Estudio, implementación y evaluación de la arquitectura para una herramienta de integración continua de investigaciones en ingeniería de líneas de productos de software

Analysis, implementation and evaluation of the architecture for a continuous integration of research tool focused on software product line engineering

MAURICIO AGUDELO ZAPATA

TESINA

Asesor, docente

PhD. Raúl Mazo Peña

UNIVERSIDAD EAFIT

ESCUELA DE INGENIERÍAS

MAESTRÍA EN INGENIERÍA

MEDELLÍN

2023

Tabla de contenido

Lista de ilustraciones

Lista de tablas

Lista de gráficos

Resumen

En repetidas ocasiones, las compañías se ven enfrentadas a tomar decisiones respecto a qué tipo de tecnología utilizar o a cómo rediseñar su base tecnológica; en algunos casos, dichas decisiones se toman de manera acelerada con el único afán de generar tiempos de respuesta óptimos y competitivos en el mercado con productos funcionales que satisfagan las necesidades puntuales del mercado. Bajo este enfoque, sin buscarlo, se incorpora una visión nula de extensibilidad, de crecimiento y de oportunidad de mejora de una gran cantidad de atributos de calidad, generando así altos índices de deuda técnica, acoplamiento entre componentes e índices nulos de reusabilidad. Todo esto provoca en las empresas productoras de software un mayor esfuerzo en mantener, extender o reconstruir los catàlogos de productos en un futuro cercano.

En este trabajo proponemos y evaluamos la arquitectura de una aplicación Web susceptible de permitir el modelado y la simulación de sistemas (de software y ciber físicos a varios niveles de abstracción) en el marco de un programa de investigación llamado VariaMos. El contexto de VariaMos, por ser un programa de integración continua de resultados de investigación en un marco de ingeniería, está caracterizado por un crecimiento constante de los lenguajes de modelado y de los mecanismos de razonamiento sobre los sistemas o las familias de sistemas que se creen y simulen con la aplicación Web asociada al programa (VariaMos). Por lo tanto,

la arquitectura de dicha aplicación web debe incorporar estándares adecuados para su correcta mantenibilidad, extensibilidad y crecimiento mediante la integración de las nuevas propuestas del equipo de trabajo de VariaMos. Para lograrlo, el proyecto de maestría que reportamos en este manuscrito siguió el siguiente proceso: Primero, identificamos los atributos de calidad de mayor relevancia siguiendo las recomendaciones presentadas en la norma (ISO/IEC 25000). Posterior a esto, construimos el marco referencial de la solución para con base en él derivar la arquitectura candidata para VariaMos, así mismo se detallará la construcción en diversas capas de la arquitectura requerida. El tercer paso consiste en familiarizar la propuesta ante en equipo de trabajo de VariaMos y realizar una retrospectiva de los beneficios de esta nueva propuesta con respecto a la versión precedente de la herramienta VariaMos. En el cuarto paso, implementamos la versión mejorada de la arquitectura y procedemos a realizar una evaluación de usabilidad de la aplicación Web resultante. Le evaluación pretende responder a la siguiente pregunta de investigación: ¿Cuáles son las características arquitecturales de una plataforma de modelado y de simulación de ingeniería que se pueda extender sin tener que redefinir la arquitectura de la plataforma a medida que los lenguajes de ingeniería o los mecanismos de razonamiento aumentan y en qué medida se pueden aplicar en una aplicación Web de trabajo colaborativo? Estos resultados constituyen una prueba empírica preliminar de los beneficios que esta nueva arquitectura ha aportado en favor de los objetivos estratégicos y técnicos de la nueva VariaMos y

nos dan una serie de recomendaciones que permiten mejorar aún más la herramienta. Para terminar, tomamos en consideración los resultados de la evaluación realizada para mejorar la aplicación Web que servirá de plataforma de trabajo para el grupo de trabajo de VariaMos en los próximos años y documentamos el nuevo resultado en este manuscrito.

Palabras clave: Extensibilidad, mantenibilidad, microservicios, atributos de calidad, arquitectura, calidad del software, líneas de productos de software.

Abstract

On repeated occasions, companies are faced with making decisions regarding what type of technology to use or how to redesign their technological base. In some cases, these decisions are taken in an accelerated manner with the sole aim of generating optimal and competitive response times in the market with functional products that satisfy specific market needs and which, in turn, without seeking to do so, incorporate a zero vision of extensibility, growth, opportunity for improvement and a large number of quality attributes that provide high quality to their products, thus generating high rates of technical debt, coupling between components and rates of improvement, incorporate a null vision of extensibility, growth, opportunity for improvement and a large number of quality attributes that provide high quality to their products, thus generating high rates of technical debt, coupling between components and null rates of reusability causing a greater effort to maintain, extend or even think about rebuilding their software capabilities in the near future.

In this work, we propose and evaluate the architecture of a Web application capable of enabling the modelling and simulation of systems (software and cyber-physical at various levels of abstraction) within the framework of a research software called VariaMos. The context of VariaMos, being a software of continuous integration of research results in an engineering framework, is characterized by a constant growth of modelling languages and reasoning mechanisms on the systems or families of

systems to be created and simulated with the web application associated with the software (VariaMos). Therefore, the architecture of this web application must incorporate adequate standards for its correct maintainability, extensibility, and growth through the integration of the new proposals of the VariaMos work team. To achieve this, the master project reported in this manuscript followed the following process: First, we identified the most relevant quality attributes following the recommendations presented in the standard (ISO/IEC 25000). Subsequently, we built the solution framework to derive the candidate architecture for VariaMos, and we will detail the construction of the required architecture in several layers. The third step consists of familiarizing the proposal to the VariaMos team and a retrospective review of the benefits of this new proposal with respect to the previous version of the VariaMos tool. In the fourth step, we implement the improved version of the architecture and proceed to perform a usability evaluation of the resulting web application. The evaluation aims to answer the following research question: What are the architectural features of an engineering modelling and simulation platform that can be extended without having to redefine the platform architecture as engineering languages or reasoning mechanisms grow, and to what extent can they be applied in a collaborative Web application? These results constitute a preliminary empirical proof of the benefits that this new architecture has brought in favour of the strategic and technical objectives of the new VariaMos and give us a series of recommendations that allow us to further improve the tool. Finally, we consider the

results of the evaluation to improve the web application that will serve as a working platform for the VariaMos working group in the coming years and document the new result in this manuscript.

Keywords: Extensibility, maintainability, microservices, quality attributes, architecture, software quality, software product lines.

Capítulo 1. Introducción

Motivación

El auge de la producción de software es cada vez más evidente e impactante en diversos entornos e industrias del mundo, como, por ejemplo, en los sectores financiero, académico y de comercio en línea. Es muy común ver como los productores de software amplían sus portafolios de productos, incrementan la cantidad de clientes que atienden y abren nuevas sucursales para expandir y consolidar sus negocios. A medida que crecen, muchos se preguntan si sus ofertas de software si están preparados para responder correctamente ante la alta demanda de sus clientes.

Planteamiento del problema

El mundo ha estado enfrentándose a una evolución constante a todo nivel, tanto en los productos y servicios que ofrece como en su cultura organizacional, la cual es impactada por tendencias ágiles que convierten a la producción de software en una industria acelerada, competitiva y con deseos de "ser los primeros". Pero ese querer ser los primeros ha producido resultados que, aunque previsibles, no creo que hayan sido los que se buscaban con tanto afán: ser los primeros en tener productos de software con un enfoque muy funcional, con ausencia de calidad y con poca claridad

anticipar el futuro y así poder satisfacer necesidades previsibles, pero no previstas (por ejemplo, hacer frente a amenazas de seguridad cada vez más hostiles y adaptarse a entornos de trabajo cada vez más colaborativos).

Si bien es cierto que si un software no hace lo que el usuario está esperando, puede no ser un software exitoso, también un software que no posee características de calidad adecuadas cómo lo son la eficiencia de desempeño, la seguridad, la extensibilidad o la mantenibilidad, tampoco puede llegar a ser usado de manera exitosa o dejará de ser usado en periodos de tiempo muy cortos. Esto se debe, en parte, a que incorporar nuevas capacidades funcionales o realizar ajustes a lo existente, resultará siendo demasiado costoso en términos de tiempo por la cantidad de código duplicado que pueda encontrarse, por el acoplamiento y dependencias generadas en el proceso de construcción del software, y por la ausencia de patrones de diseño de software. Todo esto terminará por determinar que el software tampoco cumplirá las expectativas de uso.

Lo anterior se intensifica aún más, cuando el contexto del software, como en el caso que se evaluará en esta ocasión, posee en su esencia el hecho de estar expuesto a una comunidad de investigadores que constantemente están incrementando y refinando sus capacidades funcionales. Esta dinámica de trabajo incita a la modularidad excesiva, es decir, a crear monolitos, que a su vez incrementan el tiempo de entendimiento para poder ser analizados, modificados y probados. Tiempo que a su vez se deriva en costo del personal calificado para enfrentar cada

etapa del ciclo de vida del software, costos admirativos, costos de plataforma, costos de un probable desperdicio de código que no cumple las expectativas o dejó de ser vigente para las necesidades de negocio.

Es precisamente allí, donde los pilares de extensibilidad y mantenibilidad se convierten esenciales para evidenciar las capacidades del software para ser modificado efectiva y eficientemente ante necesidades evolutivas o correctivas.

Preguntas de investigación

Es por lo anterior que esta tesina plantea las siguientes preguntas:

- ¿Cuáles son las características arquitecturales de una plataforma de modelado y de simulación de ingeniería que se pueda extender sin tener que redefinir la arquitectura de la plataforma a medida que los lenguajes de ingeniería o los mecanismos de razonamiento aumentan?
- ¿En qué medida se pueden aplicar las características de arquitectura en una aplicación Web de trabajo colaborativo?

Hipótesis de investigación

Adicionalmente a construir una arquitectura que permita reducir el impacto en desarrollo de una plataforma con problemas de extensibilidad y mantenibilidad, se espera que al incorporar una solución con principios arquitectónicos permita resolver

los retos esenciales del negocio, garantizar un crecimiento acelerado de la misma y generar los habilitadores técnicos requerido para que el equipo de trabajo pueda extender las capacidades de la plataforma.

Método de investigación

Este proyecto de investigación se ha desarrollado según las prerrogativas de la **ciencia del diseño de paradigmas**. Tres razones nos motivaron para seleccionar este método de investigación: (i) según (Rittel & Webber, 1973) es un método que tiene como objetivo tratar científicamente problemas llamados "espinosos", en los que a menudo se requieren fases exploratorias; (ii) es una metodología de investigación utilizada a menudo en las áreas de tecnología de la información basada en resultados y validaciones empíricas; y (iii) proporciona pautas específicas para desarrollar y evaluar proyectos de investigación.

Rittel y Webber (1973) argumentan que la **ciencia del diseño de paradigmas** permite el tratamiento científico de problemas como la planificación urbana, la gestión de procesos industriales y algunos problemas de las TIC (tecnologías de la información y de las telecomunicaciones) que no se prestan a la lógica analítica y lineal (Hevner, 2004). Estos problemas espinosos, que son típicamente (pero no solo) los del arquitecto o ingeniero de diseño, tienen características muy particulares (Ritchey, 2013):

i. Son fundamentalmente únicos.

ii. No pueden ser objeto de una formulación definitiva.

iii. Pueden explicarse y resolverse de muchas maneras diferentes.

iv. No involucran soluciones del tipo "verdadero o falso" sino del tipo "mejor o peor".

v. Soluciones que no pueden describirse exhaustivamente siempre tienen una serie de consecuencias que modifican el problema y requieren nuevas soluciones.

(Peffers, 2007) proponen una metodología para realizar investigaciones con el paradigma de la **ciencia del diseño de paradigmas** según el cual un proyecto de investigación puede dividirse en cuatro etapas: (i) diseño y desarrollo de soluciones, (ii) demostración, (iii) evaluación, y (iv) comunicación.

Por lo tanto, como un **primer paso** y basado en las mejores prácticas de la literatura, el estado del arte y el conocimiento de los colegas del laboratorio de acogida (GIDITIC), este proyecto propone el cambio de la arquitectura de una plataforma web, de manera que ésta permita el trabajo colaborativo y la integración continua de los resultados de investigaciones en ingeniería de líneas de productos de software de todos los miembros de un equipo de trabajo.

En un **segundo paso**, demostramos que la solución propuesta puede ser usada por los investigadores del grupo de trabajo para crear fácilmente nuevos módulos en la herramienta y que el uso de la herramienta escala correctamente a nivel de diversos aspectos definidos por el dominio de negocio

En el **tercer paso**, evaluamos la eficiencia y la usabilidad de la implementación de la solución.

Finalmente, en el **cuarto paso**, comunicamos y transferimos el conocimiento y los resultados del proyecto a través de publicaciones (esta disertación, por ejemplo), cursos universitarios y seminarios (en empresas y otras universidades que aún no hacen parte del grupo de trabajo de VariaMos).

Resultados

En el mundo de la tecnología existen diversas opciones para construir software. No podemos permitir que la elección entre tantos lenguajes de programación, tantas plataformas que cumplen con el mismo propósito y tantos modelos de arquitectura nublen las necesidades específicas de los negocios. Es por esto que se debe dedicar el tiempo suficiente a analizar de manera detallada el crecimiento a corto y largo plazo del software que se quiere construir o actualizar. Con esta investigación, se identificaron estrategias para una correcta adopción de arquitecturas modernas a un entorno específico de negocio como lo es el de las líneas de productos de software. Acorde a la hipótesis planteada en esta investigación, poco a poco los integrantes del equipo fueron adquiriendo conocimiento del modelo arquitectónico definido y se evidenció poco a poco cómo el crecimiento de la plataforma comenzaba a ser más sencillo y rápido.

Respecto a lo que se esperaba en contraste con los resultados, se logra apreciar que la eficacia, satisfacción y la estética, fueron los atributos que mayor impacto positivo tuvieron, por lo tanto, en los resultados se logra evidenciar que, adicionalmente de contar con una interfaz amigable y agradable, tanto antiguos cómo nuevos integrantes de la plataforma, podrán entender e interpretar con facilidad el modelo arquitectónico definido, como es su funcionamiento técnico y de negocio, y por ende lograr que la celeridad en el crecimiento de la plataforma sea más dinámico y veloz.

De lo anterior, se logra concluir que invertir en un enfoque arquitectónico desde el inicio, permite enfocarnos en aplicar estándares de calidad adecuados para las aplicaciones que se construyan y, lo más importante, que cumplan con los objetivos estratégicos a corto y largo plazo del negocio.

Conclusiones y trabajos futuros

El deseo de las compañías por implementar tecnologías emergentes y por siempre usar tecnologías modernas no puede impedir que se tenga una visión arquitectónica completa y holística de la solución de software. Al combinar una tendencia tecnológica y una arquitectura moderna con una visión estratégica del dominio (visión del negocio y de los dueños del producto), genera como resultado soluciones increíbles, diseñadas y creadas para perdurar en el tiempo, sin sobrecostos, sin esfuerzos adicionales y sin reconstrucción de software a corto plazo.

En esta ocasión, la reconstrucción de la solución VariaMos fue satisfactoria. Con el nuevo enfoque arquitectónico se habilitó la posibilidad de expandir las capacidades funcionales y técnicas llevando así a la nueva versión a un enfoque industrial, con la capacidad de soportar a largo plazo un gran número de usuarios y un gran número de nuevas características que anteriormente no eran posibles de incorporar a corto plazo en sus versiones precedentes. Algunas de las características que soportará serán: la gestión de identidades, compartir proyectos de líneas de productos entre usuarios, generar un entorno colaborativo entre usuarios y, permitir seguir creciendo bajo un enfoque de módulos periféricos conectados a los módulos centrales de la aplicación.

Organización del documento

El resto de esta tesina se organiza bajo la siguiente estructura: En el segundo capítulo se revisará el estado del arte de las arquitecturas que han marcado la industria del software y además las arquitecturas modernas que son aplicadas a entornos industriales de gran escala con el fin de plasmar un marco referencial y así poder seleccionar los elementos que aplican al caso de uso de líneas de productos de software. En el capítulo tres, se identifican los atributos esenciales de la solución, o también llamados drivers de arquitectura y se identifican los principios de arquitectura que impactan positivamente dichos drivers. Además, con dichos drivers y con un marco de referencia trazado, en dicho capítulo se revisará la propuesta de

solución, en la que estará incluido un conjunto de vistas técnicas cómo lo son el diagrama de arquitectura general, el stack tecnológico utilizado, los estilos y patrones de arquitectura, el modelo de base de datos y los requisitos de negocio (e.g., de la línea de productos) que permitirán que la solución cumpla con las necesidades de los interesados. En el capítulo cuatro, se evaluará la usabilidad de la arquitectura aplicada a una solución. Por último, en el capítulo cinco y seis, se evaluarán los resultados de la evaluación y posteriormente las conclusiones y trabajos futuros.

Capítulo 2. Estado del arte

Con el transcurrir del tiempo, la evolución de los negocios ha provocado que la industria del software se mueva al mismo ritmo, la demanda de los clientes y de las compañías por ofrecer nuevos servicios y capacidades de negocio está creciendo de manera exponencial año tras año, tal y cómo se puede evidenciar en estadísticas y reportes publicados por compañías como Forbes (Forbes, 2022), en el que se logra evidenciar que las compañías están cada vez generando más ingresos y por ende mayores ventas hacia los clientes finales. Pero ¿y su software? ¿Creció con las empresas? ¿Hicieron uno nuevo?

Algunas de esas preguntas serán revisadas a través de algunos modelos arquitectónicos que han marcado la industria del software tal como lo son: Cliente – servidor, Capas, SOA y Microservicios. Dichos modelos han sido revisados y probados por ingenieros en diversos y variados proyectos en todo el mundo (Butani, 2020).

Para entender la validez y el escenario especifico de uso de cada modelo arquitectónico es importante revisar cómo funciona cada uno, en qué escenarios se usa, y cuáles son sus beneficios, limitaciones y riesgos.

Arquitectura Cliente - servidor

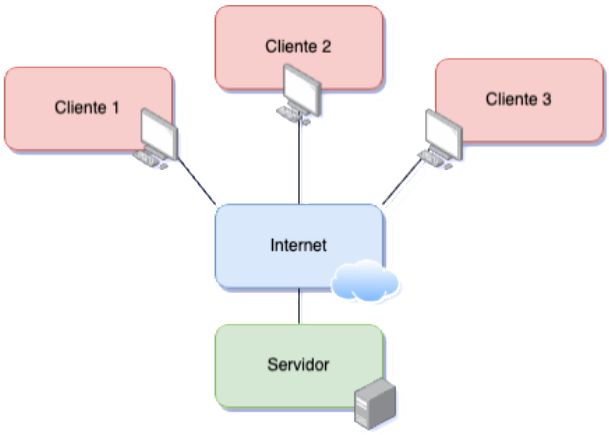

Ilustración 1. Arquitectura Cliente-Servidor

¿Cómo funciona la arquitectura cliente-servidor? En este modelo las tareas se distribuyen entre los servidores (que proveen los servicios) y los clientes (que demandan dichos servicios).

