Software architecture for the Catolica de Cuenca University.

Arquitectura de software para la Universidad Católica de Cuenca.

Autores:

Rubio Cedillo, Víctor Manuel
UNIVERSIDAD CATÓLICA DE CUENCA
Estudiante de maestría en tecnologías de la información
Cuenca – Ecuador
vrubioc@ucacue.edu.ec
https://orcid.org/0000-0001-9268-8276

Zhindón Mora, Martín Geovanny
UNIVERSIDAD CATÓLICA DE CUENCA
Docente en la maestría en tecnologías de la información
Cuenca – Ecuador
mgzhindonm@ucacue.edu.ec
https://orcid.org/0000-0003-4475-830X

Campana Ortega, Eduardo Mauricio
UNIVERSIDAD CATÓLICA DE CUENCA
Docente en la maestría en tecnologías de la información
Cuenca – Ecuador
eduardo.campana@ucacue.edu.ec
https://orcid.org/0000-0001-7720-5213

Citación/como citar este artículo: Rubio, V., Zhindón, M. y Campana, E. (2022). Arquitectura de software para la Universidad Católica de Cuenca. MQRInvestigar, 6(3), 1597-1617.
https://doi.org/10.56048/MQR20225.6.3.2022.1597-1617

Fechas de recepción: 29-AGO-2022 aceptación: 13-SEP-2022 publicación: 15-SEP-2022

https://orcid.org/0000-0002-8695-5005
http://mqrinvestigar.com/

Resumen

Este artículo, pretende determinar una arquitectura de software para la Universidad Católica de Cuenca, por medio de la adopción de estándares, políticas, convenciones, buenas prácticas y metodologías estandarizadas, con la finalidad de aminorar la curva de aprendizaje y mejorar la adaptabilidad al ecosistema de desarrollo de nuevos colaboradores, estos pueden ser externos así como internos, realizando los entregables solicitados por las diferentes áreas de la comunidad universitaria en un menor tiempo, con mayor calidad, eficacia y eficiencia de una forma organizada.

El modelo de desarrollo de software permitirá la expansión organizada de los diversos proyectos que conforman el ecosistema informático institucional, sabiendo que este está en constante crecimiento con el objetivo de satisfacer las diferentes necesidades operativas que surgen con el crecimiento de la institución.

Palabras clave: Arquitectura de Software, modelado, diseño de software, desarrollo de software.

Abstract

This article aims to determine a software architecture for the Catolica de Cuenca Universit, through the adoption of standards, policies, conventions, good practices and standardized methodologies, in order to reduce the learning curve and improve the adaptability to the ecosystem of development of new collaborators, these can be external as well as internal, carrying out the deliverables requested by the different areas of the university community in less time, with greater quality, effectiveness and efficiency in an organized manner.

The software development model will allow the organized expansion of the various projects that make up the institutional computing ecosystem, knowing that it is constantly growing in order to satisfy the different operational needs that arise with the growth of the institution.

Keywords: Software architecture, modeling, software design, software development.

Introducción

Posterior al nacimiento del software, y con la evolución del mismo, para una correcta maniobrabilidad de este, un crecimiento adecuado y mantenibilidad en el tiempo, se concibe la arquitectura del software, la cual da los lineamientos propicios para tener un producto de software de calidad y escalable, haciendo que este sea fácil y entendible de manejar, lo que además ayuda reducir costos tanto de desarrollo como de mantenimiento.

No existe un consenso en cuanto al concepto de arquitectura de software, esto se debe a que la misma es cambiante en el tiempo, por lo que su definición no sería la misma después de un tiempo, esto obedece al constante crecimiento de la ingeniería del software. Hay que tener en cuenta que esta última es relativamente nueva con respecto a las otras ingenierías, las cuales se consideran ciencias "estáticas" (Richards & Ford, 2020).

Se considera que entre más avanza la ingeniería de software, el área de la arquitectura sigue siendo relativamente inmadura, por el tiempo de estudio y aplicación de la misma con respecto a la ciencia como tal(Garlan, 2014).

Últimamente la arquitectura de software se ha convertido en un factor importante debido a los buenos resultados obtenidos de una correcta implementación de la misma, ayudando a diversos factores como la seguridad, fiabilidad de la información, robustez de la aplicación, entre otros. Si bien una buena implementación de la arquitectura de software da excelentes resultados, una mala aplicación de la misa conlleva a resultados desastrosos y costos, ya que afecta directamente al desempeño, mantenibilidad, disponibilidad y demás factores que empobrecen el sistema.

La arquitectura de software es un eje fundamental para el desarrollo de aplicaciones empresariales, y por ende para la Universidad Católica de Cuenca (UCACUE), que cuenta con un software, el ERP University, el cual es usado para la gestión y control de la institución, este está en constante expansión debido al crecimiento institucional y al cambio continuo del régimen académico en el país, por tal motivo es necesario que el ERP evolucione en su arquitectura de tal manera que el desarrollo de las nuevas necesidades o adecuaciones se realicen de una manera efectiva y eficaz.

Este documento consta de cuatro partes, la primera es un breve estudio a la arquitectura de software (Análisis general), la segunda trata del diagnóstico a la arquitectura de software presente en el ya mencionado ecosistema que comprende el ERP University, en

la tercera parte se emite una propuesta para la arquitectura, concluyendo con la parte cuatro, las conclusiones obtenidas.

