UNIVERSIDAD EAN
FACULTAD DE INGENIERIA

QUIMISOFT: DESARROLLO DE SOFTWARE PARA EL DISEÑO DE REACTORES EN INGENIERÍA QUÍMICA

MAURICIO OROZCO SIERRA
JESUS DAVID GARCIA CUELLO
RAUL REDONDO DURAN

JOHN JAIRO PORRAS

BOGOTÁ, D.C., 3 DE MARZO DE 2024

Tabla de contenidos

RESUMEN

En la industria química, se hace uso de aplicaciones y múltiples softwares para simular reactores, y para dimensionarlos a partir de variables de entrada como temperatura, presión, condiciones de la reacción química, entre otros. Al ser programas de propósito general, estos son de alcance limitado, usan licencias que suelen ser costosas limitando la accesibilidad al público y en ocasiones su uso es poco intuitivo, dificultando el aprendizaje de dichas herramientas.

El proyecto expuesto a continuación, se trata del desarrollo de un producto mínimo viable (MVP) para simulación de reactores de reacción única, en condiciones de idealidad y sin intercambiador de calor. El software se enfoca en la parametrización, eficiencia y facilidad de uso; ofrece una secuencia simple para ingresar datos, permitiendo al usuario definir las condiciones de entrada, las cuales, a través de un procesamiento interno basado en ecuaciones de diseño ajustadas a las condiciones indicadas por el usuario, se genera una salida que describe el comportamiento de todas las variables necesarias, derivadas de la resolución de ecuaciones diferenciales de diseño.

De esta forma, se busca generar una herramienta de aprendizaje para la simulación de reactores químicos, entregando un software, gratuito y de código abierto el cual estará disponible al público en general.

INTRODUCCIÓN

Los reactores químicos se pueden definir como un recipiente diseñado para que en su interior se produzcan reacciones químicas o biológicas.” (Asale & Rae, n.d.-a). Para la construcción de un reactor es necesario la simulación de éste, usando modelos matemáticos por medio de los cuales se puede determinar el comportamiento de las reacciones a partir de la variación de parámetros que se encuentran en escenarios del mundo real, con el fin de recrear los posibles comportamientos, efectos o cambios derivados de las variables como: velocidad de reacción, cambios de temperatura o presión, cambios en flujos, así como los cambios de reacción en función del volumen del reactor.

Actualmente, se observa que para la simulación de reactores se recurre a la creación de tablas y gráficos en plataformas como Excel, MATLAB o Aspen, para resolver los modelo matemáticos empleados para la simulación de reactores químicos, sin embargo, estas herramientas presentan algunos aspectos que de alguna forma limitan a las personas que desean aplicar los modelos matemáticos con los que se desarrollan las simulaciones, la principal barrera que se identifica es el costo de productos como Matlab o Aspen, entre tanto al emplear Excel, el margen de error en la simulación puede ser amplio en razón a que los resultados que se obtienen para simular el reactor dependen del registro manual de datos por parte de las personas, lo cual requiere especial cuidado y un mayor conocimiento de los conceptos que subyacen a la simulación de reactores.

Con base en lo anterior y en el contexto de la Universidad Ean, que fomenta el espíritu emprendedor y acoge a estudiantes de diversos campos de la ingeniería, se identificó la oportunidad de generar un producto mínimo viable, en un software de simulación de reactores que involucra ámbitos de la ingeniería química y la ingeniería de software con los cuales se aborda la pregunta: *¿Es posible crear una herramienta de software que facilite la simulación de reactores químicos como recurso de aprendizaje para los estudiantes de la Universidad Ean y el público en general, bajo la premisa del acceso abierto y uso gratuito del software?*.

El MVP generado usa tecnologías para desarrollo de software adecuadas para la ejecución en entornos web incorporando desarrollos de software para el backend y frontend, por medio de los cuales se resuelven las ecuaciones diferenciales usadas en el modelado de los reactores y se entrega una vista al usuario en la cual se representan múltiples gráficos por medio de los cuales se pueden analizar los comportamientos del reactor a partir de los datos cargados en los parámetros y variables que el usuario indique.

En el presente documento se presenta los temas que subyacen a la simulación de reactores desde el ámbito de la ingeniería química e igualmente se presentan conceptos de la ingeniería de software y la arquitectura de sistemas informáticos, que al aplicarlos permitió la materialización de un MVP con el cual se pueden hacer simulaciones de reactores generando un recurso de aprendizaje que promueve la democratización del conocimiento y el acceso a tecnologías para el aprendizaje.

OBJETIVOS

Objetivo General

Desarrollar un producto mínimo viable (MVP) conformado por un software web desacoplado para la simulación de reactores de reacción única, en condiciones de idealidad y sin intercambiador de calor.

Objetivos Específicos

- Examinar los enfoques, aplicaciones y metodologías usadas para diseñar reactores químicos, identificando los elementos relevantes para integrarlos en un MVP de simulación de reactores.
- Realizar el diseño y arquitectura del software, junto a la codificación y desarrollo del MVP, según el contexto y necesidades funcionales identificadas para los cálculos, gráficas y tablas necesarias en la simulación de los reactores químicos CSTR, PFR, PBR y Batch.
- Crear escenarios de prueba y validación con diferentes tipos de reactores que permitan la comprobación del correcto funcionamiento del aplicativo en la simulación

DEFINICIÓN DEL PROBLEMA

En la industria química la simulación y dimensionamiento de reactores se hace con aplicaciones y software de propósito general. Estas son alternativas funcionales que, sin embargo, presentan limitaciones como licencias de pago, y al ser herramientas no especializadas en el modelado de reactores, su uso consiste en usar el software para que el usuario mismo construya el modelo matemático necesario para simular el reactor. El factor humano en la construcción de simuladores genera variabilidad y subjetividad generando posibles discrepancias en los resultados.

Se puede entender el proceso de construcción de simulación en reactores como la solución de un sistema de múltiples ecuaciones diferenciales, las cuales están condicionadas por las características propias del reactor a generar, por ejemplo, si hay o no caída de presión, temperatura, que tipo de reactor es y en base a que variable se está calculando (Conversión o flujo). El cambio de la variable independiente genera cambios en los cálculos de ecuaciones algebraicas internas del reactor, que a su vez dependen de los inputs o constantes ingresadas por el usuario. Por esta razón, se sabe que es posible definir a partir de inputs de qué forma se va a definir el sistema de ecuaciones diferenciales, y el sistema de ecuaciones algebraicas. Facilitando este problema presente en los programas matemáticos adaptados a la simulación de reactores.

Al ofrecer una herramienta que combina flexibilidad, precisión y facilidad de uso, este proyecto aborda un vacío crítico en el diseño de reactores, y establece un estándar y un precedente ante la creación de software libre para procesos en un entorno académico. La implementación exitosa de este prototipo de software puede transformar el proceso de diseño de reactores y contribuir significativamente a un entendimiento más eficiente, sostenible y adaptativo al contexto de ser capaz de entender estos problemas.

La iniciativa de este proyecto se centra en el desarrollo de un prototipo de diseño de reactores, un software que aborde la parametrización precisa y la facilidad de uso. Este enfoque simplifica la entrada de datos, permitiendo al usuario definir condiciones de entrada. A través de procesamiento interno basado en ecuaciones de diseño ajustadas a las condiciones indicadas, el software genera resultados que describen el comportamiento de variables clave, derivadas de la

resolución de ecuaciones diferenciales de diseño. Con este planteamiento se pretende resolver la pregunta: ¿Es posible desarrollar una herramienta de software que, al facilitar la simulación de reactores químicos mediante un acceso abierto y uso gratuito, sirva como un recurso educativo efectivo tanto para los estudiantes de la Universidad Ean como para el público en general?

JUSTIFICACIÓN

La industria química enfrenta desafíos sin precedentes en eficiencia operativa, sostenibilidad y adaptabilidad a procesos innovadores. Una de las piedras angulares para abordar estos desafíos es el diseño de reactores químicos, crucial en la optimización de los procesos de producción. Tradicionalmente, el diseño de reactores se ha basado en aplicaciones y softwares de propósito general que, aunque útiles, a menudo carecen de la robustez y especificidad necesarias para abordar las necesidades particulares de cada proceso químico. Esta falta de especialización puede llevar a ineficiencias operativas, aumentando el consumo de recursos y el impacto ambiental, así como limitando la capacidad de innovación dentro de la industria.

El diseño de reactores químicos radica en su capacidad para adaptarse específicamente a las condiciones únicas de cada reacción, incluyendo variables como la temperatura, presión, y la naturaleza de los reactivos y productos. Además, la adaptabilidad y especificidad en el diseño de reactores no solo mejora la eficiencia y efectividad de los procesos químicos, sino que también contribuye significativamente a la reducción de costos operativos.