Adicionalmente, la comunicación entre el cliente y el servidor se hace a través de internet o, en el caso de entornos industriales, de una red privada.

Respecto a sus beneficios, podemos encontrar:

- Requiere poco presupuesto para implementarse.
- Fácil de configurar.
- Fácil de administrar.

Algunas de sus limitaciones son:

- Alta complejidad para realizar actualizaciones del software.
- Alto acoplamiento entre el alojamiento web y los datos.
- Alto costo al momento de escalar y extender las capacidades del software.

Riesgos

Existen dos riesgos a considerar a la hora de elegir o revisar este modelo arquitectónico. El primer riesgo se basa en la complejidad en la actualización de la versión de software de todos los clientes. Cuando es requerida una actualización de software, siempre será requerida una ventana de mantenimiento, es decir, el software no podrá estar disponible para su uso durante un periodo de tiempo determinado, lo que implica que, en algunos casos, se cuenta con un tiempo limitado para que el software esté offline, por lo tanto, existirán casos en los que no se logre el 100% de la actualización de los clientes.

Por otro lado, el segundo riesgo se basa en el acoplamiento generado entre el alojamiento web y los datos. A la hora de evaluar escenarios de caos, si revisamos el caso de esta arquitectura, existe una mayor probabilidad de fallo. Es decir que si falla el alojamiento web, todo el sistema fallará, si falla algún componente específico de la base de datos, seguramente todo el sistema fallará, o en casos en los que solo sea requerido el reinicio de algún

componente de la base de datos, se deberá reiniciar por completo todo el servidor. Al final del día, lo anterior representa más horas en las que el software no estará disponible para ser usado.

Arquitectura por capas

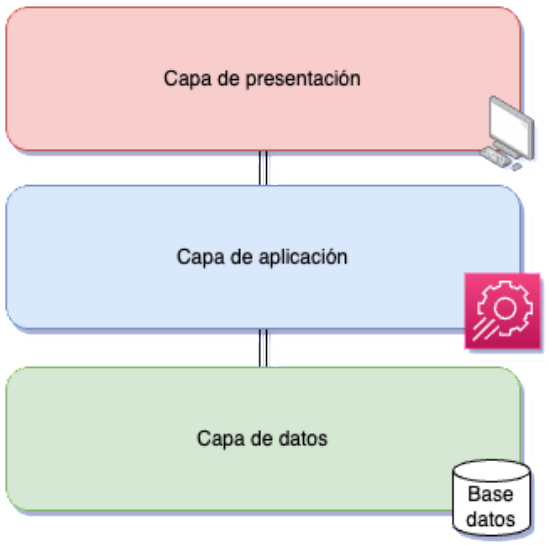

Ilustración 2. Arquitectura Capas

¿Cómo funciona? El modelo arquitectónico por capas toma su nombre de la segregación de responsabilidades propias de las capas principales de un software: la capa de presentación, la capa de negocio y la capa de base de datos.

Es muy común ver escenarios en los que los arquitectos de software aplican derivaciones a esta arquitectura; por ejemplo, dividir la capa de negocio en

la capa de reglas del negocio, la capa de la lógica de la aplicación, la capa funcional, la capa de acceso a datos, etc.

Respecto a sus beneficios, podemos citar:

- Es simple y fácil de implementar.
- Consistencia en el despliegue del código.
- Segregación de responsabilidades.

Algunas de sus limitaciones son:

- Alta complejidad a la hora de escalar.
- Puede ser difícil para mantener.
- Interdependencias entre cada capa.

Riesgos

Aunque la segregación de responsabilidades juegue un papel esencial en este modelo arquitectónico, la interdependencia que se genera entre cada capa convierte un beneficio en un riego, es decir, si bien cada capa está separada, si una de ellas deja de estar disponible debido a una falla o a un posible escalamiento de hardware, afectará a todas las demás capas, por lo tanto, se generarán índices de indisponibilidad.

Arquitectura orientada a servicios (SOA)

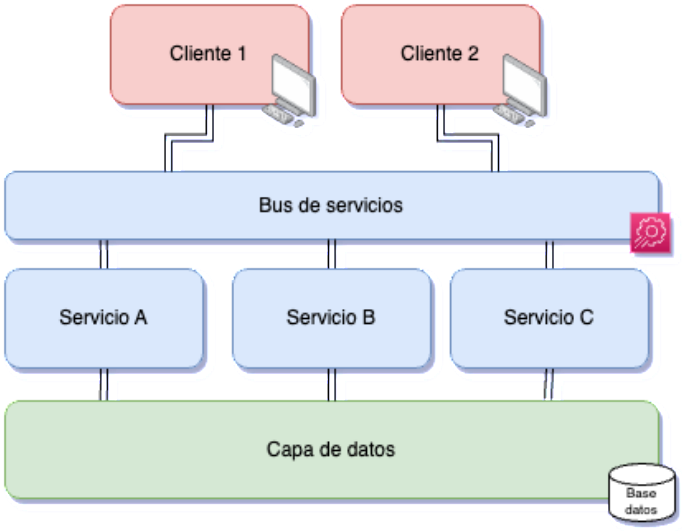

Ilustración 3. Arquitectura SOA

Cómo funciona? La arquitectura orientada a servicios es una de las arquitecturas que más se ha usado en la industria del software. Con el transcurrir del tiempo las compañías a nivel mundial han valorado la capacidad con la que dicha arquitectura permite habilitar una correcta sincronía entre las plataformas, los usuarios y los procesos de negocio. Mediante la estructuración de servicios, acompañado de los estándares de calidad adecuados, gran cantidad de compañías ha escalado significativamente.

Desde el punto de vista de los procesos de negocio, la arquitectura SOA se ha destacado por incorporar propósito y homogeneidad a los artefactos de software que se construyen, es decir, el concepto de "servicio" se convierte en "capacidad de negocio", lo cual quiere decir que no es solo un artefacto más de software, es una capacidad que atiende y soluciona una necesidad del negocio.

Respecto a su uso, su gran popularidad y excelentes resultados han permitido que sea posible evidenciar dicha arquitectura en gran parte de las industrias, como por ejemplo, en el sector de seguros, en el sector de las finanzas.

Respecto a sus beneficios, podemos encontrar:

- Alta reutilización.
- Escalabilidad mejorada.
- Oportunidad para desarrollar software de manera paralela.

Algunas de sus limitaciones son:

- Gobierno de los servicios.
- Inversión de tiempo inicial.
- Mayor carga y tiempos de respuesta.

Riesgos

Si no se estructura de manera adecuada desde su diseño y creación, algunos aspectos de su reutilizados se verán afectados debido al impacto que puede generar un mal diseño. Por ejemplo, la ausencia de gobiernos provoca que

existan servicios con propósitos similares o iguales, lo cual quiere decir que el índice de reúso se ve afectado y al mismo tiempo la escalabilidad del software mismo.

Al tener un índice de reúso alto, la actualización de las capacidades de negocio suele causar grandes retos a la hora de generar una actualización en los contratos de consumo, es decir, a mayor reúso, mayor índice de dependencia y a mayor índice de dependencia, mayor impacto en los tiempos de desarrollo y actualización.

Arquitectura Microservicios

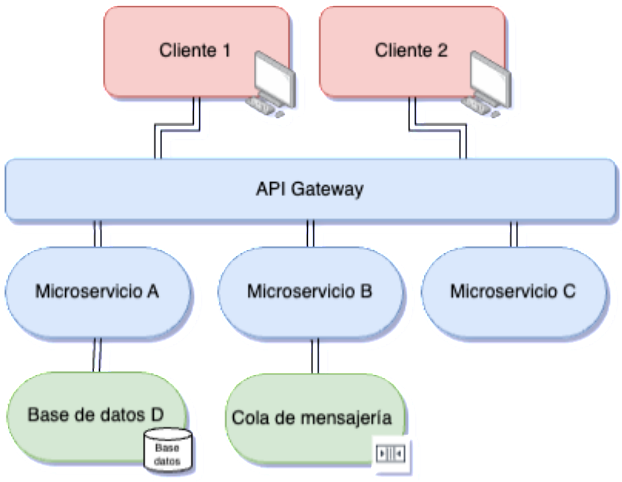

Ilustración 4. Arquitectura Microservicios

¿Cómo funciona? Gran parte de su funcionamiento consiste en las bases presentadas en la arquitectura SOA. Adicional a la segregación de responsabilidades y reúso, la arquitectura de microservicios es reconocida por la independencia de sus capacidades, es decir, cada capacidad de negocio expuesta mediante un microservicio está habilitada para funcionar sin tener ningún tipo de dependencia con otro componente, incluyendo componentes como bases de datos centrales, otros servicios u otros sistemas. Lo anterior garantiza el aislamiento de cada capacidad de negocio y permite un alto índice de disponibilidad y de resiliencia.

La arquitectura de Microservicios es comúnmente usada en escenarios que requieren de un índice alto de rendimiento, un índice alto de capacidades de negocios y en algunos escenarios en los que existan gran cantidad de equipos de trabajo.

Respecto a sus beneficios, podemos encontrar:

- Independencia en caso de fallo.
- Segregación de responsabilidades.
- Reusabilidad.
- Mayor agilidad.

Algunas de sus limitaciones son:

- Mayor esfuerzo en el despliegue.
- Alta complejidad en el diseño.

- Complejidad al crear escenarios de prueba.

Riesgos

La independencia de cada Microservicio trae consigo beneficios altamente valorados pero aun así, su complejidad en diseño y esfuerzo al requerido para su despliegue es el reto que se debe asumir, por lo general es usado en aplicaciones grandes, aplicaciones en las que la dependencia entre componentes genere fallas en el mismo. La falta de diseño y una definición correcta de su despliegue, provocará que la agilidad de su recuso y crecimiento disminuyan considerablemente.

Al día de hoy, es una arquitectura que sigue en crecimiento y evolución, lo cual significa que es debidamente necesario revisar frecuentemente las actualizaciones que surjan con el propósito de resolver errores o mejoras en cada uno de sus principios (Mantenibilidad, performance y resiliencia).
Habiendo revisado un poco los conceptos básicos de este estilo arquitectónico en el capítulo anterior, procederemos con una revisión desde el punto de vista de los beneficios que se obtendrán derivados de la adopción de este modelo arquitectónico y de los costos que esto trae consigo.

Tal y cómo lo menciona el ingeniero Martin Fowler, quien es el creador de este estilo arquitectónico: "Any architectural style has trade-offs: strengths

and weaknesses that we must evaluate according to the context that it's used", (Fowler, 2019). Lo anterior quiere decir que, cada estilo arquitectónico tiene un balance, por lo tanto, en la mayoría de los casos se observa que se debe sacrificar "X" para ganar "Y". Llevando lo anterior a un ejemplo, se puede entender como "X" el requisito de aumentar la seguridad de la información aplicando un algoritmo de cifrado a cierto conjunto de datos, por consiguiente, "Y", en este caso será el rendimiento de la aplicación el cual se verá afectado al incrementar el tiempo de procesamiento del conjunto de datos que ahora estará cifrado.

El estilo arquitectónico de microservicios no es la excepción a la regla anteriormente descrita, es por lo que a continuación se revisarán sus principales beneficios y el costo que se debe asumir por ello (o bien, pensar en cómo mitigar ese costo).

- **Fuerte modularidad y distribución:** La modularidad de software se ha convertido en uno de los pilares más grandes de este estilo arquitectónico debido a la estrategia que incorpora de desacoplar las capacidades de un dominio en artefactos de software, permitiendo así que el mantenimiento y crecimiento del mismo sea más eficiente y eficaz, no obstante, al contar con mayor cantidad de artefactos, se debe considerar la distribución de los mismos puede jugar en contra,

en primer lugar, en entornos en los que hay equipos pequeños, el mantenimiento puede requerir una inversión alta de tiempo para suplir el fin de mantenerlos y soportarlos, en segundo lugar, el costo que se debe asumir por el rendimiento al generar consumos entre artefactos de software, si el software no se construye correctamente, puede generar intermitencias, y por último, la estrategia de comunicación entre artefactos debe ser considerada, diseñada y creada de manera idónea para que este beneficio no se convierta en una desventaja.

- **Despliegue independiente y consistencia eventual:** La consistencia es uno de los grandes retos a la hora de elegir el estilo arquitectónico de microservicios. Al ser un estilo desacoplado en el que sus capacidades de negocio están desacopladas entre sí, se han evidenciado escenarios en los que los problemas de consistencia son generados por etapas anteriores del procesos, en decir, mientras un proceso "A" está siendo ejecutado, el proceso "B" , en el que sea requerida la respuesta del proceso "A", se ejecutará y fallará por la ausencia de datos. Vale la pena resaltar que las soluciones que aún son en su esencia un monolito, también deben superar este reto. Adicional a este reto, los monolitos deben superar los cambios y evoluciones del producto, incluso cuando son cambios pequeños, al estar altamente acoplados, pueden generar fallos fácilmente.

Por otro lado, el principio de independizar cada capacidad de negocio en un componente de software trae consigo un incremento en la celeridad de construcción de capacidades de negocio, la estructuración y delimitación del alcance de las capacidades, y además, altos índices de enfrentar cualquier cambio, bien sea grande o pequeño, en el producto de manera exitosa. Es allí donde el balance es evaluado. Al elegir este estilo arquitectónico, se debe asumir el reto de la consistencia, pero se ganará lo anteriormente descrito.

- **Diversidad tecnológica y complejidad operacional:** Día a día nacen nuevas tecnologías que ayudan a resolver problemas específicos de la industria, lo cual es muy positivo. No obstante, la complejidad no se da al momento de implementar dichas tecnologías, sino al momento de integrar la tecnología a una solución actual, es decir, en los casos en lo que se tenga un monolito, su integración podrá no ser exitosa. Caso contrario sucede con el beneficio de diversidad tecnológica que posee este estilo arquitectónico, en donde es posible implementar capacidades de negocio independientes con la libertad de elegir cualquier tipo de tecnología, siempre y cuando sea la adecuada para resolver de manera eficiente el reto en cuestión, sin causar algún problema de integración con otras capacidades. Un punto a favor es

adicional, es la celeridad al momento de definir que tecnología usar, debido a que no hay que pasar por procesos y análisis extensos de compatibilidad y mantenibilidad frente a las demás componentes de software previamente construidos.

En entornos empresariales, altos índices de libertad tecnológica trae más retos que beneficios. Los costos de operación y de soporte suelen elevarse al momento de aumentar una tecnología más a su contrato de soporte o equipo de operaciones. Adicionalmente, al momento de enfrentar actualizaciones o incremento en las capacidades ya construidas, la curva de aprendizaje de cada una de las tecnologías puede ser un costo alto en términos de tiempo para algunas compañías, cuya rotación de personal es media o alta.

Posterior a la revisión general de algunas de las arquitecturas más comunes en el mundo, podemos deducir que algunas empresas, debido a su complejidad de negocio, al tiempo requerido para una mejora técnica y debido a que su software "aún funciona", prefieren conservar su software actual; aceptando las implicaciones que esto trae consigo. Por ejemplo, en el caso de Woolworths en Australia (Evaluando ERP) (CNET, 2002), la empresa quiso integrar su sistema construido hace 30 años con uno reciente y eso resultó en un gran fracaso.

Por otro lado, en compañías en las que se logró un diseño detallado desde su concepción evidencian arquitecturas con capacidades para escalar al ritmo de las necesidades del negocio. Como por ejemplo, el caso de Amazon (Pothula, Medium, 2020) y Netflix (Pothula, Medium, 2020). En ambos casos, se construyeron arquitectura completamente desacopladas, en las cuales se logra evidenciar, que a la fecha, son arquitecturas que siguen vigentes, siguen creciendo y permiten la extensión de sus capacidades de negocio día tras día. Adicionalmente, en ambas se logra evidenciar que incluyen capacidades como lo son la escalabilidad, la mantenibilidad y la resiliencia.

Algunos de los retos que las compañías anteriormente mencionadas asumieron al usar estilos arquitectónicos cómo el de microservicio, fueron mitigados mediante el uso de servicios en la nube y una sólida estrategia de integración y despliegue continuo (CI y CD). Es por lo anterior que, a continuación, se describirá brevemente cómo estos apoyan a que la adopción de un enfoque moderno de arquitectura sea satisfactorio.

Servicios en la nube

Para la presente propuesta, evitar procesos extensos de configuración y administración son los principales motivos del uso de servicios en la nube. Además, encontrar capacidades al alcance de un par de clics, cómo lo son, servidores web,

variedad de motores de bases de datos o inclusivo colas de mensajería, en las cuales se puede implementar patrones y principios como lo son publicador-suscriptor (Microsoft Azure, s.f.), suelen ser más fáciles de implementar.

Para el caso de esta propuesta, el uso de servicios en la nube incrementará la celeridad de la construcción del software de manera en que los interesados podrán ver de manera rápida los cambios que se generarán día a día por los desarrolladores, es por ellos que, inicialmente, se hará uso de servicios como: servidores web, motores de bases de datos y gestor de contenedores, en cual ayudará enormemente a la estrategia de integración y despliegue continuo que será descrita a continuación.

Integración continua (CI) y despliegue continuo (CD)

La contenerización de componentes (Docker, s.f.) se ha convertido en el pilar de toda estrategia de CI y CD, debido a que incorpora la flexibilidad que se requiere al momento de tener una diversidad tecnología amplia, es decir, el concepto de encapsular únicamente los componentes requeridos para que una solución, o en este caso, un servicio, funcione adecuadamente ha sido un gran apoyo para que estilos de arquitectura como el de microservicios logre funcionar con éxito.

Adicionalmente, en escenarios en los que existe un alto índice de componentes distribuidos, no es una opción descartar el uso de esta estrategia, debido a que la capacidades de orquestar el despliegue de cada componente brinda seguridad y

mitiga la complejidad operacional que pueda generarse al administrar cada uno de los componentes construidos.

Capítulo 3. Selección de plataforma

Habiendo revisado lo anterior, durante la búsqueda de una plataforma que se hubiera enfrentado a cambios arquitectónicos y a constantes necesidades de negocio, fue elegida la plataforma VariaMos, debido a que, en su historia, cuenta con diversas versiones en la que en cada una de ellas posee un modelo arquitectónico distinto, cada versión con sus limitaciones y con sus motivantes del por qué se eligió dicho modelo arquitectónico en el momento de su construcción.

A continuación, se realizará una breve revisión del propósito de la plataforma y las diferentes versiones de la plataforma y se analizarán sus motivantes y sus puntos de mejora.

VariaMos

Para comenzar, el principal motivante de la creación de la plataforma, tal y cómo lo describen sus creadores (Mazo, Salinesi, & Diaz, 2012), fue la falta de métodos y herramientas para soportar el modelado, integración, razonamiento y configuración compleja en el dominio de las líneas de productos. Dicha ausencia, es más evidente cuando el modelo es compuesto por varias vistas de un mismo producto.

A lo largo de los años, la plataforma ha experimentado grandes cambios, ha escalado y ha tenido la oportunidad de tener a varios investigadores que han aportan su conocimiento para el crecimiento de la misma. Es por lo anterior, que a continuación

se describirán las principales características de las versiones 1.0, 2.0 y 3.0 de la plataforma, y además se revisarán sus limitaciones para extender y mantener la misma.

