trata a las capacidades del dominio, técnicas como el diseño basado en dominios pueden permitirle estructurar el código para representar mejor el dominio del mundo real en el que opera el software.

Nuestro dominio empresarial se convierte en la fuerza principal que impulsa la arquitectura del sistema, con la esperanza de que sea más fácil hacer cambios y de que sea más fácil organizar nuestros equipos en torno a nuestro dominio empresarial.

Cultura de Automatización. La implementación de una cultura de automatización permite agilizar la manera en el que las operaciones entorno al software que se produce, por lo cual incrementa la velocidad en la cual las tareas rutinarias se sistematizan de manera que se vuelven monótonas. Las operaciones candidatas a esta automatización son las siguientes:

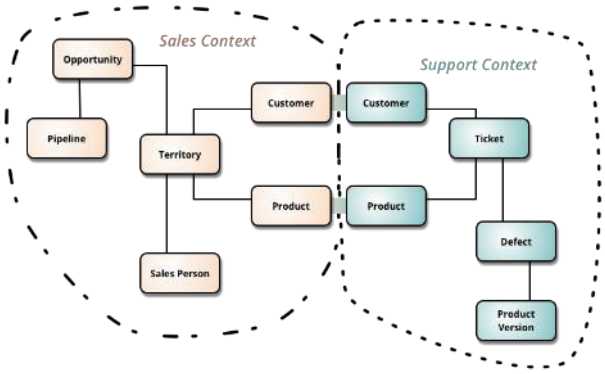
- Infraestructura
- Pruebas
- Entrega Continua

Ocultar Detalles de Implementación. Permite aislar de manera coherente los contextos en los cuales se desarrolla nuestra aplicación, por lo que en general se aconseja a que el único medio de comunicación entre servicios sean las interfaces de red REST API obteniendo los siguientes beneficios:

- Tiempo de Desarrollo Mejorado: Al permitir que los módulos se desarrollen de forma independiente, podemos permitir que se realice más trabajo en paralelo y reducir el impacto de agregar más desarrolladores a un proyecto.
- Compresibilidad: Cada módulo puede verse de forma aislada y entenderse de forma aislada. Esto facilita la comprensión de lo que hace el sistema en su conjunto.
- Flexibilidad: Los módulos se pueden cambiar independientemente uno del otro, lo que permite realizar cambios en la funcionalidad del sistema sin requerir que otros módulos cambien.

Figura 13

Contexto Acotado



Nota: Evidenciamos que para un contexto totalmente diferente pueden ocurrir polisemias en algunas entidades referencia (Fowler, Martin Fowler Blog, 2016)

“El contexto limitado es un patrón central en el diseño impulsado por dominio. Es el foco de la sección de diseño estratégico de DDD que se trata de tratar con grandes modelos y equipos. DDD trata con modelos grandes dividiéndolos en diferentes contextos limitados y siendo explícito sobre sus interrelaciones”. (Fowler, Martin Fowler Blog, 2016)

Descentralización. Uno de los beneficios que permite la arquitectura es poder servirse de recursos por su propia cuenta lo que conocidamente se llama self-service permitiendo a las personas desplegar el software bajo demanda, haciendo el desarrollo y las pruebas lo más fácil posible y evitar dividir el equipo para realizar estas actividades. (Newman, Building Microservices, 2015)

Permite a los equipos trabajar en diferentes tipos de tecnologías, y que sean estos los que se encarguen de desplegarlos y soportarlos a través del ciclo de vida del servicio.

Adicionalmente habilita a los equipos de desarrollo poder provisionar su ambiente de desarrollo de una manera descentralizada sin mucho gobierno de tecnología, todo esto con la mira de poder desenvolverse de mejor manera.

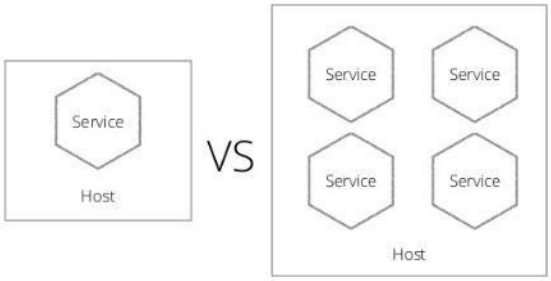
Descentralizar incluso los llamados ESB (Enterprise Service Bus) comúnmente utilizados en las arquitecturas orientadas a servicios que lo único que han permitido es que se haga más compleja la lógica del negocio centralizándola en el ESB. La comunidad de microservicios favorece un enfoque alternativo: puntos finales inteligentes y tuberías tontas.

“Las aplicaciones creadas a partir de microservicios pretenden ser lo más desacopladas y cohesivas posibles: poseen su propia lógica de dominio y actúan más como filtros en el sentido clásico de Unix: reciben una solicitud, aplican la lógica según corresponda y producen una respuesta. Estos son coreografiados utilizando protocolos RESTish simples en lugar de protocolos complejos como WS-Choreography o BPEL u orquestación mediante una herramienta central” (Fowler & Lewis, Microservices, 2014)

Despliegue Independiente. Permitir el despliegue de nuevas características en cada uno de los servicios de forma independiente posibilita a poder identificar rápidamente el comportamiento del servicio y sus posibles fallos en el caso de presentarse, este principio es uno de los más importantes y sobre el cual se cimienta la construcción de servicios, habilitando así que el sistema sea más resilente. Adoptando el modelo un servicio por host, se reduce los efectos secundarios que pueden causar que el despliegue de un servicio afecte a otro que no se encuentre relacionado. (Newman, Building Microservices, 2015)

Figura 14

Un Servicio por Host



Nota: Una de las principales recomendaciones es la intentar administrar un microservicio dentro una sola maquina ya se virtual o contenerizada (Newman, Sam Newman & Associates, 2020)

Primero el Consumidor. La principal motivación de crear servicios es que existan porque alguien los necesita un “consumidor”, no descuidar este aspecto habilita la satisfacción del mismo entregando información no solo a través de los endpoints, sino a través de una buena documentación que permita identificar rápidamente la información que se necesita para poder intercambiar entre servicios.

Existen herramientas que son útiles las cuales implementan el patrón de diseño Service Discovery que son muy útiles sobre todo en entornos altamente escalables (Ej. 600 microservicios y mas)

Aislar Fallos. Un punto importante a entender es que los microservicios no son resistentes por defecto, pueden existir fallas en los servicios, estas pueden ocurrir debido a fallas en los servicios dependientes. Lo cual pueden surgir por una variedad de razones, como errores en el código, tiempos de espera de la red, etc.

Lo que es crítico con una arquitectura de microservicios es garantizar que todo el sistema no se vea afectado o se caiga cuando haya errores en una parte individual del sistema. Hay patrones como Bulkhead y Circuit Breaker que pueden ayudarlo a lograr una mejor resistencia.

Altamente Observable. Una arquitectura de microservicios requiere un medio para visualizar el estado de salud de todos los servicios en el sistema y las conexiones entre ellos. Esto le permite localizar rápidamente y responder a los problemas que puedan ocurrir. Las herramientas que permiten la visualización incluyen un mecanismo de registro completo para registrar, almacenar y hacer que los registros se puedan buscar para mejorar el análisis. (Overview of Microservices, 2016)

“No podemos confiar en observar el comportamiento de una única instancia de servicio o el estado de una sola máquina para ver si el sistema funciona correctamente. En cambio, necesitamos una visión conjunta de lo que está sucediendo” (Newman, Building Microservices, 2015).

Norma ISO/IEC 25000

La podemos definir como: “Proceso eficaz de software que se aplica de manera que crea un producto útil que proporciona valor medible a quienes lo producen y a quienes lo utilizan” (Bessin, 2005).

Uno de los atributos por los cuales se ha determinado el presente proyecto es el perteneciente a la mantenibilidad, esta nos permite evolucionar los sistemas de software de manera rápida y eficiente lo que es fundamental para organizaciones de hoy en día.

Figura 15

Organigrama de la Norma ISO 25010

Nota: La elección de uno de los determinados enfoques queda a libre elección y conveniencia del producto para quienes lo elaboran y mantienen.

Mantenibilidad

"Esta característica representa la capacidad del producto software para ser modificado efectiva y eficientemente, debido a necesidades evolutivas, correctivas, perfectivas o adaptativas" (Valenciano Lopéz, 2015)

Evolutivas y/o Perfectivas

Involucra a las nuevas funcionalidades y prestaciones al software, con el objetivo de: "mejorar el rendimiento de las existentes y readaptar nuevos procedimientos de trabajo, permiten conseguir que el sistema sea más operativo.

Correctivas

La corrección de los fallos y defectos de los programas, del mismo modo los efectos fruto de las etapas tempranas de desarrollo.

Adaptativas

Cambios a realizar los cuales en ocasiones son derivados de las actualizaciones en los sistemas operativos de acuerdo con las normas ISO.

Por lo que pueden corresponder a un porcentaje bajo en comparación con los restantes mantenimientos.

Factores Relacionados con la Mantenibilidad

Proceso de Desarrollo

- Parte integral del proceso de desarrollo del software.
- Técnicas implementadas poco intrusivas con el software existente.
- Cultura de mantenibilidad.

Documentación

- Falta de especificaciones de diseño.
- Documentación inexistente o incompleta.

Subcaracterísticas de la Mantenibilidad

Figura 16

Principales Subcaracterísticas de la Mantenibilidad

Subcaracterísticas	Descripción
Modular	Capacidad de un sistema o programa de ordenador (compuesto de componentes discretos) que permite que un cambio en un componente tenga un impacto mínimo en los demás.
Reusable	Capacidad de un activo que permite que sea utilizado en más de un sistema software o en la construcción de otros activos.
Analizabilidad	Nivel de comodidad en la cual se permite analizar el grado de afectacion por los cambios sucedidos, asi como enmarcar las posibles raices de tales males
Capacidad para ser modificado	Habilidad que consigue su maleabilidad de manera concreta, no introduciendo males o errores en su desenvolvimiento
Capacidad para ser probado	Facilidad con la que se pueden establecer criterios de prueba para un sistema o componente y con la que se pueden llevar a cabo las pruebas para determinar si se cumplen dichos criterios.

Nota: Información a partir de las normas ISO (ISO/IEC 25010, 2020)

Modularidad. Las principales métricas para evaluar esta subcaracterísticas son: Capacidad de condensación y el acoplamiento entre clases.

Analizabilidad. La característica de la analizabilidad permite considerar métricas que involucren el diagnostico de las deficiencias o las mismas raíces del problema y mostrar los componentes que requieren de alteración.

Modificabilidad. Se define como: “la relación entre dos o más responsabilidades que posee un sistema y su acoplamiento entre las mismas” (Valenciano Lopéz, 2015).

“Si una responsabilidad A tiene un alto grado de acoplamiento con la responsabilidad B, el costo de cambiar A es menor si las dos responsabilidades son asignadas al mismo módulo” (Valenciano Lopéz, 2015).

- Acoplamiento
- Cohesión
- El Costo promedio.

Por lo que se puede inferir que los cambios detectados en etapas tempranas del desarrollo y su implementación serán mucho menos costosos que los realizados en sus etapas finales.

Capacidad de ser probado. Existen distintos modelos de desarrollo de software, así como modelos de pruebas. A cada uno le corresponde un nivel distinto de involucramiento en las actividades de desarrollo (ISO/IEC 25010, 2020).

- Pruebas Estáticas:

“Son el tipo de pruebas que se realizan sin ejecutar el código de la aplicación, pueden referirse a la revisión de documentos, ya que no se hace una ejecución de código, esto se debe a que se pueden realizar “pruebas de escritorio” con el objetivo de seguir los flujos de la aplicación” (ISO/IEC 25010, 2020).

- Pruebas Dinámicas

“Todas aquellas pruebas que para su ejecución requieren la ejecución de la aplicación, permiten el uso de técnicas de caja negra y caja blanca con mayor amplitud. Debido a la naturaleza dinámica de la ejecución de pruebas es posible medir con mayor precisión el comportamiento de la aplicación desarrollada” (ISO/IEC 25010, 2020).

Reusabilidad. De acuerdo a las normas ISO los principales elementos involucrados en la reusabilidad son:

- Paquetes de software de propósito general.
- Diseños previamente definidos (Estructuras de datos, algoritmos, etc.)
- Código anteriormente testeado y adicionalmente filtrado.
- Personal cualificado
- Especificaciones de requisitos previamente concebidas.

Tipos de reutilización:

- Oportunística

“El ingeniero de software reutiliza piezas que él sabe que se ajustan al problema” (ISO/IEC 25010, 2020).

- Sistemática:
    - Esfuerzo planificado con antelación bajo el control de la empresa.
    - Los componentes deben ser en lo posible de manera genérica de tal forma que permita su reutilización en otros contextos.
    - Los beneficios se los obtiene a futuro, invirtiendo en el presente.
    - Priorizar una componetización prioritaria a bajo nivel.

  - Imprescindible llevar un registro de los componentes a través de un sistema de control de versiones.

- Top-Down
  - Presentar componentes cohesivos unos con otros.
  - Desarrollo incremental y progresista
  - Alta inversión en etapas tempranas del desarrollo
  - Beneficios en el futuro

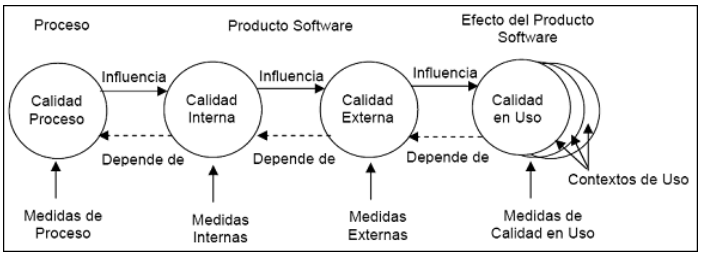
A pesar que el modelo SQuaRE no especifica un conjunto de métricas para evaluar la mantenibilidad de productos software, sí establece las características deseables de dichas mediciones. Las cuales son independientes, objetivas, y reproducibles y adicionalmente deben estar expresadas por escalas válidas y suficientemente precisas para comparaciones fiables.