Por lo tanto, este proyecto de grado propone el desarrollo de un producto mínimo viable de simulación de reactores químicos que se centre en la parametrización como su característica distintiva. Dicho software permitirá al usuario, ingresar especificaciones y ajustar las variables de diseño a las necesidades exactas de sus proyectos.

Este proyecto se posiciona en la intersección de la ingeniería química y la ingeniería de sistemas, aprovechando los avances en software y simulación para ofrecer una solución tangible a un problema complejo y multidimensional. Al hacerlo, no solo se espera mejorar los procesos existentes sino también inspirar una nueva generación de innovaciones de software libre para el diseño y operación de reactores químicos.

REQUERIMIENTOS

A continuación, se presentan los requerimientos funcionales y no funcionales del prototipo funcional de software. Así pues, los requerimientos del simulador son:

Requerimientos funcionales

- **Entrada de datos**: el software permitirá al usuario ingresar variables como temperatura, presión, volumen, y condiciones reactivas.
- **Personalización de parámetros**: se permitirá a los usuarios ajustar y personalizar parámetros de simulación para adaptarse a diferentes escenarios y múltiples tipos de reactores.
- **Procesamiento y cálculo**: basándose en los datos de entrada, el software utilizará ecuaciones diferenciales, ecuaciones de diseño y principios de ingeniería química para simular el comportamiento del reactor.
- **Salida**: generación de tablas, gráficos y resultados detallados sobre las condiciones del reactor.

Requerimientos no funcionales

- **Usabilidad**: el programa dispondrá de una interfaz intuitiva para el usuario.
- **Rendimiento**: tiempos de respuesta rápidos (máx. 10.000 ms de respuesta a una determinada acción del usuario) incluso con cálculos complejos.
- **Compatibilidad**: el software estará disponible para navegadores web modernos como Chrome, Edge o Firefox, y para múltiples dispositivos, entre ellos Android, Windows y Mac.
- **Software libre y código abierto**: se le facilitará el acceso al código fuente del prototipo funcional para permitir la modificación y mejora por parte de la comunidad.

SELECCIÓN DE ALTERNATIVAS

Algunos de los productos disponibles en el mercado para simulación de reactores tienden a tener un alcance limitado, ya que hacen uso de licencias que suelen ser costosas limitando la accesibilidad al público y en ocasiones su uso es poco intuitivo, dificultando el aprendizaje de dichas herramientas.

Por otro lado, no suelen ser programas construidos específicamente para orientar al usuario a obtener el modelo deseado, sino que más bien son programas que facilitan el uso de modelos matemáticos de cualquier tema, y es el usuario mismo quien debe construir la simulación, así como definir qué input debe entrar, de qué forma va a ingresar, cómo se procesa ese conjunto de inputs y cómo va a llegar a un resultado. Dando como consecuencia que la manipulación de datos y dependencia de la construcción del modelo se convierta en una desventaja significativa en estas alternativas funcionales.

En vista del costo de las herramientas, y el riesgo de error derivado del error humano cometido en la construcción de modelos. Nuestra alternativa es un desarrollo a medida de un MVP en la cual se incorpora las características para la simulación de reactores de reacción única, en condiciones de idealidad y sin intercambiador de calor. Otorgando las siguientes ventajas:

- Acceso libre
- Su uso se hace por medio de un navegador de internet
- Código abierto (Abre la posibilidad de que cualquier interesado en esta alternativa pueda mejorar y evolucionar el producto)
- Puede obtener mejoras funcionales en el futuro (Ej: Generar un instrumento de aprendizaje con respecto a la temática de diseño de reactores y programación)

Desde un punto de vista comparativo, podemos destacar las siguientes características de cada software:

Excel	Matlab	Quimisoft
Excel es conocido por su interfaz amigable y familiar para la mayoría de los usuarios. Su estructura de hojas de cálculo facilita la organización y manipulación de datos.	MATLAB se usa mucho en aplicaciones de ingeniería y ciencias por su capacidad para manejar cálculos numéricos y algoritmos complejos de manera eficiente.	La ventaja principal de esta aplicación es su enfoque en la facilidad de uso. Una interfaz intuitiva y amigable facilita la interacción de los usuarios sin necesidad de tener conocimientos especializados únicamente en el uso del software.
Permite crear modelos personalizados utilizando fórmulas y funciones incorporadas. También es posible integrar gráficos para visualizar los resultados.	Ofrece herramientas y funciones para modelado y simulación y para visualización de datos.	Al priorizar la eficiencia en el uso de recursos, la aplicación puede ser más liviana y rápida en comparación con herramientas más generales como Excel o MATLAB.
Para modelos complejos, Excel puede ser limitado en cuanto a capacidad de cálculo y procesamiento de grandes cantidades de datos.	Puede requerir cierto tiempo para familiarizarse con su sintaxis y funcionalidades, especialmente para usuarios nuevos en programación.	La aplicación puede ser adaptada para que se ajuste a las necesidades específicas de los usuarios y del proceso de modelado de reactores.
Aunque es ligero en recursos en comparación con herramientas especializadas, puede volverse lento con modelos grandes y complejos.	MATLAB puede consumir más recursos computacionales en comparación con otras herramientas, especialmente en modelos	La aplicación pretende ser usada exclusivamente para términos de simulación, por lo tanto, se espera que tenga un bajo consumo de recursos.

	complejos o grandes conjuntos de datos.	

MARCO TEÓRICO

8.1 Ecuación de diseño: Una ecuación de diseño se refiere a toda aquella ecuación que sea derivada o "adaptada" a un reactor a partir de la ecuación general del balance molar (Fogler, 2008, pp. 8–10).

$$Fj0 - Fj + Gj = \frac{dNj}{dt}$$

Ecuación 8.1: Ecuación general del balance molar (Fogler, 2008), Recuperado de: Elements of Chemical Reaction Engineering (4th ed.) (p. 8).

8.2 Reactor: Un reactor corresponde a un "Recipiente diseñado para que en su interior se produzcan reacciones químicas o biológicas." (Asale & Rae, n.d.-a). Hay que considerar que el modelado de cada reactor proviene de una ecuación de diseño, que proviene de la ecuación general del balance molar.

8.2.1 Reacción química: Una reacción química es la acción de haber cambiado la estructura molecular de uno o más compuestos, por este motivo se dice que "una reacción química ha ocurrido cuando un número detectable de moléculas de una o más especies han perdido su identidad y han asumido una nueva forma, por un cambio en el tipo o número de átomos en el compuesto, o por un cambio en la estructura o configuración de dichos átomos." (Fogler, 1986).

8.2.2 Conversión: La conversión es una medida del progreso que ha tenido la reacción hacia la transformación completa, toma valores entre 0 y 1 sin incluir al 1. "Este es el resultado de dividir las moles de reactivo que han reaccionado con las moles de entrada de reactivo" (Foutch, 2003).

8.2.3 Reactor batch: Un reactor batch corresponde a un recipiente de volumen constante, por esa razón también es conocido como "Recipiente presurizado" a nivel de laboratorio. (Smith et al., 2013, pp. 55–119). En el cual de acuerdo a Fogler (2008, pp. 11–12) podemos modelar con la siguiente ecuación de diseño:

$$\frac{dNj}{dt} = rjV$$

Ecuación 8.2: Ecuación de diseño para reactor batch (Fogler, 2008), Recuperado de: Elements of Chemical Reaction Engineering (4th ed.) (p. 11-12).

Hay que tener en cuenta que este tipo de reactor tiene como variables "input" el "ingreso de calor, temperatura, tiempo, composición molar y presión" (Agbebi & Sandrock, 2015), siendo el volumen una constante y la velocidad de reacción una función propia de la reacción que a su vez va en función de la constante de Arrhenius que va en función de la temperatura.

8.2.4 Reactor CSTR: Este tipo de reactor tiene una corriente de entrada y salida constante, en la que la corriente de salida tiene las mismas características que el contenido interno del tanque (Trambouze et al., 2005). Fogler (2008, pp. 13–14) Define la ecuación de diseño para este reactor como:

$$\int^{V} rjdV = Vrj$$

Ecuación 8.3: Ecuación de diseño para reactor CSTR [Integral] (Fogler, 2008). Recuperado de: Elements of Chemical Reaction Engineering (4th ed.) (p. 13-14).

Donde se resuelve la integral como:

$$V = \frac{Fj0 - Fj}{-rj}$$

Ecuación 8.4: Ecuación de diseño para reactor CSTR (Fogler, 2008). Recuperado de: Elements of Chemical Reaction Engineering (4th ed.) (p. 11-12).

El reactor CSTR tiene como inputs de entrada el ingreso de calor, temperatura, presión, Flujo molar de entrada y de salida. Y al igual que el reactor batch, tiene un volumen constante y la velocidad de reacción va en función de la constante de Arrhenius y la temperatura.