Versión 1.0. Plug-in Eclipse

La primera versión de VariaMos fue creada para ser instalada en el famoso software Eclipse (Eclipse, 2001), el cual fue creado en el año 2001 por la compañía IBM y suple la función de ser una interfaz de desarrollo integrada. A pesar de que se logró un funcionamiento a nivel del propósito de negocio: el modelado de requisitos de dominio usando modelos de características y el soporte automatizado para su subsiguientes verificación y configuración usando un solver de programación lógica por restricciones. Esta versión fracasó debido a sus altos índices de dependencia con los plug-ins de la plataforma; es decir, adicional a depender de la plataforma Eclipse en sí, también dependía de algunos plugin adicionales, cómo lo eran el Eclipse Modeling Framework (Eclipse EMF, s.f.) y el ATLAS Transformation Language (Eclipse, s.f.) para poder crear los lenguajes, construir los modelos y transformar los modelos construidos en programas lógicos y en programas de restricciones.

Limitaciones y retos:

- Al tener un alto índice de dependencia, la **extensibilidad** y la **mantenibilidad** de la plataforma se limitaba a las características que ofrecieran las dependencias.
- Mitigar la dependencia con plataformas o plugin adicionales fue el reto a tener en cuenta para evolucionar hacia la versión 2.0.

Versión 2.0. Java Stand Alone.

En la segunda versión de VariaMos se logró independizar un poco más la plataforma, eliminando así gran parte de la dependencia frente a Eclipse. En la presente versión, logramos evidenciar cómo se adapta la solución a una arquitectura tipo cliente-servidor, tal y cómo es expuesto en (Mazo, Muñoz Fernández, Rincón, Salinesi, & Tamura, 2015), en la que la solución es empaquetada en un archivo ejecutable para permitir la instalación del mismo en cada uno de los usuarios que desean usarlo. No obstante, el tedioso procedimiento que se debía seguir para que cada usuario pudiera añadir una nueva característica (o lenguaje) a la plataforma fue el que limitó su crecimiento e incitó a sus creadores a pensar en llevar la plataforma a otro nivel.

Limitaciones y retos:

- En esta ocasión la **extensibilidad** y la **mantenibilidad,** desde el punto de vista de la complejidad en los procesos, jugó en

contra de la aplicación, provocando así que los tiempos para mejorarla o extenderla fueran considerablemente altos.

- En el documento "VariaMos: an extensible tool for engineering (dynamic) product lines" (Muñoz Fernández, Second Guide to Define New Dynamic Operations in VariaMos, 2015) y en la guía para definir nuevas operaciones dinámicas (Muñoz Fernández, Second Guide to Define New Dynamic Operations in VariaMos) se logra evidenciar el tedioso proceso que se debía seguir para extender las capacidades de la plataforma.
- Simplificar el proceso de extensión de capacidades de la herramienta, disminuir la curva de aprendizaje para poder utilizar y extender la herramienta, además de aligerar el proceso de mantenimiento de la herramienta, fueron las principales motivaciones para crear la versión 3.0.

Versión 3.0. Web Page Vue JS.

En la tercera versión, se adoptó una arquitectura un poco más moderna, cómo lo podemos evidenciar en la sección de arquitectura del sitio oficial de VariaMos (VariaMos, s.f.). Construir la plataforma en una tecnología web, permitió que la plataforma mitigara y superara algunos retos de mantenibilidad y extensibilidad al momento de ser usado por los usuarios.

Fue así como nuevas capacidades fueron incorporadas a la herramienta; por ejemplo, la de "Control para sistemas continuos y discretos basados en OBJ-OBJ. Adicionalmente, fue incorporado un marco de trabajo sobre programación orientado a fragmentos (Correa, Mazo, & Giraldo), el cual impulsó el reto de especificar los componentes de dominio y de aplicación, además de generar una relación entre los requisitos y los componentes la cual permitía (i) obtener productos no solo a partir de los requisitos de dominio, sino también a partir del código correspondiente a los requisitos configurados, y (ii) derivar productos finales gracias los procesos asistidos de configuración y de personalización de los componentes de dominio correspondientes.

Por otro lado, a medida que aumentaban las extensiones, también aumentaba el índice de duplicidad de código y crecía el malestar de los desarrolladores por haber pasado más tiempo modificando el código de manipulación de la librería gráfica de la herramienta, que haciendo código bien estructurado que agregara nuevo valor a la herramienta.

Adicionalmente, se evidenció una considerable curva de aprendizaje de la librería gráfica de la herramienta, MxGraph (mxGraph, s.f.). Esta librería es usada para diagramar los objetos de los lenguajes, y de VueJS (VueJS, s.f.), el cual, en su momento, era un marco de trabajo que estaba en crecimiento y aún no contaba con una comunidad de desarrollo lo suficientemente grande y activa.

Limitaciones y retos:

- Al duplicar el código al interior de una aplicación, no solo se afecta la extensibilidad sino también la mantenibilidad. Esto es debido a que al requerir un cambio, bien sea mayor o menor, el impacto que éste tendrá en la solución de software será considerable; es decir, que requerirá de tiempo y de un detallado análisis para lograr un hallazgo correcto de los componentes que deben ser intervenidos.
- La mitigación de la duplicidad de código y ausencia de estructura a la hora de extender la herramienta fueron dos de los mayores motivantes para que el equipo de VariaMos pensara en una versión 4.0 de su plataforma.

Tomando como base lo revisado previamente, los siguientes dos capítulos serán claves para lograr el objetivo de mejorar la plataforma, en primer lugar, se revisará un proceso estándar para la documentación de la arquitectura, y en segundo lugar se construirá y revisará una arquitectura de referencia, la cual será pensada para garantizar una correcta proyección de crecimiento y adaptación de necesidades a futuro. El uso de ambas, la guía de documentación y la arquitectura de referencia, se encargará de guiar al equipo de VariaMos para identificar las verdaderas necesidades de negocio y construir una arquitectura escalable y mantenible.

Capítulo 4. Procedimiento para documentar una arquitectura

Adicionalmente a la revisión de arquitecturas y a la selección de la plataforma, en este capítulo, resolveremos una de las dudas que surgen al momento de enfrentar un problema, dicha pregunta es: ¿Por dónde empezamos?. Hoy en día, es muy común ver en escenarios industriales como equipos se enfrentan a problemas complejos de manera desordenada y/o aleatoria, es decir, no tienen ningún conjunto de pasos a seguir para llegar a una correcta solución o definición del problema. Por lo anterior, en el presente capítulo, se revisará un modelo estándar de documentación de arquitecturas a partir de las verdaderas necesidades tanto a nivel técnico cómo de negocio.

Para documentar la propuesta de modernización de VariaMos utilizaremos una guía llamada Arc42 (Starke & Hruschka, s.f.). Esta guía fue construida por dos personas que hacen parte de la Junta Internacional de Calificación de Arquitectura de Software (iSAQB, s.f.) con el fin de simplificar y estandarizar la comunicación efectiva, práctica y pragmática de la documentación de la arquitectura del software. A continuación será revisada cada una de las etapas que componen la guía de documentación de arquitectura.

Modelo ARC 42

En la siguiente gráfica, podemos observar que el estándar ARC42 (Starke & Hruschka, s.f.), consta de 12 etapas, las cuales serán revisadas brevemente a continuación:

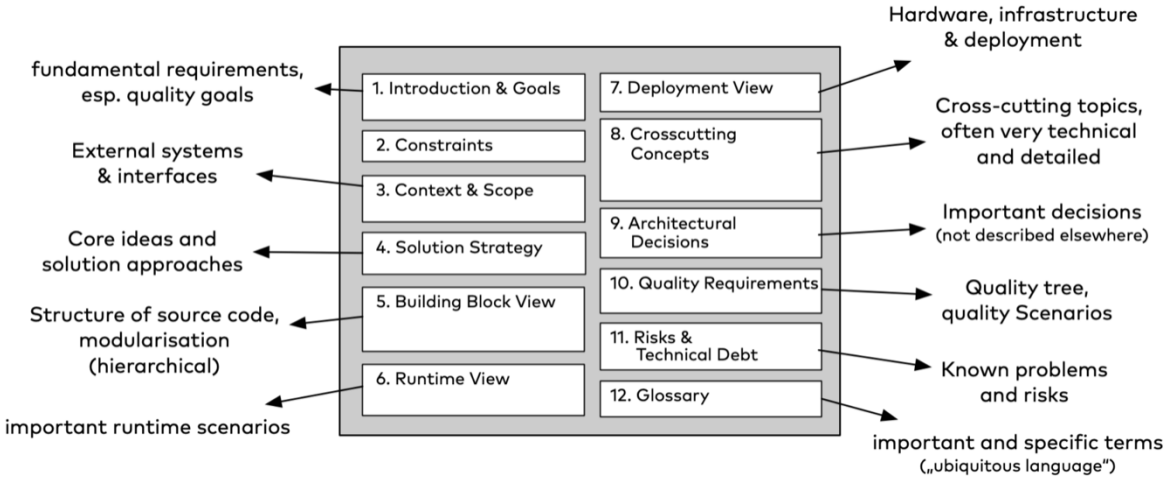

Ilustración 5. Proceso Arc42

- **Etapa 1: Introducción y metas.** Esta etapa consiste en describir brevemente los requisitos clave para el éxito de la arquitectura, adicionalmente, incluye la descripción de las metas de calidad esperadas y una lista con el nombre de los interesados y/o patrocinadores relevantes.
- **Etapa 2: Restricciones.** En esta etapa se debe especificar cualquier cosa que restrinja al equipo en términos de diseño, implementación o procedimiento.

- **Etapa 3: Contexto y alcance.** En esta etapa se debe especificar y delimitar las comunicaciones que se tendrán con sistemas internos, externos o inclusive usuarios. Se debe incluir una perspectiva de negocio y/o técnica.
- **Etapa 4: Estrategia de la solución.** En esta etapa se debe describir las decisiones fundamentales de diseño, la arquitectura que será implementada, las tecnologías que serán utilizadas y un acercamiento a cómo se lograrán las metas de calidad.
- **Etapa 5: Construyendo vista en bloques.** En esta etapa se especifica, mediante diagramas abstractos, una vista que permita identificar herencias y dependencias que posee el sistema.
- **Etapa 6: Vista del tiempo de ejecución.** En esta etapa se describen algunos de los procesos más relevantes, en cada diagrama construido se evidencian escenarios y su respectivo comportamiento e interacción con otros componentes o partes del proceso. Esta vista es muy valiosa a la hora de encontrar caminos alternos o excepciones que no son consideradas en primera instancia.
- **Etapa 7: Vista de despliegue.** En esta etapa se construye un diagrama que permita identificar los componentes de infraestructura, cómo lo son, equipos de cómputo, ambientes de despliegue (por ejemplo: desarrollo, pruebas y producción), topologías de red, software y demás componentes que sean valiosos para cada caso de uso.

- **Etapa 8: Conceptos transversales.** En esta etapa se debe especificar o diagramar los principales enfoques de la solución, desde el punto de vista de las regulaciones o relaciones entre componentes. Adicionalmente, se incluyen las especificaciones del modelo de dominio, estilos de arquitectura e inclusive, reglas para el uso adecuado de las tecnologías y su implementación.
- **Etapa 9: Decisiones de arquitectura.** En esta etapa son descritas las decisiones más importantes, las cuales pueden ser críticas, costosas, riesgosas o de gran escala. Cada decisión debe contar con su respectiva justificación.
- **Etapa 10: Requerimientos de calidad.** En esta etapa, por lo general se construye un árbol de calidad de alto nivel, el cual incluye escenarios expresados como requerimientos que impactan las métricas deseadas.
- **Etapa 11: Riesgos y deuda técnica.** En esta etapa se deben dar a conocer los riesgos técnicos o potenciales problemas que existen o pueden ser causados alrededor del sistema. Adicionalmente, si es del caso, especificar la deuda técnica que debe ser gestionada y ajustada posteriormente.
- **Etapa 12: Glosario:** En esta etapa, se debe definir cada término que esté sujeto a interpretación, por lo general son definidos términos de dominio y técnicos que son usado por los interesados para su correcto entendimiento e interpretación.

Habiendo revisado lo anterior, es de aclarar que este estándar fue diseñado para abarcar la mayor cantidad de documentación posible y con diferentes tipos de propósitos, así mismo, cómo para diversos interesados, tales como equipo técnico y directivos de negocio. Es por lo anterior que en el caso de esta propuesta, debido al número reducido de integrantes del equipo de VariaMos y a las especialidades que existen dentro de él, se han seleccionado las siguientes etapas: en primera instancia la etapa uno (introducción y metas) y tres (Contexto y alcance) serán revisadas en el capítulo 6, y las etapas cuatro (Estrategia de la solución), seis (Vista del tiempo de ejecución), siete (Vista de despliegue) y nueve (Decisiones de diseño) serán revisadas en el capítulo 7. Las etapas anteriormente descritas permitirán transmitir al equipo de VariaMos el nuevo enfoque de arquitectura y las prioridades funcionales y no funcionales con base en las cuales se deberán tomar las futuras decisiones.

Capítulo 5. Arquitectura de referencia

Antes de comenzar con las etapas establecidas en el capítulo anterior, en este capítulo se construirá una arquitectura de referencia, la cual es de vital importancia para el equipo por que permitirá tener un punto de referencia sobre que pueden llegar a usar y que no. Es por lo anterior que, este capítulo se construirá y revisará un modelo arquitectónico de referencia basado en un conjunto de tecnologías, principios y patrones de arquitectura usados en entornos industriales, y posterior a ello en el capítulo 6, se revisarán las etapas seleccionadas de la guía presentada en el capítulo anterior.

Explorando arquitecturas de referencia para soluciones industriales, como lo son el caso de Amazon (Pothula, Medium, 2020) y Netflix (Pothula, Medium, 2020), podemos encontrar que es utilizado el estilo arquitectónico de microservicios puesto que el principio de distribución brinda un valor excepcional e impulsa el crecimiento acelerado que tienen ambas plataformas. Adicionalmente, se logra identificar en sus arquitecturas que para mantener de manera adecuada sus plataformas, existen algunos módulos adicionales como lo son el de observabilidad y seguridad.

Adicionalmente, en ambos casos se identifican los requisitos funcionales y no funcionales, seguidamente se identifica un conjunto de herramientas que son usadas

y por último se describe cómo es usada cada herramienta para cumplir con el requisito.

Es por lo anterior que para esta propuesta, ha sido construido un marco referencial de arquitectura que permitirá garantizar tanto los aspectos funcionales cómo técnicos y necesidades de evolución constante, para esto, son requeridos algunos de los módulos que serán descritos a continuación:

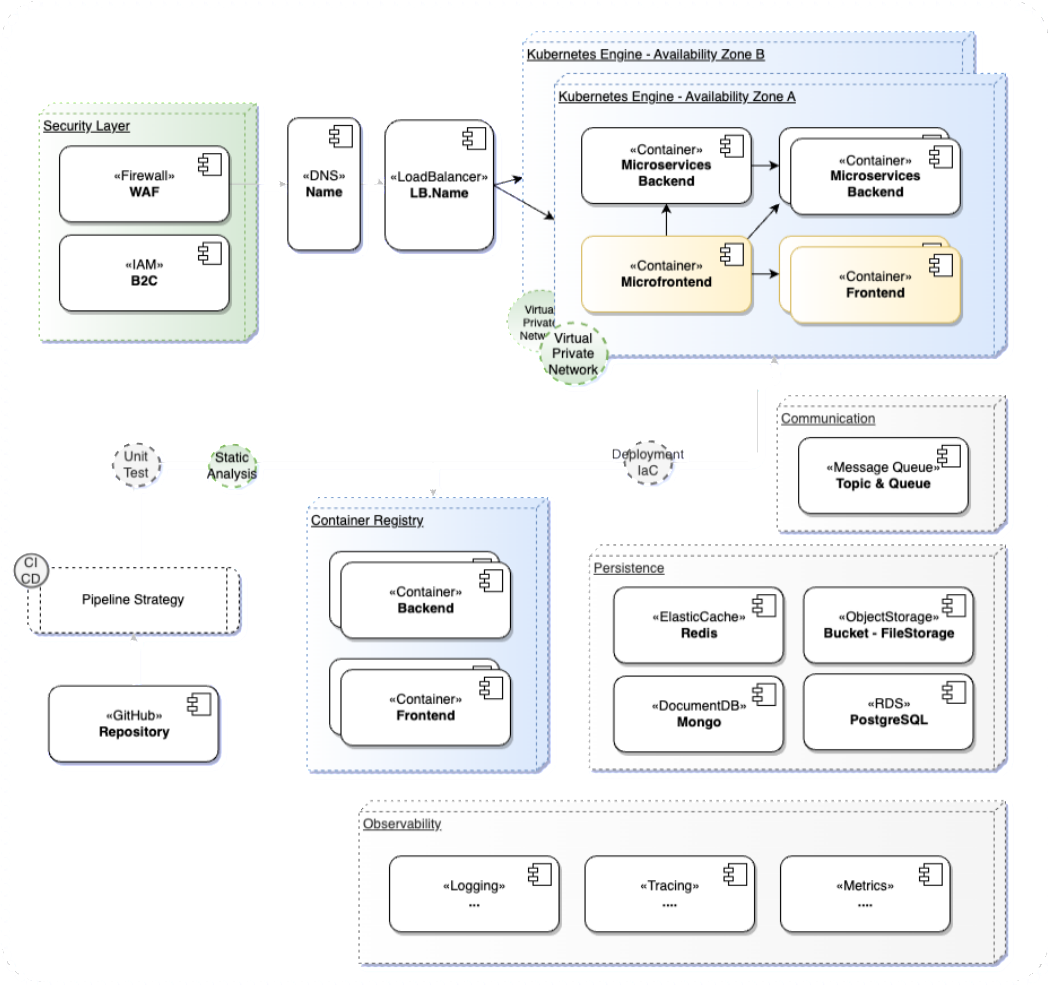

Ilustración 6. Arquitectura de referencia

En la ilustración 6, evidenciamos diversos módulos que impactan la seguridad, la eficiencia de desempeño, la mantenibilidad, la extensibilidad y la integración y el despliegue continuo.

Cada uno de los módulos satisface un propósito específico. A continuación, se revisará cada uno de los módulos del marco referencial propuesto, los cuales son seguridad, comunicación, procesamiento, persistencia, observabilidad y por último, integración y despliegue continuo.

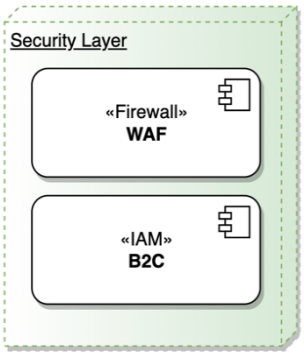

Ilustración 7. Módulo de referencia - Seguridad

- **Seguridad**: En este módulo se incluyen componentes como el web application firewall (WAF), el cual nos brinda protección ante ataques de denegación de servicio (DDoS) y el gestor de identidad, que brinda las capacidades de autenticación y autorización de uso a la solución.