Material y métodos

Al igual que en el desarrollo del software en la arquitectura de software existen las buenas y malas prácticas, si no se aplica adecuadamente o peor aún si se carece de la arquitectura de software en un proyecto, se puede generar un verdadero caos. a estas malas prácticas se las conoce como "*smelly*", pero ¿que comprende la arquitectura de software?, es muy usual la confusión entre arquitectura y diseño de software, por lo que es necesario establecer una diferencia entre las dos. Como ya se mencionó anteriormente enmarcar la arquitectura de software en un concepto o definición es complejo por la continua evolución que esta sufre, pero basados en la teoría y experiencia se intenta establecer un concepto cercano a lo que puede referirse esté termino.

Arquitectura de software: comprende el esqueleto del proyecto, al más alto nivel de abstracción del negocio; determina también el almacenamiento de datos, como se interrelacionan los componentes, las capas o niveles que va a poseer, de igual manera las restricciones pertinentes, se trata de aquellas decisiones clave que, de llegar a cambiarse más adelante, generan un costo (Garlan, 2014; Jaiswal, n.d.; Oyedeji et al., n.d.; Richards & Ford, 2020; Wang et al., 2005; Yong, 2021).

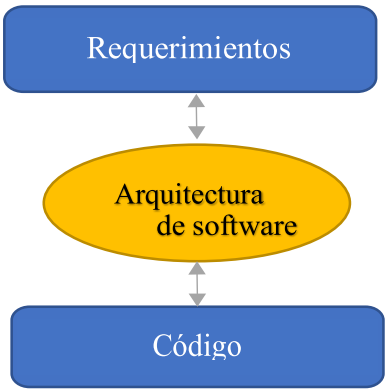
Toda decisión acerca de la arquitectura de software se basa en las compensaciones que esta pueda retribuir al proyecto, siendo siempre más importante el por qué, más que el cómo.

Diseño de Software: corresponde al diseño de los componentes, módulos o clases, se pueden cambiar más adelante para dar salida y atención a necesidades operativas o comerciales (Jaiswal, n.d.; Le et al., 2018; Richards & Ford, 2020; Yong, 2021)

Para un mejor entendimiento de la diferencia entre estos dos términos, arquitectura y diseño de software, si hablamos de duplicidad de interfaces o clases, enlaces rotos o sin funcionamiento, esto obedece a un problema provocado por un diseño erróneo de software, mientras que si existen componentes cuya abstracción no es la adecuada, conectores mal definidos, entre otros, esto es provocado por una arquitectura de software errónea (Richards & Ford, 2020).

Figura 1

La arquitectura de software como puente hacia el producto final(Garlan, 2014)



Fuente: Elaboración propia

Como se muestra en la figura 1, la arquitectura de Software es el puente entre los requerimientos y el código del sistema, la arquitectura empieza a tomar forma basada en los requerimientos no funcionales de la aplicación, por ejemplo, si se requiere manejar en el sistema información sensible, debemos dotar de las respectivas seguridades a la aplicación por medio de la arquitectura, por lo mismo los requerimientos no funcionales son conocidos también como características de la arquitectura, entre algunas a mencionar están las siguientes:

Disponibilidad, Fiabilidad, Testeabilidad, Escalabilidad, Seguridad, Agilidad, Tolerancia a fallos, Elasticidad, Recuperabilidad, Desempeño, Capacidad de aprendizaje.

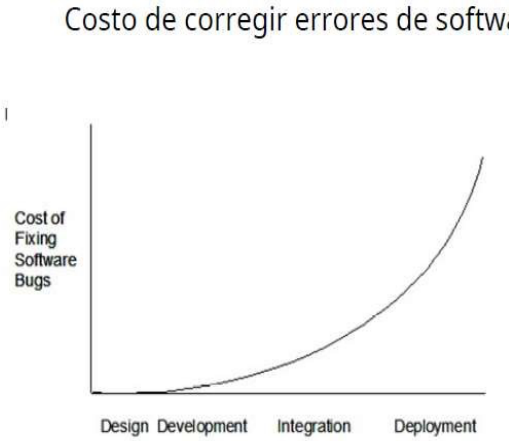
Existen seis problemas presentes en la arquitectura de software de los cuales debemos estar atentos(Le et al., 2018), estos son:

1. Sobrecarga de preocupaciones.
2. Ciclo de dependencia.
3. Sobrecarga de enlace.
4. Interfaz no usada.
5. Delegación descuidada.
6. Acoplamiento de cambio conjunto.

Incurrir en estos problemas puede desembocar en un costo monetario muy alto, como se indica en la figura 2.

Figura 2

Curva del costo del desarrollo del software (Miyachi, 2011)



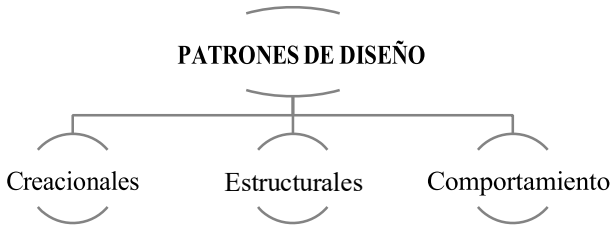
Patrones de diseño de arquitectura

Dentro del mundo de la arquitectura de software encontramos una gran variedad de patrones de diseño de arquitectura que ayudan al objetivo de tener una arquitectura de software óptima y adaptada, estos encapsulan soluciones específicas para ser aplicados en diferentes proyectos (Wang et al., 2005).