8.2.5 Reactor PFR: En un reactor de flujo en pistón (PFR), los reactivos fluyen a través de un tubo. La reacción ocurre a medida que los reactivos avanzan a lo largo del tubo. Este tipo de reactor es útil cuando se necesita un control preciso sobre las condiciones de reacción (Jaibiba et al., 2020, Chapter 10.3.2). Y se suele utilizar en reacciones en estado gaseoso.

De acuerdo con Fogler (2008, pp. 14–17) la ecuación de diseño para el reactor PFR corresponde a:

$$dV = \frac{dFj}{rj}$$

Ecuación 8.5: Ecuación de diseño para reactor PFR (Fogler, 2008). Recuperado de: Elements of Chemical Reaction Engineering (4th ed.) (p. 17).

Los inputs para este tipo de reactor corresponden a la composición del flujo, calor, temperatura, velocidad de reacción, y presión.

8.2.6 Reactor PBR: En los Biorreactores de Película Adherida (PBR), se llenan tubos o cilindros con materiales de soporte adecuados, seguidos de la inoculación con el cultivo para formar una biopelícula. El tiempo necesario para que la biopelícula madure depende del cultivo, el soporte y la disponibilidad de nutrientes. El uso de PBR suele mejorar el contacto entre la biopelícula y el sustrato.

Por último, la ecuación de diseño para este reactor corresponde a:

$$dW = \frac{dFj}{r'j}$$

Ecuación 8.6: Ecuación de diseño para reactor PBR (Fogler, 2008). Recuperado de: Elements of Chemical Reaction Engineering (4th ed.) (p. 17).

Teniendo en cuenta que tiene los mismos inputs que un PFR, y que obtenemos como resultado, la variación de W (Peso de catalizador) con respecto a la composición del flujo.

8.3 Caída de presión: Es una condición determinada en el diseño de reactores, en la cual se asume que la presión varia en diferentes puntos del modelo (Green & Southard, 2019). La caída de presión la determina una ecuación diferencial que describe el comportamiento con un reactor según la teoría de mecánica de fluidos, y el balance de energía mecánica (Anaya-Durand et al., 2014, p. 128).

Software, hardware y sistemas de simulación

Según la Organización Internacional para la Estandarización, el término “hardware” hace referencia a: - Toda o parte de los componentes físicos de un sistema de procesamiento de información (ISO, 2015).

Por otra parte, el software puede definirse como el producto digital creado y mantenido por profesionales del software para su uso a largo plazo. Incluye programas que se ejecutan en ordenadores de diversos tamaños y tipos, junto con el contenido mostrado durante la ejecución del programa y la información de apoyo en formatos físicos y digitales a través de diferentes medios electrónicos. El software es crucial, ya que influye significativamente en varios aspectos de nuestras vidas, profundamente arraigado en nuestras actividades diarias, el comercio y la cultura (Pressman & Maxim, 2021).

Según la Organización Internacional para la Estandarización [ISO] (2015), el término “software” hace referencia a: - todo o parte de los programas, procedimientos, reglas y documentación asociada de un sistema de procesamiento de información. - El software es una creación intelectual independiente del medio en el que este almacenado [o ejecutándose].

Así pues, la ingeniería de software es la disciplina centrada en crear y mantener productos de software informático de calidad a lo largo del tiempo. Involucra el desarrollo de programas que se ejecutan en computadoras de diferentes tamaños y arquitecturas, así como el contenido asociado e información que respalda estos programas. La ingeniería de software es un proceso integral que incorpora herramientas, métodos, un proceso y un enfoque de calidad (Pressman & Maxim, 2021).

Figura 1. *Capas de la ingeniería de software*

Nota. Adaptada de *Capas de la ingeniería de software* [Figura], Pressman, R. S., & Maxim, B., *Ingeniería del software: Un enfoque práctico*, McGraw-Hill Interamericana.

Así pues, de acuerdo con Pressman, R. S., & Maxim, B., (2021), la capa de *herramientas* hace referencia al soporte automático o semiautomático que las herramientas CASE pueden brindar al proceso y a los métodos. Las herramientas CASE (Computer-Aided Software Engineering) son herramientas informáticas que asisten a los ingenieros en el desarrollo de sistemas de información. Estas herramientas proporcionan soporte automatizado o semiautomatizado para actividades como el modelado, la generación de código, la depuración y la documentación de software. Por su parte, la capa de *métodos* hace referencia a las instrucciones técnicas para desarrollar el software. Así pues, esta capa involucra la recolección de requerimientos, el modelado y la codificación del programa (Pressman & Maxim, 2021). De forma central, tanto en la figura de representación de capas anterior, como en la propia ingeniería de software, la capa de ***proceso*** establece un marco de trabajo para la ejecución efectiva de la ingeniería de software, dando así una base para el control administrativo de los proyectos de software (estableciendo métodos técnicos, productos de trabajo, hitos, criterios de calidad y gestión del cambio); en este mismo sentido, un ***proceso de software*** es el marco de trabajo utilizado para desarrollar software de alta calidad (Pressman & Maxim, 2021).

Según Pressman & Maxim (2021), existen múltiples modelos de proceso de ingeniería de software, pero todos tienen en común las siguientes actividades fundamentales:

1. Especificación de software: la funcionalidad y las restricciones del software.
2. Diseño e implementación: diseño y construcción del sistema de acuerdo con la especificación.

3. Validación: asegurar que el sistema cumple con lo que el cliente quiere.
4. Evolución: el software mejora para adaptarse a las necesidades del cliente.

De forma muy relacionada al software y la ingeniería de software, el término requisitos/requerimientos hace referencia a aquellos atributos necesarios que se definen para un sistema antes de desarrollar un diseño del mismo. Así pues, la elicitación de requisitos (SRA System requirements analysis; por sus siglas en inglés) es una metodología que permite identificar el conjunto adecuado de requisitos de tal forma que satisfagan una necesidad o característica esencial (requisito). De esta manera, se busca resolver las necesidades del cliente mediante la definición expresa, clara y sistemática de lo que el sistema debe hacer (Grady, 2014).

En este sentido, al desarrollarse las metodologías de desarrollo de software y el software permeaba otras industrias haciéndolas más eficientes y escalables, cada vez se hizo más importante un marco de trabajo común que permita el desarrollo de aplicaciones de forma eficiente y ágil. En respuesta, el desarrollo ágil de software se configura como un enfoque dinámico de la creación de software que se centra en la flexibilidad, colaboración con el cliente y capacidad de adaptación a cambiantes necesidades empresariales.

Las metodologías ágiles de desarrollo de software, según el Holcombe (2008), se centran en los siguientes puntos: 1. Responden rápidamente a los cambios y evoluciones de la empresa, sin dejar de aspirar a alcanzar los objetivos del cliente y satisfacer sus necesidades empresariales. 2. Estrechan relaciones con el cliente, por lo que se responde más rápidamente a la naturaleza cambiante de las necesidades del cliente. 3. No existe un "producto acabado", sino una relación continua con el cliente, que implica una corrección, ampliación, adaptación o desarrollo continuos.

Por otra parte, los "servicios web" son una de las formas más comunes en las cuales se presenta el software en los aplicativos webs modernos. Estos son módulos de software auto-descriptivos y autocontenidos que están disponibles a través de una red, como Internet, y que realizan tareas, resuelven problemas o llevan a cabo transacciones en nombre de un usuario o aplicación. Estos servicios web constituyen una infraestructura informática distribuida

compuesta por diversos módulos de aplicación que interactúan y tratan de comunicarse entre sí (Papazoglou, 2008).

Previamente al desarrollo de software, y posterior a la elicitación de requisitos, se debe realizar el modelamiento y la arquitectura del software. Según la IEEE (2000), la arquitectura se define como la organización fundamental de un sistema, plasmada en sus componentes, sus relaciones entre sí y con el entorno, y los principios que rigen su diseño y evolución.

Esta modelización y análisis de simulación se define como el proceso de usar modelos informáticos para replicar sistemas del mundo real para estimar métricas de rendimiento, abordar escenarios hipotéticos y mejorar la toma de decisiones. Este enfoque ha evolucionado como una herramienta crucial en diversos ámbitos desde la Segunda Guerra Mundial, desempeñando un papel vital en actividades como evaluar el rendimiento del sistema, capacitar al personal y optimizar procesos operativos. A través de la modelización de simulación, las organizaciones pueden simular diversos escenarios como sistemas de producción, gestión de inventarios, redes de comunicación, operaciones militares, logística portuaria, servicios de salud, operaciones financieras y sistemas de transporte (Altiok & Melamed, 2007).