“La organización que elabora el software obtiene valor agregado por que el software de alta calidad requiere un menor esfuerzo de mantenimiento, menos errores que corregir y poca asistencia al cliente. Esto permite que los ingenieros de software dediquen más tiempo a crear nuevas aplicaciones y menos a repetir trabajos mal hechos” (Pressman, 2010)

Las aplicaciones de estas normas pueden estar deliberadamente modificadas o ser creadas por su necesidad, sin embargo, se deberá proporcionar la manera en como se ha adecuado dicha métrica al modelo de calidad o ser especifico al indicar cuál será sustituida.

Figura 17

Marco Conceptual para el Modelo de Calidad



Nota: Relación existente entre las fases de calidad con sus respectivas métricas y su influencia y dependencia entre ellas.

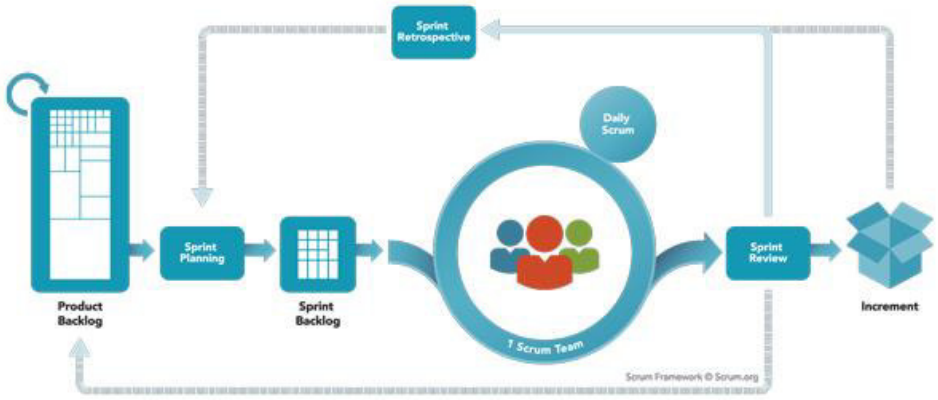
Por lo tanto, se realizará la definición de características, subcaracterísticas y atributos de calidad interna y externa, en lo que con lleva a la característica de la mantenibilidad descrita en las normas ISO/IEC 25010, describiendo las métricas a utilizarse en el estudio comparativo.

Metodología Ágil de Desarrollo

Ágil es un término utilizado para describir: “enfoques para el desarrollo de software que enfatizan la entrega incremental, la colaboración en equipo, la planificación continua y el aprendizaje continuo, en lugar de tratar de entregar todo de una vez cerca del final” (visual-paradigm, 2020).

Figura 18

Proceso de Iteración



Nota: Los tiempos de las iteraciones generalmente se ajustan a 1 o 4 semanas dependiendo de la complejidad del proyecto ilustración tomada de: (Scrum Org, 2017).

Eventos Scrum

De acuerdo con la literatura plasmada por la organización Scrum: Los eventos de Scrum tienen el objetivo de minimizar la necesidad de reuniones no definidas en Scrum además de establecer una medida que permita al equipo fomentar la comunicación y colaboración reduciendo el tiempo en reuniones extensas además de reducir los procesos restrictivos y predictivos (Schwaber & Sutherland , 2016).

Todos los eventos tienen una caja de tiempo cuando se inicia un sprint este tiene una duración fija y no se puede acortar o alargar. Los siguientes eventos pueden terminar siempre que se logre el propósito del evento, pero dentro de la caja de tiempo y asegurando el fomento de la transparencia (Scrum Org, 2017).

Los eventos son:

Sprint. Es un bloque de tiempo (time-box) de un mes o menos durante el cual se crea un incremento de producto terminado utilizable y potencialmente desplegable (Schwaber & Sutherland , 2016).

Sprint Planning. Durante la planificación del Sprint, todo el equipo Scrum colabora y discute el trabajo de alta prioridad deseado para el Sprint y define el objetivo del Sprint (Scrum Alliance, 2020).

Daily Scrum. El Equipo de Desarrollo se reúne durante 15 minutos (o menos) todos los días del Sprint para inspeccionar el progreso hacia el Objetivo del Sprint (Scrum Alliance, 2020).

Sprint Review. Las revisiones de Sprint se centran en el producto que se está desarrollando, específicamente en el incremento de producto potencialmente entregable creado durante el Sprint (Scrum Alliance, 2020).

Sprint Retrospective. El Equipo Scrum analiza lo que salió bien y las áreas de mejora en el Sprint. Hacen planes tangibles sobre cómo mejorar sus propios procesos, herramientas y relaciones. (Scrum Alliance, 2020).

Artefactos

Los artefactos tienen los objetivos de fomentar la transparencia y las oportunidades.

Estas están específicamente definidas para fomentar la transparencia de la información de tal manera que todos tengan el mismo entendimiento de lo que se está llevando a cabo a través de los artefactos (Scrum Alliance, 2020).

Los artefactos Scrum son:

Product Backlog. Lista ordenada de todo lo que se sabe que se necesita en un producto y se encuentra en constante evolución y nunca se completa. (Scrum Alliance, 2020)

Sprint Backlog. Es una lista de todo lo que el equipo se compromete a lograr en un Sprint determinado. Una vez creado, nadie puede agregar al Sprint Backlog excepto el Equipo de Desarrollo. (Scrum Alliance, 2020)

Increment. Al final de cada Sprint, el equipo debe completar un incremento de producto que sea potencialmente liberable, lo que significa que cumple con la definición acordada de terminado. (Scrum Alliance, 2020)

Beneficios de Scrum

Los principales beneficios del marco de trabajo para el desarrollo se los describe por medio de la **Tabla 1**.

Tabla 1

Beneficios de Scrum

Beneficio	Reto
Gestión Expectativas del Cliente	Lista de requisitos priorizada
Resultados anticipados (“time to market”)	Priorización de requisitos por valor y coste
Flexibilidad y adaptación	Replanificación en el inicio de cada iteración.
Retorno de inversión	Priorización de requisitos por valor.
Mitigación de riesgos	Desarrollo iterativo e incremental.
Productividad y calidad	Mejora Continua, comunicación diaria del equipo y Time Boxing.

Beneficio	Reto
Equipo Motivado	Equipo Autogestionado
Alineamiento Cliente - Equipo	Cliente y equipo trabajando en conjunto

Nota: Tabla resumida tomada del blog https://proyectosagiles.org/

Roles Involucrados

La metodología SCRUM plantea la conformación de un equipo de trabajo conocido también como SCRUM Team, por medio del cual se dará vida al producto.

Product Owner. Define el que se tiene que hacer en cuanto al desarrollo del producto, el aspecto del mismo y las características que este debería contener, a su vez es responsable por mantener actualizado la pila de acumulación del producto (Product Backlog) y se asegura que todos los miembros del Scrum Team conozcan las prioridades. (Scrum Alliance, 2020)

Scrum Master. Ayuda al equipo Scrum a desempeñarse en su nivel más alto. También protegen al equipo de distracciones tanto internas como externas. El Scrum Masters responsabiliza al equipo Scrum de sus acuerdos de trabajo, los valores de Scrum y del marco de Scrum en sí.

Team Development. El equipo de desarrollo decide cómo realizar el trabajo establecido por el propietario del producto. Los equipos de desarrollo están estructurados y facultados para organizar y gestionar su propio trabajo, mayormente conformado entre 3 y 9 personas.

Capítulo III

Análisis y Diseño de la Solución

En base a los objetivos principales que persigue el presente estudio comparativo se ha presentado el correspondiente análisis tanto de la situación actual como la propuesta.

Análisis

La responsabilidad de ofrecer productos que adopten arquitecturas de software más optimizadas tanto para en el despliegue como a través de los servicios de infraestructura, con la finalidad de incrementar los recursos en base a la demanda.

Las empresas desarrolladoras que implementan y consideran la aplicación de arquitecturas optimizadas para sus productos; marcas y empresas de nivel mundial son casos de éxito en la implementación de arquitecturas principalmente basadas en microservicios.

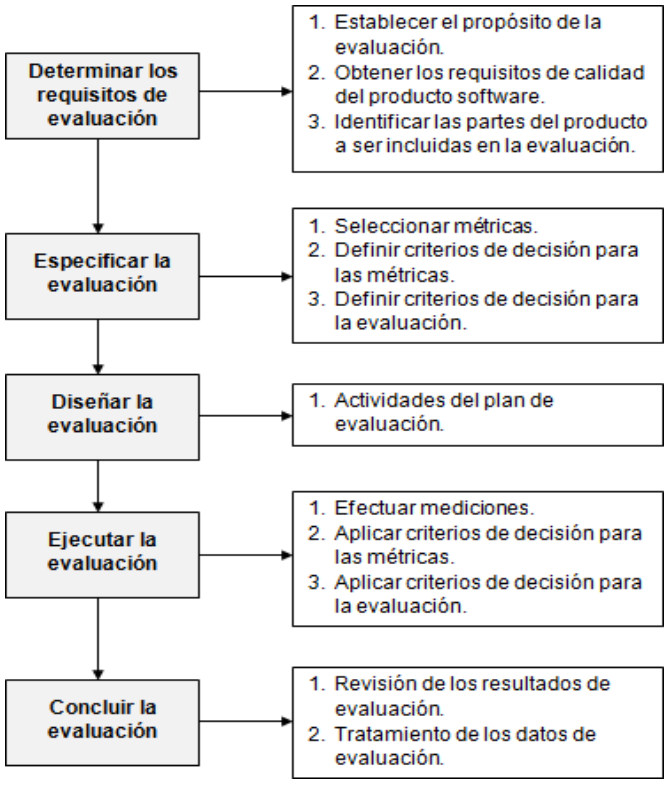
La maximización de los recursos de hardware y la alineación del software hacia los objetivos estratégicos de las empresas, así como: la elasticidad, el aislamiento de los fallos entre otras características y/o beneficios, son un factor de éxito para: Netflix, Hulu, Uber, Spotify, Gift, Amazon, etc.

Proceso de Evaluación

Determinar los procesos principales e identificar las actividades necesarias para la evaluación de la calidad, en la cual intervienen tareas, propósitos, entradas, resultados e información complementaria.

Figura 19

Proceso de Evaluación Aplicado al Producto Software



Nota: Describe el flujo por el cual se puede implementar la evaluación del producto software, esto puede elaborarse durante o posterior al proceso de desarrollo tomado de (Normas de Calidad ISO 25040, 2020)

Métricas propuestas

Las características de calidad propuestas en la norma se pueden aplicar para todos los sistemas de software, sin embargo, se debe considerar el tipo de sistema de software a evaluar, estas determinan el grado de importancia mayor o menor que otros.

Métricas de Calidad

Por medio de la **Tabla 2** se presentan las principales métricas de mantenibilidad a tomar en cuenta en el proceso de evaluación.

Tabla 2

Métricas Mantenibilidad

Subcaracterísticas	Métricas
Modularidad	Capacidad de condensación
	Acoplamiento
Reusabilidad	Ejecución de reusabilidad
Analizabilidad	Capacidad de pistas de auditoría
	Diagnóstico de funciones suficientes
Modificabilidad	Complejidad ciclomática
	Profundidad de Herencia
	Grado de localización de corrección de impacto
	Complejidad de modificación
	Índice de éxito de modificación
Testabilidad	Completitud funcional de funciones de pruebas.
	Capacidad de pruebas autónomas
	Capacidad de reinicio de pruebas

Nota: Tomado de la fuente: (Pardo Mesias, 2018)

Ponderación

Se ha establecido la **Tabla 3** correspondiente a la ponderación para establecer los parámetros comparativos y sobretodo de interés.

Tabla 3

Ponderación de Importancia

NIVEL	SÍMBOLO	PORCENTAJE
ALTO	A	80 - 100
MEDIO	B	40 - 79
BAJO	C	1 – 39
No Aplica	NA	0

Nota: Tabla la cual es de referencia para aplicar en las características del modelo de calidad.