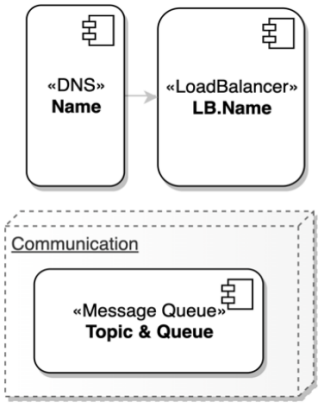

Ilustración 8. Módulo de referencia - Comunicación

- **Comunicación**: En este módulo, como primer punto de conexión, se incluye un gestor de DNS acompañado de un balanceador de carga con el fin de distribuir la carga que se proyecte a futuro y así evitar caídas de la solución;

por otro lado, se incluye un modelo de comunicación, enfocado en mensajería asíncrona impulsado por patrones de diseño como el publicador - suscriptor.

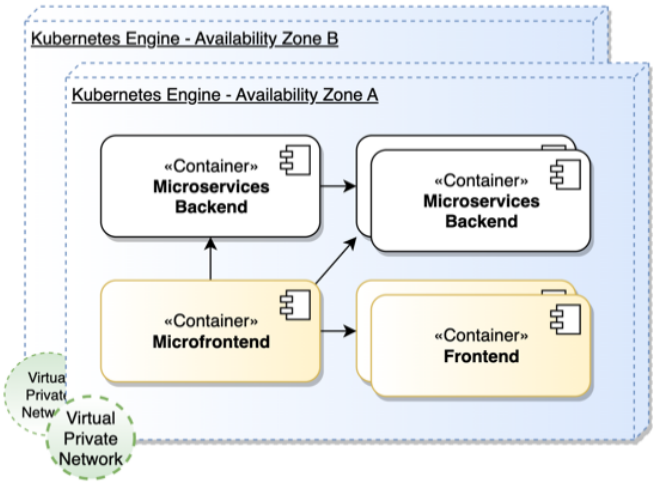

Ilustración 9. Módulo de referencia - Procesamiento

- **Procesamiento**: En este módulo encontramos un motor de kubernetes que incorpora capacidades elásticas (incrementan o disminuyen) y automáticas dependiendo de los niveles de demanda en periodos de tiempo específicos.

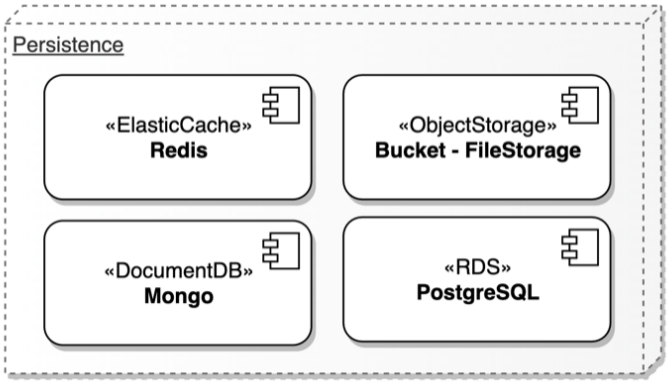

Ilustración 10. Módulo de referencia - Persistencia

- **Persistencia**: En este módulo encontramos diversos tipos de persistencia: desde información en caché para aumentar el rendimiento al momento de recuperar información, hasta motores de bases de datos no relacionales y relacionales. Todo esto con el fin de abarcar todo tipo de información que el negocio pueda llegar a necesitar.

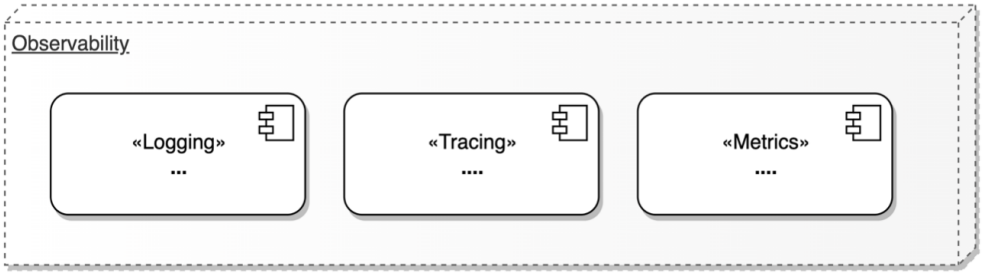

Ilustración 11. Módulo de referencia - Observabilidad

- **Observabilidad:** Este módulo es la encargada de correlacionar cada uno de los eventos y acciones generadas en el sistema, monitorear de punta a punta con el fin de identificar puntos de fallo y probables mejoras de la solución.

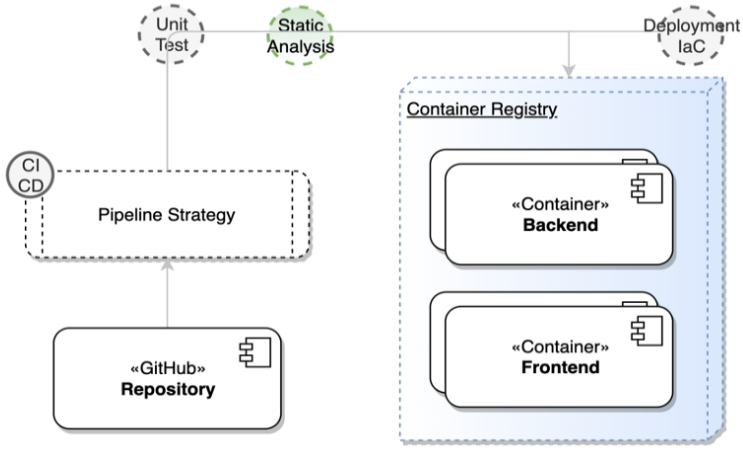

Ilustración 12. Módulo de referencia - CI y CD

- **Integración y despliegue continuo:** Este módulo se convierte en un pilar muy importante de la adopción de tecnologías modernas, como lo es el estilo arquitectónico de microservicios, debido a que dicho estilo requiere de precisión al momento de monitorear y desplegar cada uno de sus componentes.

Capítulo 6. Contexto y necesidades de negocio

Tomando como base la revisión de la arquitectura de referencia y las bases de la guía ARC42, a continuación, primer lugar los objetivos y las metas, seguidamente, el contexto y alcance de la solución. Adicionalmente, en el capítulo 7, será revisadas las etapas 3, 4, 5 y 6, las cuales consisten en, definir la estrategia de la solución, definir la vista de tiempo de ejecución, definir la vista de despliegue, y recolectar las decisiones de diseño, respectivamente.

Etapa 1. Presentar el objetivo y sus metas.

La presente propuesta tiene como finalidad rediseñar y mejorar la plataforma del equipo de investigación de VariaMos, mediante un enfoque arquitectónico moderno y enfocado netamente en las necesidades del dominio. Es por esto, que a continuación se describirán los atributos de calidad que fueron considerados esenciales para el éxito de la versión 4.0 de VariaMos.

Los atributos de calidad que serán mencionados a continuación son altamente relevantes, debido a que estos serán considerados cómo los architectural driver, los cuales permitirán al equipo de VariaMos tomar decisiones y a realizar trade off de manera imparcial.

La prioridad número uno para el equipo de VariaMos, en términos de atributos, es la extensibilidad, la usabilidad y la mantenibilidad. En la ilustración 13 se relaciona cada atributo con su respectiva motivación.

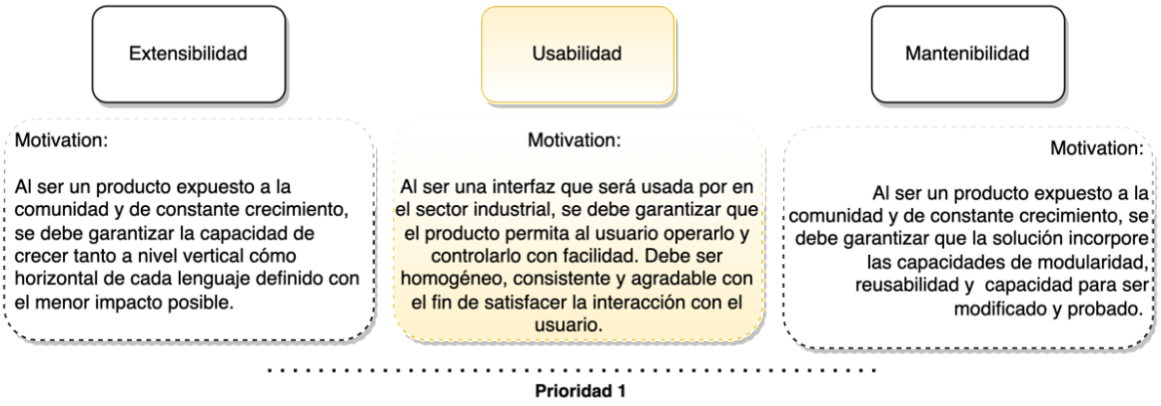

Ilustración 13. Atributos de calidad - Prioridad 1

En segundo lugar, la seguridad y la compatibilidad juegan un papel muy importante a la hora de extender y exponer a un entorno más grande e industrial a la plataforma VariaMos. En la ilustración 14 se relaciona cada atributo con su respectiva motivación.

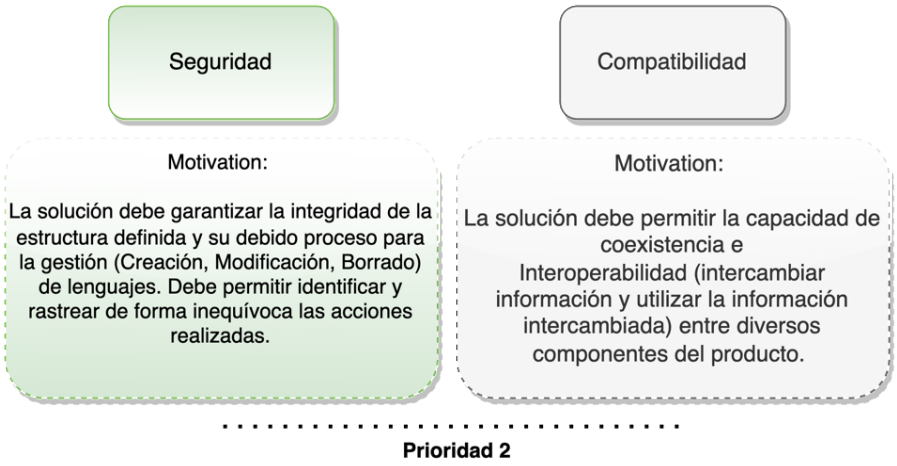

Ilustración 14. Atributos de calidad - Prioridad 2

En tercer lugar, la elasticidad y la eficiencia de desempeño, siguen siendo importantes, no obstante, si llegase a existir una decisión que impacte negativamente los atributos de prioridad 1 (Extensibilidad, Usabilidad y Mantenibilidad) o 2 (Seguridad y compatibilidad), a cambio de impactar positivamente atributos de prioridad 3 (Elasticidad y eficiencia de desempeño), dicha decisión deberá ser descartada. De igual forma, en la ilustración 15 se relaciona cada atributo con su respectiva motivación.

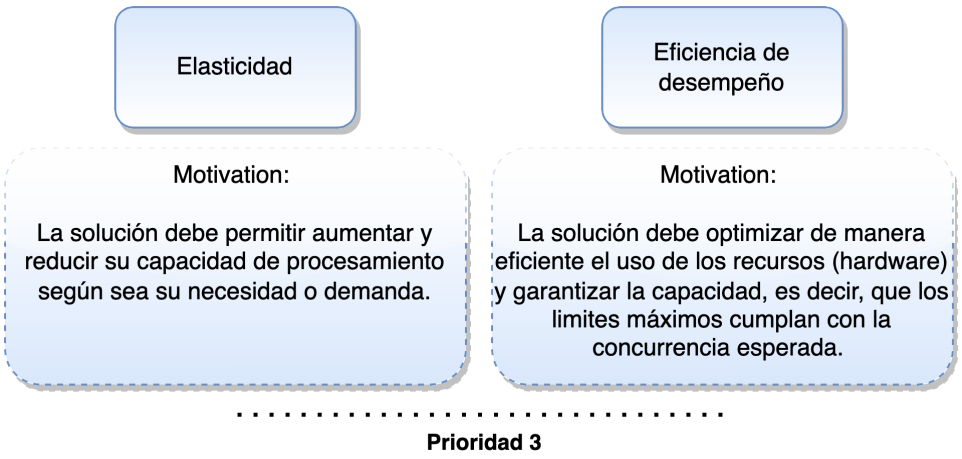

Ilustración 15. Atributos de calidad - Prioridad 3

En repetidas ocasiones, a raíz de la ausencia de contexto y prioridad de las necesidades de las compañías, se difumina el verdadero objetivo de un software, es decir, al combinar un enfoque técnico y de negocio derivado en un adecuado conjunto de atributos de calidad, las compañías lograran crear asombrosos productos con altas capacidades de durar, crecer y atender las necesidades de los usuarios que usaran el producto.

Etapa 2. Definir el contexto y alcance de la solución

Actualmente VariaMos cuenta con una comunidad activa, la cual está conformada por industriales e investigadores (tanto estudiantes como docentes universitarios) que constantemente extiende las capacidades y las funciones de la plataforma,

adicionalmente, el equipo crece constantemente y se espera llevar la plataforma a entornos industriales en los que puedan usar la plataforma de manera colaborativa. Por consiguiente, tal y cómo fue descrito en el capítulo anterior, la prioridad mayor en este caso será la extensibilidad, mantenibilidad y usabilidad.

Revisando un poco más en detalle las especificaciones del dominio, se han construido dos gráficos ilustrativos con el fin de traducir las necesidades de negocio a un enfoque un poco más técnico. El primero gráfico ilustra los contextos delimitados de la plataforma y el segundo gráfico ilustra la columna vertebral de estructuración de cada proyecto al interior de la plataforma. A continuación, ambos serán ilustrados y descritos.

Segregación de dominios

Cada que nace una nueva tecnología, esta se convierte en inspiración para otras personas a crear formas de abordar u optimizar la tecnología misma mediante un conjunto de prácticas, principios, métodos o marcos de trabajo. Tal y como lo fue el caso del diseño dirigido por dominios (Evans, 2003) , en el que encontramos una guía bastante completa sobre cómo podemos identificar y segregar correctamente los dominios de contexto determinado. Tomando como base los principios de segregación de responsabilidad y delimitación de contextos, a continuación se ilustra un diagrama de dominios para la plataforma VariaMos.

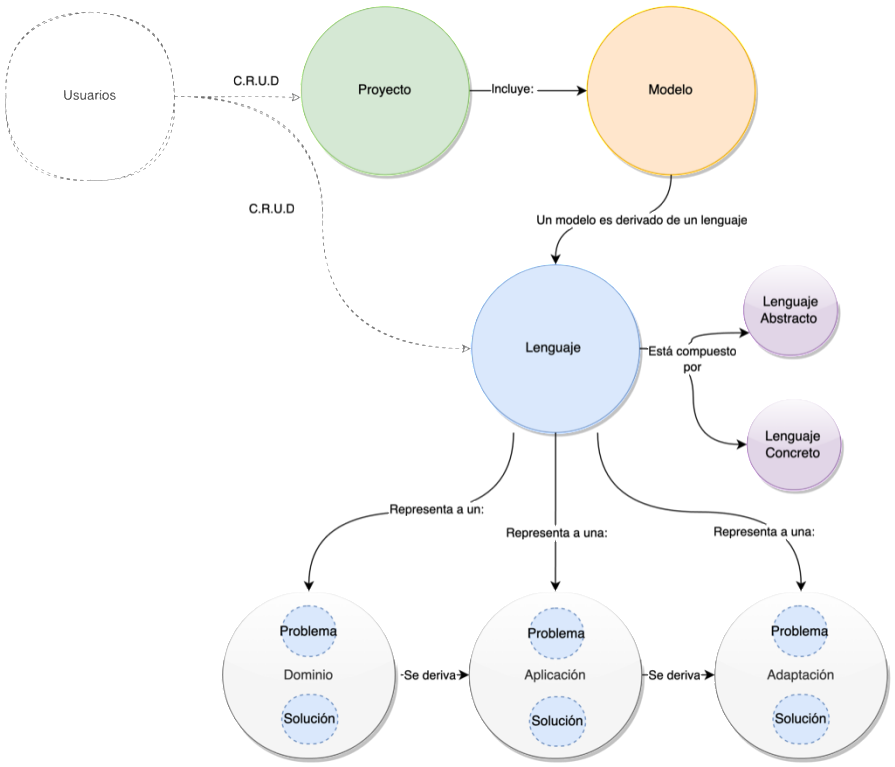

Ilustración 16. Diagrama de dominios

En la gráfica anterior, podemos observar diversos contextos de dominio, los cuales serán descritos a continuación:

- **Usuarios y proyectos**: Ambos dominios se vuelven relevantes debido a la proyección que tiene la plataforma de ser utilizada en entornos industriales, segregar ambos dominios en cada uno de forma independiente permitirá habilitar a futuro el trabajo colaborativo entre varios usuarios.

  En primer lugar, el dominio de usuario estará habilitado para interactuar con los dominios de proyectos y lenguajes, mediante la

creación, eliminación, modificación o lectura de los mismos. En segunda lugar, el dominio de proyectos será el encargado de concatenar cada modelo que un usuario desee crear.

- **Lenguaje y Modelo**: Ambos dominios, son el centro y la razón principal de las líneas de productos de software, el cual es el tema de interés y profundización del equipo de investigadores del equipo de VariaMos. La segregación de ambos dominios permitirá al equipo concentrarse en extender cada una de sus capacidades sin afectar lenguajes o modelos previamente construidos.

  En cuanto al dominio de lenguajes, podemos evidenciar que este estará compuesto por un lenguaje abstracto y un lenguaje concreto. Adicionalmente, este podrá representar un a un dominio, una aplicación o una adaptación. A su vez, un dominio podrá ser derivado en una aplicación y una aplicación podrá ser derivada en una adaptación. Por otro lado, evidenciamos el dominio de Modelo estará ligado a un proyecto y, además, este podrá ser derivado de un lenguaje existente.

Como resultados de independizar cada uno de los dominios previamente descritos, podremos habilitar y garantizar una correcta extensibilidad en cada uno de ellos. Vale la pena aclarar que, al ser dominios independientes, se mitiga la probabilidad de afectaciones entre dominios al momento de realizar actualizaciones o mejoras en cada uno de ellos.