Los sistemas de control de versiones (VCS) son herramientas sofisticadas creadas para supervisar las alteraciones en documentos, programas de software, sitios web expansivos y otras compilaciones de datos. Desempeñan un papel crucial en el proceso de desarrollo de software al servir de repositorio central de la versión definitiva del código fuente y sus modificaciones. Los VCS fomentan la colaboración entre desarrolladores al permitir fusionar sin fisuras los cambios de distintos colaboradores e identificar las modificaciones conflictivas para resolverlas cuando sea necesario. Este sistema no sólo protege contra la sobreescritura accidental del trabajo, sino que también proporciona un registro detallado de cada cambio realizado en un archivo, incluido el autor y la marca de tiempo (Spinellis, 2005).

Virtualización, contenedores y computación en la nube

La virtualización se refiere a la tecnología que permite la abstracción de componentes físicos en objetos lógicos, lo que permite una mayor utilidad de los recursos que estos objetos proporcionan. En concreto, la virtualización de computadoras ofrece un entorno idéntico al de una máquina original, garantizando el aislamiento y control total de los recursos del sistema. La virtualización permite crear una versión virtual del sistema, en lugar de física. Esto permite, por ejemplo, crear múltiples instancias, entornos o máquinas virtuales (VMs) en un solo sistema físico. Cada máquina virtual puede ejecutar sus propios sistemas operativos y aplicaciones como si estuviera en un hardware dedicado, independiente de las otras VMs y del hardware subyacente. De esta forma, no solo se optimiza el uso de recursos informáticos, sino que también facilita una gestión más eficiente y flexible de los mismos (Portnoy, 2016).

Por otra parte, los contenedores, aunque son similares en su propósito, son diferentes en su concepción y funcionamiento. Los contenedores son paquetes de software independientes y autocontenidos, que, a diferencia de las máquinas virtuales, funcionan a nivel de sistema operativo. Esto permite agrupar múltiples cargas de trabajo dentro de una única entidad, mejorar la portabilidad y reducir el consumo de recursos en comparación con las máquinas virtuales tradicionales. Tecnologías como Docker y Kubernetes han estandarizado la gestión y el despliegue de contenedores, convirtiéndolos en una fuerza disruptiva en la virtualización y la computación en nube para mejorar la eficiencia y la escalabilidad en el despliegue de aplicaciones (Portnoy, 2016).

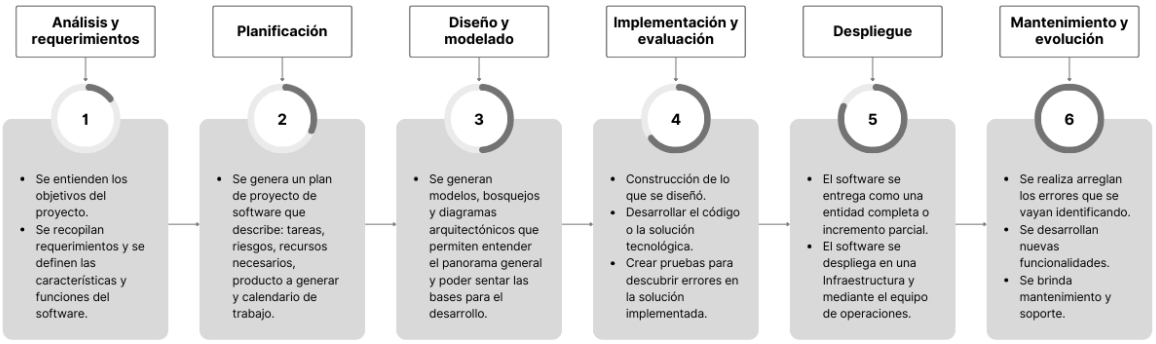
Las tecnologías de virtualización, “containerización”, entre otras han dado paso a la computación en la nube y los modelos de entrega de distribución de software como SaaS (Software as a Service). De acuerdo al NIST (National Institute of Standards and Technology), la computación en la nube hace "se trata de un modelo que permite un acceso cómodo y a la carta a un conjunto compartido de recursos informáticos configurables (redes, servidores, almacenamiento, aplicaciones y servicios, por ejemplo), que pueden ser rápidamente aprovisionados y liberados con un mínimo esfuerzo de gestión o interacción con el proveedor de servicios" (2011).

Desarrollo de software: ciclo de vida, arquitectura y patrones de diseño, principios SOLID, modelo de arquitectura C4

A medida que el software se fue haciendo más complejo, con mayor número de partes interesadas, más requerimientos –a veces conflictivos entre sí– y con demandas de calidad cada vez mayores, se hizo imperativo acudir a una metodología que permitiera concluir de forma exitosa los intrincados proyectos de software. En este contexto, surge el ciclo de desarrollo de software para abarcar estos retos, se trata de una metodología y marco de referencia que permite "la explotación y el mantenimiento de un producto de software, abarcando desde la definición hasta la finalización de su uso" (ISO, 2017). Es decir, facilita un modo sistemático de llevar a cabo un proyecto de software, de tal forma que se cumpla con los objetivos de sus partes interesadas. Es por esto por lo que, en el contexto del desarrollo de un simulador químico que automatiza y simplifica el proceso de diseño de reactores químicos, entender las fases del proceso de desarrollo de software es de vital importancia.

El ciclo de vida del desarrollo de software moderno se divide típicamente en seis etapas o fases: (1) análisis y recopilación de requerimientos, (2) planificación, (3) diseño y modelado, (4) implementación –entendida como desarrollo o construcción– y evaluación, (5) despliegue y (6) mantenimiento y evolución. Cada una de estas fases cuenta con un conjunto de tareas, entradas y salidas definidas que posibilitan el avance a las siguientes etapas y facilita la administración del proyecto. (Bernd Bruegge & Dutoit, 2002; Pressman & Maxim, 2021; Sommerville, 2011).

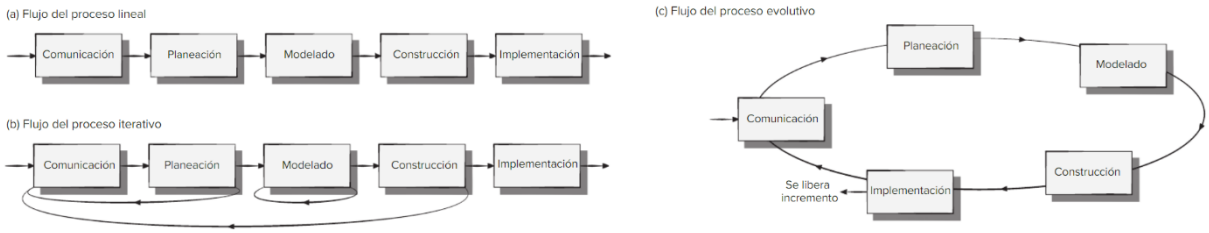
Figura 2. *Fases del ciclo de vida del desarrollo de software*



Nota. En la figura anterior se detallada las actividades principales llevadas a cabo en cada una de las etapas del ciclo de desarrollo de software. Elaboración propia con base en Pressman & Maxim (2021).

Tal como se mencionó en el apartado teórico de ingeniería de software, no existe un único modelo de proceso de ingeniería de software. Es decir, no existe un único modelo de ciclo de desarrollo software, sino múltiples que se pueden adaptar mejor o no según la naturaleza del proyecto, naturaleza de la aplicación, los métodos y herramientas a aplicar y/o los controles y estrategias requeridas (Pressman & Maxim, 2021). Sin embargo, estos modelos se pueden clasificar según su alcance del ciclo de vida; la cualidad y cantidad de las etapas; y la estructura y sucesión de las etapas. En este sentido, los modelos más comunes incluyen: ciclo de vida lineal, en cascada, por componentes, en V, iterativo, por prototipos, evolutivo, incremental, en espiral y orientado a objetos.

Figura 3. *Modelos de proceso de ingeniería de software*

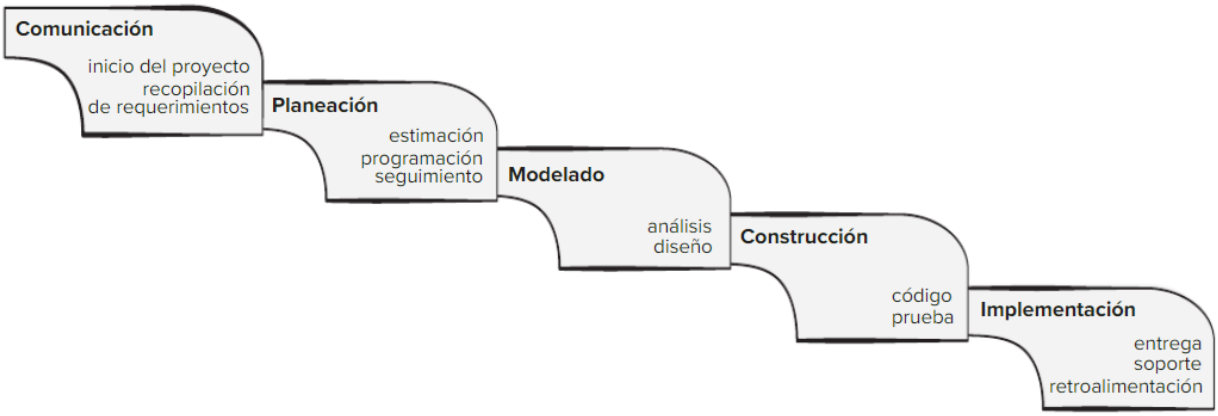


Nota. En la figura anterior se describe el flujo de proceso de software para los tres de los modelos de ciclo de vida de desarrollo de software más populares. Adaptada de Pressman & Maxim (2021).