Figura 20

Métricas Subcaracterísticas Modularidad

Métrica	Capacidad de condensación	Acoplamiento de clases
Propósito de la Métrica	Determinar numéricamente la dureza en la relación de los componentes del software	Medir el grado de acoplamiento en base a sus dependencias y la forma de registralas.
Método Aplicado	Contabilizar el número de componentes que no son afectados por los cambios de otros componentes y el número total de componentes específicos	Establecer el numero de relaciones que tiene una funcion con respecto a otras clases
Fórmula	Z= N / M Donde: N = # de artefactos no intervenidos por alteraciones externas M = Total artefactos SIEMPRE: M > 0	Z = N N = # de dependencias presentadas en un funcion vs otros artefactos
Valor deseado	0	1
Recursos utilizados	Código Fuente	Código Fuente

Nota: Tabla elaborada a partir de las normas (ISO/IEC 25010, 2020)

Figura 21

Métricas Subcaracterísticas Reusabilidad

Métrica	Ejeccion de Reusabilidad
Propósito de la Métrica	Componentes de software reutilizables
Método Aplicado	Enlistar los componentes reutilizables y los componentes de la biblioteca
Fórmula	Z= N / M Donde: N = # de componentes software reutilizables M = # total de componentes de la bilbioteca SIEMPRE: M > 0
Valor deseado	1
Recursos utilizados	Código Fuente

Nota: Información tomada de las normas (ISO/IEC 25010, 2020)

Figura 22

Métricas Subcaracterísticas Analizabilidad

Métrica	Capacidad de seguimiento de auditoría	Diagnóstico de funciones suficientes
Propósito de la Métrica	Los operarios del sistema identifacan de manera sencilla el origen del problema	Las funciones que permiten identificar problemas, muestran un detalle del origen del problema
Método Aplicado	Detellar la información que debe persistir frente a la que efecitvamete ha almacenado	Identificar las operaciones que permiten analizar la información presentada contra las que son necesarias en el correspondiente RF
Fórmula	Z= N / M N = Información evidenciada como almacenada M = Información requerida a ser almacenada SIEMPRE: M > 0	Z = N/M N = # de operaciones implementadas M = # de funciones de diagnóstico requeridas en la especificación de requerimientos SIEMPRE M > 0
Valor deseado	1	1
Recursos utilizados	Especificación de requerimientos, Código fuente, Desarrolladores, Tester	Especificación de requerimientos, Código fuente, Desarrolladores, Tester

Nota: Información elaborada tomando en consideración las normas (ISO/IEC 25010, 2020)

Figura 23

Métricas Subcaracterística Capacidad de ser Modificado

Métrica	Complejidad ciclomática	Profundidad de herencia	Grado de localización de corrección de impacto	Complejidad de modificación	Índice de éxito de modificación
Propósito de la Métrica	¿Cuál es la complejidad estructural de un código fuente?	¿Profundidad en la jerarquía de la herencia de las clases involucradas?	Grado de relacion entre un fallo y su correspondiente operación de solución	¿Con qué facilidad el desarrollador puede modificar el software para resolver problemas?	¿Hasta qué punto puede el sistema ser operado sin fallas después del mantenimiento?
Método Aplicado	Detallar los controles de decisión a traves de los operadores utilizados como condicionantes	Contar las jerarquías empleadas en una determinada función o método.	Detallar los problemas surgidos cuando se soluciona los mismos, describiendo el total efectivamente solventado	Tomar el tiempo de trabajo que le toma al desarrollador modificar y contar el número de modificaciones	Contar el número de problemas dentro de un determinado período antes de mantenimiento y contar el número de problemas en el mismo período después del mantenimiento.
Fórmula	Z = N+1 N = # de instrucciones condicionales que tiene una función	Z = N N = # de jerarquías empleadas para una determinada función.	Z = N/M N = # de errorres a ser solucionados despues de realizar una operación de intervención M = # de fallas solventadas SIEMPRE B > 0	X = N / T N = # de modificaciones T = Tiempo de trabajo que le toma al desarrollador modificar SIEMPRE T > 0	X = N/M N = # de problemas dentro de un determinado período antes de mantenimiento M = # de problemas en el mismo período después del mantenimiento SIEMPRE: M > 0
Valor deseado	1	0	0	0/T	0
Recursos utilizados	Código Fuente	Código Fuente	Requisitos, Código fuente, Desarrollador, Tester	Desarrollador	Desarrollador

Nota: Información tomada de las normas (ISO/IEC 25010, 2020)

Figura 24

Métricas Subcaracterística Capacidad de ser Probado

Métrica	Completitud funcional de funciones de pruebas	Capacidad de prueba autónoma	Capacidad de reinicio de pruebas
Propósito de la Métrica	¿Son las funciones de prueba completas y fáciles de implementar?	¿Qué tan independiente es el software al ser probado?	¿Con qué facilidad se puede llevar a cabo las pruebas nuevamente después del mantenimiento?
Método Aplicado	Enlistas las funciones de verificacion desarrolladas vs las especificas para la operación	Describir las funciones de verifcacion acopladas a entornos involucrados vs las operaciones de verificacion con sus correctos entornos	Contar el número de casos en los cuales el mantenedor puede pausar y restaurar las pruebas y contar el número de casos de pausa en la ejecución de pruebas
Fórmula	Z = N / M N = # de funciones de prueba implementadas M = # de funciones de prueba requeridas Dónde: M > 0	Z = N / M N = # de pruebas que dependen de otros sistemas. M = # total de pruebas dependientes de otros sistemas Dónde: M > 0	Z = N / M N = # de casos en los cuales el mantenedor puede pausar y restaurar las pruebas M = # de casos de pausa en la ejecución de pruebas Dónde: M > 0
Valor deseado	1	1	1
Recursos utilizados	Código fuente, Tester	Código fuente, Tester	Desarrollador, Tester

Nota: Información conformada de acuerdo a las normas (ISO/IEC 25010, 2020)

Ámbito de la Investigación

En el presente proyecto de investigación se aborda las arquitecturas monolíticas y orientadas a microservicios con la finalidad de optimizar recursos en mantenimiento, adicionalmente realizar evaluaciones conforme a métricas enfocadas en cumplir el modelo de calidad ISO 25010 en marcadas únicamente a la subcaracterísticas de la mantenibilidad.

Aplicado al software de facturación electrónica ya existente el mismo que se encuentra basado en una arquitectura monolítica y su desarrollo basado en la arquitectura basada en microservicios.

Definiciones, Acrónimos y Abreviaturas. Los términos relevantes asociados a su contexto se lo presentan a través de la siguiente lista.

De la tecnología. Términos correspondientes a las tecnologías a utilizar.

- PHP
- PostgreSQL
- RESTFul
- JSON

Del negocio. Términos pertenecientes a las reglas de negocio.

- Facturación Electrónica
- Comprobantes Electrónicos
- Emisor
- Receptor
- Contribuyente

- Firma Electrónica
- RIDE

Del sistema. Termines correspondiente a los sistemas involucrados en el desarrollo.

- Pruebas
- Integración Continua
- Entrega Continua
- BackOffice

Definición de Requisitos

Se determinan los requisitos funcionales y no funcionales de la aplicación utilizando los siguientes acrónimos:

- RE: Requerimiento Específico
- NR: Nombre del Requerimiento

Requisitos Funcionales

Los requisitos funcionales muestran una descripción, el nombre y un código asignado a cada uno de los requisitos funcionales, mismo que fueron realizados en base a las necesidades de la empresa de acuerdo a la **Tabla 4**.

Tabla 4

Requisitos Funcionales

RF	NR	DESCRIPCIÓN
RF01	Iniciar Sesión	Permitir el acceso al usuario con sus credenciales validas

RF	NR	DESCRIPCIÓN
RF02	Visualizar Menú Principal	Mostrar las opciones disponibles al usuario
RF03	Registrar una Factura	Registrar los datos de la factura con un solo ítem de detalle (servicios profesionales) y enviarla al SRI para su correspondiente registro.
RF04	Generar XML	Generar la información correspondiente a la factura en formato XML, considerando la clave de acceso y su correspondiente registro secuencial.
RF05	Firmar XML	Incluir en el formato XML la firma electrónica perteneciente al emisor de la factura electrónica.
RF06	Generar RIDE	Representación impresa de un documento electrónico.
RF07	Envió de Notificaciones	Notificar al receptor de la factura por medio de correo electrónico los datos y el documento RIDE
RF08	Cerrar Sesión	Salir de la sesión perteneciente que el usuario inicialmente ha creado.

Nota: Información tomada a partir de las historias de usuario.

Requisitos No Funcionales

Atributos no funcionales que deberá cumplir la aplicación, estos factores garantizarán que la misma tenga: rendimiento, seguridad, fiabilidad, disponibilidad y usabilidad del producto.

Rendimiento. La aplicación web garantiza un correcto desempeño de todos los procesos que se realizarán con el sistema de base de datos, estos deberán estar diseñados para que la respuesta sea rápida y no comprometa el rendimiento del software.

Seguridad. Cada usuario es responsable de contar con sus credenciales para acceder a la aplicación y deben ser válidas al momento de iniciar sesión.

Fiabilidad. El almacenamiento de la información en la base de datos deberá garantizar la integridad y correcta relación entre los datos de manera que facilite su consulta.

Disponibilidad. La aplicación será implementada para estar disponible para los usuarios, esta condición dependerá del servicio de Internet su conectividad y el navegador que se utilice.

Usabilidad. La aplicación debe contar con diseños amigables e intuitivos para el usuario, además de un diseño “Responsivo y Adaptativo” a fin de garantizar la adecuada visualización en múltiples dispositivos.

Planificación Scrum

Manteniendo como guía la metodología seleccionada para el desarrollo, se procede con la elaboración de la pila del producto (Product Back Log), la cual tiene como principal fuente de alimentación de los requerimientos funcionales descritos en el presente capítulo.

Se procede a estructurar el presente formato para la documentación de la pila de producto, el cual consta de los siguientes literales:

- Identificador correspondiente al ítem del listado de requerimientos.
- Historia de Usuario: Descripción del requerimiento.
- Tiempo estimado: Tiempo aproximado para el desarrollo.
- Prioridad: Grado de relevancia.

- Criterio de aceptación: Descripción de la funcionalidad esperada.

Product Backlog

A través de la tabla 5, se presenta la pila de producto (Product Backlog) tomando los datos de los requerimientos planteados

Tabla 5

Product Backlog

ID	HISTORIA DE USUARIO	TIEMPO ESTIMADO	PRIORIDAD	CRITERIO DE ACEPTACIÓN
1	**Como**: Usuario registrado **Quiero**: Ingresar las credenciales asignadas **Para**: Comprobar su validez	1	5	1. Presentar el formulario de inicio de sesión para el ingreso del nombre de usuario y contraseña. 2. Iniciar una sesión validando las credenciales ingresadas. 3. Si las credenciales son incorrectas mostrar nuevamente el formulario de inicio de sesión con un mensaje de error.
2	**Como**: Usuario autenticado **Quiero**: Visualizar las opciones disponibles del menú principal **Para**: Identificar las opciones disponibles	1	3	1. Mostrar un botón de emitir factura. 2. Mostrar un link de cerrar la sesión.
3	**Como**: Usuario Autenticado	4	5	1. Presentar un formulario de ingreso de información con todos los campos pertenecientes a una factura.

ID	HISTORIA DE USUARIO	TIEMPO ESTIMADO	PRIORIDAD	CRITERIO DE ACEPTACIÓN
	Quiero: Registrar los datos de una factura con un solo ítem de detalle. **Para**: Emitir una factura a un cliente **Como**: Usuario Autenticado			2. Validar la información ingresada en los campos de la factura.
4	**Quiero**: Exportar la información de una factura válida en formato XML **Para**: Validar su estructura con los archivos XSD del SRI.	4	5	1. Presentar los datos de una factura de acuerdo al formato XML del SRI. 2. Validar el formato obtenido de acuerdo al formato XSD del SRI.
5	**Como**: Usuario Autenticado **Quiero**: Enviar la información de un formato XML válido al SRI **Para**: Registrarla en la plataforma del SRI.	4	5	1. Incluir la firma electrónica perteneciente al emisor de la factura en el formato XML. 2. Enviar una factura en formato XML firmada digitalmente a través del servicio web correspondiente dispuesto por el SRI.
6	**Como**: Usuario Autenticado **Quiero**: Generar la representación impresa de una factura en formato PDF **Para**: Poder visualizar la factura en formato PDF	1	4	1. Incorporar los datos de una factura en una plantilla genérica de acuerdo al formato del SRI. 2. Presentar la factura en formato PDF.
7	**Como**: Usuario Autenticado **Quiero**: Enviarle una notificación al correo	1	4	1. Envié una notificación por correo electrónico indicando que se le ha emitido una factura electrónica.

ID	HISTORIA DE USUARIO	TIEMPO ESTIMADO	PRIORIDAD	CRITERIO DE ACEPTACIÓN
	electrónico del receptor de la factura. **Para**: Que la pueda registrar de acuerdo a lo que le establezca la ley. **Como**: Usuario Autenticado			2. Adjuntar la factura firmada electrónicamente en formato XML. 3. Adjuntar el archivo correspondiente a la representación impresa de un documento electrónico RIDE.
8	**Quiero**: Cerrar la sesión vigente **Para**: Asegurar el uso de las credenciales asignadas.	1	2	1. Cerrar una sesión activa, y presentar nuevamente la página de inicio sesión.

Nota: Información tomada a partir de las historias de usuario.

La información presentada en la tabla anterior puede ser optimizada para facilitar las iteraciones por lo cual se presenta los siguientes valores para esta actividad:

- Tiempo Estimado: 1 a 5 días.
- Prioridad: 1 a 5 siendo 5 la de mayor importancia.

Sprint Planning

La actividad de planificación (Sprint Planning), es en donde los miembros del equipo eligen una o varias historias de usuario del Product Backlog con la finalidad de ser desarrolladas en una iteración. Por lo que este listado de tareas a cumplir es denominado como pila de tareas o Sprint Backlog.

Iteración 1. Historias de usuario seleccionadas para la iteración número 1.

Tabla 6

Historias de Usuario para la Iteración 1

ID	HISTORIA DE USUARIO	TIEMPO ESTIMADO	PRIORIDAD	CRITERIO DE ACEPTACIÓN
1	**Como**: Usuario registrado **Quiero**: Ingresar las credenciales asignadas **Para**: Comprobar su validez	1	5	1. Presentar el formulario de inicio de sesión para el ingreso del nombre de usuario y contraseña. 2. Iniciar una sesión validando las credenciales ingresadas. 3. Si las credenciales son incorrectas mostrar nuevamente el formulario de inicio de sesión con un mensaje de error.
2	**Como**: Usuario autenticado **Quiero**: Visualizar las opciones disponibles del menú principal **Para**: Identificar las opciones disponibles	1	3	1. Mostrar un botón de emitir factura. 2. Mostrar un link de cerrar la sesión.
3	**Como**: Usuario Autenticado **Quiero**: Registrar los datos de una factura con un solo ítem de detalle. **Para**: Emitir una factura a un cliente	4	5	1. Presentar un formulario de ingreso de información con todos los campos pertenecientes a una factura. 2. Validar la información ingresada en los campos de la factura.