Podemos entender por patrones de diseño a las comunicaciones entre los objetos y las clases que se adecuan para solventar los problemas de diseño general en un escenario puntual, la industria y academia continuamente han demostrados los múltiples beneficios del uso de los patrones de diseño(Wang et al., 2005).

Figura 3

Tipos de patrones de diseño



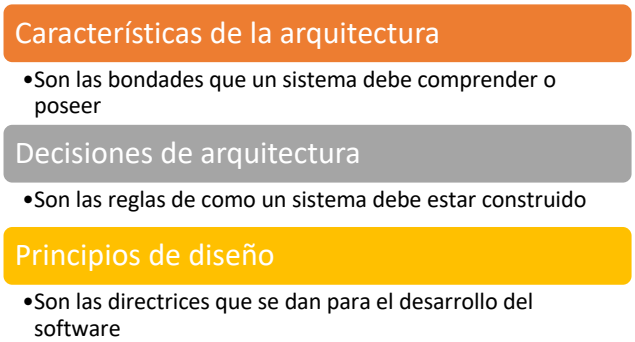
Fuente: Elaboración propia

Como se indica en la figura 3, hay tres tipos de patrones de diseño de arquitectura que se clasifican por su naturaleza, estos son: creacionales, estructurales y de comportamiento.

Hay una correlación entre los patrones de diseño de arquitectura y el diseño del software, esta vinculación se establece a través de los objetivos de la arquitectura, los cuales a su vez se dividen en subjetivos, por lo tanto, la arquitectura y el diseño van de la mano en base a los propósitos de los requisitos del sistema (requisitos no funcionales).

Existen varios patrones de diseño, entre estos están los últimamente populares, Microservicios, por su uso en grandes aplicaciones como la plataforma de streaming Netflix, también encontramos los patrones de diseño orientados a eventos y los *server-less* (sin servidor).

La arquitectura de software comprende tres aristas que son importantes considerar.

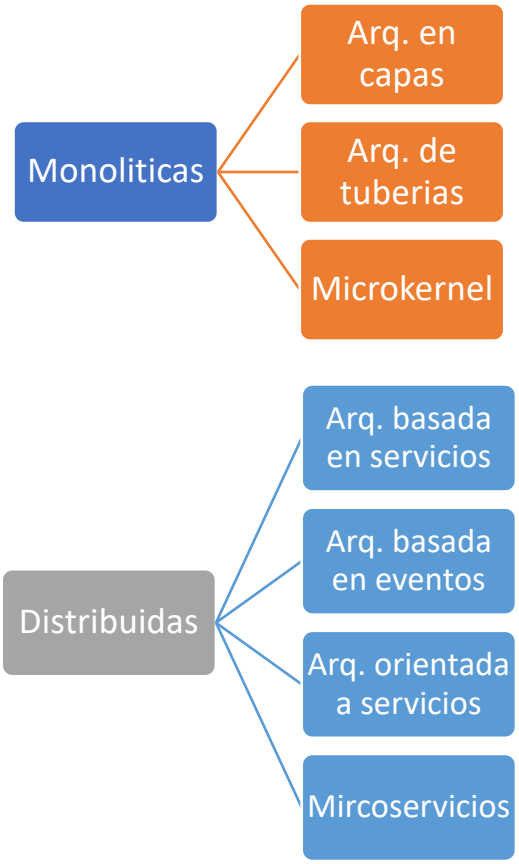


La modularidad es otro pilar importante dentro de la estructura del software, ya que otorga desacoplamiento parcial o total de los módulos a desarrollar, actualmente entre más desacoplados estén estos, más libertad se tiene para el desarrollo comunitario, pudiendo estos módulos dividirse en los grandes dominios de la empresa, una gran ventaja de la modularidad es la disminución del riesgo a fallos totales, ya que en el caso extremo de fallar un módulo los demás continúan operativos para las otras partes del negocio sin causar un cese total de actividades hasta la mitigación del error que lo haya causado, la modularidad se es medible en base a tres parámetros: la cohesión, el acoplamiento y la conciencia entre módulos.

Dentro de las arquitecturas actualmente contamos con dos grupos, que se dividen según la figura 4 en (Richards & Ford, 2020):

Figura 4

Tipos de arquitecturas.