Estructuración de proyecto

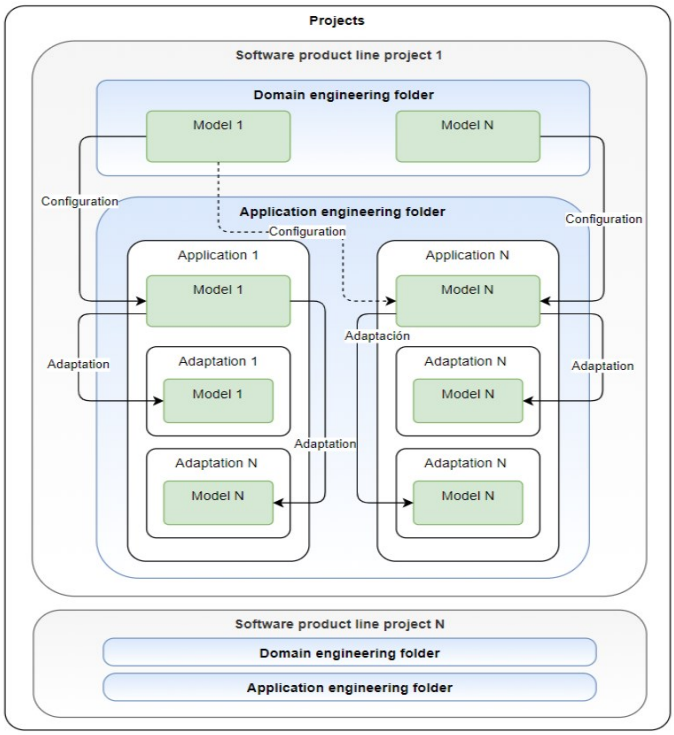

Ilustración 17. VariaMos - Gestor de Proyectos

Habiendo evaluado y segregado correctamente los dominios, en la siguiente gráfica observaremos cómo un proyecto incluye una o más líneas de

productos de software, las cuales a su vez incluyen por defecto una carpeta de domain engineering y una carpeta de application engineering. En cuanto a la primera carpeta, observamos como es posible tener uno o varios modelos derivados de lenguajes de dominio. En cuanto a la segunda carpeta, tenemos la posibilidad de crear aplicaciones (sub-carpetas), que a su vez nos permiten almacenar modelos derivados de lenguajes de aplicación. Posteriormente, la estructura habilita la creación de una sub-carpeta adicional para estructurar los modelos de adaptación, allí se podrán crear uno o más modelos derivados de un modelo de aplicación previamente construido.

Capítulo 7. Arquitectura Propuesta

Dando continuidad a la ejecución de la guía ARC42, y tomando como base la especificación de las etapas 1 (metas de calidad) y 2 (contexto de negocio) descrito en el capítulo anterior (6), a continuación se especificará la estrategia de la solución, un par de vistas en tiempo de ejecución, la vista de despliegue, y por último, las decisiones de diseño más relevantes que fueron revisadas en la construcción de la arquitectura de la versión 4.0 de VariaMos.

Etapa 3. Definir la estrategia de la solución:

Para la estrategia de la solución, como punto de partida se han considerado un conjunto de tecnologías, o también llamado Tecnology Stack, que apoyan los objetivos estratégicos y atributos de calidad definidos en etapas previas. Seguidamente, se expone un acercamiento al diagrama de arquitectura a alto nivel, el cual brindará una visión de la distribución de los componentes y el uso de las tecnologías.

Tecnology Stack

A continuación, inicialmente veremos de manera gráfica la selección de tecnologías y seguidamente una breve descripción de su motivante.

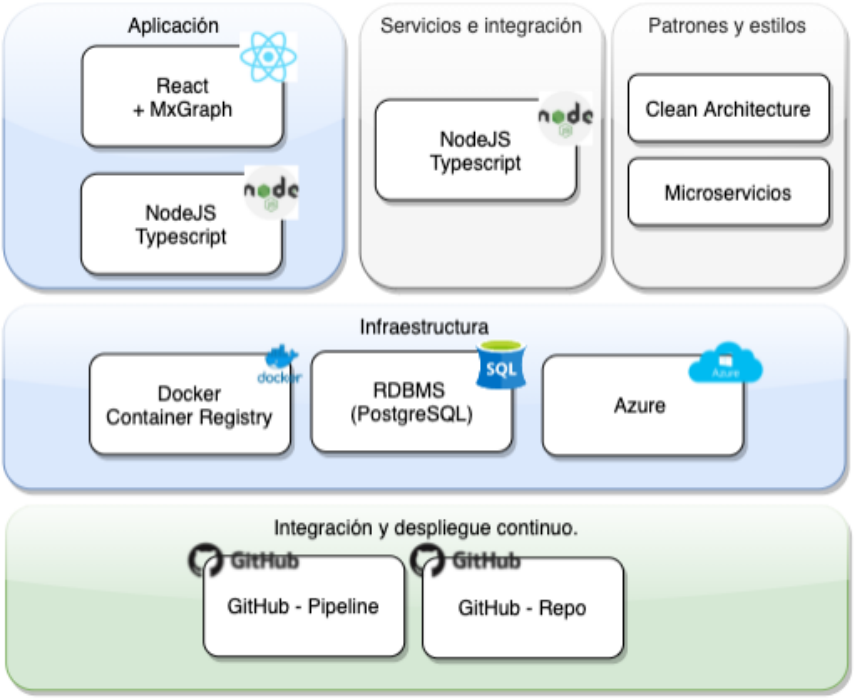

Ilustración 18. Propuesta - Technology Stack

- **Tecnologías de aplicación**: En la sección de aplicación es utilizado un marco de trabajo de frontend llamado React (React, s.f.) y una librería llamada MxGraph (JGraph Ltd, 2020) la cuál es utilizada para la construcción de gráficos, dicha librería permitirá que se construyan los modelos derivados de los lenguajes creados en la plataforma de VariaMos.
- **Tecnologías para los servicios e integración:** En esta sección se utiliza el motor de compilación de NodeJs (Nodejs, s.f.) en conjunto con el lenguaje de programación TypeScript (Microsoft Corporation, s.f.). Todos los

microservicios, inicialmente, son construidos bajo estos lineamientos. Es de aclarar, que debido a su alto índice de desacople, los servicios a futuro podrán ser construidos en cualquier lenguaje o plataforma que soporte comunicación HTTP.

- **Tecnologías para la infraestructura**: En esta sección, es utilizada la nube de Microsoft, llamada Azure (Microsoft Corporation, s.f.). Cada uno de los componentes (Frontend y Backend) es desplegado en un appservice (Microsoft Corporation, s.f.) independiente. Es usado un motor de bases de datos relacional llamado PostgreSQL (The PostgreSQL Global Development Group, s.f.) para resolver la persistencia Y además, cómo gestor de contenedores es usado DockerHub (Docker Inc, s.f.).
- **Tecnologías para la integración y despliegue continuo**: En cuanto a esta sección, se utilizarán las ventajas que el gestor de repositorios GitHub posee. Para este caso, utilizaremos GitHub Actions (GitHub, Inc, s.f.), la cual será de gran ayuda para garantizar que cada actualización que el equipo realice de manera exitosa a la plataforma, se vea reflejada en un ambiente que todos puedan acceder y usar.
- **Patrones y estilos de arquitectura:** Cómo lo hemos evidenciado en el transcurrir de esta propuesta, será utilizado el estilo arquitectónico de microservicios y adicionalmente serán utilizados gran parte de los principios expuesto en Clean Architecture (Martin, 2017), el cual es un marco de trabajo

que nos permite generar de forma estándar y ordenada un enfoque de dominio dentro de nuestro código.

Arquitectura a alto nivel

La propuesta de la arquitectura a alto nivel es derivada de las tecnologías previamente descritas. Para esta arquitectura, la cual es orientada a microservicios, abordaremos las capas de aplicación, lógica de negocio, persistencia, y por último, integración continua (CI) y despliegue continuo (CD).

La siguiente gráfica muestra la interacción de cada una de las capas entre sí. Seguidamente, se revisarán algunos motivantes para que fueron considerados para el uso de cada una de las tecnologías.

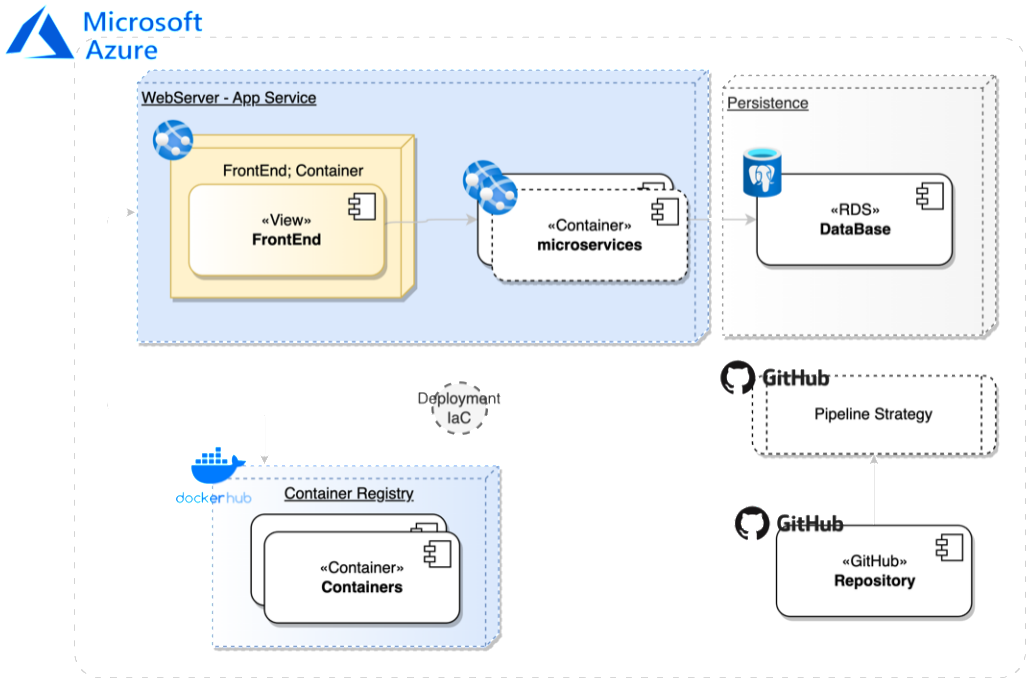

Ilustración 19. Propuesta - Arquitectura a alto nivel

A nivel de **infraestructura**, vale la pena aclarar, que tal y como se ve en el gráfico anterior, todos los componentes de software han sido desplegados en la nube de Microsoft, la cual es llamada Azure.

En cuanto a la **capa de aplicación**, esta capa se considera como la etapa de procesamiento, es aquí en donde se ejecuta la parte visual y lógica de la plataforma. El despliegue del frontend y de los microservicios se realiza en el servicio de servidor web de Azure, el cual es llamado AppService

Seguidamente, en la **capa de persistencia**, será elegida una base de datos relacional, debido a que la plataforma manejará un modelo de datos simple y reducido.

Por último, en la **capa de integración y despliegue continuo**, será usada la plataforma de DockerHub, la cual permitirá almacenar y desplegar los contenedores registrados en la plataforma de Azure, adicionalmente, será usada la característica GitHub Actions (de GitHub) para sincronizar los contenedores y los servicios de Azure con el fin de realizar el despliegue al momento de identificar un cambio en el repositorio de código.

Etapa 4. Definir la vista de tiempo de ejecución.

Cuando la complejidad de los procesos son altos, los diagramas de tiempo de ejecución brindan una visión un poco más clara de las interacciones entre componentes y usuarios del sistema.

Es por lo anterior, que a continuación se revisarán dos ilustraciones en las que veremos de manera gráfica el flujo de algunas de las transacciones y la interacción de los usuarios (estudiante y diseñador) con la plataforma, así mismo, la comunicación de los microservicios hacia los demás componentes de la plataforma.

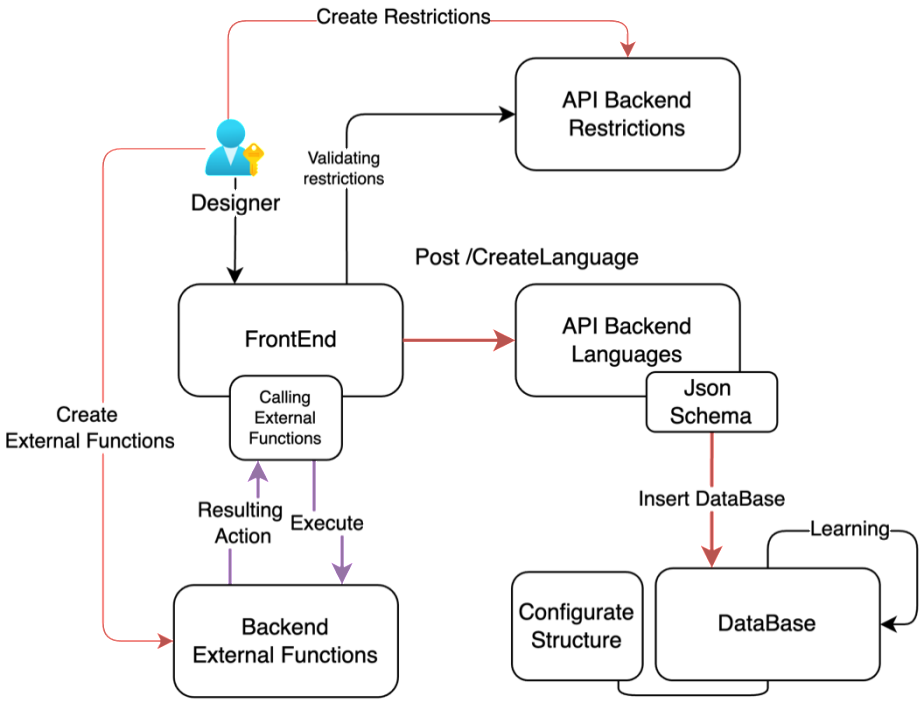

Ilustración 20. Propuesta - Diagrama tiempo de ejecución - Diseñador

Desde el punto de vista del diseñador, en la ilustración 20 podemos observar que este tendrá la posibilidad de interactuar con varias funcionalidades cómo lo son las restricciones, los lenguajes y las funciones externas, las cuales están construidas en microservicios independientes. Para cada funcionalidad, el diseñador tendrá la posibilidad crear, modificar, actualizar y eliminar capacidades. Adicionalmente, en

cuanto a las funciones externas, el frontend cuenta con un método que abstrae el consumo de servicios HTTP, lo cual quiere decir, que con un conjunto de parámetros la plataforma tendrá la capacidad de consumir microservicios externos, lo anterior fue diseñado y construido con el fin suplir la capacidad de extender a futuro cualquier tipo de procesamiento externo que un diseñador quiera realizar.

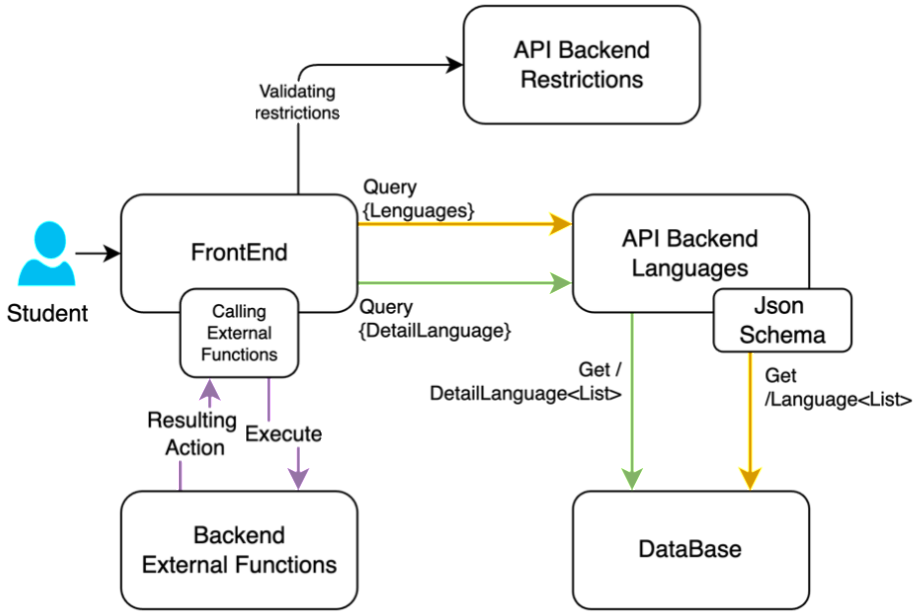

Ilustración 21. Propuesta - Diagrama tiempo de ejecución - Estudiante

Desde el punto de vista del estudiante, en la ilustración 21 podemos observar cómo este interactuará con un rol de lector y podrá consumir los microservicios de lenguajes y restricciones, los cuales serán usados en la medida en que el estudiante

use la plataforma. Adicionalmente, el estudiante podrá usar las funciones externas que son creadas por los diseñadores.

Con el fin de garantizar la correcta comunicación hacia los futuros integrantes del equipo de investigadores de VariaMos sobre el funcionamiento de la plataforma, se deberá velar por la construcción de diagramas como los anteriormente expuestos. Es de aclarar que este tipo de diagramas es recomendable construirlos en escenarios en los que existan conceptos clave y/o procesos complejos de entender.

Etapa 5. Definir la vista de despliegue.

En esta etapa se revisará la arquitectura definitiva de VariaMos, la cual será la versión 4.0, seguido a esta arquitectura se revisará la estructura de arquitectura limpia que fue agregada a cada microservicio construido.

Arquitectura VariaMos 4.0.

La arquitectura presentada a continuación habilita la versión 4.0 de la plataforma de VariaMos, en la presente versión fueron abarcadas las necesidades básicas y principales de versiones precedentes, cómo lo son la gestión de lenguajes, restricciones, capa visual, persistencia y gestión de despliegue continuo.

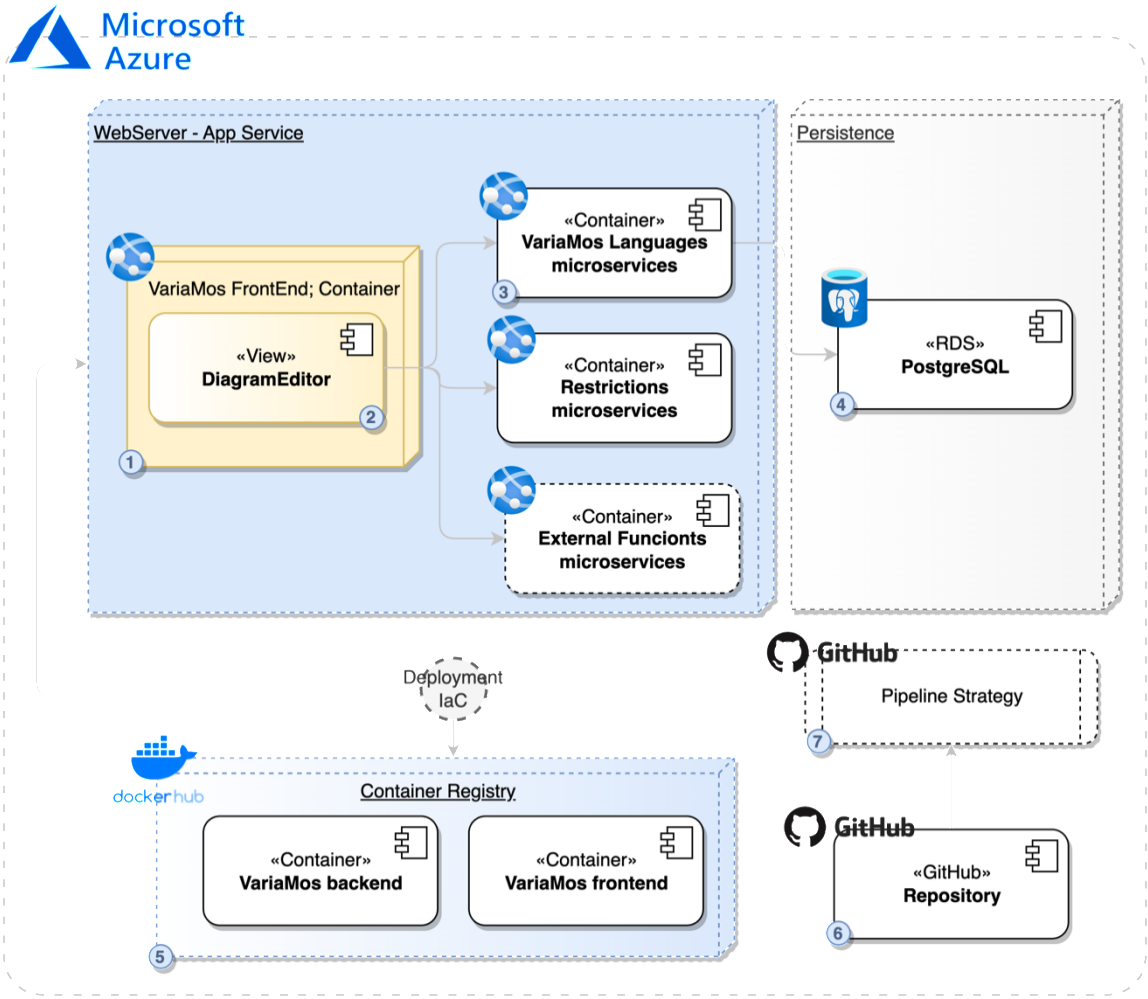

Ilustración 22. Propuesta - Arquitectura definida.