ID	HISTORIA DE USUARIO	TIEMPO ESTIMADO	PRIORIDAD	CRITERIO DE ACEPTACIÓN
8	**Como**: Usuario Autenticado **Quiero**: Cerrar la sesión vigente **Para**: Asegurar el uso de las credenciales asignadas.	1	2	1. Cerrar una sesión activa, y presentar nuevamente la página de inicio sesión.

Nota: Información tomada a partir de las historias de usuario.

Sprint Backlog 1

Una vez identificadas las tareas para la iteración 1 procedemos a registrar los responsables de realizarlas para cumplir con las historias de usuario.

Tabla 7

Sprint Back Log 1

ID	Tarea	Responsable	N° Product Back Log	27 jul - 31 jul	3 ago – 7 ago
1	Implementación del Modelo de Datos Autenticación	MT	1	X	
2	Diseñar el formulario de Inicio de sesión	MT	1	X	

ID	Tarea	Responsable	N° Product Back Log	27 jul - 31 jul	3 ago – 7 ago
3	Aplicar reglas de validación para el campo del nombre de usuario y contraseña	MT	1	X	
4	Iniciar una sesión a través de credenciales válidas.	MT	1	X	
5	Comprobar sesiones activas y aplicar las políticas de seguridad correspondientes.	MT	1	X	
6	Enviar mensaje de error al formulario de inicio de sesión si las credenciales son incorrectas.	MT	1	X	
7	Diseño del menú principal de la aplicación	MT	2	X	
8	Ingresar con una sesión activa al menú principal	MT	2	X	
9	Diseñar el formulario de registro factura	MT	3		X
10	Acceder al formulario de registro de factura.	MT	3		X
11	Validar campos del formulario	MT	3		X

ID	Tarea	Responsable	N° Product Back Log	27 jul - 31 jul	3 ago – 7 ago
12	Cerrar una sesión activa por medio del icono correspondiente.	MT	8		X

Nota: Información tomada a partir de las historias de usuario.

Sprint Review 1

Para el artefacto Sprint Review se procede a transparentar los resultados del Sprint Back Log 1, detallando su avance.

Tabla 8

Sprint Review 1

ID	Tarea	Iteración	Avance	Fecha Entrega
1	Implementación del Modelo de Datos Autenticación	1	100%	27-JUL-2020
2	Diseñar el formulario de Inicio de sesión	1	100%	27-JUL-2020
3	Aplicar reglas de validación para el campo del nombre de usuario y contraseña	1	100%	28-JUL-2020
4	Iniciar una sesión a través de credenciales válidas.	1	100%	28-JUL-2020
5	Comprobar sesiones activas y aplicar las políticas de seguridad correspondientes.	1	100%	29-JUL-2020

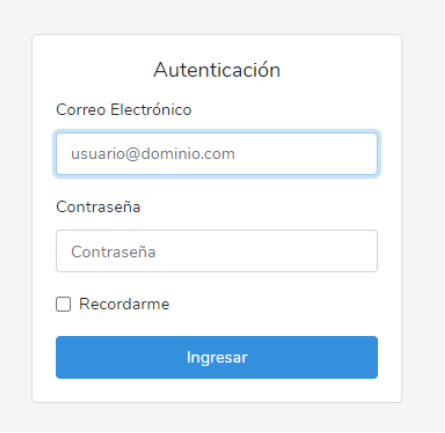
ID	Tarea	Iteración	Avance	Fecha Entrega
6	Enviar mensaje de error al formulario de inicio de sesión si las credenciales son incorrectas.	1	100%	29-JUL-2020
7	Diseño del menú principal de la aplicación	1	100%	30-JUL-2020
8	Ingresar con una sesión activa al menú principal	1	100%	31-JUL-2020
9	Diseñar el formulario de registro factura	1	100%	04-AGO2020
10	Acceder al formulario de registro de factura.	1	100%	04-AGO-2020
11	Validar campos del formulario	1	100%	06-AGO-2020
12	Cerrar una sesión activa por medio del icono correspondiente.	1	100%	07-AGO-2020

Nota: Información tomada a partir de las historias de usuario.

Producto Entregable Demo 1

Figura 25

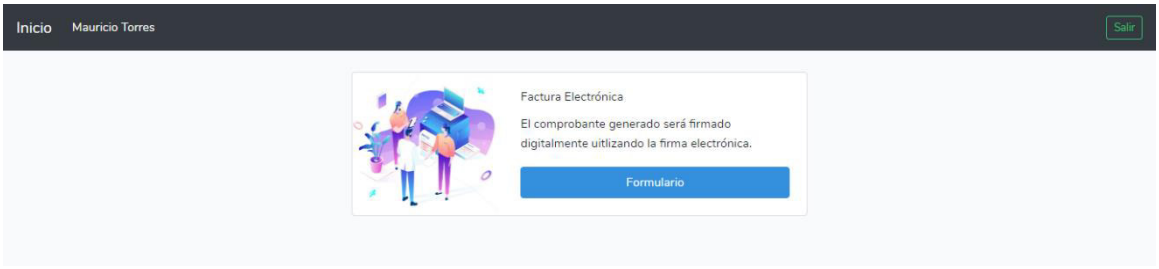
Pantalla de Inicio de Sesión Usuario Registrado



Nota: Los usuarios se encuentran previamente registrados en la base de datos.

Figura 26

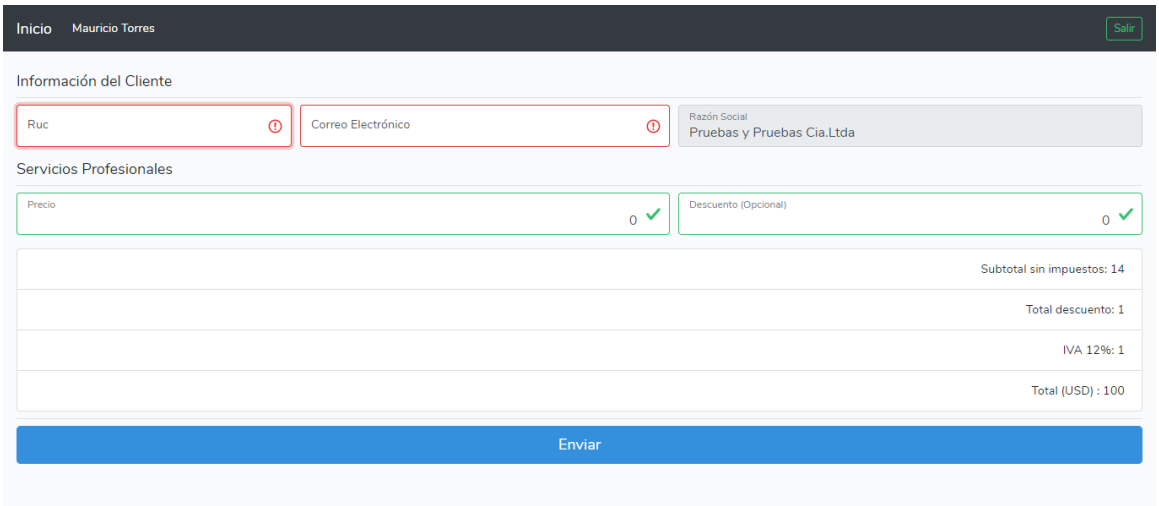
Menú Principal



Nota: En el menú principal se encuentran las opciones disponibles para el usuario identificado.

Figura 27

Formulario de Registro de Factura



Nota: Registro de la información correspondiente a una factura.

Iteración 2. Historias de usuario seleccionadas para la iteración número 1.

Tabla 9

Historias de Usuario la Iteración 2

ID	HISTORIA DE USUARIO	TIEMPO ESTIMADO	PRIORIDAD	CRITERIO DE ACEPTACIÓN
4	**Como**: Usuario Autenticado **Quiero**: Exportar la información de una factura válida en formato XML **Para**: Validar su estructura con los archivos XSD del SRI.	4	5	1. Presentar los datos de una factura de acuerdo al formato XML del SRI. 2. Validar el formato obtenido de acuerdo al formato XSD del SRI.
5	**Como**: Usuario Autenticado **Quiero**: Enviar la información de un formato XML válido al SRI **Para**: Registrarla en la plataforma del SRI.	4	5	1. Incluir la firma electrónica perteneciente al emisor de la factura en el formato XML. 2. Enviar una factura en formato XML firmada digitalmente a través del servicio web correspondiente dispuesto por el SRI.
6	**Como**: Usuario Autenticado **Quiero**: Generar la representación impresa de una factura en formato PDF **Para**: Poder visualizar la factura en formato PDF	1	4	1. Incorporar los datos de una factura en una plantilla genérica de acuerdo al formato del SRI. 2. Presentar la factura en formato PDF.
7	**Como**: Usuario Autenticado **Quiero**: Enviarle una notificación al correo electrónico del receptor de la factura.	1	4	1. Enviar una notificación por correo electrónico indicando que se le ha emitido una factura electrónica. 2. Adjuntar la factura firmada electrónicamente en formato XML.

ID	HISTORIA DE USUARIO	TIEMPO ESTIMADO	PRIORIDAD	CRITERIO DE ACEPTACIÓN
	Para: Cumplir con lo establecido en la ley tributaria.			3. Adjuntar el archivo correspondiente a la representación impresa de un documento electrónico RIDE

Nota: Información tomada a partir de las historias de usuario.

Sprint Backlog 2

Una vez identificadas las tareas para la iteración 2 procedemos a registrar los responsables de realizarlas para cumplir con las historias de usuario.

Tabla 10

Sprint Back Log 2

ID	Tarea	Responsable	N° Product Back Log	10 ago - 14 ago	17 ago – 21 ago
1	Estructurar la información en base al formato XML	MT	4	X	
2	Validar la estructura del archivo contra el XSD del SRI	MT	4	X	
3	Presentar los datos de una factura en base al XML factura versión 1.0.0	MT	4	X	

ID	Tarea	Responsable	N° Product Back Log	10 ago - 14 ago	17 ago – 21 ago
4	Implementación del algoritmo de módulo 11 para la clave de acceso	MT	4	X	
5	Implementar firmador de documentos XML conforme a la ficha técnica del SRI	MT	5	X	
6	Validar la estructura de la firma de acuerdo al algoritmo XADES-BES	MT	5	X	
7	Incluir la firma al comprobante factura.	MT	5	X	
8	Validación de envió al entorno de pruebas del SRI	MT	5	X	
9	Diseño de la plantilla HTML en formato RIDE en base a la ficha técnica del SRI	MT	6		X
10	Incorporar los datos de la factura generada de acuerdo a la plantilla RIDE	MT	6		X

ID	Tarea	Responsable	N° Product Back Log	10 ago - 14 ago	17 ago – 21 ago
11	Presentación de la información en el formato RIDE	MT	7		X
12	Diseño de plantilla HTML para él envió de la notificación.	MT	7		X
13	Envió de la notificación por correo electrónico al receptor de la factura.	MT	7		X
14	Adjuntar al envió de la notificación el archivo XML autorizado por el SRI y el RIDE	MT	7		X

Nota: Tabla elaborada a partir de las historias de usuario.

Sprint Review 2

Para el artefacto Sprint Review se procede a transparentar los resultados del Sprint Back Log 2, detallando su avance.

Tabla 11

Sprint Review 2

ID	Tarea	Iteración	Avance	Fecha Entrega
1	Estructurar la información en base al formato XML	2	100%	11-AGO-2020

ID	Tarea	Iteración	Avance	Fecha Entrega
2	Validar la estructura del archivo contra el XSD del SRI	2	100%	12-AGO-2020
3	Presentar los datos de una factura en base al XML factura versión 1.0.0	2	100%	13-AGO-2020
4	Implementación del algoritmo de módulo 11 para la clave de acceso	2	100%	14-AGO-2020
5	Implementar firmador de documentos XML conforme a la ficha técnica del SRI	2	100%	15-AGO-2020
6	Validar la estructura de la firma de acuerdo al algoritmo XADES-BES	2	100%	17-AGO-2020
7	Incluir la firma al comprobante factura.	2	100%	18-AGO-2020
8	Validación de envió al entorno de pruebas del SRI	2	100%	19-AGO-2020
9	Diseño de la plantilla HTML en formato RIDE en base a la ficha técnica del SRI	2	100%	20-AGO2020
10	Incorporar los datos de la factura generada de acuerdo a la plantilla RIDE	2	100%	20-AGO-2020
11	Presentación de la información en el formato RIDE	2	100%	21-AGO-2020
12	Diseño de plantilla HTML para él envió de la notificación.	2	100%	21-AGO-2020
13	Envió de la notificación por correo electrónico al receptor de la factura.	2	100%	24-AGO-2020

ID	Tarea	Iteración	Avance	Fecha Entrega
14	Adjuntar al envió de la notificación el archivo XML autorizado por el SRI y el RIDE	2	100%	24-AGO-2020