Fuente: Elaboración propia

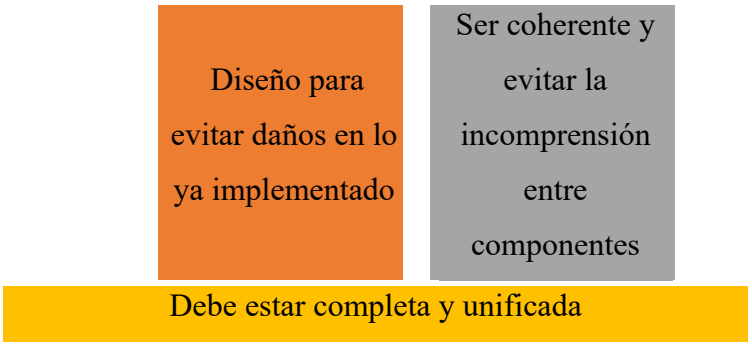
Arquitectura ágil

Con las nuevas metodologías de desarrollo, nace la arquitectura de software ágil, esta está alineada de igual manera con el manifiesto ágil, sus principios y objetivos, este tipo de arquitectura es evolutiva, lo que quiere decir que a medida que evoluciona el proyecto la arquitectura de software lo hace de igual manera, si a medida que avanza el proyecto los cambios funcionales requieren el uso de componentes externos por ejemplo, la arquitectura debe adaptarse en función de satisfacer esta necesidad, estableciendo la forma en la que la aplicación interactuara con el mencionado módulo, esto puede ser por una nueva capa de comunicación brindando el desacoplamiento suficiente para hacer el módulo reemplazable de ser requerido (Yong, 2021).

La arquitectura ágil busca seguir tres principios, según lo expuesto en la figura 5:

Figura 5

Principios de la arquitectura ágil



Fuente: Elaboración propia

Los tres principios expuestos deben estar alineados de tal manera que el producto final cumpla con las expectativas y requisitos del cliente, en el caso de cambiar las necesidades del cliente, se vuelve a iterar todo el proceso hasta lograr una satisfacción plena con el resultado, siendo la refactorización del código un tema clave en este tipo de metodología. Es totalmente correcto afinar detalles en cuanto a la arquitectura en el ciclo de vida del desarrollo, ya que no importa cuán bien se crea que se encuentra estructurado el desarrollo, la misma interacción con las partes interesadas en el proceso hará que dicha arquitectura tienda a modificarse leve o incrementalmente; esto no quiere decir que no haya existido software que se haya desarrollado tal cual se presentó en la infraestructura inicial; sin embargo, eventualmente dichos proyectos se cancelan, si cuenta con gran documentación, pero no con un sistema en funcionamiento óptimo y exitoso, toda la documentación es inútil, en este mismo aspecto se debe tener cuidado con las denominadas arquitecturas de torre de marfil, ya que las mismas son creadas por arquitectos en relativo aislamiento del equipo de trabajo, fueron infraestructuras aparentemente bien sentadas y documentadas, pero lamentablemente no siguieron el adecuado proceso ni mucho menos se consideran los principios del manifiesto ágil (Miyachi, 2011; Nord & Tomayko, n.d.; Yong, 2021).

Migración de arquitecturas

Cuando se tiene aplicaciones legadas que continúan siendo de utilidad, es común encontrarse con la necesidad de realizar nuevos módulos o adaptaciones a los ya existentes, generalmente la primera idea es el cambio de software, pero esto no siempre es óptimo, ya que el software legado puede tener un *core* bastante ajustado a las necesidades de la empresa, algo difícil de encontrar en el mercado ofertante de soluciones de este tipo; por consiguiente lo que se busca es la actualización del software y la adaptación o adición de las nuevas funcionalidades, para esto se requiere un análisis minucioso en el cual se debería estudiar inicialmente la variación que va a sufrir la arquitectura del software.

La migración de sistemas heredados a una nueva arquitectura de software puede ser muy fuerte e invasiva, siendo la arquitectura más común la monolítica. Si se desea la transición de una arquitectura de software monolítica hacia una basada en microservicios, se debe considerar incluso la afección en la abstracción de la aplicación (da Silva et al., 2019; Wang et al., 2005).

Después del análisis del estado inicial a la arquitectura heredada y listos para la migración se plantean varias preguntas que ayudaran en este proceso:

- ¿Cuáles son los pasos efectivos para la migración?
- ¿Cuáles son las características del software que deben llevarse a la modularización, y cómo migrar las mismas?
- ¿Cómo migrar las mejoras?

Posterior al análisis y la resolución de estas preguntas, se puede denotar que no siempre es mejor o más optimo realizar la migración entre arquitecturas, esto puede deberse a una serie de factores como el costo del cambio, la adaptabilidad del proyecto a la nueva tecnología, la vida del proyecto después del cambio, incluso cuando los enfoques de la nueva arquitectura posee reconocimientos a sus bondades, se puede tener como resultado del análisis que el cambio no es idóneo, sino en su lugar se podrían modularizar ciertos componentes y mejorar utilidades del software legado (da Silva et al., 2019).

Diseño de la arquitectura del software

Al momento del diseño de la arquitectura de software, es importante analizar en los requerimientos los aspectos importantes como la seguridad, privacidad de los datos y el sistema en sí, el realizar pruebas de latencia, estudios de concurrencia, evaluaciones de disponibilidad de la infraestructura, tazas de transacción y cualquier riesgo que se pueda presentar durante el desarrollo del proyecto, así como durante la vida del software en sí. Con respecto a la arquitectura del software, cabe estimar los siguientes seis aspectos(Garlan, 2014).

1) Comprensión: La arquitectura de software ayuda a comprender sistemas de gran tamaño al mostrarlos en un nivel de abstracción muy entendible.
2) Reutilizar: La arquitectura permite la reutilización de componentes, marcos, bibliotecas y complementes a lo largo del sistema, lo que hace que sea mucho más fácil su mantenimiento.