Tal como se mencionó en el apartado teórico de ingeniería de software, no existe un único modelo de proceso de ingeniería de software. Es decir, no existe un único modelo de ciclo de desarrollo software, sino múltiples que se pueden adaptar mejor o no según la naturaleza del proyecto, naturaleza de la aplicación, los métodos y herramientas a aplicar y/o los controles y estrategias requeridas (Pressman & Maxim, 2021). Sin embargo, estos modelos se pueden clasificar según su alcance del ciclo de vida; la cualidad y cantidad de las etapas; y la estructura y sucesión de las etapas. Por otra parte, también se pueden clasificar o en modelo prescriptivo o modelo ágil. En este sentido, los modelos prescriptivos más comunes incluyen: ciclo de vida lineal o en cascada, por componentes, en V, iterativo, por prototipos, evolutivo, incremental, en espiral y orientado a objetos; y los modelos agiles incluyen: Scrum, XP, Kanban y DevOps.

El modelo de cascada es una elección adecuada cuando los requerimientos de un problema se entienden bien, están bien definidos y son en cierta medida estables. Se trata de un modelo secuencia y sistemático compuesto por las fases: definición de requerimientos, planeación, modelado, construcción e implementación (Pressman & Maxim, 2021).

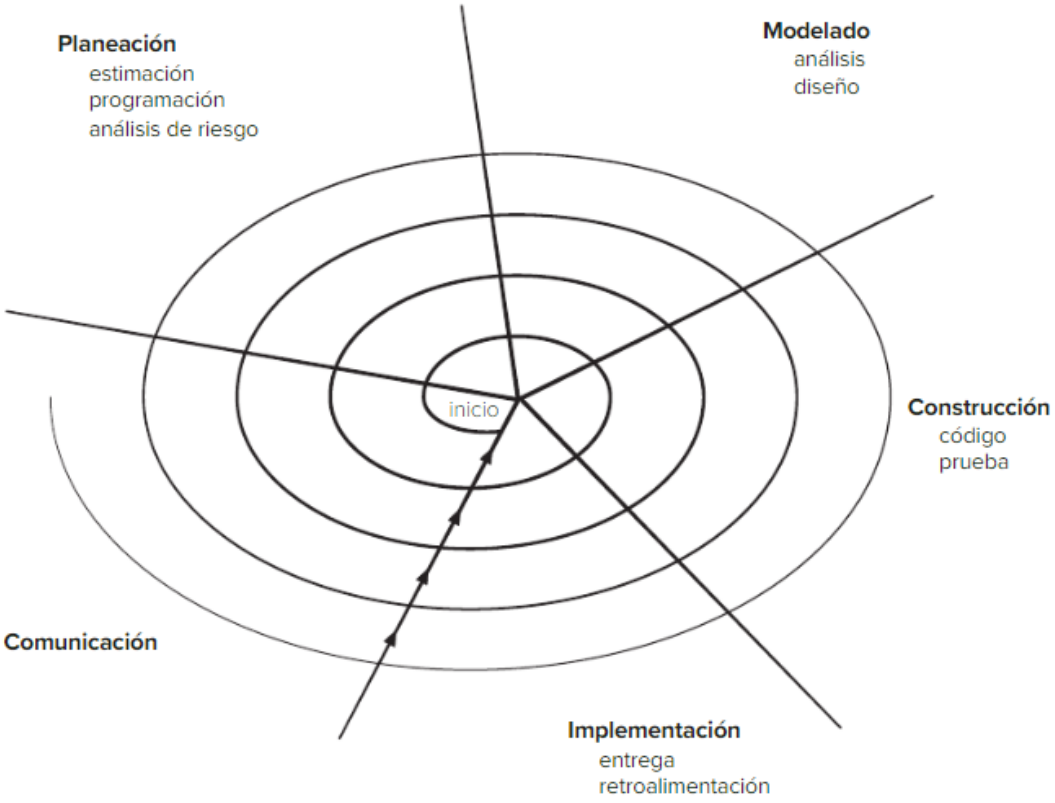
Figura 4. *Modelo de cascada*



Nota. En la figura anterior se describe el flujo de proceso de software para los tres de los modelos de ciclo de vida de desarrollo de software más populares. Adaptada de Pressman & Maxim (2021).

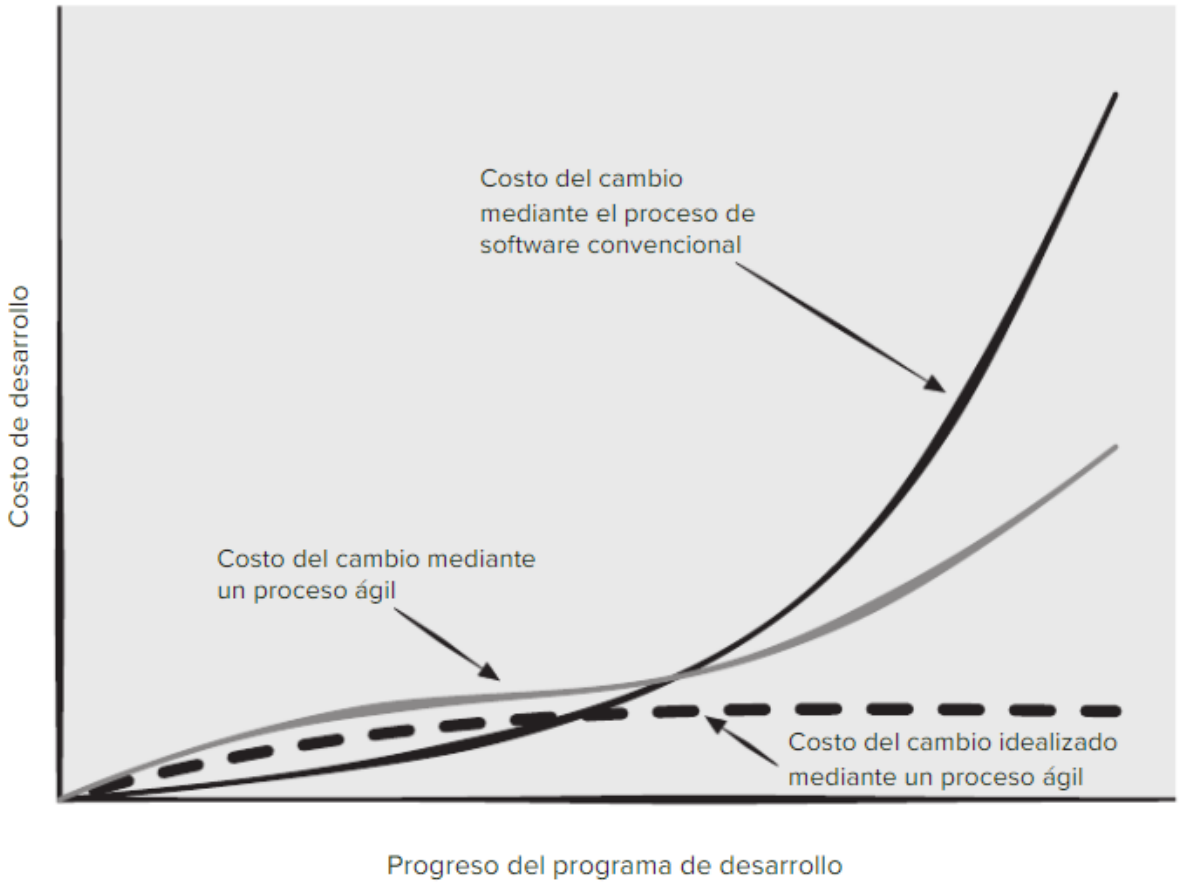
En la industria del software gestionar el cambio es algo inaudible; es decir, se disponen de dos opciones en un proyecto para abordarlo: evitarlo o tolerarlo. El modelo en espiral es un modelo incremental, basado en versiones o incrementos, que busca tolerar el cambio y de esta forma reducir los riesgos y el costo asociado a los cambios en el software. Se trata de un modelo que reconoce la naturaleza iterativa de los proyectos de software modernos y toma los elementos sistémicos y secuenciales del modelo lineal. En este modelo, cada incremento o versión añade funcionalidad al anterior, y de esta manera se va avanzando poco a poco, en contraste con las metodologías más lineales en las cuales se presentaba un producto de software terminado al final en vez de prototipos incrementales (Pressman & Maxim, 2021; Sommerville, 2011).