Nota: Información estructurada a partir de las historias de usuario

Producto Entregable 2

Figura 28

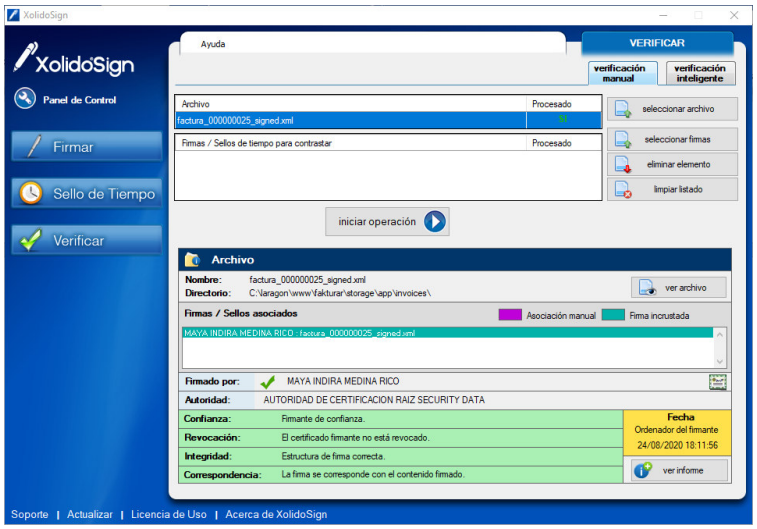
Estructura Archivo XML

```
<?xml version="1.0" encoding="UTF-8"?>
<factura id="comprobante" version="1.0.0">
   <infoTributaria>
      <ambiente>1</ambiente>
      <tipoEmision>1</tipoEmision>
      <razonSocial>SERVICIO DE RENTAS INTERNAS</razonSocial>
      <nombreComercial>LE HACE BIEN AL PAIS</nombreComercial>
      <ruc>1760013210001</ruc>
      <claveAcceso>0503201201176001321000110010030009900641234567814</claveAcceso>
      <codDoc>01</codDoc>
      <estab>001</estab>
      <ptoEmi>003</ptoEmi>
      <secuencial>000990064</secuencial>
      <dirMatriz>AMAZONAS Y ROCA</dirMatriz>
   </infoTributaria>
   <infoFactura>
      <fechaEmision>05/03/2012</fechaEmision>
      <dirEstablecimiento>SALINAS Y SANTIAGO</dirEstablecimiento>
      <contribuyenteEspecial>12345</contribuyenteEspecial>
      <obligadoContabilidad>SI</obligadoContabilidad>
      <tipoIdentificacionComprador>05</tipoIdentificacionComprador>
      <razonSocialComprador>EGUIGUREN PENARRETA GABRIEL FERNANDO</razonSocialComprador>
      <identificacionComprador>1103029144</identificacionComprador>
      <totalSinImpuestos>100.00</totalSinImpuestos>
      <totalDescuento>0.00</totalDescuento>
      <totalConImpuestos>
         <totalImpuesto>
            <codigo>2</codigo>
            <codigoPorcentaje>2</codigoPorcentaje>
            <baseImponible>100.00</baseImponible>
            <valor>12.00</valor>
         </totalImpuesto>
      </totalConImpuestos>
      <propina>0.00</propina>
      <importeTotal>112.00</importeTotal>
```

Nota: Estructura de una factura en formato XML fuente ficha técnica SRI. (Servicio de Rentas Internas, 2020)

Figura 29

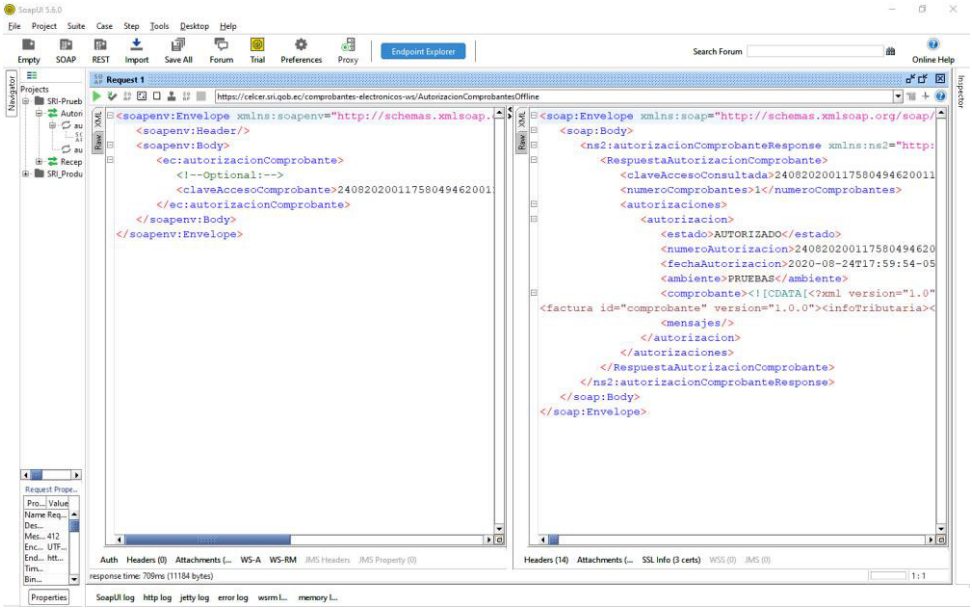
Validación Firma Digital



Nota: Herramienta gratuita de validación de certificado de firma digital. (Xolido, 2020)

Figura 30

Validación de la Factura Electrónica



Nota: Herramienta SoapUI permite generar peticiones a los servicios Web del SRI (SoapUI, 2020)

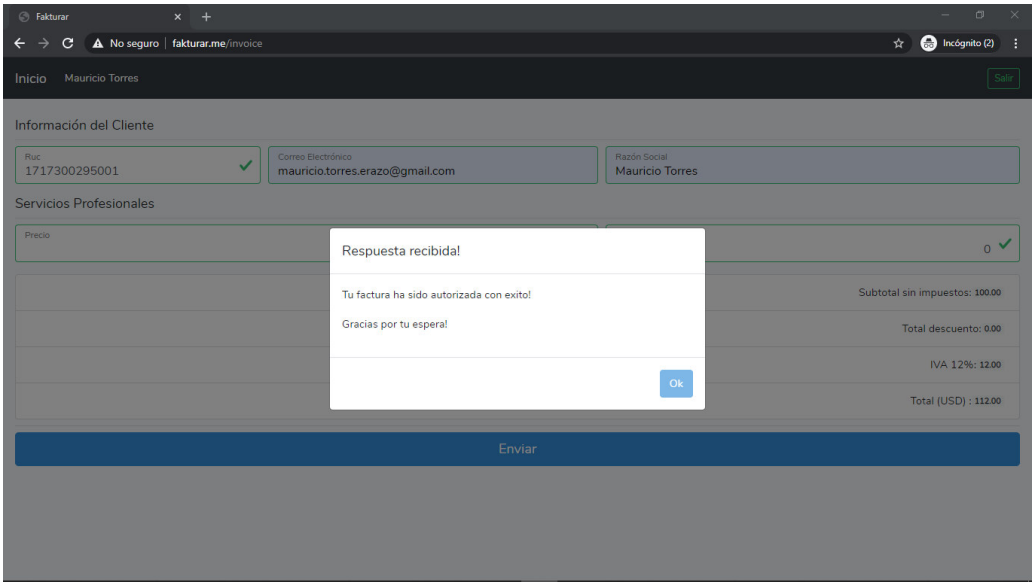
Figura 31

Notificación de Correo Electrónico

Nota: Correo electrónico enviado al beneficiario del producto o servicio.

Figura 32

Envió de Factura Electrónica



Nota: Factura enviada satisfactoriamente al SRI.

Capítulo IV

Estudio Comparativo

De acuerdo a la obtención de las métricas implementadas a las arquitecturas tanto monolíticas como basadas en microservicios aplicadas al caso de estudio en consideración podemos evaluar los siguientes resultados.

Por lo tanto, se presente la siguiente tabla con las ponderaciones en las subcaracterísticas para la mantenibilidad tanto para el sistema monolítico como para los dos principales microservicios.

Análisis del Producto Software

El producto software como objeto de estudio de comparación entre las arquitecturas monolíticas y basadas en microservicios es una aplicación de tipo web para la facturación electrónica.

Principales Componentes

Los principales componentes software para el objeto del estudio comparativo han sido tomados de acuerdo al contexto de la aplicación de tipo web facturación electrónica, los cuales se los presenta a través de la siguiente tabla:

Tabla 12

Principales Componentes del Software

N.	Nombre	Contexto	Descripción
1	Firmador	Negocio	Permite aplicar la firma electrónica basado en el algoritmo XADES-BES

N.	Nombre	Contexto	Descripción
2	Notificador	Negocio	Permite enviar notificaciones a través de correo electrónico.

Nota: Componentes de software a ser evaluados.

Herramientas de Evaluación de Código Fuente

Las herramientas seleccionadas son de software libre y su utilización está disponible exclusivamente para el lenguaje de programación PHP, a continuación, se detallan sus características.

Tabla 13

Herramientas de Evaluación de Código

N.	Nombre	Descripción	Sitio Web
1	PHP Mess Detector	Extensión de composer para la calidad de código	https://phpmd.org/
2	PHP Metrics	Permite obtener métricas del proyecto software	https://phpmetrics.org/
3	PHP Loc	Permite obtener estadísticas de la calidad del código fuente	https://github.com/sebastianbergmann/phploc

Nota: Herramientas disponibles para la calidad del código fuente.

Evaluación del Producto Software

Para el proceso de evaluación se aplica lo detallado en los capítulos 2 y 3 en lo que concierne a las métricas pertenecientes a las subcaracterísticas determinadas por el enfoque de mantenibilidad de la norma ISO25000.

Figura 33

Componente de Notificaciones para la Arquitectura Monolítica

ARQUITECTURA MONOLÍTICA
NOMBRE DEL COMPONENTE: NOTIFICACIONES
DESCRIPCION: Permite enviar notificaciones de correo electrónico a un destinario adjutando archivos

N.	NOMBRE DE LA CLASE	DESCRIPCIÓN	FRAMEWORK	DOMINIO DE LA SOLUCIÓN	PROFUNDIDAD DE HERENCIA	HEREDA DE	REUTILIZABLE
1	Router	Enruta una petición	SI	NO	1	AbstractRouter	SI
2	Request	Abstrae una solicitud	SI	NO	1	StandarRequest	SI
3	Receiver	Destinatario de una notificación	NO	SI	0	NA	NO
4	IvoiceSent	Notificación	SI	SI	0	NA	SI
5	Notification	Clase base para el envio de una notificación	SI	SI	0	NA	SI
6	Storage	Implementacion del Sistema de archivos	SI	SI	1	FlySystem	SI
7	InvoiceReady	Abstrae el objeto Mail	NO	SI	1	Mailable	SI
8	View	Abstrae un objeto vista que sera utilizado como plantilla para el envio del correo	SI	SI	1	View	SI
9	Response	Envia una respuesta a un plantilla HTML	SI	SI	1	BaseResponse	SI

Nota: Principales clases que intervienen para enviar una notificación de correo electrónico en la arquitectura monolítica.

Figura 34

Notificaciones para la Arquitectura Basado en Microservicios

ARQUITECTURA: MICROSERVICIOS
NOMBRE DEL COMPONENTE: NOTIFICACIONES
DESCRIPCION: Permite enviar notificaciones de correo electrónico a un destinario adjutando archivos

N.	NOMBRE DE LA CLASE	DESCRIPCIÓN	FRAMEWORK	DOMINIO DE LA SOLUCIÓN	PROFUNDIDAD DE HERENCIA	HEREDA DE	REUTILIZABLE
1	Router	Enruta una petición	SI	NO	1	AbstractRouter	SI
2	Request	Abstrae una solicitud	SI	NO	1	StandarRequest	SI
3	Receiver	Destinatario de una notificación	NO	SI	0	NA	NO
4	IvoiceSent	Notificación	SI	SI	0	NA	SI
5	Notification	Clase base para el envio de una notificación	SI	SI	0	NA	SI
6	Storage	Implementacion del Sistema de archivos	SI	SI	1	FlySystem	SI
7	InvoiceReady	Abstrae el objeto Mail	NO	SI	1	Mailable	SI
8	View	Abstrae un objeto vista que sera utilizado como plantilla para el envio del correo electrónico	SI	SI	1	View	SI
9	Response	Envia una respuesta a una solicitud en formato json	SI	SI	1	BaseResponse	SI

Nota: Principales clases que intervienen para enviar una notificación de correo electrónico en la arquitectura basada en microservicios.

Figura 35

Notificaciones para la Arquitectura Basado en Microservicios

ARQUITECTURA: MICROSERVICIOS
NOMBRE DEL COMPONENTE: NOTIFICACIONES
DESCRIPCION: Permite enviar notificaciones de correo electrónico a un destinario adjutando archivos

N.	NOMBRE DE LA CLASE	DESCRIPCIÓN	FRAMEWORK	DOMINIO DE LA SOLUCIÓN	PROFUNDIDAD DE HERENCIA	HEREDA DE	REUTILIZABLE
1	Router	Enruta una petición	SI	NO	1	AbstractRouter	SI
2	Request	Abstrae una solicitud	SI	NO	1	StandarRequest	SI
3	Receiver	Destinatario de una notificación	NO	SI	0	NA	NO
4	IvoiceSent	Notificación	SI	SI	0	NA	SI
5	Notification	Clase base para el envio de una notificación	SI	SI	0	NA	SI
6	Storage	Implementacion del Sistema de archivos	SI	SI	1	FlySystem	SI
7	InvoiceReady	Abstrae el objeto Mail	NO	SI	1	Mailable	SI
8	View	Abstrae un objeto vista que sera utilizado como plantilla para el envio del correo	SI	SI	1	View	SI
9	Response	Envia una respuesta a una solicitud en formato json	SI	SI	1	BaseResponse	SI

Nota: Principales clases que intervienen para enviar una notificación por correo electrónico en la arquitectura monolítica.