3) Construcción: Establece el esqueleto principal de la aplicación denotando los componentes principales y como estos interactúan entre sí.
4) Evolución: Por medio del diseño arquitectónico se pueden establecer los lineamientos de expansión del software que lo comprende.
5) Análisis: Las descripciones arquitectónicas dan paso al análisis, así como al a validación de la coherencia del proyecto.
6) Gestión: El correcto diseño de la arquitectura de software ayuda a la evaluación crítica de la misma, y guía al establecimiento de estrategias de implementación a más de evaluar los potenciales riesgos lo que se traduce en una menor refactorización del código.

En base a estos aspectos podemos decir que gracias a una buena arquitectura de software es posible modularizar de tal forma que se use la persistencia de un proveedor, la vista de otro, un marco de desarrollo de otro y así sucesivamente, gracias al desacoplamiento establecido en el diseño, y lograr que todo funcione en perfecta armonía.

Es útil el tener presente la diferencia entre estructuras de codificación y estructuras de tiempo de ejecución, la primera ayuda a razonar sobre el mantenimiento y el costo de la refactorización, en cambio la segunda es útil para evaluar características como el rendimiento, la escalabilidad y la confiabilidad.

Es importante recalcar que no hay reglas ni consejos generales para todos los proyectos, aun habiendo muchos intentos de formalizarlos con miras a llegar a la excelencia(Jaiswal, n.d.), por lo tanto, lo que determina el éxito en el diseño de esta es la práctica y experiencia de los ingenieros al momento de la concepción de una arquitectura como solución a un sistema (Oyedeji et al., n.d.). El desarrollo de las arquitecturas sostenibles es de alguna manera un paso en la evolución de las mimas, reduciendo el impacto negativo que pudiese surgir con el tiempo y malograr el software, para eso la implementación de estándares y buenas prácticas es un punto obligatorio.

Lamentablemente al momento no hay una buena guía para el desarrollo de una arquitectura sostenible en la cual se indiquen las medidas a tomar para lograr este objetivo. Lo que si se tiene claro es que la sostenibilidad se la debe aplicar desde el levantamiento de requisitos, (para lo cual se dispone ciertas plantillas que ayudan en este punto), hasta el fin del ciclo de vida, por medio del involucramiento de los usuarios y desarrolladores hacia este estilo, compartiendo inclusive compromisos emocionales.

En un proyecto en el cual los requerimientos atraviesan por cambios constantes, es ideal la aplicación de una metodología ágil. Una de estas alternativas es implementar XP, por lo que las decisiones de cambio pueden afectar los conectores, configuraciones, o componentes de la arquitectura, pero por medio de la XP se van afinando continuamente, esta es una metodología ágil y de esta forma no representara un costo de refactorización posterior (Nord & Tomayko, 2006).

En la actualidad se ha creado el paradigma de sistema de sistemas (SOS), es un sistema orientado a integrar algunos sistemas legados con otros actualizados y con diferentes dominios, orientando los recursos y capacidades en pro del éxito de la integración (Mohsin & Janjua, 2018). El gran desafío que esto implica, es el diseño de la arquitectura dinámica, intentando prever las conductas imprevistas en ejecución, por lo que los SOS, generan acoplamientos tolerantes y la interoperabilidad. Para esto se plantean cuatro tipos de modelos:

- Modelado de arquitectura estática
- Modelado de arquitectura dinámica
- Reconfiguraciones de tiempo de ejecución
- Representación de atributos de calidad

Resultados y discusión

Para realizar una propuesta adecuada de una arquitectura de software, es necesario basarse en las características que la misma debe tener, se han listado 8 de estas consideradas aplicables en el software universitario.

Usabilidad, Escalabilidad, Mantenibilidad, Fiabilidad, Extensibilidad, Rendimiento, Disponibilidad, Observabilidad

Las características indicas fueron evaluadas cualitativamente, por el grupo de desarrollo en base a las necesidades institucionales, llegando a un consenso total, como resultado se priorizaron: la Usabilidad, la Escalabilidad y la Mantenibilidad, para cada una de ellas se determina como se debería adaptar el software o proceder para hacer más robusta cada característica.

Usabilidad: Para mejorar la experiencia de usuario se debe inicialmente contar con una guía de diseño del sistema, en el cual se indiquen las reglas generales de usabilidad y sus

buenas prácticas, así como: la paleta de colores, tipografía y demás elementos, de tal forma que los desarrolladores del sistema mantengan concordancia en la elaboración del producto; el rol de las reglas de diseño en un factor clave ya que estas determinan la carga visual, la forma de construcción de los interfaces, etc.

Escalabilidad: Siendo esta la capacidad que tenga el sistema para la adaptación a un nuevo cambio así como el crecimiento del mismo de forma efectiva, se debe estimar el software como tal y la infraestructura que este soporta, siendo la última la menos compleja de atacar, la aplicación debe reposar en una infraestructura que pueda escalar horizontalmente y bajo demanda, encontrando puntos de operación óptimos en cualquier proceso en el que se encuentre (Ej: época de matrículas en la cual el sistema tiene un alto grado de concurrencia por parte del estudiantado), por el lado del software institucional debe tener la posibilidad de implementar nuevas funcionalidades sin que el sistema pierda operatividad en desarrollos anteriores, siendo para esto un requisito el desacoplamiento y la modularidad.