Figura 4. *Modelo en espiral*



Nota. En la figura anterior se describe el m odelo en espiral, se lee en sentido horario y comenzando en el centro (desde *Comunicación*). Adaptada de Pressman & Maxim (2021).

Figura 5. *Costo del cambio según el modelo de proceso*



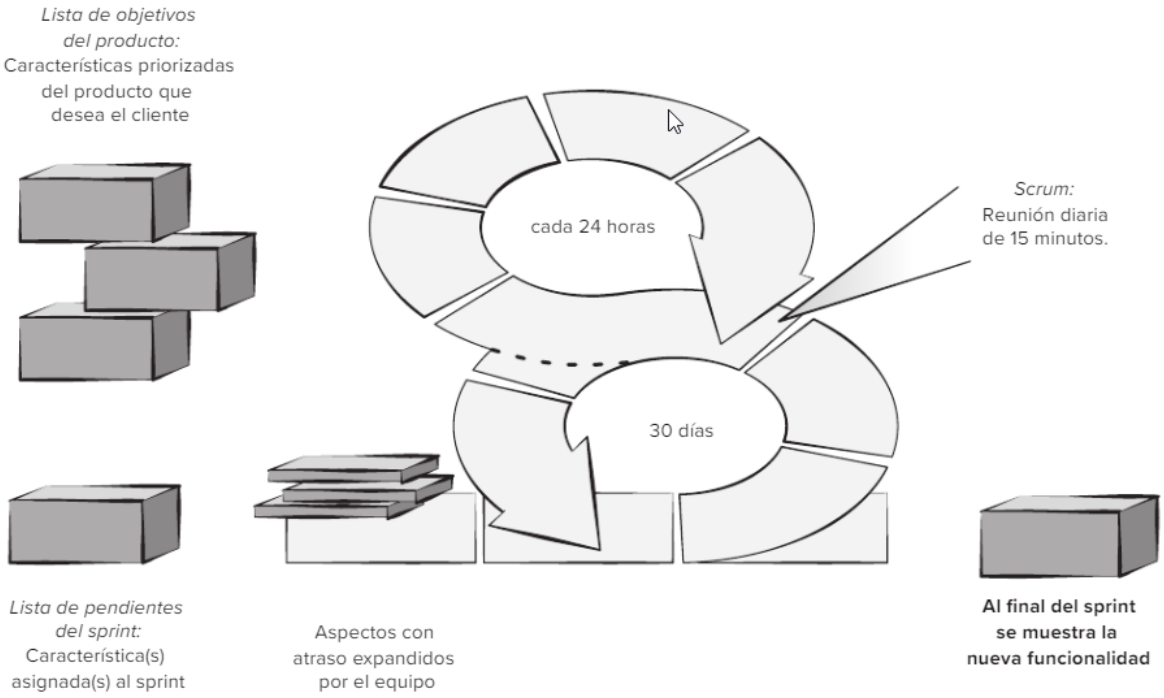
Nota. En la figura anterior se describe el costo de implementar un cambio en un proyecto de software a medida que avanza el tiempo. Adaptada de Pressman & Maxim (2021).

Por su parte, el modelo SCRUM es una metodología que se enfoca en la agilidad como mecanismo para enfrentar cambios de manera efectiva. Este enfoque no solo facilita la comunicación entre las partes interesadas, sino que también se centra en la entrega rápida de software operacional, caracterizándose por su adaptabilidad y flexibilidad. Las características generales de los modelos ágiles, y en particular de SCRUM, incluyen la integración de actividades de especificación, diseño e implementación, lo que permite una mayor fluidez y eficiencia en el desarrollo. Además, el sistema se desarrolla en versiones incrementales,

facilitando la adaptación a cambios y la mejora continua (Pressman & Maxim, 2021; Sommerville, 2011).

En el contexto actual, donde los entornos de negocio están en constante evolución, la idea de conjuntos de requerimientos de software estables a lo largo de todo el proyecto se ha vuelto poco viable. Los procesos tradicionales que buscan especificar completamente los requerimientos antes de pasar a las etapas de prueba y desarrollo no se alinean con la necesidad de desarrollo rápido de software. En este sentido, SCRUM ofrece una respuesta ágil mediante el desarrollo en incrementos, donde cada uno representa una mejora sobre el sistema anterior (Pressman & Maxim, 2021; Sommerville, 2011).

Figura 6. *Flujo del proceso de Scrum*



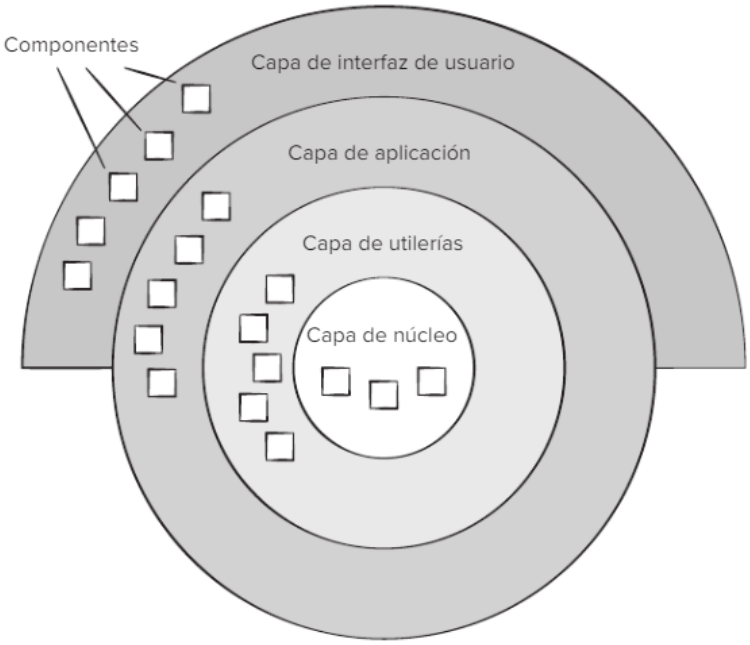
Nota. En la figura anterior se describe el flujo de proceso para el modelo de proceso Scrum. Adaptada de Pressman & Maxim (2021).

El flujo de proceso de Scrum se compone de varias etapas clave. Comienza con la Planificación del Sprint, donde se define el trabajo a realizar en un período fijo llamado sprint. Durante el sprint, el equipo se reúne diariamente en el Scrum Diario para discutir el progreso y los obstáculos. Se avanza en el desarrollo de las tareas seleccionadas, seguido de la Revisión del Sprint, donde se presenta el trabajo realizado, y la Retrospectiva del Sprint, que permite al equipo reflexionar sobre su desempeño y realizar mejoras. El proceso se cierra con el Refinamiento del Backlog, donde se preparan las tareas para futuros sprints. Este enfoque iterativo y colaborativo de Scrum promueve la entrega continua de valor y la mejora constante en los proyectos de desarrollo de software (Pressman & Maxim, 2021; Sommerville, 2011).

Por otra parte, la arquitectura de software proporciona el esqueleto y la guía conceptual para el desarrollo de sistemas de software, enfatizando la estructura y diseño de sus componentes, sus relaciones y cómo estos cumplen con los requisitos del sistema, tanto funcionales como no funcionales. Esta disciplina es esencial para garantizar la calidad, eficiencia y escalabilidad del software, al tiempo que facilita la comunicación entre los diferentes stakeholders del proyecto y proporciona una base común para los desarrolladores. Una parte importante de la arquitectura de software son las descripciones arquitectónicas, que pueden ser formales e informales, y ofrecen una visión integral de los sistemas, detallando sus componentes principales, relaciones e interacciones para cumplir con los requisitos establecidos. Estas descripciones, que incluyen diagramas, modelos y documentación relevante, son esenciales para comprender el diseño y la implementación del sistema. La selección de tecnologías y la adopción de patrones y estilos arquitectónicos específicos son decisiones arquitectónicas importantes en la organización del software. Dichas decisiones afectan directamente la calidad, el rendimiento, la mantenibilidad y la escalabilidad del sistema. Por lo tanto, deben tomarse con un enfoque

metódico y teniendo en cuenta las necesidades y limitaciones del proyecto (Pressman & Maxim, 2021; Sommerville, 2011).