Figura 36

Componente Firmador de Documentos para la Arquitectura Monolítica

ARQUITECTURA MONOLÍTICA
NOMBRE DEL COMPONENTE: FIRMADOR
DESCRIPCIÓN: Permite firmar electronicamente con el algoritmo xades-bes un archivo xml utilizando un certificado de firma digital

N.	NOMBRE DE LA CLASE	DESCRIPCIÓN	FRAMEWORK	DOMINIO DE LA SOLUCIÓN	PROFUNDIDAD DE HERENCIA	HEREDA DE	REUTILIZABLE
1	Router	Enruta una petición	SI	NO	1	AbstractRouter	SI
2	Request	Abstrae una solicitud	SI	NO	1	StandarRequest	SI
3	DOMDocument	Representa un documento HTML o XML en su totalidad;	NO	SI	0	NA	SI
4	File	Clas de utilidad para trabajar con archivos	SI	SI	1	SplFileInfo	SI
5	Signer	Firma un archivo xml	SI	SI	0	NA	SI
6	XMLSecurityDSig	Implementacion de firma digital para archivos web	NO	SI	0	NA	SI
7	XMLSecurityKey	Clase de utilidad para trabjar con certificados de firma	NO	SI	0	NA	SI
8	Response	Envia una respuesta a un plantilla HTML	SI	SI	1	BaseResponse	NO

Nota: Principales clases que intervienen para firmar un documento XML (Extensible Markup Language) en la arquitectura monolítica.

Figura 37

Firmador de Documentos para la Arquitectura Basada en Microservicios

ARQUITECTURA: MICROSERVICIOS
NOMBRE DEL COMPONENTE: FIRMADOR
DESCRIPCIÓN: Permite firmar electronicamente con el algoritmo xades-bes un archivo xml utilizando un certificado de firma digital

N.	NOMBRE DE LA CLASE	DESCRIPCIÓN	FRAMEWORK	DOMINIO DE LA SOLUCIÓN	PROFUNDIDAD DE HERENCIA	HEREDA DE	REUTILIZABLE
1	Router	Enruta una petición	SI	NO	1	AbstractRouter	SI
2	Request	Abstrae una solicitud	SI	NO	1	StandarRequest	SI
3	DOMDocument	Representa un documento HTML o XML en su totalidad;	NO	SI	0	NA	SI
4	File	Clas de utilidad para trabajar con archivos	SI	SI	1	SplFileInfo	SI
5	Signer	Firma un archivo xml	SI	SI	0	NA	SI
6	XMLSecurityDSig	Implementacion de firma digital para archivos web	NO	SI	0	NA	SI
7	XMLSecurityKey	Clase de utilidad para trabjar con certificados de firma digital	NO	SI	0	NA	SI
8	Response	Envia una respuesta a una solicitud en formato json	SI	SI	1	BaseResponse	SI

Nota: Principales clases que intervienen para firmar un documento XML (Extensible Markup Language) en la arquitectura basada en microservicios

Ponderación de Porcentajes

Tabla 14

Ponderación en las Subcaracterísticas

Subcaracterística	Nivel de importancia	Ponderación	Modo de ponderación
Modularidad	A	30%	Se realiza una ponderación del 30% debido a la importancia entre los componentes fundamentales.
Reusabilidad	M	15%	Se propone una ponderación del 15% debido a la reutilización de componentes de software en otros proyectos.
Capacidad de Análisis	A	15%	Se propone una ponderación del 15% debido a los recursos que se invierten en auditorias de sistema.
Capacidad de ser Modificado	A	35%	Se realiza una ponderación del 35% debido a las nuevas necesidades de funcionalidad de los clientes.
Capacidad de ser Probado	B	5%	Se toma el valor del 5% debido a la inexperiencia de los desarrolladores al trabajar con herramientas pruebas

Nota: Valores a ser ponderados en las métricas seleccionadas.

Aplicación de Métricas

A continuación, se presentan el resultado de las métricas seleccionadas para cada una de las arquitecturas, así como la evaluación de los componentes que intervienen en el proceso.

Arquitectura Monolítica

Componente Firmador

Figura 38

Métricas Aplicadas al Componente Firmador

N.	Métrica	Resultados
1	Capacidad de condensación	0,88
2	Acoplamiento de Clases	1
3	Ejecución de Reusabilidad	0,88
4	Capacidad de seguimiento de auditoría	1
5	Diagnóstico de funciones suficientes	0,6
6	Complejidad ciclomática	2
7	Profundidad de herencia	2
8	Grado de localización de corrección de impacto	0
9	Complejidad de modificación	3/95min
10	Índice de éxito de modificación	1
11	Completitud funcional de funciones de pruebas	1
12	Capacidad de prueba autónoma	1
13	Capacidad de reinicio de pruebas	1

Nota: Resultados de la aplicación de las métricas al componente firmador (ISO/IEC 25010, 2020)

Componente Notificaciones

Figura 39

Métricas Aplicadas al Componente Notificaciones

N.	Métrica	Resultados
1	Capacidad de condensación	0,78
2	Acoplamiento de Clases	3
3	Ejecución de Reusabilidad	0,88
4	Capacidad de seguimiento de auditoría	1
5	Diagnóstico de funciones suficientes	0,2
6	Complejidad ciclomática	3
7	Profundidad de herencia	1
8	Grado de localización de corrección de impacto	0
9	Complejidad de modificación	2/135min
10	Índice de éxito de modificación	0,5
11	Completitud funcional de funciones de pruebas	1
12	Capacidad de prueba autónoma	1
13	Capacidad de reinicio de pruebas	1

Nota: Resultados de la aplicación de las métricas al componente notificaciones (ISO/IEC 25010, 2020)

Arquitectura Basada en Microservicios

Componente Firmador

Figura 40

Métricas Firmador

N.	Métrica	Resultados
1	Capacidad de condensación	0,63
2	Acoplamiento de Clases	1
3	Ejecución de Reusabilidad	1
4	Capacidad de seguimiento de auditoría	1
5	Diagnóstico de funciones suficientes	0,8
6	Complejidad ciclomática	2
7	Profundidad de herencia	1
8	Grado de localización de corrección de impacto	0
9	Complejidad de modificación	3/60min
10	Índice de éxito de modificación	0
11	Completitud funcional de funciones de pruebas	1
12	Capacidad de prueba autónoma	1
13	Capacidad de reinicio de pruebas	1

Nota: Resultados de la aplicación de las métricas al componente firmador (ISO/IEC 25010, 2020)

Componente Notificaciones

Figura 41

Métricas Notificaciones

N.	Métrica	Resultados
1	Capacidad de condensación	0,56
2	Acoplamiento de Clases	3
3	Ejecución de Reusabilidad	1
4	Capacidad de seguimiento de auditoría	1
5	Diagnóstico de funciones suficientes	0,8
6	Complejidad ciclomática	2
7	Profundidad de herencia	1
8	Grado de localización de corrección de impacto	0
9	Complejidad de modificación	2/45min
10	Índice de éxito de modificación	1
11	Completitud funcional de funciones de pruebas	1
12	Capacidad de prueba autónoma	1
13	Capacidad de reinicio de pruebas	1

Nota: Resultados de la aplicación de las métricas al componente notificaciones (ISO/IEC 25010, 2020)

Figura 42

Métricas Aplicadas a la Arquitectura de Monolítica

SUBCARACTERÍSTICA	MÉTRICA	FÓRMULA	VALOR DESEADO (UMBRAL)	APLICA	VALOR OBTENIDO	PONDERACIÓN / 10	VALOR PACIAL TOTAL / 10	NIVEL DE IMPORTANCIA	PORCENTAJE DE IMPORTANCIA	VALOR FINAL	CALIDAD DEL SISTEMA / 10
Modularidad	Capacidad de Condensación	X = A / B Donde B>0	0 <= x <= 1	SI	0,83	10	6,65	A	30%	20%	5,75
	Acoplamiento de clases	X = A	1 <= X <= 4	SI	2	10					
Reusabilidad	Ejecución de Reusabilidad	X = A / B Donde B>0	0 <= X <= 1	SI	0,88	10	8,8	M	15%	13%	
Capacidad de Análisis	Capacidad de seguimiento de auditoría	X = A / B Donde B>0	0 <= X <= 1	SI	1	10	5,5	M	15%	8%	
	Diagnostico de funciones suficientes	X = A / B Donde B>0	0 <= X <= 1	NO	0,1	10					
Capacidad de ser modificado	Complejidad ciclomática	X = A + 1	1 <= X <= 15	SI	2,5	5	3,17	A	35%	11,08%	
	Profundidad de herencia	X = A	1 <= X <= 4	SI	1,5	5					
	Grado de localizacion de corrección de impacto	X = A / B Donde B>0	0 <= X <= 1	NO	0	10					
	Complejidad de modificación	X = A / T	Mejor de los casos 1/60min; Peor de los casos 0/60min	SI	0,02	10					
	Indice de éxito de modificación	X = A / B Donde B>0	0 <= X <= 1	NO	0,75	10					
Capacidad de ser probado	Completitud funcional de funciones de pruebas	X = A / B Donde B>0	0 <= X <= 1	NO	1	10	10	B	5%	5%	
	Capacidad de prueba autonoma	X = A / B Donde B>0	0 <= X <= 1	NO	1	10					
	Capacidad de reinicio de pruebas	X = A / B Donde B>0	0 <= X <= 1	NO	1	10					

Nota: Cuadro elaborado en base a las métricas obtenidas Anexo 1

Figura 43

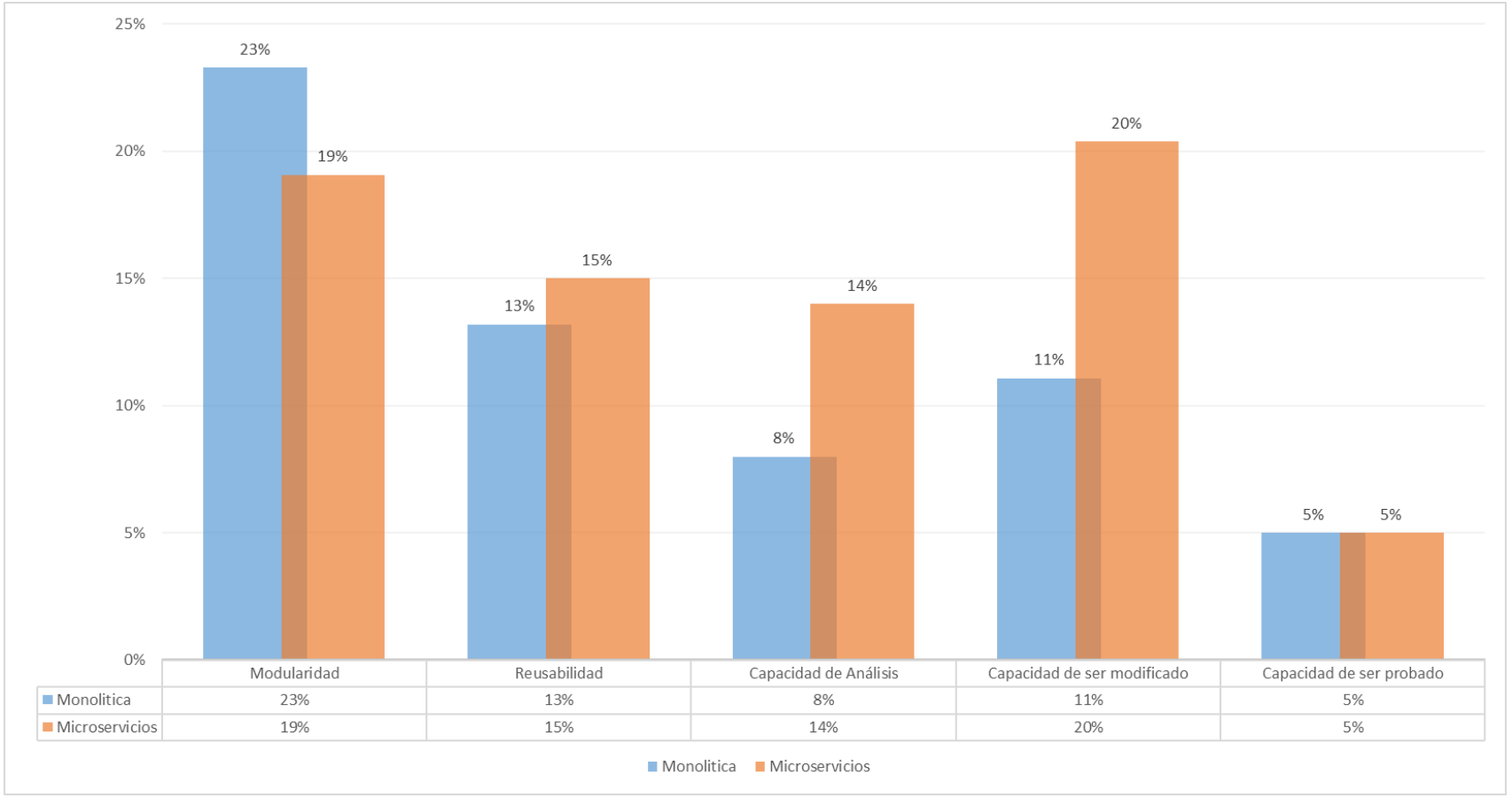
Métricas Aplicadas a la Arquitectura de Microservicios