Mantenibilidad: Con la constante expansión que sufre el proyecto por las diferentes circunstancias, el mismo tiene que ser de fácil mantenimiento, sabiendo que la tarea de mantenerlo puede no ser de quien hizo la implementación inicial, por tal razón, la forma de desarrollar debe ser estandarizada, homogénea y transversal para todos los desarrolladores.

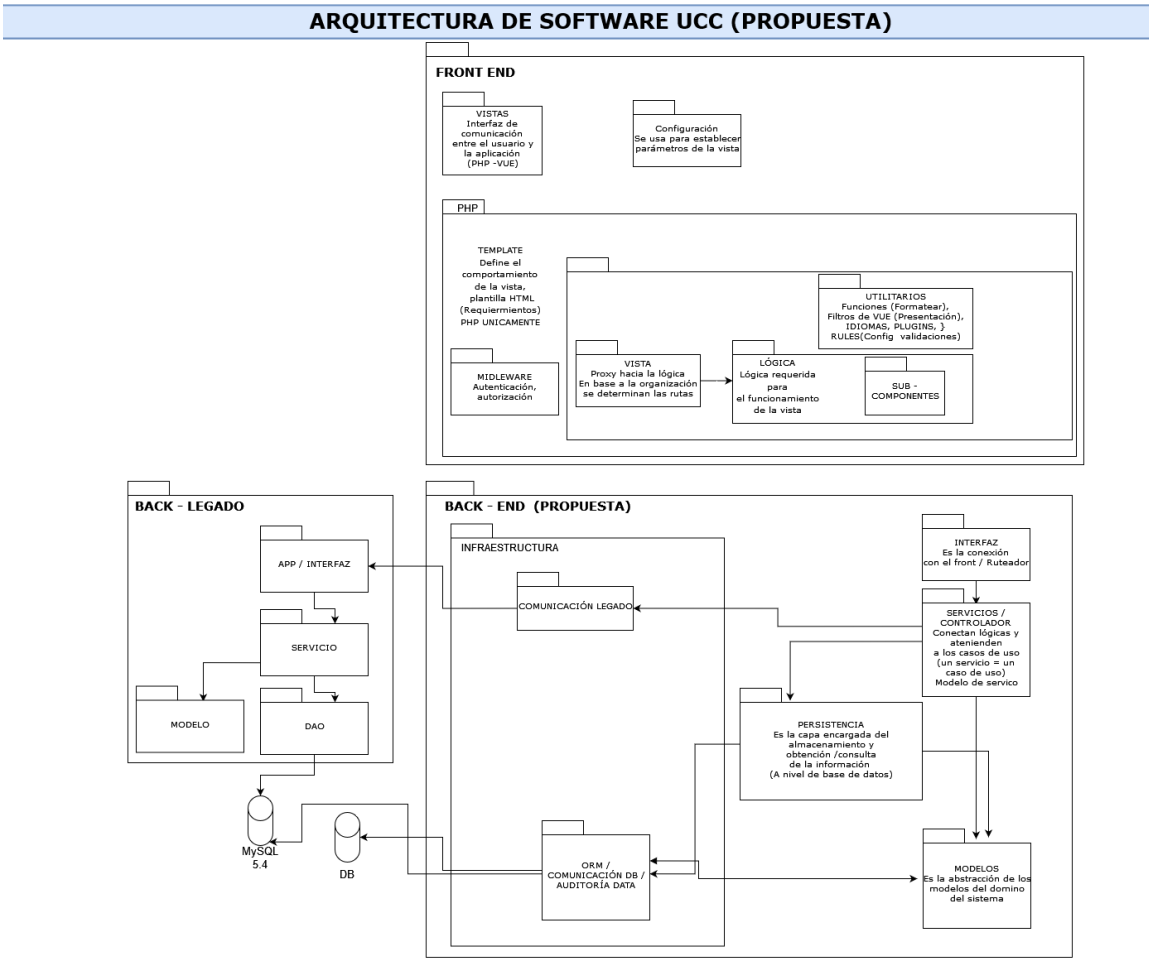
Propuesta

En base al análisis realizado a las arquitecturas y a la literatura se propone una arquitectura en capas modularizada en dominios, potenciando la reutilización de componentes aplicando conceptos de arquitectura basada en servicios y un uso adecuado de la segmentación y segregación de las responsabilidades de cada capa.

Modelo de aplicación

Figura 6

Arquitectura Propuesta



Fuente: Elaboración propia

El modelo planteado en la figura 6, puede ser replicado por cualquier software independiente del lenguaje o framework que se use, de igual manera con esta estructura se puede lograr una arquitectura distribuida en dominios replicando el mismo modelo en proyectos independientes, la interconexión del mismo puede ser por medio de un bus de eventos o un API *GATEWAY* que sirva de enrutador de las peticiones, sin embargo por medio de librerías como AXIOS, la parte del front se puede conectar a *n* proyectos de

back por medio de la instanciación de la misma usando diferentes parámetros de conexión.

Como se puede observar en la figura 6 el Back-end propuesto se basa en un modelo de capas, con la ventaja de un desacoplamiento entre las mismas, la capa de infraestructura se comunica con la aplicación legada, la ventaja de tener un ORM a más de la facilidad de la comunicación con la base de datos es el desacoplamiento al dato por medio de los modelos, sin restar importancia que se pueden separar los modelos en diferentes esquemas y yendo un poco más allá en varios motores de bases de datos, haciendo de la aplicación un ente completamente agnóstico a cualquier herramienta que se pudiese usar para las soluciones que se brinden.