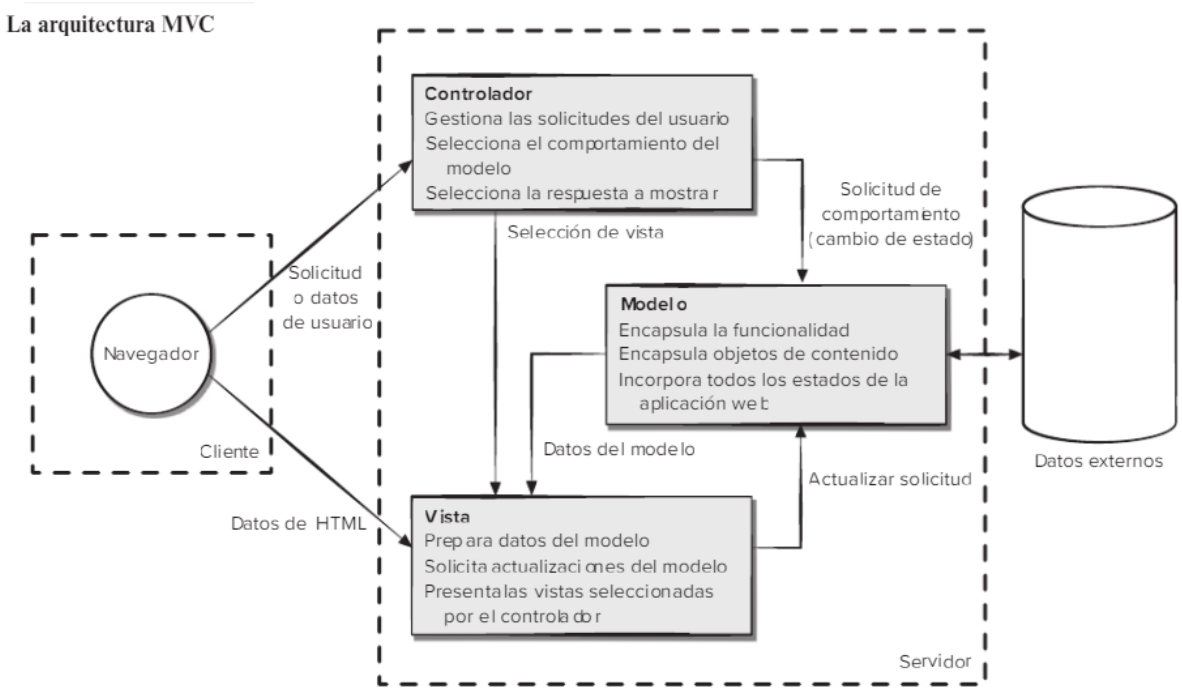
Figura 6. *Arquitectura por capas*



Nota. En la figura anterior se describe la arquitectura por capas. Adaptada de Pressman & Maxim (2021).

Las arquitecturas en capas, orientadas a servicios (SOA), basadas en eventos y microservicios son los estilos arquitectónicos más populares porque ofrecen soluciones a problemas comunes de diseño mediante enfoques probados que promueven cualidades como modularidad, reusabilidad y flexibilidad. El paradigma centrado a objetos y la programación orientada a objetos (OOP) se destacan por su enfoque en definir las entidades del dominio como objetos con responsabilidades y comportamientos específicos. El OOP también promueve principios como el polimorfismo, la herencia y la encapsulación para el desarrollo de sistemas modulares y extensibles (Pressman & Maxim, 2021; Sommerville, 2011).

***Figura** 7. Arquitectura MVC (Modelo-Vista-Controlador)*



Fuente: adaptado de Jacyntho, Mark Douglas, Schwabe, Daniel y Rossi, Gustavo, "An Architecture for Structuring Complex Web Applications", 2002, disponible en http://www-di.inf.puc-rio.br/schwabe/papers/OOHDMJava2%20Report.pdf

Robert Martin, Bertrand Meyer y Barbara Liskov (entre otros) en los años 2000 propusieron los principios SOLID (; por sus siglas en inglés). Estos principios surgen como una guía para el desarrollo de software de calidad, promoviendo código más limpio, escalable y resistente a cambios. A continuación, se describen cada uno de los principios (Michaels, 2022).

***Figura** 7. Arquitectura MVC (Modelo-Vista-Controlador)*

Nota. En la figura anterior se listan los principios SOLID (con sus significados en inglés), se trata de un acrónimo compuesto de cinco principios para la elaboración de software de calidad. Elaboración propia con base en Michaels (2022).

El principio de responsabilidad única, también conocido como principio de responsabilidad única, establece que una clase debe tener solo una razón para cambiar. En otras palabras, una clase debe ser responsable de una sola función del software, y esa función debe estar encapsulada en la clase. Una clase se vuelve más susceptible a cambios y más difícil de mantener si asume más de una responsabilidad (Michaels, 2022).

El principio abierto-cerrado, también conocido como principio abierto-cerrado, establece que las entidades de software, como clases, módulos, funciones, etc., deben estar abiertas para extensión, pero cerradas para modificación. Esto significa que el comportamiento de una entidad se puede extender sin cambiar su código fuente (Michaels, 2022).

El principio de sustitución de Liskov, también conocido como sustitución de Liskov, establece que los objetos de una superclase pueden ser sustituidos por los objetos de sus subclases sin afectar la ejecución adecuada del programa. En otras palabras, las subclases deben poder interactuar plenamente con las clases base de las que se derivan (Michaels, 2022).

El principio de segregación de interfaces, también conocido como "principio de segregación de interfaces", sostiene que ningún cliente debería verse obligado a depender de métodos que no usa. Las interfaces deben adaptarse a los requisitos del cliente y no obligar a las clases a usar métodos que no necesitan (Michaels, 2022).

El principio de inversión de dependencias, también conocido como principio de inversión de dependencias, establece que los módulos de alto nivel no deben depender directamente de los módulos de bajo nivel; en cambio, ambos deben depender de abstracciones. Además, las abstracciones no deben depender de los detalles, sino que los detalles deben depender de las abstracciones (Michaels, 2022).

Para poder entender la estructura y funcionamiento de un sistema es fundamental contar con una representación visual de su arquitectura. Tradicionalmente, el Lenguaje Unificado de Modelado (UML) es la herramienta estándar para esta tarea. Sin embargo, con la evolución de las tecnologías y metodologías, surgió la necesidad de un enfoque más moderno y adaptativo. Es aquí donde el modelo C4, creado por Simon Brown, se presenta como una respuesta innovadora (Richards & Ford, 2020).

El modelo C4 se basa en cuatro niveles de abstracción, conocidos como los cuatro C's: Contexto, Contenedor, Componente y Clase. Cada uno de estos niveles ofrece una perspectiva diferente y complementaria de la arquitectura del sistema, permitiendo a los diferentes interesados obtener la información que necesitan.

1. Contexto: La Vista Panorámica del Sistema

El Contexto es el nivel más alto del modelo C4 y proporciona una visión general del sistema en su conjunto. Aquí se identifican los usuarios finales, los sistemas externos y las

interacciones entre ellos y el sistema en cuestión. Esta vista de alto nivel es esencial para entender el propósito y la justificación del sistema, así como su interacción con el entorno más amplio (Richards & Ford, 2020).}

2. Contenedor: Definiendo la Infraestructura

El nivel de Contenedor detalla los diferentes entornos de despliegue, como servidores web, bases de datos y servicios en la nube. En esta etapa, se visualiza cómo se organizan y se comunican los componentes a un nivel más técnico. Esta vista es crucial para aquellos involucrados en la gestión de infraestructura y operaciones, ya que aclara cómo se distribuye el sistema en el hardware y otros recursos computacionales (Richards & Ford, 2020).

3. Componente: La Estructura Interna

El tercer nivel, Componente, se sumerge en la estructura interna del sistema. Aquí se desglosan las responsabilidades y se muestra cómo los diferentes componentes colaboran para realizar las funciones del sistema. Esta vista es especialmente relevante para los arquitectos de software y desarrolladores, ya que proporciona una guía clara sobre cómo está organizado el código y cómo fluyen los datos a través del sistema (Richards & Ford, 2020).

4. Clase: Detalles de Implementación

Finalmente, el nivel de Clase se asemeja a los diagramas de clases tradicionales en UML. Aunque no es un reemplazo directo de UML, este nivel complementa los anteriores al ofrecer una representación detallada de las clases y sus relaciones dentro del sistema. Esto es particularmente útil para los desarrolladores durante la implementación y mantenimiento del código (Richards & Ford, 2020).

Stack tecnológico del simulador de reactor químico: Python, Javascript, Flask, Vue.js, Docker.