1. El servidor web es el que se encarga de la correcta ejecución del front-end y el back-end. En esta ocasión está siendo utilizado el servicio de App-Services dentro de la nube de Microsoft Azure.
2. El front-end, está siendo desplegado dentro de un App-Service independiente a los servicios del back-end.

3. En cuanto al back-end, inicialmente fueron creados 3 microservicios, los cuales son:
    i. Lenguajes: Es el encargado de gestionar todo lo relacionado a los parámetros de los lenguajes, es decir, insertar, modificar, leer o eliminar.
    ii. Restricciones: Es el encargado de validar las restricciones que han sido parametrizadas en el servicio cada que se ejecutan acciones en específico dentro de la acción de diagramar en la plataforma.
    iii. Funciones externas: Es un microservicio que fue creado como base para que futuros integrantes del equipo puedan tomarlo como plantilla y así crear sus propias funciones externas.
4. Toda la información concerniente a lenguajes, usuarios y restricciones están siendo almacenadas en una base de datos PostgreSQL, igualmente desplegada con los servicios de Azure. El acceso a esta base de datos solo está permitido a través de los microservicios.
5. El front-end y cada uno de los servicios del back-end tiene su propio contenedor, los cuales están siendo almacenados en DockerHub.
6. Está siendo utilizado GitHub cómo repositorio central del código fuente tanto del front-end como de cada uno de los microservicios.

7. Está siendo aprovechada la funcionalidad Actions que tiene GitHub cómo puente entre Microsoft Azure y GitHub para generar el despliegue continuo al momento de realizar un push en la rama master del código fuente.

Arquitectura limpia

La arquitectura limpia (Martin, 2017) se ha considerado un estándar de desarrollo de software por varios años. El incremento de la legibilidad del código, la fácil estructuración de los proyectos y la segregación de responsabilidades de cada uno de módulos son algunas de las ventajas que son aprovechadas al hacer uso de este estándar. A continuación, revisaremos la adaptación de esta estructura a los microservicios de VariaMos.

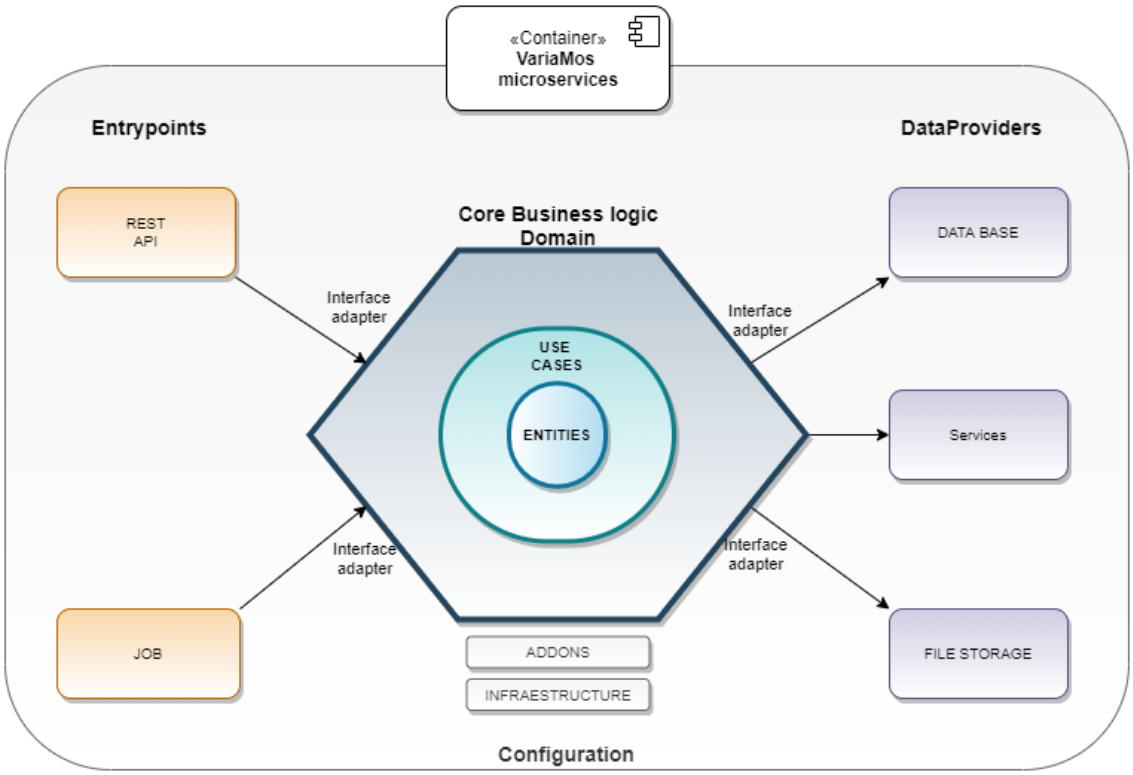

Ilustración 23. Propuesta - Arquitectura limpia

En primer lugar, tenemos el núcleo principal de la lógica del dominio, en esta sección debemos crear el dominio y dentro de él, debemos crear los respectivos casos de uso y entidades. Seguidamente, los proveedores de datos, cómo su nombre lo indica, son las interfaces que se encargaran de establecer la conexión con servicios o bases de datos externas.

Posteriormente, los puntos de entrada, tienen el propósito de exponer las capacidades construidas en el núcleo principal, por lo general se exponen mediante el protocolo HTTP o se ejecutan mediante tareas programadas.

Finalmente, de manera transversal tendremos una sección de utilerías e infraestructura, en estas serán configurados los archivos como el archivo de Docker, parámetros de configuración, o bien, librerías generales cómo generadores de identificadores, gestores de descarga de archivos o inclusive cifrado de información.

Etapa 6. Recolectar las decisiones de diseño

En la sexta y última etapa, se estructuran las decisiones que se toman en el transcurso del diseño de la solución; es decir, se evidencia el porqué de la selección de la tecnología que se usó, cuál fue su finalidad y se describe que otro tipo de tecnologías fue evaluada.

A continuación se revisarán algunas de las decisiones más importantes que surgieron durante el diseño y el desarrollo de la plataforma VariaMos 4.0. Inicialmente se revisará cómo se llegó a la conclusión de usar el estilo arquitectónico de microservicios, el marco de trabajo de React y arquitectura limpia, y para finalizar el por qué fue elegida la nube de Microsoft Azure.

¿Por qué microservicios?

Posterior a la revisión de los diversos estilos arquitectónicos que podemos encontrar en la industria, debido a sus beneficios respecto a su capacidad para crecer e independizar las capacidades de negocio, se tomó la decisión de seleccionar el estilo

arquitectónico de microservicios y aprovechar al máximo la segregación de responsabilidades que está implícita en su esencia.

¿Por qué React?

El punto de partida para la selección de un marco de trabajo para el front-end de la plataforma VariaMos fue la revisión de las tecnologías usadas en sus versiones precedentes. Posterior a dicha revisión, se ha encontrado que VueJS fue utilizado en la versión 3.0. Por lo tanto, se procedió a revisar fuentes como (State of JS, 2022), en las que se logra evidenciar estadísticas de diversos marcos de trabajo de front-end. Luego de revisar dichas estadísticas, se evalúa el ranking del enfoque de uso y manejo de cada marco de trabajo, dando como resultado que React lleva siendo el número uno por más de 5 años consecutivos. En conclusión, al ser un marco de trabajo que lleva muchos años siendo el número uno en términos de uso, significa que la curva de aprendizaje para usarlo y aprenderlo, es mucho menor respecto a otros cómo VueJS o Angular. Adicionalmente, al tener presente que VariaMos es un equipo que está en constante crecimiento y cambio de integrantes, se considera pertinente React debido a que esté disminuirá el tiempo de entendimiento del marco de trabajo a los futuros y actuales integrantes del equipo, dando como resultado un incremento en los índices de mantenibilidad (el cual es uno de los atributos prioritarios de VariaMos).

¿Por qué una arquitectura limpia?

Estandarización e incremento en los índices de mantenibilidad son las razones fundamentales para el uso de una arquitectura limpia. La reducción del entendimiento del código, la capacidad para extender, modificar o crear nuevas características en un microservicio se convierten vitales a la hora de contar con un equipo que está en constante crecimiento.

¿Por qué Azure?

En esta ocasión, la selección de la nube de Azure tuvo lugar a un aprovechamiento de un recursos existente, debido a que uno de los integrantes del equipo poseía beneficios en cuanto a los costos de mantener la plataforma completa de VariaMos funcionando en la nube.

No obstante, es muy común ver casos en los que se cuenta con diversas opciones y todas ellas pueden suplir las necesidades de negocio, es por tal motivo, que variables como el costo, la experiencia de los miembros del equipo o simplemente relaciones comerciales, son las que nos permiten tomar una decisión al respecto.

Capítulo 7. Evaluación

Protocolo de la evaluación

La evaluación de la presente investigación fue realizada mediante el protocolo de evaluación sugerido en el estándar internacional 2006-ISO-IEC-25062 (Software product quality requirements and evaluation for usability test reports, SQuaRE). Dicho protocolo aduce que la usabilidad de una aplicación computacional puede ser evaluada mediante cinco pasos. Cada uno de los pasos, aplicado a la versión 4.0 de VariaMos, son presentados a continuación.

Paso 1. Definir los objetivos de la prueba

- Evaluar en qué medida la creación de una línea de productos con la nueva versión 4.0 de la herramienta es más fácil respecto a la creación de líneas de productos con la versión precedente.
- Evaluar en qué medida la creación de un nuevo lenguaje de modelado con la versión 4.0 de la herramienta es más fácil respecto a la creación de lenguajes de modelado con la versión precedente.
- Evaluar en qué medida extender la versión 4.0 de la herramienta es más fácil respecto a la versión precedente.
- Evaluar los niveles de usabilidad de la plataforma.

Es de aclarar que se evaluarán únicamente las características anteriormente descritas debido a que la versión 4.0 de VariaMos sigue en constante crecimiento. Por ejemplo, las características de multiusuario y multiproyecto aún no pueden ser evaluadas puesto que están en construcción.

Paso 2. Descripción de los participantes

Para este estudio, se seleccionó a un conjunto de participantes que cumplieran con las siguientes características:

- Pertenecer al grupo de trabajo de VariaMos. Esta característica es importante para poder tener el punto de vista comparativo con la versión precedente de la herramienta que fue bien conocida y usada por los miembros del grupo de trabajo.
- Conocimiento intermedio en líneas de productos de software.
- Conocimiento intermedio en desarrollo de software.

Participante	**Rol**	**Años de Experiencia Variamos**	**Tipo de investigación**
1..5	Diseñador de líneas de productos. Desarrollador de lenguajes. Desarrollador Core.	0..10	Maestría o Doctorado

Paso 3. Definición de tareas

Contexto de uso del producto en la prueba

A cada participante le fue asignado un rol, el criterio de asignación del rol fue basado en el aporte que cada participante le generó y/o generará a la plataforma. Por tal motivo, las tareas especificadas en este paso están categorizadas dependiendo del rol asignado al participante. En particular, al rol de diseñador de líneas de productos le fue asignada la tarea "Crear una línea de productos" y al rol de Desarrollador de lenguajes, le fue asignada la tarea "Crear un lenguaje".

¿Por qué se seleccionaron esas tareas?

La tarea "crear una línea de productos" fue seleccionada para probar la facilidad de uso en la navegación, distribución de funcionalidad y el entendimiento intuitivo de su uso.

La tarea "crear un lenguaje" fue seleccionada para probar el grado de facilidad para crear nuevos lenguajes. Además, pretendemos tener el punto de vista de los participantes al comparar la creación de lenguajes con el nuevo VariaMos con respecto a la misma capacidad ofrecida por la versión precedente de la herramienta.

Adicionalmente, las tareas seleccionadas previamente, fueron seleccionadas porque son dos de las tareas principales que le brindan a la herramienta un constante crecimiento.

Criterios de cumplimiento de las tareas

La tarea número uno se considera exitosa cuando el participante logra evidenciar en su proyecto de líneas de producto, mínimamente, un modelo en el que se logre evidenciar uno o más elementos en el espacio de trabajo.
La tarea número dos, se considera exitosa cuando el participante reciba un mensaje "Language created successfully" al finalizar las acciones para crear un lenguaje.
La tarea número tres, se considera exitosa cuando el participante logra crear y publicar una capacidad nueva a la plataforma. La capacidad proyectada para la prueba fue la creación de la interpretación de un nuevo tipo/categoría de lenguajes; es decir, incorporar la capacidad de crear lenguajes escritos, quedando así la plataforma con la capacidad para crear lenguajes gráficos y escritos.

Escenario de la prueba

Durante la ejecución de la prueba, los participantes accedieron al sitio web en donde se encuentra publicada y funcionando la plataforma que fue diseñada e implementada con la arquitectura propuesta en este trabajo. Por

lo tanto, los dispositivos utilizados por los participantes durante la prueba fueron: computadores con sistema operativo Windows y MacOS, monitores y/o pantallas integradas y una conexión a internet

Herramientas del administrador de la prueba

Las tareas fueron evaluadas mediante un cuestionario. Dicho cuestionario se construyó basado en los pilares expuestos por la (ISO/IEC 25000), de inteligibilidad, operabilidad y estética, en el que, en primera instancia, se identificaron los aspectos personales de cada participante; es decir, nombre, rol asignado, tarea por evaluar, años de experiencia y tipo de investigación. Seguidamente, se identificaron los aspectos del entorno informático del participante; es decir, qué equipo de cómputo utilizó, si fue un dispositivo portátil o de escritorio, qué tamaño de pantalla y solución fue configurada, qué sistema operativo utilizó, qué navegador y qué versión de navegador utilizó, y para finalizar, se indagó sobre la duración total de la tarea, la completitud de la información suministrada, la facilidad de uso de la plataforma, la cantidad de errores identificados, y el nivel de agrado y satisfacción al usar la herramienta. Para el apartado final, los participantes calificaron la herramienta con una escala de Likert de 7 puntos e hicieron observaciones adicionales escritas y verbales.

Diseño experimental

El diseño experimental se diseñó basado en las características y libertades que la plataforma brinda por ser una plataforma 100% web; es decir, no se tuvieron en cuenta variables de control cómo por ejemplo sistemas operativos, hardware utilizado y espacio de trabajo, por el contrario, se evaluó la usabilidad y la experiencia en general de principio a fin con la herramienta. En dicha evaluación se identificaron variables cómo: grado de satisfacción, grado de eficiencia, duración de las tareas, tasa de ocurrencia de errores, tasa de error y tasa de éxito.

Procedimiento

1. Antes de iniciar la ejecución de las tareas, el participante deberá garantizar que cumple con los conocimientos catalogados cómo prerrequisitos para la ejecución de la tarea asignada.
2. Se asigna una tarea al participante.
3. Se realiza el envío de la documentación requerida para la ejecución de cada una de las tareas asignadas.
4. Se brindan indicaciones sobre el tiempo máximo para la culminación de la tarea, el cual será de una semana, iniciando a partir del cumplimiento de los prerrequisitos.

5. Al finalizar la tarea, se envía un cuestionario para recopilar información sobre datos personales, datos de contexto y medición de usabilidad.

Instrucciones para los participantes

Cada participante debe ingresar a la wiki publicada en el repositorio de código de GitHub de VariaMos. En dicho lugar se encuentra la descripción detallada de cada una de las tareas y el paso a paso para culminar de manera exitosa cada una de las tareas.

Se le informa a cada participante que, en caso de tener dudas, puede contactar al administrador de la prueba para dar solución a la misma.

Paso 4. Definición de las métricas de usabilidad

Las métricas utilizadas para la evaluación de la herramienta construida con el nuevo enfoque arquitectónico son:

Eficacia

Es la capacidad de producir u obtener un resultado deseado. Algo se considera efectivo cuando se obtiene un resultado previsto o esperado.

- Tasa de finalización:
    - Variables requeridas, número de tareas asignadas, número de tareas completadas parcial o totalmente.

- Resultado: Porcentaje de participantes que completaron correctamente cada tarea.

- Errores:
  - Variables: Cantidad de tareas ejecutadas. Cantidad de tareas que no fueron ejecutadas correctamente.
  - Resultados: Porcentaje de tareas que no fueron completadas o fueron parcialmente completadas.
- Ayudas:
  - Variables: Cantidad de veces que fue requerida ayuda / Cantidad de veces que no fue necesario requerir ayuda. ¿Requirió ayuda para terminar la tarea? ¿Requirió información adicional a la que se encuentra en la documentación?
  - Resultados: Porcentaje de tareas asistidas y no asistidas.

Eficiencia

Es la capacidad de lograr un objetivo final con la menor cantidad de desperdicio, esfuerzo o energía.

- Tiempo de duración de la tarea.
  - Variables: Tiempo total de duración de cada tarea.
  - Resultados: Promedio de ejecución de cada tarea. Desviación estándar del tiempo por cada participante (Tareas completadas dividido el tiempo de duración de cada tarea).

Satisfacción

Percepción subjetiva de cada participante a nivel de utilidad, apariencia y facilidad para ser usado.

- Variables: Calificación subjetiva de 0 a 7.
- Resultados: Promedio de las calificaciones.

Inteligibilidad

Se identificó la capacidad de la plataforma para permitir al participante entender si la plataforma es adecuada para resolver sus necesidades. ¿Se logró finalizar la tarea? ¿Cumplió el objetivo plasmado en la tarea?