Escalamiento Horizontal

Aplicación

Con miras a dar solución al escalamiento horizontal, la aplicación puede ser llevada a servicios IaaS como el de AWS, AZURE o GCP, en estos se tiene la facultad de replicar servidores virtuales de la aplicación, incrementando de forma paralela, el ancho de banda, memoria RAM y demás recursos que se pueden saturar bajo la alta concurrencia de los usuarios, para equilibrar dicha saturación y con la replicación de instancias se puede usar como puerta de entrada un balanceador de carga para que se distribuya el tráfico equitativamente entre los servidores, compartiendo así el estrés que pudiesen llegar a tener.

Base de datos

Las bases de datos pueden tener un escalamiento horizontal, dividiendo la escritura y la lectura de sus datos, para esto se necesita de una base de datos principal y las réplicas de las mismas, siendo estas últimas usadas para la lectura de datos; por medio de varias herramientas como MaxScale, ProxySQL y MySQL Proxy se puede lograr el balanceo.

Segregación de servidores por función

Archivos estáticos

Para que los archivos estáticos propios del giro de negocio de la instrucción, no afecten el rendimiento ni costos al momento de la replicación de instancias, se recomienda el uso de un servidor dedicado para los mismos, con acceso único desde la aplicación.

Aplicación – Front/Back

Al realizar la arquitectura distribuida en dominios, se puede separar el front del back, y a su vez el back en los n dominios que tenga la institución, siendo cada uno de estos proyectos del back desplegables en un servidor único, incluso en un lenguaje diferente si se requiere, esto se traduce en un desacoplamiento modular que serviría para el escalamiento bajo demanda para cada uno ellos o a su vez remplazo con mayor facilidad de ser necesario, y un escalamiento, aminorando costos y esfuerzos por parte del equipo.

Base de datos

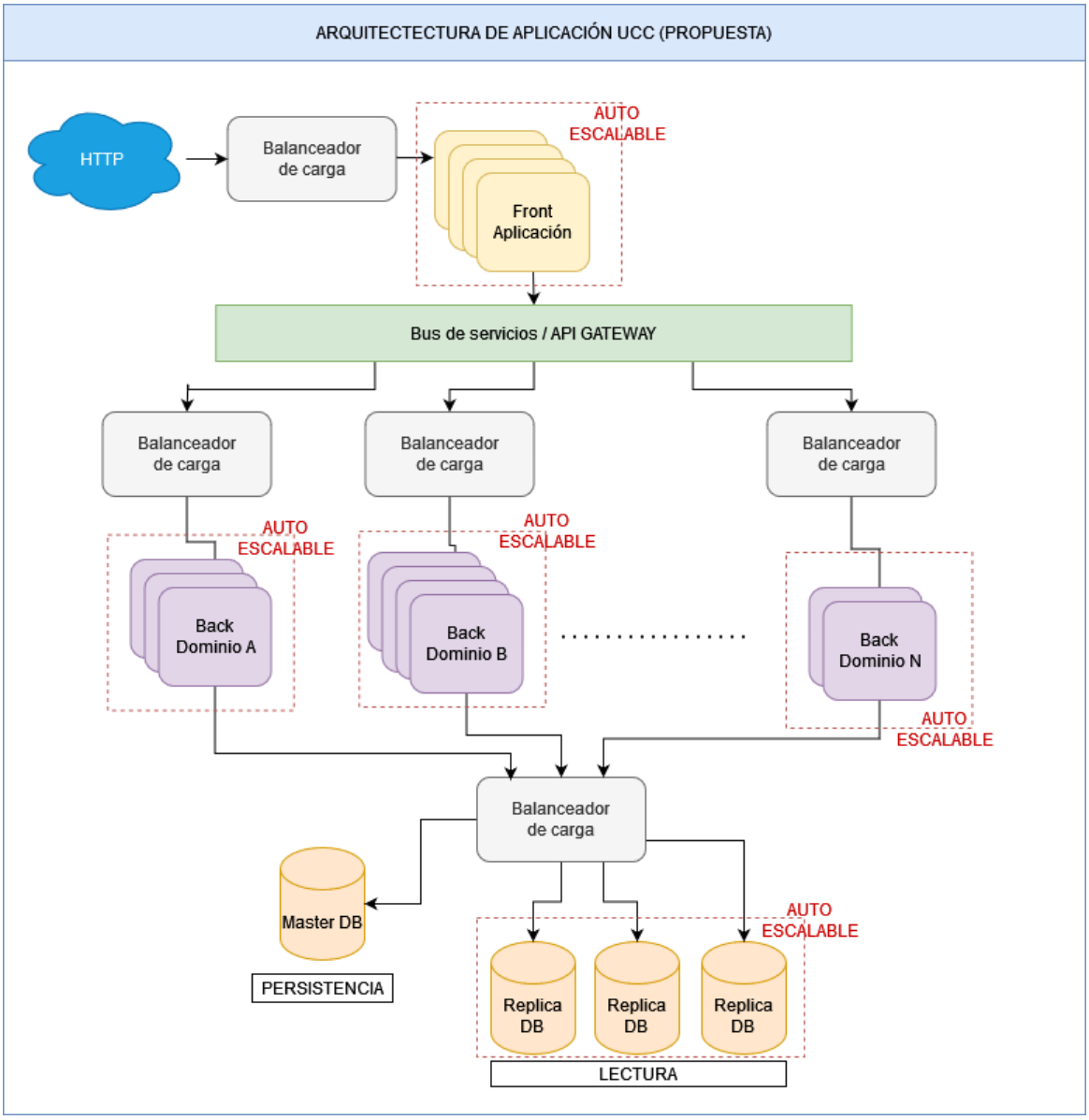
El servidor del motor de base de datos estará alojado de manera única e independiente en un servidor, a este al ser independiente se le puede implementar reglas de seguridad más estrictas ayudando a la seguridad de la información, de igual forma con sus réplicas en el caso de tenerlas. Cabe recalcar que la redundancia de la información en una base de datos en espejo es ideal para solventar posibles inconvenientes.