SUBCARACTERÍSTICA	MÉTRICA	FÓRMULA	VALOR DESEADO (UMBRAL)	APLICA	VALOR OBTENIDO	PONDERACIÓN / 10	VALOR PACIAL TOTAL / 10	NIVEL DE IMPORTANCIA	PORCENTAJE DE IMPORTANCIA	VALOR FINAL	CALIDAD DEL SISTEMA / 10
Modularidad	Capacidad de Condensación	X = A / B Donde B>0	0 <= x <= 1	SI	0,59	10	5,45	A	30%	16%	7,02
	Acoplamiento de clases	X = A	1 <= X <= 4	SI	2	10					
Reusabilidad	Ejecución de Reusabilidad	X = A / B Donde B>0	0 <= X <= 1	SI	1	10	10	M	15%	15%	
Capacidad de Análisis	Capacidad de seguimiento de auditoría	X = A / B Donde B>0	0 <= X <= 1	SI	1	10	9	M	15%	14%	
	Diagnostico de funciones suficientes	X = A / B Donde B>0	0 <= X <= 1	NO	0,8	10					
Capacidad de ser modificado	Complejidad ciclomática	X = A + 1	1 <= X <= 15	SI	2	5	5,83	A	35%	20,39%	
	Produndidad de herencia	X = A	1 <= X <= 4	SI	1	5					
	Grado de localizacion de corrección de impacto	X = A / B Donde B>0	0 <= X <= 1	NO	0	10					
	Complejidad de modificación	X = A / T	Mejor de los casos 1/60min; Peor de los casos 0/60min	SI	0,04	10					
	Indice de éxito de modificación	X = A / B Donde B>0	0 <= X <= 1	NO	0	10					
Capacidad de ser probado	Completitud funcional de funciones de pruebas	X = A / B Donde B>0	0 <= X <= 1	NO	1	10	10	B	5%	5%	
	Capacidad de prueba autonoma	X = A / B Donde B>0	0 <= X <= 1	NO	1	10					
	Capacidad de reinicio de pruebas	X = A / B Donde B>0	0 <= X <= 1	NO	1	10					

Nota: Cuadro elaborado en base a las métricas obtenidas Anexo 1

Figura 44

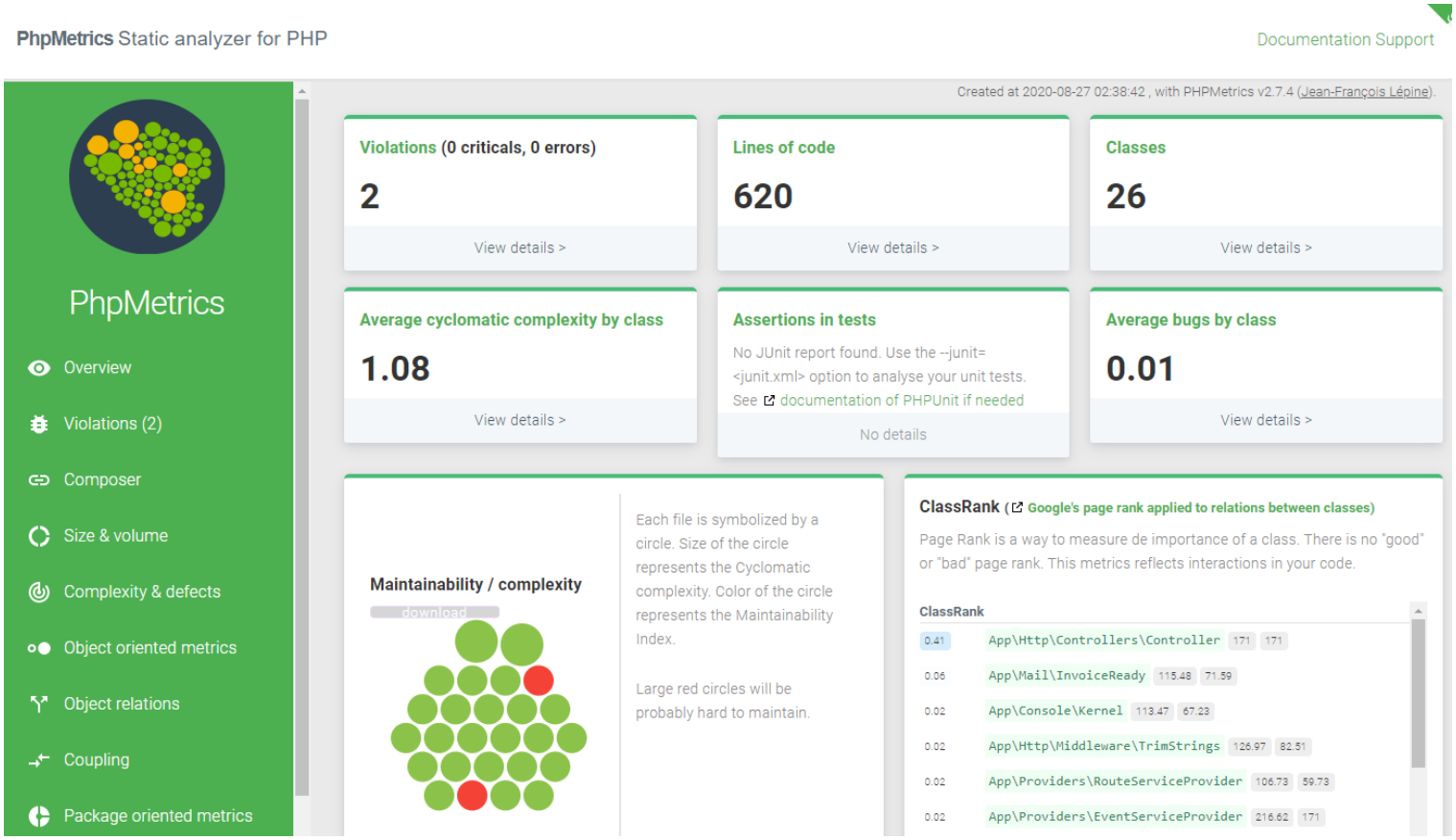
Gráfico Comparativo entre Arquitecturas



Nota: El gráfico muestra el resultado del estudio entre las métricas de las arquitecturas.

Figura 45

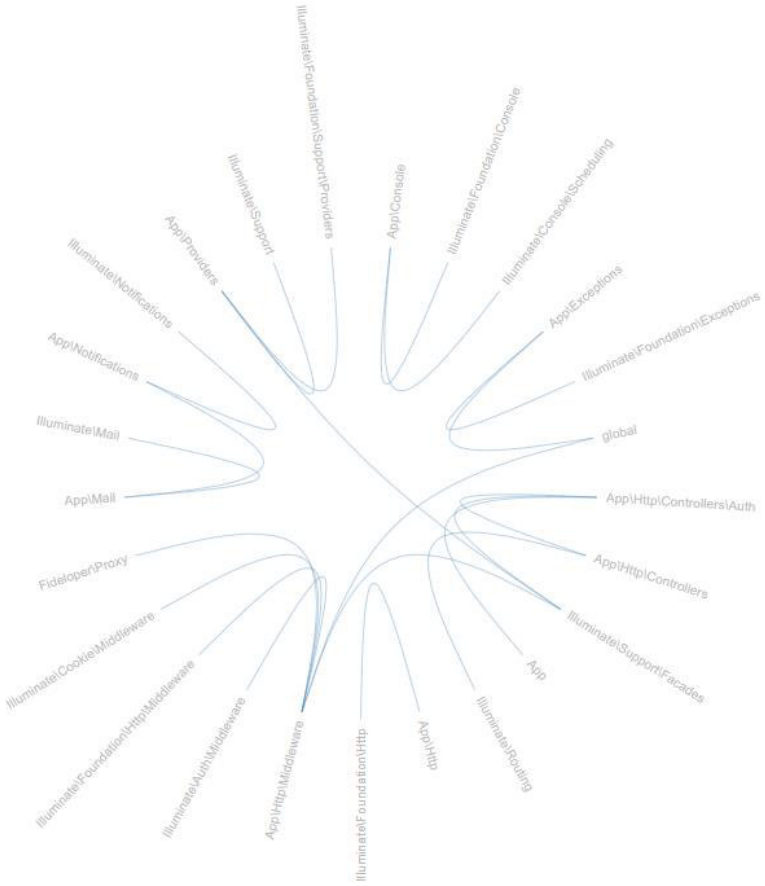
Métricas Generales Componente Notificaciones



Nota: Resultados del análisis por medio de la herramienta phpmetrics (Lépine, 2020).

Figura 46

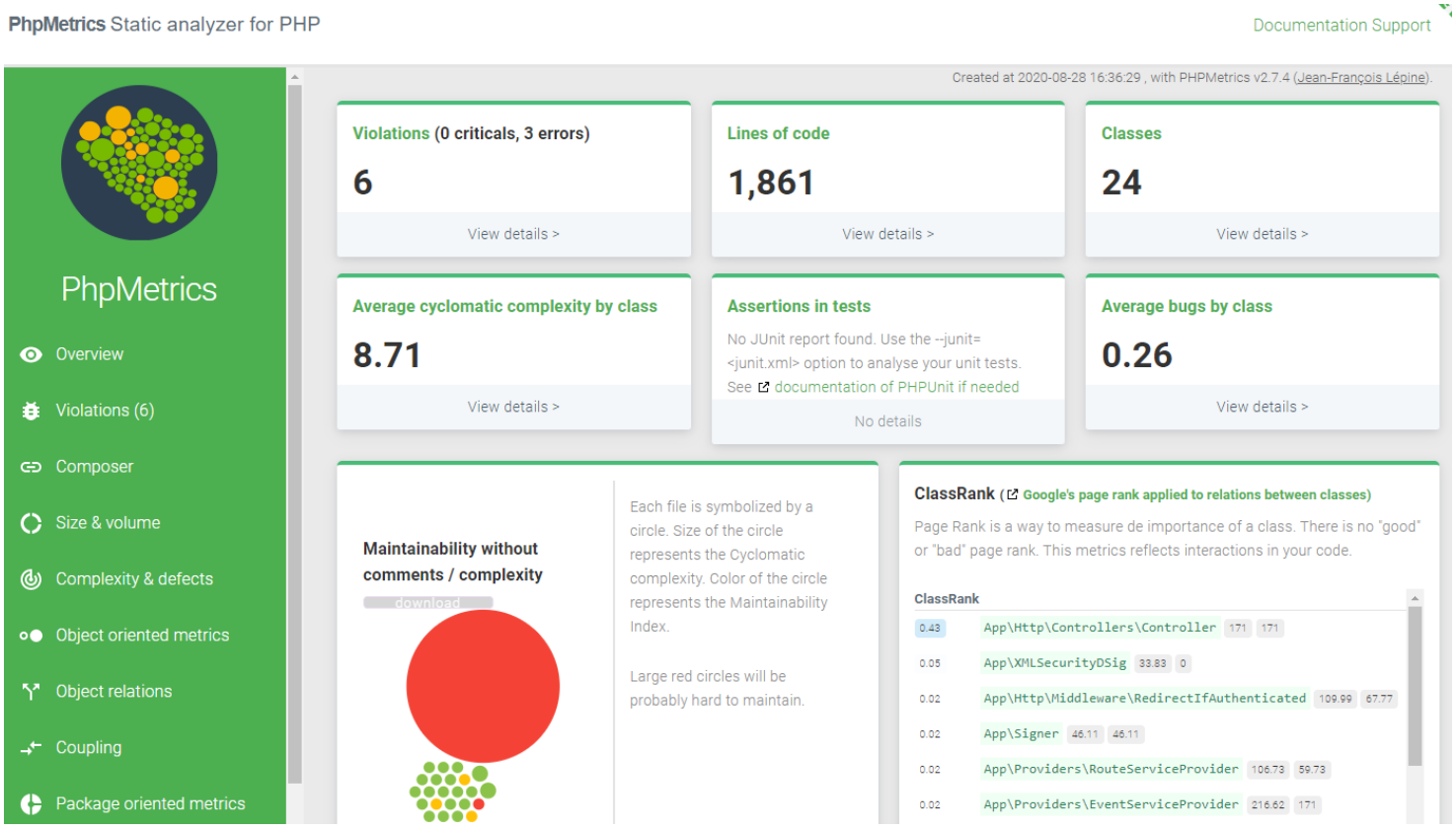
Relaciones entre Paquetes Componente Notificaciones



Nota: Tomado de la herramienta phpmetrics (Lépine, 2020).