- Variables: Calificación subjetiva de 0 a 7.
- Resultados: Promedio de las calificaciones.

Operabilidad

Se identificó el grado en el que la plataforma permite al participante realizar actividades con facilidad. Adicionalmente, se calculó el tiempo total promedio que duran los participantes para culminar la tarea.

- Variables: Tiempo total requerido para culminar la tarea
- Resultados: Promedio del total de cada participante.

Estética

Se identificó la capacidad que tiene la interfaz de la plataforma de agradar y satisfacer las interacciones del participante.

- Variables: Calificación subjetiva de 0 a 7.
- Resultados: Promedio de las calificaciones.

Paso 5. Ejecución del protocolo

Para la ejecución del protocolo, se ha utilizado un cuestionario creado en la plataforma de formularios de Google (Google, s.f.). El cuestionario fue dividido en tres secciones, la primera es información personal, la segunda es información de contexto y por último la recolección de las métricas de usabilidad. A continuación serán revisadas cada una de las secciones.

Sección 1: Información personal.

En esta sección, fue recopilada la información del nombre, el correo, tipo de estudio cursado y hace cuánto tiempo ha estado usando la plataforma de VariaMos. Lo anterior es relevante para la encuesta debido a que servirá como base para contactar al participante y adicionalmente para validar las perspectivas de una persona que lleva poco o mucho tiempo usando la plataforma.

Sección 2: Información de contexto

En esta sección, fue recopilada la información del rol que fue ejecutado en la prueba, que tipo de sistema operativo fue utilizado, qué tarea ejecutó y una breve descripción de componentes de hardware o software adicionales que fueron utilizado durante la prueba, como por ejemplo pantallas, audífonos, software específicos para pruebas, entre otros.

Sección 3: Métricas de usabilidad.

En esta sección, fue recopilada toda la información concerniente a los cálculos para hallar y calcular las métricas de usabilidad. Variables como cantidad de errores, duración de ejecución de la tarea, cantidad de horas dedicadas para la misma tareas en etapas precedentes de VariaMos, nivel de apariencia y aspecto físico, fueron algunas de las más relevantes para tener un resultado concreto de los índices de usabilidad de la versión 4.0 de VariaMos.

Adicionalmente a lo anterior, en la encuesta se suministra un conjunto de enlaces que direccionan al participante a un instructivo detallado con el paso a paso para ejecutar la tarea que ha seleccionado. Dichos enlaces fueron construidos con el fin de tener una guía estándar con el procedimiento de dichas tareas para la plataforma de VariaMos.

Para finalizar, se le informa a los participantes de la encuesta algunas variables que deben tener que calcular durante la ejecución de la tarea que les ha sido asignada. Algunas de las variables que deben calcular son:

- Calcule el tiempo que ha empleado en completar con éxito la tarea seleccionada.
- Cuente cuántos errores se encontraron durante la ejecución de su tarea.
- Cuente cuántas veces fue necesaria la ayuda del evaluador para completar su tarea.

Capítulo 8. Resultados

Posterior a la ejecución del protocolo de evaluación, a continuación se realizará un análisis de los datos recolectados y posteriormente se presentarán los resultados obtenidos.

8.1. Análisis de datos

El análisis de los datos obtenidos por los participantes de la prueba se estructurará inicialmente, revisando los datos recolectados, posteriormente la puntuación y reducción de los datos y por último, un análisis estadístico de los datos.

8.1.1. Recolección de datos

Respecto a la recolección, se observa que los datos que se estaban esperando obtener fueron los datos que los participantes diligenciaron en el cuestionario enviado, es decir, desde el punto de vista de los datos, no hubo ninguna diferencia entre los resultados esperados y los resultados obtenidos.

8.1.2. Puntuación de los datos

Para las preguntas en donde se ha evidenciado un porcentaje sobre el total de las respuestas, se ha utilizado la siguiente tabla para traducir un resultado cuantitativo a un resultado cualitativo. En la siguiente

tabla, podemos observar la división por cuartiles y la asociación de un resultado cualitativo por cada uno de ellos.

Tabla 1. Resultados - Puntuación de datos - Valor cualitativo

Cuartil	Valor cuantitativo		Valor cualitativo
	>	=<	
1	0%	25%	Pobre
2	25%	50%	Promedio
3	50%	75%	Bueno
4	75%	100%	Excelente

En cuanto a la puntuación de los datos, en las últimas 5 preguntas del cuestionario, se ha solicitado a los participantes seleccionar dentro de una escala de uno a siete un valor que identifique su inclinación al resultado. Para dichas preguntas del cuestionario el valor uno indicará que están muy en desacuerdo con la premisa expuesta en la pregunta y el valor siete indicará que están muy de acuerdo, adicionalmente, se ha añadido un valor cualitativo a cada respuesta, el cual se distribuirá de la siguiente manera:

Tabla 2. Resultados - Puntuación de datos - Escala de Likert

Escala de Likert	Valor cuantitativo	Valor cualitativo
Strongly Disgree	1	Muy negativo
Disagree	2	Negativo
Somewhat Disagree	3	Algo negativo
Neither Agree nor Disagree	4	Ni negativo ni positivo
Somewhat Agree	5	Algo positivo
Agree	6	Positivo
Strongly Agree	7	Muy positivo

Las demás preguntas del cuestionario fueron, en primera instancia, variables requeridas para cálculos posteriores de las métricas de usabilidad y en segunda instancia, datos informativos que le permitirán al equipo de Variamos tener una certeza de los participantes que aportaron al desarrollo de la prueba, por tal motivo, en el valor cualitativo de algunos resultados será "Informativo".

8.1.3. Reducción de los datos

A pesar de que el participante tuvo la opción de elegir entre dos tareas (Crear una línea de producto o crear un lenguaje), los datos no fueron divididos debido a que para ambas tareas las preguntas del cuestionario fueron las mismas, de decir, el principal motivante de tener dos tareas diferentes fue brindar un campo de acción más amplio para los diferentes roles que encontramos en el equipo de VariaMos.

8.1.4. Análisis estadístico

En cuanto al análisis estadístico, en la presentación de resultados se verá reflejado, en su gran mayoría, el promedio sobre cada conjunto de datos obtenido por pregunta. Adicionalmente, en algunas de ellas, se visualizará el valor mínimo y máximo obtenido de cada pregunta (siempre y cuando aplique).

8.2. Presentación de los resultados

8.2.1. Resultados de la sección 1:

A continuación es presentado el conjunto de datos recolectado por los participantes.

Tabla 3. Resultados sección 1.

You are a	Have you ever used variamos?
Master student	Less than 1 year
Master student	More than 2 years.
Teacher	Less than 1 year
Undergraduate student	Less than 1 year
Undergraduate student	Less than 1 year

A continuación es presentado de manera gráfica el promedio de las respuestas realizadas por los participantes.

- La siguiente gráfica muestra la distribución en porcentajes de cada tipo de estudio cursado por el participante.

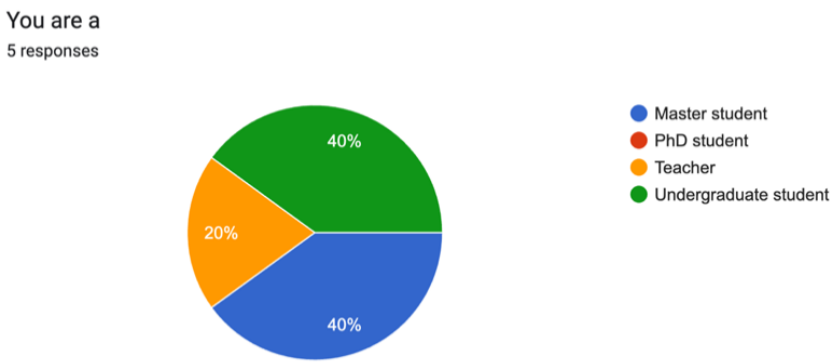

Gráfico 1. Resultados sección 1 - Tipo estudio

- La siguiente gráfica muestra la distribución en porcentajes del tiempo que los participantes han hecho uso de VariaMos.

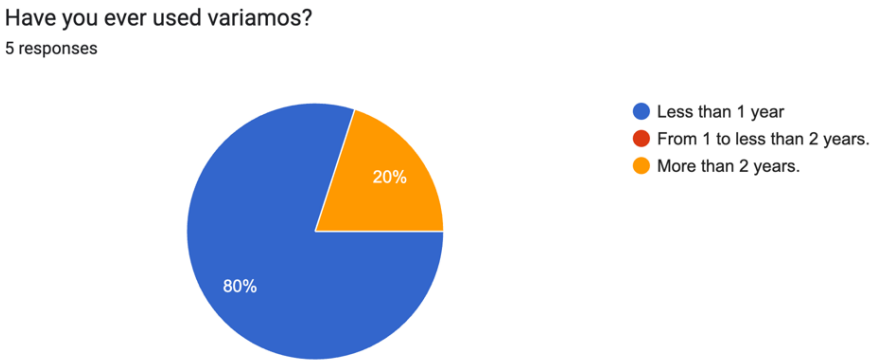

Gráfico 2. Resultados sección 1 - Uso VariaMos

8.2.2. Resultados de la sección 2:

A continuación es presentado el conjunto de datos recolectado por los participantes.

Tabla 4. Resultados sección 2.

Which of the following roles do you identify with?	Which of the following operating systems did you use during the test ?	Which of the following tasks did you do?	Describe briefly If you used another components (Hardware or Software) that you consider relevant during the execution of the selected task
Core developer	Windows	Create a product line	
Core developer	Windows	Create a language	GOOGLE CHROME
Language developer	Windows	Create a language	
Product line designer	MacOS	Create a language	Hardware running ARM64 Architecture (Apple M2)
Product line designer	MacOS	Create a product line	Hardware running ARM64 Architecture (Apple M2)

A continuación es presentado de manera gráfica el promedio de las respuestas realizadas por los participantes.

- La siguiente gráfica muestra la distribución en porcentajes del tipo de rol seleccionado por cada participante.

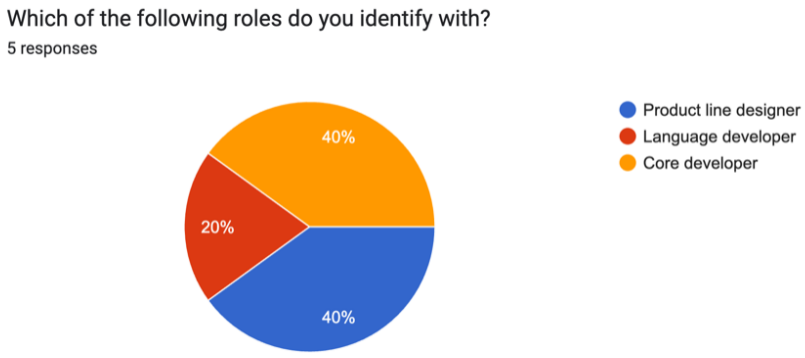

Gráfico 3. Resultados sección 2 - Tipo de rol

- La siguiente gráfica muestra la distribución en porcentajes del sistema operativo utilizado por cada participante.

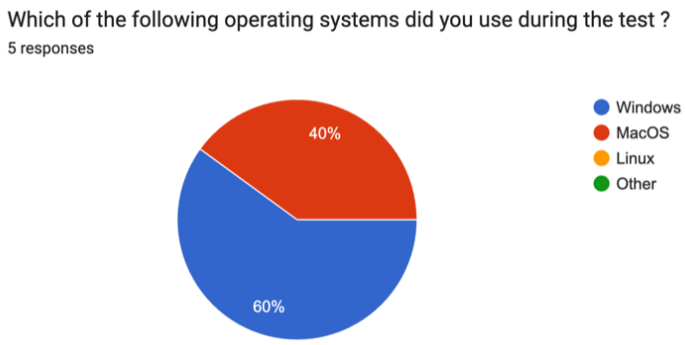

Gráfico 4. Resultados sección 2 - Sistema operativo

- La siguiente gráfica muestra la distribución en porcentajes de la tarea seleccionada por cada participante.

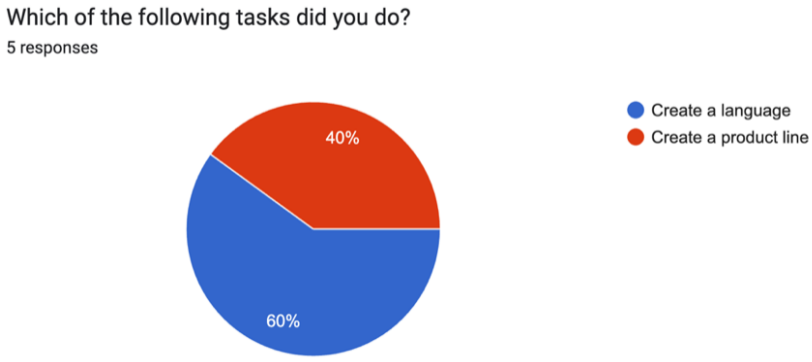

Gráfico 5. Resultados sección 2 - Tarea seleccionada

- En la revisión de descripción de hardware y software adicional utilizado por cada participante, fueron encontrados los siguientes:

Hardware running ARM64 Architecture (Apple M2)

GOOGLE CHROME

Gráfico 6. Resultados sección 2 - Descripción adicional

8.2.3. Resultados de la sección 3:

A continuación es presentado el conjunto de datos recolectado por los participantes.

Tabla 5. Resultados sección 3.

The task was	How many errors did you find doing the test?	Did you require evaluator's help doing the task?	How many time did you use to complete your task?	Do you remember how many hours (days) you used to complete the same task in previous versions of variamos?	VariaMos 4.0 has a user-friendly interface	VariaMos 4.0 has a nice appearance	VariaMos 4.0 has a nice color palette	VariaMos 4.0 is more useful than its previous versions	In VariaMos 4.0 you are able to build languages and product lines faster than its previous versions
partially completed.	15	No	0:15:00	0:00	4	4	6	4	4
completed.	0	No	0:15:00	23:59	5	6	6	7	7
completed.	0	No	0:30:00	23:59	5	5	5	5	6
not completed.	1	No	0:20:00	N/A	4	7	6	3	3
partially completed.	1	No	0:10:00	N/A	4	7	6	4	4

Seguidamente es presentado de manera gráfica el promedio de las respuestas realizadas por los participantes, y además, dividido por las métricas de eficiencia, eficacia y satisfacción, las cuales fueron definidas en el capítulo anterior.

Eficiencia

- La siguiente gráfica muestra la distribución en porcentajes del estado final de la tarea por cada participante, es decir, si no fue completada, parcialmente completada o completada en su totalidad.

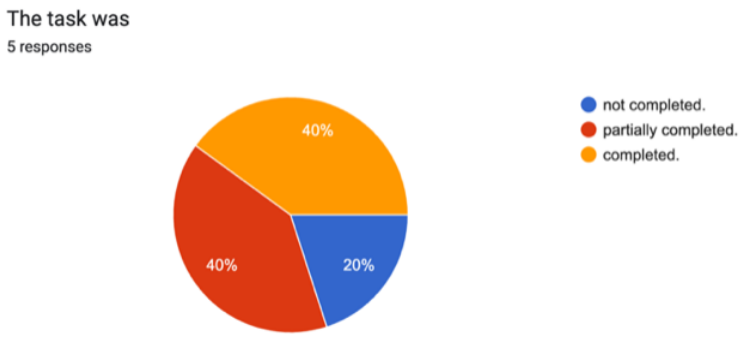

Gráfico 7. Resultados sección 3 - Tareas completadas

- La siguiente gráfica muestra la distribución en porcentajes de la cantidad de errores encontrados por cada participante.

Gráfico 8. Resultados sección 3 - Errores encontrados

- La siguiente gráfica muestra la distribución en porcentajes del apoyo por parte del evaluador que requirió el participante durante la ejecución de la tarea.

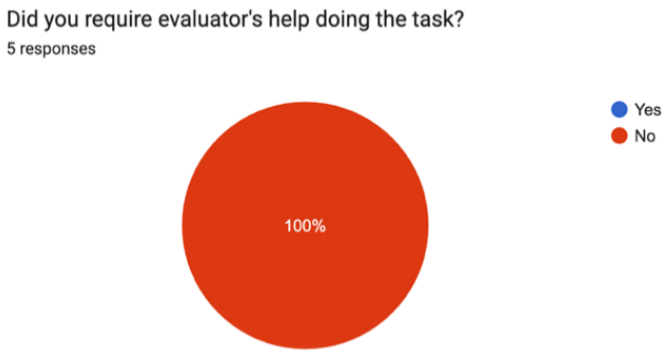

Gráfico 9. Resultados sección 3 - Apoyo del evaluador

A continuación, son descritos los resultados que impactan las variables para el cálculo de usabilidad de la plataforma.

Eficacia

- La siguiente ilustración muestra la cantidad de tiempo utilizado para completar la tarea por cada participante.

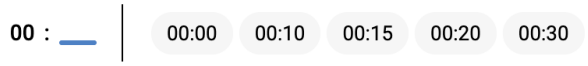

Ilustración 24. Resultados sección 3 - Tiempo utilizado

- La siguiente ilustración muestra la cantidad de tiempo utilizado para completar la tarea en versiones precedentes de VariaMos

Ilustración 25. Resultados sección 3 - Tiempo utilizado versiones precedentes

Satisfacción

- La siguiente gráfica muestra la distribución en cantidad y porcentaje de la percepción de cada participante respecto al fácil uso de la interfaz de usuario.

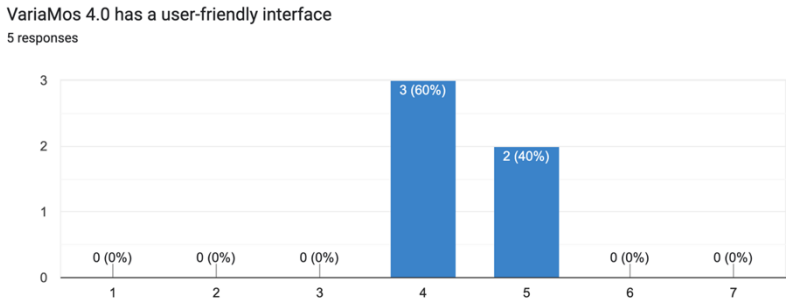

Gráfico 10. Resultados sección 3 - Fácil uso

- La siguiente gráfica muestra la distribución en cantidad y porcentaje de la percepción de cada participante respecto a su apariencia en diseño.

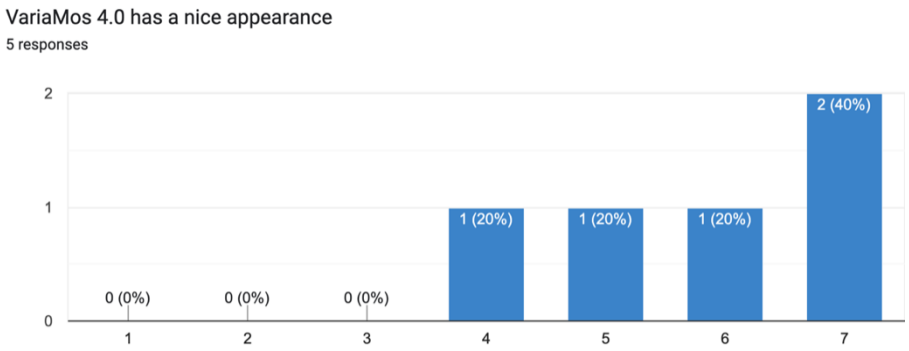

Gráfico 11. Resultados sección 3 - Apariencia en diseño

- La siguiente gráfica muestra la distribución en cantidad y porcentaje de la percepción de cada participante respecto a la paleta de colores utilizada en el diseño.