Arquitectura de la aplicación

Con base a lo anteriormente descrito se propone la siguiente arquitectura de aplicación como solución al sistema de la universidad, indicada en la figura 7.

Figura 7

Diagrama de la arquitectura de la aplicación (propuesta)



Fuente: Elaboración propia

Conclusiones

Una de las conclusiones relevantes es nunca buscar la mejor arquitectura, sino la más adecuada, intentado abarcar la menor cantidad de características para asegurar en cierto grado el éxito del software.

En las arquitecturas agiles, se puede tener una gran cantidad de versiones de la arquitectura, en la cual pueden cambiar algunos factores como el dominio, las capas, los conectores, y esto en base a los requerimientos de cada iteración.

Como indica Da Silva en su experiencia, no toda arquitectura es óptima para todo proyecto, cada una viene concebida en base a los requerimientos no funcionales y demandas del cliente sobre el funcionamiento y demás aspectos dignos de la misma.

Sin una arquitectura limpia el proyecto puede verse complicado en fiabilidad, costos de mantenimiento, dificultades para la integración de nuevo personal, desempeño y demás situaciones que comprometen todo el sistema.

No se encuentra literatura específica que determine la mejor arquitectura de software para cada proyecto, esta se define con el conocimiento y experiencia del analista o encargado, los diferentes ciclos de vida del software pueden comprometer la arquitectura, así como el desconocimiento de la misma por parte del equipo de desarrollo.

Existen sistemas que están diseñados para cambios imprevistos en ejecución, y estos deben ser resilientes al cambio o a los mismos imprevistos, para este tipo de paradigmas (SOS), existen enfoques que pueden ayudar en su modelado y concepción.

En base al análisis de lo que son las arquitecturas de software, se puede decir que dentro de la universidad no se ha analizado hasta la fecha la arquitectura del software en sí del sistema sino más bien el diseño.

El escalamiento horizontal ayuda de gran manera cuando la aplicación posee un alto número de usuarios concurrentes.

Referencias bibliográficas

da Silva, H. H. S., Carneiro, G. de F., & Monteiro, M. P. (2019). An experience report from the migration of legacy software systems to microservice based architecture. Advances in Intelligent Systems and Computing, 800 Part F1, 183–189. https://doi.org/10.1007/978-3-030-14070-0_26

Garlan, D. (2014). Software architecture: A travelogue. Future of Software Engineering, FOSE 2014 - Proceedings, 29–39. https://doi.org/10.1145/2593882.2593886

Jaiswal, M. (n.d.). Software Architecture and Software Design. https://ssrn.com/abstract=3948301

Le, D. M., Link, D., Shahbazian, A., & Medvidovic, N. (2018). An Empirical Study of Architectural Decay in Open-Source Software. Proceedings - 2018 IEEE 15th International Conference on Software Architecture, ICSA 2018, 176–185. https://doi.org/10.1109/ICSA.2018.00027

Miyachi, C. (2011). Agile software architecture. ACM SIGSOFT Software Engineering Notes, 36(2), 1–3. https://doi.org/10.1145/1943371.1943388

Mohsin, A., & Janjua, N. K. (2018). A review and future directions of SOA-based software architecture modeling approaches for System of Systems. Service Oriented Computing and Applications, 12(3–4), 183–200. https://doi.org/10.1007/s11761-018-0245-1

Nord, R. L., & Tomayko, J. E. (n.d.). Software Architecture-Centric Methods and Agile Development.

Oyedeji, S., Adisa, M. O., Penzenstadler, B., & Wolf, A. (n.d.). Validation Study of a Framework for Sustainable Software System Design and Development.

Richards, M., & Ford, N. (2020). Software Architecture an Engineering Approach.

Wang, J., Song, Y.-T., & Chung, L. (2005). From Software Architecture to Design Patterns: A Case Study of an NFR Approach.

Yong, D. (2021). Design and Practice of Software Architecture in Agile Development. Journal of Physics: Conference Series, 2074(1), 012008. https://doi.org/10.1088/1742-6596/2074/1/012008

Conflicto de intereses:

Los autores declaran que no existe conflicto de interés posible.

Financiamiento:

No existió asistencia financiera de partes externas al presente artículo.

Agradecimiento:

N/A

Nota:

El artículo no es producto de una publicación anterior, tesis, proyecto, etc.