Figura 47

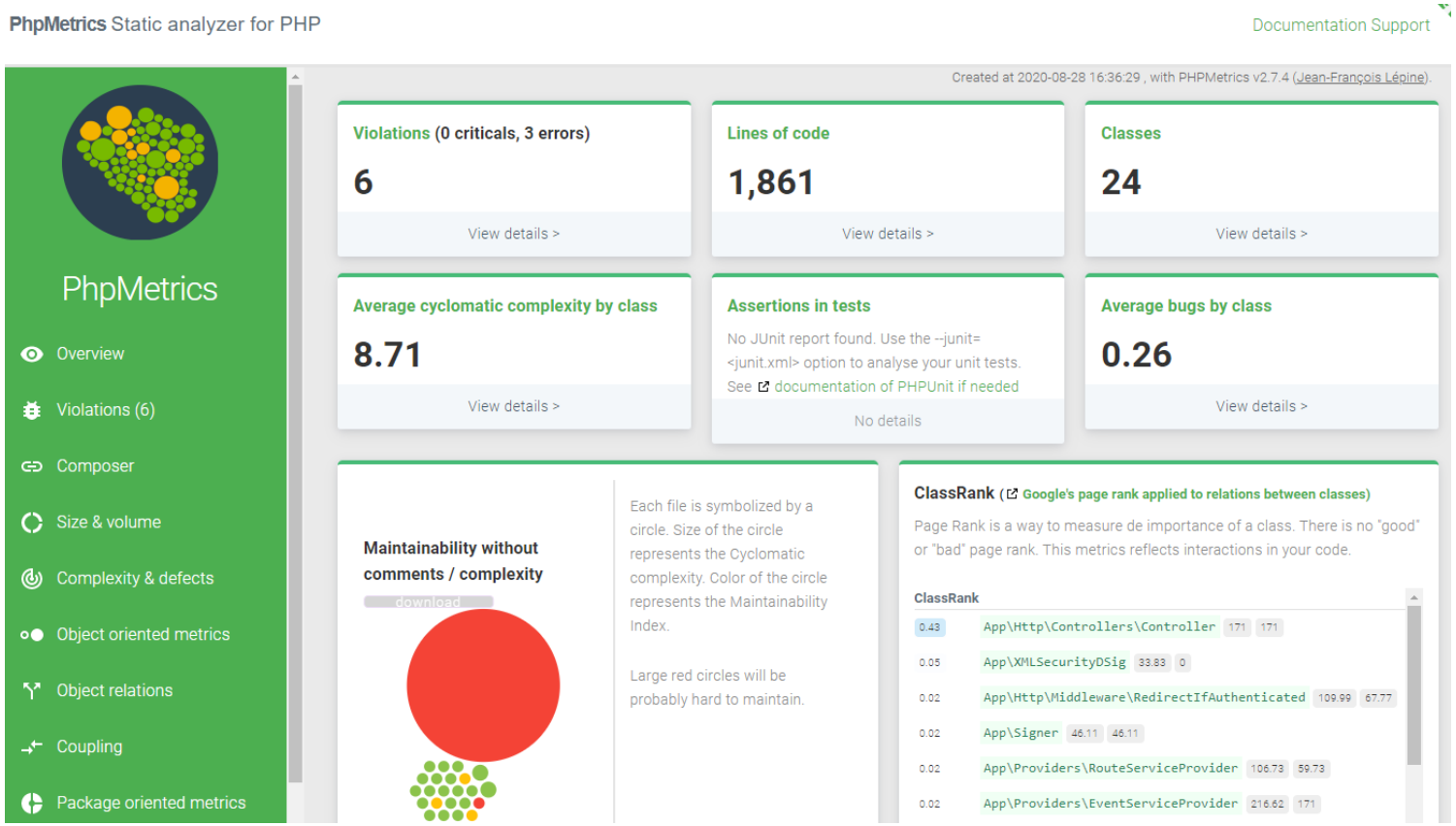
Métricas Generales Componente Firmador



Nota: Tomado de la herramienta phpmetrics (Lépine, 2020).

Figura 48

Relaciones entre Paquetes Componente Firmador



Nota: Tomado de la herramienta phpmetrics (Lépine, 2020)

Análisis de los Resultados

Las normas ISO/IEC 2500 nos permiten tener una visión más clara acerca de la arquitectura que brinde los mejores requisitos para el desarrollo de productos de software, permite obtener una transparencia de los principales componentes críticos de la aplicación, los cuales nos permiten anticiparnos a futuro.

Valores Presentados

En las tablas 22 Y 23 nos indican que las métricas de reusabilidad involucran principalmente a las arquitecturas basadas en microservicios, esto claramente puede ser para las organizaciones que requieren que sus productos puedan ser evolucionados tomando como factor la reusabilidad de componentes previamente desarrollados, en contraste con los valores presentados por la métrica de reusabilidad que es una de las características más importantes de los arquitecturas monolíticas en las cuales una alta modularidad de sus componentes pueden dar lugar al a su correspondiente coeficiente de acoplamiento, sin embargo en una organización en la cual la evolución del producto software tenga una estabilidad y una rigidez a lo largo del tiempo puede volverse altamente empleada esta arquitectura.

Evaluaciones Adicionales

Nos permiten considerar los factores como el nivel de conocimientos que debería tener el equipo de desarrollo en las tecnologías que requiere el producto software, siendo una métrica determinante la complejidad de modificación en el código, sin duda alguna un factor a tomar en consideración cuando se desarrolla el producto software ya que independientemente del tipo de producto este factor es tomado en cuenta para la futura mantenibilidad del mismo.

Capítulo V

Conclusiones y Recomendaciones

El presente proyecto de investigación se ha basado en la aplicación de las normas ISO2500 con enfoque en la mantenibilidad específicamente el apartado 25010, del cual se desprende las siguientes conclusiones y recomendaciones.

Conclusiones

- Las normas internacionales ISO/IEC 25010 permiten acoger un modelo de calidad entorno en la mantenibilidad del producto de software de forma estructurada y permitiendo involucrar a sus métricas en las decisiones de diseño en etapas tempranas del desarrollo, así como en las posteriores de pruebas y mantenimiento.
- Las arquitecturas monolíticas presentan un alto grado de acoplamiento, basados en los resultados obtenidos se puede evidenciar que su complejidad aumenta considerablemente conforme a la integración de nueva funcionalidad y grado de madurez, el estudio presentado ha mostrado un alto índice en la modularidad de esta arquitectura con llevando a una mayor posibilidad de incorporar un componente de software de baja calidad afectado así a todo el comportamiento del sistema y sus principales módulos dependientes.
- Las arquitecturas basadas en microservicios presentan resultado en favor de la descomposición afectando directamente a su modularidad ya que esta se disminuye al tener menos acoplamiento entre los módulos entorno a las necesidades del negocio, adicionalmente permite una mejor comprensión de los principales componentes de software ya que mantienen su cohesión y dentro de su contexto.

- Los resultados finales del estudio comparativo aplicadas a las arquitecturas propuestas muestran una favorable tendencia a un entorno basado en microservicios obteniendo un valor de 7.02 sobre 10; siendo un valor considerable como aceptable.
- En base a las conclusiones antes descritas, podemos concluir que la arquitectura basada en microservicios es aplicable a la facturación electrónica como un primer paso para considerar la evolución hacia plataformas de tipo web documental con base a certificados de firma digital; otorgando la capacidad de mantener un producto que evoluciona constantemente.

Recomendaciones

- Sostener la aplicación de las métricas a lo largo del ciclo de vida del producto software, mas no solo en el ciclo de vida de desarrollo del mismo; con la finalidad de obtener valores más ajustados a la evolución del producto software habilitando de esta manera una mejor comprensión de los beneficios de continuar invirtiendo en capacitación e innovación de tecnología en la organización.
- Llevar un registro cronológico de las métricas que se implementan a lo largo del ciclo de vida de desarrollo, sobre todo con el enfoque en el mantenimiento. Así como también determinar factores de posible mejora que de tal forma minimicen los riesgos de posibles errores y que las ventanas de mantenimiento sean las más cortas posibles en beneficios de los usuarios del producto software.
- Recomendar la aplicación de las normas de calidad ISO/IEC 25000 entorno a todos sus enfoques propuestos, obteniendo los valores numéricos de las métricas en base a herramientas que permitan presentación.

- Incentivar la realización de un posterior estudio comparativo sobre como la aplicación de las Normas ISO 25000 contribuyen a la sostenibilidad y la reducción del fracaso en los proyectos de desarrollo de software.

Bibliografía

Berners-Lee, T., Fielding, R., & Frystyk, H. (1996, Mayo). *rfc-editor*. Retrieved from RFC 1945.

Berners-Lee, T., Fielding, R., & Masinter, L. (2005, Febrero). *ietf*. Retrieved from https://tools.ietf.org/html/rfc3986

Bessin, G. (2005, Junio 15). *The business value of software quality.* Retrieved from IBM developerWorks: https://www.ibm.com/developerworks/rational/library/4995.html

Boniface, M., Bassem, N., Stephen C, P., Servin, A., Yang, X., & Zlatev, Z. (2010). Platform-as-a-Service Architecture for Real-time. *eprints*, 6.

Booth, D., Haas, H., McCabe, F., Newcomer, E., Champion, M., Ferris, C., & Orchard, D. (2014, FEBRERO 11). *w3.* Retrieved from W3C technical reports index: https://www.w3.org/TR/ws-arch

Callejas-Cuervo, M., Alarcon Adana, A., & Alvarez Carreño, A. (2016, Junio 22). *Scielo*. Retrieved from Scielo: http://www.scielo.org.co/scielo.php?script=sci_arttext&pid=S1900-38032017000100236

CALLEJAS-CUERVO, M., ALARCÓN-ALDANA, A., & ÁLVAREZ-CARREÑO, A. (2017). Modelos de calidad del software, un estado del arte. 236-250.

Fielding, R. T. (2000). Architectural Styles and the Design of Network-based Software Architectures. Retrieved from https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

Fielding, R., & Reschke, J. (2014, Junio). *ietf*. Retrieved from https://tools.ietf.org/html/rfc7231#section-4

Fowler, M. (2016). *Martin Fowler Blog*. Retrieved from https://martinfowler.com/articles/microservices.html

Fowler, M., & Lewis, J. (2014). *Microservices*. Retrieved from www.martinfowler.com/articles/microservices.html

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1995). *Design Patterns: Elements of Reusable Object-Oriented Software.* Pearson Education.

ISO/IEC 25010. (2020, Julio 10). Retrieved from ISO25000: https://iso25000.com/index.php/normas-iso-25000/iso-25010?limit=3&start=6

Krassner, G., & Pope, S. (1988). A Description of the Model-View-Controller UserInterface Paradigm in the Smalltalk-80 System. *Journal of Object-Oriented Programming*.

Lépine, J.-F. (2020, Agosto 20). *PHP Metrics*. Retrieved from Phpmetrics.org: https://phpmetrics.org/

Linlin Wu, S. K. (2011). SLA-based Resource Allocation for Software as a Service Provider (SaaS) in Cloud. *IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, 10.

Manish Godse, S. M. (2009). An Approach for Selecting Software-as-a-Service (SaaS) Product. *IEEE International Conference on Cloud Computing*, 5.

Mell, P., & Grance, T. (2011). Definition of Cloud Computing. *The NIST*.

Navarro, M. (2017). Plataformas para la integración. *Revistabyte*.

Newman, S. (2015). *Building Microservices.* O'Reilly Media, Inc.

Newman, S. (2019). *What Are Microservices?* O'Reilly Media.

Newman, S. (2020, junio 30). *Sam Newman & Associates*. Retrieved from https://samnewman.io/talks/principles-of-microservices/

Nitu. (2009). Proceeding of the 2nd annual conference on India software engineering. *Configurability in SaaS (software as a service) applications*.

Normas de Calidad ISO 25040. (2020, 07 16). Retrieved from Normas de Calidad ISO 25040: https://iso25000.com/index.php/normas-iso-25040/iso-25040

Olsen, R. (2007). Convention Over Configuration. In R. Olsen, *Design Patterns in Ruby.* Addison-Wesley Professional.

Olsen, R. (2007). *Design Patterns in Ruby.* Addison-Wesley Professional.

Overview of Microservices. (2016). In S. De Santis, L. Florez, D. Nguyen, & E. Rosa, *Evolve the Monolith to Microservices with Java and Node.* RedBooks.

Pardo Mesias, S. (2018). *Mantenibilidad de productos de software según el modelo square iso/iec 25000.* Universidad Nacional Agraria de la Selva, TIngo María.

Pressman, R. (2010). *Ingeniería del Software.* Mexico DF: Mc Graw Hill.

Proyectosagiles. (2020, Julio 10). Retrieved from proyectosagiles: https://proyectosagiles.org/

Richards, M. (2015). *Software Architecture Patterns.* O' Reilly Media.

Rising, L. (1998). *The Patterns Handbook: Techniques, Strategies, And Applications.* Cambridge University Pres.

Rodero, M. (2011). Building Safe PaaS Clouds: A Survey on Security in Multitenant Software Platforms. *The Adaptive Choice-Based Conjoint (ACBC) Technical Paper*.

Rodriguez, J. (2006). Metodologia de la Investigacion curso general y aplicado. Bogota.

Rodríguez, M., Pedreira, Ó., & Fernández, C. M. (2020). Certificación de la Mantenibilidad del Producto Software. *Revista Latinoamericana de Ingeniería de Software*, 127-134.

Sáez, F. (2020, Junio 18). *facilethings*. Retrieved from https://facilethings.com/blog/en/convention-over-configuration

Schwaber, K., & Sutherland , J. (2016, Julio). *La Guía de Scrum.* Retrieved from ScrumGuides: https://www.scrumguides.org/docs/scrumguide/v2016/2016-Scrum-Guide-Spanish.pdf#zoom=100

Scrum Alliance. (2020, 07 01). Retrieved from Scrum Alliance: https://www.scrumalliance.org/about-us

Scrum Org. (2017, Septiembre 25). *Scrum Org.* Retrieved from The Home of Scrum: https://www.scrum.org/resources/blog/que-es-scrum

Servicio de Rentas Internas. (2020, Agosto 25). *Facturación Electrónica SRI*. Retrieved from Servicio de Rentas Internas del Ecuador: https://www.sri.gob.ec/

SoapUI. (2020, Agosto 25). *SoapUI*. Retrieved from SoapUI: https://www.soapui.org/

Stauffer, M. (2019). *Laravel Up & Running.* O'Reilly Media.

Taibi, D., Auer, F., Lenarduzzi, V., & Felderer, M. (2019). From Monolithic Systems to Microservices: An Assessment Framework. *Arxiv*, 1-7.

Tiwana, A., & Konsynski, B. (2010). Platform Evolution: Coevolution of Platform. *Information Systems Research* , 675-687.

Valenciano Lopéz, J. (2015, Junio). *AUDITORÍA MANTENIBILIDAD APLICACIONES SEGÚN LA ISO/IEC 25000 (Tesis de Grado).* UNIVERSIDAD COMPLUTENSE DE MADRID, Madrid.

Vargas Cordero, Z. (2009). La Investigación Aplicada: Una forma de conocer las realidades con evidencia. San Pedro Montes de Oca, Costa Rica.

visual-paradigm. (2020, Julio 13). Retrieved from https://www.visual-paradigm.com/scrum/what-is-agile-software-development/

Xolido. (2020, Agosto 25). *Xolido Sign Desktop*. Retrieved from Xolido Sign: https://www.xolido.com/lang/xolidosign/xolidosigndesktop/