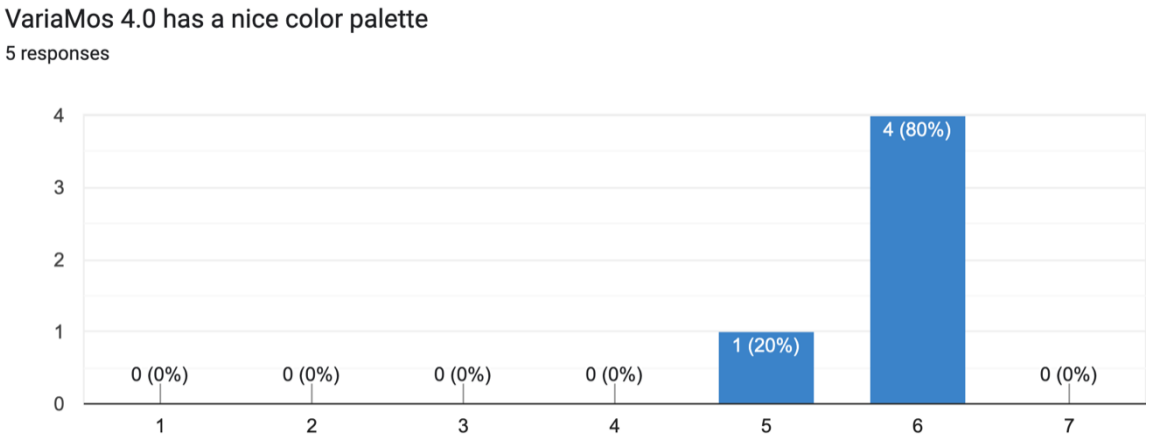

Gráfico 12. Resultados sección 3 - Paleta de colores

- La siguiente gráfica muestra la distribución en cantidad y porcentaje de la percepción de cada participante respecto a si es más útil respecto a versiones precedentes de VariaMos.

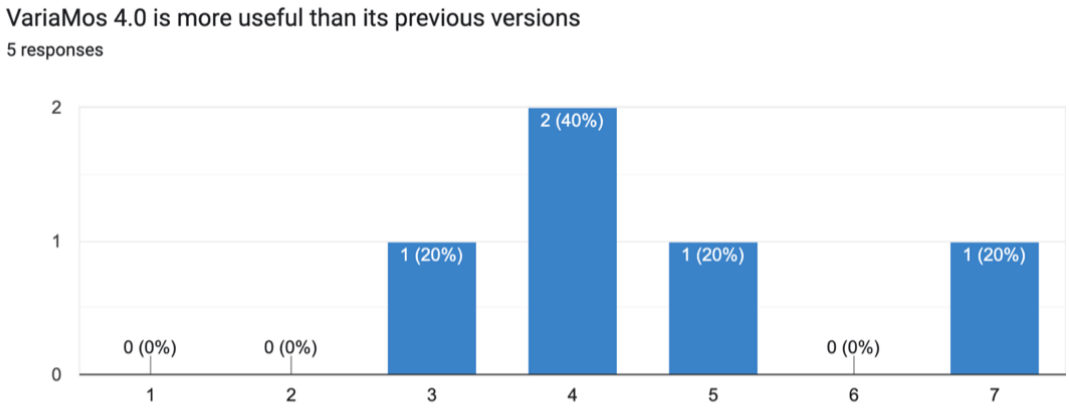

Gráfico 13. Resultados sección 3 - Utilidad

- La siguiente gráfica muestra la distribución en cantidad y porcentaje la habilidad de cada participante para construir lenguajes y líneas de producto más rápido que en versiones precedentes de VariaMos.

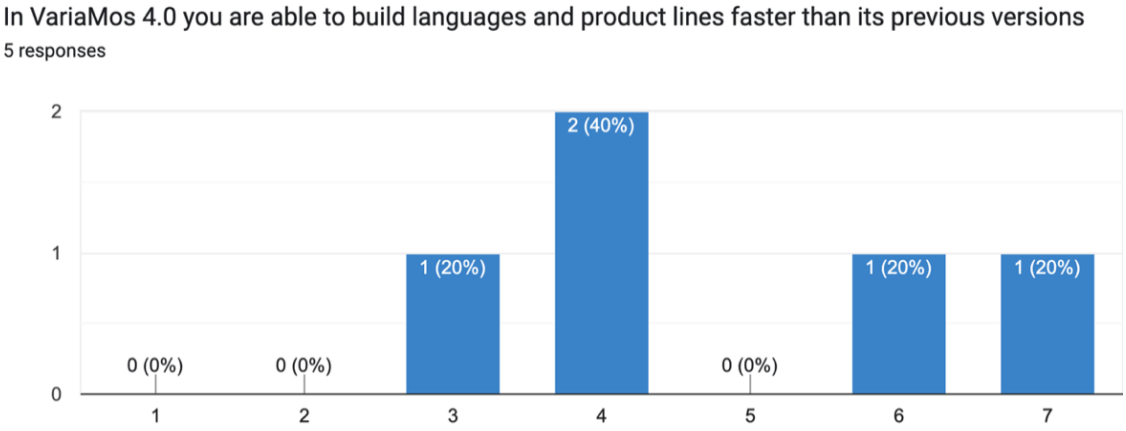

Gráfico 14. Resultados sección 3 - Velocidad

8.2.1. Rendimiento de los resultados

Posterior a la revisión de los datos y los promedios calculados de cada respuesta, a continuación, serán presentados los resultados obtenidos de los cálculos realizados para cada métrica de calidad definida, dichos cálculos incluyen el valor promedio, el valor mínimo y el valor máximo.

Tabla 6. Resultados - Promedios - Mínimos y Máximos.

	The task was completed or partially completed	How many errors did you find doing the test?	Did you require evaluator's help doing the task?	How many time did you use to complete your task?	Do you remember how many hours (days) you used to complete the same task in previous versions of variamos?	VariaMos 4.0 has a user-friendly interface	VariaMos 4.0 has a nice appearance	VariaMos 4.0 has a nice color palette	VariaMos 4.0 is more useful than its previous versions	In VariaMos 4.0 you are able to build languages and product lines faster than its previous versions
Average	80%	3,4	100%	0:18:00	11:59:30	4,4	5,8	5,8	4,6	4,8
Min	N/A	0	N/A	0:10:00	0:00:00	4	4	5	3	3
Max	N/A	15	N/A	0:30:00	23:59:00	5	7	6	7	7

Habiendo calculado lo anterior, en la siguiente tabla, los valores promedio serán traducidos a un valor cualitativo con el fin de generar unas conclusiones más claras e imparciales en capítulos posteriores. Para lograr lo anterior, son utilizadas las tablas generadas en el numeral 8.1.2.

Tabla 7. Resumen de resultados por atributo.

Atributo	Variable	Valor Cualitativo Promedio obtenido	Valor Cuantitativo obtenido
Eficacia	Porcentaje de tareas completadas	80%	Excelente
	Porcentaje de tareas no completadas	20%	Pobre
	Porcentaje de tareas sin asistencia	100%	Excelente
	Promedio de Errores	3	Informativo
Eficiencia	**Versión 4.0**		
	Promedio tiempo de ejecución de cada tareas	0:18:00	Informativo
	Tareas completadas / Promedio de tiempo de ejecución de cada tarea	0,22	Informativo
	Versiones precedentes		
	Promedio tiempo de ejecución de cada tareas	Una semana	Informativo
Satisfacción	Promedio de calificación de 1 a 7 sobre su utilidad	4,6	Ni negativo ni positivo
	Promedio de calificación de 1 a 7 sobre su apariencia	5,8	Algo positivo
	Promedio de calificación de 1 a 7 sobre su facilidad de uso	4,4	Ni negativo ni positivo
	Promedio de los resultados sobre Utilidad + Apariencia + Facilidad de uso	5,2	Algo positivo
Inteligibilidad	Promedio de calificación de 1 a 7 sobre su utilidad	4,6	Ni negativo ni positivo
	Promedio de tareas parcialmente completadas y completadas	80%	Excelente
Operabilidad	Promedio tiempo de ejecución de cada tareas	0:18:00	Informativo
	Promedio de calificación de 1 a 7 sobre la facilidad para construir lenguajes y lineas de producto	4,8	Ni negativo ni positivo
Estetica	Promedio de calificación de 1 a 7 sobre una buena apariencia	5,8	Algo positivo
	Promedio de calificación de 1 a 7 sobre la paleta de colores	5,8	Algo positivo

Algunas de las conclusiones derivadas de la tabla anterior, serán discutidas en el capítulo de conclusiones, el cual se revisará a continuación.

Capítulo 9. Trabajos futuros

En cuanto a los trabajos futuros que quedaron en el mapa de ruta de la plataforma de VariaMos, los más relevantes son, la gestión de roles, trabajo colaborativo y crecimiento y migración de lenguajes. A continuación, revisaremos brevemente cada uno de ellos.

Gestión de roles

Desde el punto de vista de la seguridad, la gestión de roles y permisos es una característica muy importante de trabajar a futuro. Con esta característica se mitigará el riesgo de pérdida de información o de inyección de características no deseadas, lo anterior es debido a que poco a poco el equipo de VariaMos crecerá y será necesario un flujo de aprobación para la creación, actualización y/o eliminación de lenguajes.

Es importante aclarar, que este punto fue pensado desde la arquitectura, por lo tanto, tanto la aplicación como la arquitectura misma tiene la capacidad de soportar dicha característica sin necesidad de generar un impacto muy grande en cuanto a cambios o refactorización de código.

Trabajo colaborativo

Una de las características más deseadas del equipo de investigación de VariaMos, es poder habilitar el trabajo colaborativo, es decir, que la plataforma cuente con la capacidad de crear proyectos de líneas de producto y poderla compartir con varios

miembros de un grupo determinado. Con estas características, la plataforma subirá a un siguiente nivel respecto a su nivel de madurez, es decir, podrá desempeñarse con gran tranquilidad en un entorno industrial.

Así mismo, al igual que la característica anterior, el diseño arquitectónico construido, está en la capacidad de soportar la construcción de una gestión de proyectos y de compartirlos con varios usuarios. No obstante, se debe aclarar que para que esta característica funcione correctamente, se deberá construir la característica de gestión de roles y permisos.

Migración y crecimiento de lenguajes

Debido a que en la versión 4.0 de VariaMos fue requerida una reconstrucción desde cero de la plataforma, en versiones precedentes existen lenguajes que aún no han sido migrados a la versión 4.0. Es por lo anterior, que en la divulgación de la nueva versión hacia el equipo de trabajo fue expuesta la necesidad de leer la documentación generada en la versión 4.0 con el único propósito de migrar los lenguajes previamente creados. A la fecha, lenguajes como el descrito en (LÓPEZ HENAO, 2022), ya ha sido migrado y la experiencia que este investigador tuvo al realizarlo fue altamente gratificante al evidenciar que los tiempos de creación de su lenguaje se redujeron considerablemente.

Capítulo 10. Conclusiones

En este capítulo, se revisarán las conclusiones respecto a los resultados de algunos de los atributos de usabilidad obtenidos en el capítulo anterior, seguidamente se revisará el contraste de la hipótesis planteada con los resultados obtenidos y para finalizar se concluirá con un conjunto de enseñanzas y beneficios de una estructuración sólida de la arquitectura de una plataforma.

Si bien el grupo de participantes en la prueba realizada fue reducido, se logra apreciar la percepción general del equipo frente a la nueva versión de la plataforma en diferentes factores dentro de la usabilidad. Cómo se pudo observar en los resultados, la eficacia, satisfacción y estética, fueron algunos de los atributos que que generaron mayor porcentaje de aceptación.

Dado lo anterior, desde el punto de vista de la eficacia, con el resultado obtenido, los integrantes del equipo de VariaMos expresan con unos altos índices de aceptación que las tareas ejecutadas se lograron sin mucha complicación y que además no requirieron ningún tipo de ayuda para culminar las tareas. Así mismo, desde el punto de vista de la satisfacción, se da a entender que la utilidad, la apariencia, y la facilidad para usar la nueva versión suple y es aceptada en gran medida. Y por último, desde el punto de vista de la estética, con los resultados

logramos concluir que la nueva versión posee una apariencia y una paleta de colores que genera un impacto positivo en los investigadores que la usaron.

Por otro lado, en contraste con las preguntas de investigación planteadas y la hipótesis generada al inicio de este documento, podemos concluir que, las características arquitecturales primarias de una plataforma de modelado y simulación de ingeniería, cómo lo es VariaMos, son la extensibilidad, usabilidad y mantenibilidad; Vale la pena aclarar que cada una de ellas ha sido revisada en el transcurrir de la construcción del modelo arquitectónico propuesto en este documento. Y, además, evidenciando su diseño, implementación y evaluación, podemos concluir que es completamente es posible aplicar en su totalidad cada una de las características de arquitectura definidas. Adicionalmente, respecto a la hipótesis, no sólo fue posible afrontar los retos previamente descritos, además de eso, se logró construir una plataforma que habilita, de manera amigable, el crecimiento constante de la plataforma, tal y cómo se pudo evidenciar en el capítulo anterior de trabajos futuros.

Posterior a recopilar todo lo anterior, la conclusión más valiosa se podría resumir en el deseo de incentivar y demostrar que si se desea construir plataformas escalables, extensibles, mantenibles y con un enfoque en el crecimiento del negocio, es de vital importancia que se estructure de manera sólida su arquitectura. Es comprensible

que al principio no se evidenciaron muchos avances debido a los esfuerzos y dedicación de tiempo que el equipo debe realizar mientras diseña la arquitectura, pero al largo plazo, el resultado de la arquitectura traerá consigo gran cantidad de beneficios y ahorros, tanto en dinero como en tiempo para sus equipos de trabajo.

Referencias

VariaMos. (n.d.). Web architecture. From VariaMos: https://variamos.com/home/variamos-web/web-documentation/

VueJS. (n.d.). VueJS. From https://vuejs.org/

mxGraph. (n.d.). mxGraph. From https://jgraph.github.io/mxgraph/

Eclipse. (n.d.). Eclipse ATL. From https://www.eclipse.org/atl/

Eclipse EMF. (n.d.). Eclipse. From https://www.eclipse.org/modeling/emf/

Evaluando ERP. (n.d.). Evaluando ERP. From https://www.evaluandoerp.com/software-erp/implementar-erp/implementaciones-fallidas-erp/

Eclipse. (2001). Eclipse Foundation. From https://www.eclipse.org/

CNET. (2002, Enero). CNET. From https://www.cnet.com/tech/tech-industry/i2-nike-fallout-a-cautionary-tale/

Forbes. (2022, Mayo). The Global 2000. From https://www.forbes.com/lists/global2000/?sh=68795ec25ac0

Correa, D., Mazo, R., & Giraldo, G. L. (n.d.). Fragment-oriented programming: a framework to design and implement software product line domain components. DYNA.

Muñoz Fernández, J. C. (n.d.). Second Guide to Define New Dynamic Operations in VariaMos. VariaMos.

Evans, E. (2003). Domain-driven Design: Tackling Complexity in the heart of software.

Fowler, M. (2019). MartinFowler. From Microservices: https://martinfowler.com/microservices/

ERAZO RAMOS, A. F. (2020). CONTROL PARA SISTEMAS CONTINUOS Y DISCRETOS BASADO EN LA INGENIERÍA DIRIGIDA POR MODELOS. VariaMos.

Mazo, R., Muñoz Fernández, J. C., Rincón, L., Salinesi, C., & Tamura, G. (2015). VariaMos: an extensible tool for engineering (dynamic) product lines. VariaMos, 1-6.

Muñoz Fernández, J. C. (2015). Second Guide to Define New Dynamic Operations in VariaMos. VariaMos.

Mazo, R., Salinesi, C., & Diaz, D. (2012). VariaMos: a Tool for Product Line Driven Systems Engineering with a Constraint Based Approach. HAL Archives Ouvertes France, 1-9.

Pothula, K. (2020, Agosto). Medium. From https://medium.com/@kethan.pothula/netflix-system-design-and-architecture-592ea72e155f

Pothula, K. (2020, Agosto). Medium. From https://medium.com/@kethan.pothula/amazon-system-design-and-architecture-d787a6572f35

Butani, A. (2020, 12). RedHat. From 5 essential patterns of software architecture: https://www.redhat.com/architect/5-essential-patterns-software-architecture

A design science research methodology for information systems research. Journal of Management Information Systems, 24(3). (2007). In K. Peffers, T. Tuunanen, M. Rothenberger, & S. Chatterjee.

Ritchey, T. (2013). Wicked problems: Modelling social messes with morphological analysis. Acta morphologica generalis, 2(1).

Hevner, A. R. (2004). Design science in information systems research. MIS Quarterly, 28(1).

Rittel, H., & Webber, M. (1973). Dilemmas in a general theory of planning. Policy sciences, 4(2).

State of JS. (2022). The State Of JS 2022. From Front-end Frameworks: https://2022.stateofjs.com/en-US/libraries/front-end-frameworks/

Martin, R. C. (2017). Clean Architecture: A Craftsman's Guide to Software Structure and Design: A Craftsman's Guide to Software Structure and Design .

Google. (n.d.). Google. From Google Forms: https://docs.google.com/forms

Microsoft Azure. (n.d.). Microsoft Azure. From Publisher-Subscriber pattern: https://learn.microsoft.com/en-us/azure/architecture/patterns/publisher-subscriber

Docker. (n.d.). Docker. From Container: https://www.docker.com/resources/what-container/

React. (n.d.). React. From https://react.dev/

React. (n.d.). React. From https://react.dev/

JGraph Ltd. (2020). mxgraph. From https://jgraph.github.io/mxgraph/

Nodejs. (n.d.). NodeJs. From https://nodejs.org/

Microsoft Corporation. (n.d.). Typescript. From https://www.typescriptlang.org/

Microsoft Corporation. (n.d.). Azure. From https://azure.microsoft.com/

Microsoft Corporation. (n.d.). Azure. From App Service: https://azure.microsoft.com/en-us/products/app-service/

The PostgreSQL Global Development Group. (n.d.). PostgresSQL. From https://www.postgresql.org/

Docker Inc. (n.d.). Docker Hub. From https://hub.docker.com/

Gib. (n.d.).

GitHub, Inc. (n.d.). Git Hub. From Actions: https://github.com/features/actions

LÓPEZ HENAO, A. O. (2022). DISEÑO DE SISTEMAS CIBER-FÍSICOS EN EL CONTEXTO DE LA INGENIERIA DE LINEAS DE PRODUCTOS DE SOFTWARE, UN ENFOQUE DE INGENIERIA DIRIGIDA POR MODELOS.