Python es un lenguaje de programación de alto nivel, conocido por su legibilidad y simplicidad. Desde su creación por Guido van Rossum en 1991, Python ha ganado una inmensa popularidad en áreas como desarrollo web, automatización, análisis de datos y aprendizaje automático. Su ventaja radica en la facilidad para aprenderlo y su amplia biblioteca de módulos. Sin embargo, puede no ser tan rápido como otros lenguajes compilados. Su competidor más directo podría ser Ruby por su uso en aplicaciones web similares.

Por otra parte, Javascript es el lenguaje de programación dominante para el desarrollo web del lado del cliente. Creado por Brendan Eich en 1995, ha evolucionado desde agregar interactividad básica a las páginas web hasta ser la base de complejas aplicaciones web. Es prácticamente indispensable en el desarrollo front-end y tiene una comunidad muy activa. Aunque es increíblemente versátil, puede resultar complejo manejar sus aspectos asíncronos y su naturaleza dinámica. Typescript, un superconjunto de Javascript que añade tipado estático, es uno de sus competidores más notables.

Desde el lado de Python, Flask, es un micro framework que permite crear aplicaciones web de manera rápida y con una base simple pero extensible. Surgido en 2010 por Armin Ronacher, Flask se ha posicionado como una opción ideal para proyectos pequeños o medianos debido a su simplicidad y flexibilidad. No obstante, podría no ser la mejor opción para aplicaciones muy grandes o con mucha carga. Django, también en Python, es su competidor más cercano ofreciendo una solución más completa "out of the box".

Y por la parte de Javascript, Vue.js, es un framework progresivo de Javascript para construir interfaces de usuario. Creado por Evan You en 2014, Vue.js ha crecido en popularidad por su facilidad de integración y su enfoque en la capa de vista y la programación *frontend*. Su comunidad está en constante crecimiento y se considera una herramienta muy amigable para los

desarrolladores. Aunque es poderoso, proyectos muy grandes pueden beneficiarse más de soluciones como React o Angular, sus principales competidores.

Para finalizar, Docker es una plataforma de contenedores que permite empaquetar aplicaciones dentro de contenedores, asegurando que funcionen en cualquier entorno. Desde su lanzamiento en 2013 por Solomon Hykes, Docker ha revolucionado la manera en que se despliegan y se escalan las aplicaciones. Su principal ventaja es la portabilidad y la eficiencia en el uso de recursos. Como desventaja, puede tener una curva de aprendizaje más pronunciada y la dificultad al monitorizar sistemas complejos (que pueden estar compuestos por cientos o miles de contenedores).

METODOLOGÍA

Enfoque: En este proyecto se usa un enfoque cualitativo para obtener una comprensión completa de la efectividad y usabilidad de la interfaz de usuario, así como para comprender las necesidades y preferencias de los usuarios finales.

Tipo de investigación: Descriptiva (Se describe un proceso de simulación plasmado en un prototipo de software) debido a que su objetivo principal es proporcionar una descripción detallada y precisa de un tema específico, identificando sus características principales, patrones y tendencias. Se hizo uso de métodos como entrevistas, observaciones y análisis de datos para recopilar información detallada sobre el objeto de estudio.

Variables de Investigación:

Nombre de variable	**Descripción de variable**
Requerimientos para el software	Estas son las características funcionales y no funcionales que el software debe poseer para cumplir con los objetivos de diseño de reactores químicos. Pueden incluir aspectos como la capacidad de cálculo, generación de gráficos, y facilidad de uso, entre otros
Parámetros y variables de entrada del software	Estas son las formas en que los usuarios ingresarán datos al software para realizar la simulación de reactores químicos. Pueden incluir entradas numéricas que indican las condiciones iniciales del reactor y constantes, selección de parámetros para la construcción
Resultados esperados en el software (Salidas)	Estos son los resultados que se esperan obtener del software después de realizar la simulación de reactores químicos. Incluye datos numéricos, gráficos, tablas de resultados.

Teniendo en cuenta que el alcance del proyecto consiste en la creación de un producto mínimo viable para realizar simulaciones de reactores de reacción única en condiciones de idealidad y sin intercambiador de calor de manera eficiente, planteamos las siguientes preguntas para saber que se espera de este simulador por parte de un usuario:

P: ¿Cuáles considera que son las funciones fundamentales que debe tener un simulador de reactores?

R: Debe ser capaz de simular reactores CSTR, PFR, PBR y batch según las condiciones dadas por el usuario, también siendo capaz de procesar ecuaciones adaptadas de acuerdo con el contexto del reactor a simular, por ejemplo, las ecuaciones para el cálculo de velocidad de reacción, ecuaciones para caída de presión determinadas por la mecánica de fluidos o de caida de temperatura.

P: En cuanto a los datos de entrada de un simulador de reactores, ¿cuáles son los más relevantes a considerar para garantizar una simulación eficiente?

R: Primero, se debe conocer qué tipo de reactor se está simulando y en función de qué variable se van a construir las ecuaciones de diseño; Debe tener todas las condiciones iniciales que son necesarias para resolver el sistema de ecuaciones diferenciales en el reactor, por ejemplo, el flujo, fracciones, ecuación de reacción con coeficientes estequiométricos y especies químicas, que condiciones tiene esa reacción y que volumen tiene el reactor, en el caso de un reactor de tipo tanque (CSTR o batch).

P: Respecto a los datos de salida de un simulador de reactores, ¿qué información considera importante que se genere como resultado del software?

R: Gráficos que permitan entender la evolución de la reacción y la distribución de las variables en el reactor. Así como gráficos que describan las condiciones físicas y químicas que están sucediendo en el reactor a medida que cambia la variable independiente, por ejemplo, cómo se comporta la presión y temperatura en el reactor, ese cambio como afecta la producción de la especie química deseada, como cambia la fracción de cada especie química a lo largo del reactor y cuál es el punto de la gráfica en la cual el usuario queda satisfecho con el resultado de llevar a cabo la reacción, por ejemplo, que volumen debe tener el reactor, cuanto tiempo debo

dejar el reactor funcionando para alcanzar la composición deseada o cuanta masa de catalizador es necesaria para que ocurra la reacción esperada.

P: ¿Qué opina sobre el uso de software libre en el desarrollo de simuladores de reactores? ¿Cuáles serían las ventajas y desventajas de esta elección?

R: Opino que dependiendo de cómo se tome dicho codigo libre, puede afectar de manera positiva a la industria y a la academia, ya que permite el acceso a alternativas funcionales que son muy utilizadas hoy día, sin el inconveniente del costo que es un gran problema a la hora de usar herramientas como MATLAB o ASPEN. Por el lado de las ventajas, el acceso gratuito o a un costo mucho menor que otras herramientas ya mencionadas, puede permitir la posibilidad de modificar el código si uno requiere algo especifico y la existencia de una comunidad que puede adicionarle características en caso de que los desarrolladores originales no continúen dando soporte. Por otro lado, las desventajas que tiene ese código libre es que el soporte técnico queda en manos de la comunidad y no es especializado, y dependiendo de los desarrolladores puede no tener funciones avanzadas que, si tienen aplicaciones de pago, además de que se requieren recursos propios para implementar y mantener dicho software.

P: En el proceso de validación de un simulador de reactores, ¿qué criterios considera importantes para asegurar la precisión y confiabilidad de los datos de entrada y salida?

R: El software debe asegurarse que la construcción del modelo matemático para llevar a cabo la simulación este correctamente estructurado, y que cada una de las variables este debidamente referenciada en el código interno de la simulación. El que el usuario ingrese los datos y requisitos mediante casillas y menús desplegables remueve el factor humano, al menos hasta cómo compilará el modelo adecuadamente, solo requiriendo del ingreso de datos numéricos y ecuaciones ya referenciados y compilados en el software.

P: ¿Qué parámetros de entrada necesitan ingresar los estudiantes para simular un reactor?
R: Los estudiantes necesitan ingresar parámetros como temperatura, presión y condiciones de la reacción para poder simular el comportamiento del reactor bajo diferentes condiciones.

P: ¿Qué tipo de reacciones químicas debería poder simular el software en su versión inicial?

R: El software debería poder simular una reacción química única en condiciones ideales para estudiar el comportamiento del reactor. De ese modo, se puede entender la teoría de diseño de reactores al menos en un entorno académico.

P: ¿Cómo debería ser la interfaz del software para que sea accesible a usuarios novatos?
R: La interfaz debería ser fácil de usar y navegar para que los usuarios novatos puedan aprender a usar el software rápidamente sin mucha dificultad.

P: ¿Qué tipos de resultados visuales esperan los usuarios obtener de la simulación?
R: Los usuarios esperan ver gráficos y tablas que representen los resultados de la simulación para analizar cómo varían las variables clave del reactor.

P: ¿Es importante que los usuarios puedan guardar y cargar sus simulaciones? ¿Por qué?
R: Sí, es importante porque los investigadores y estudiantes necesitan guardar sus simulaciones y cargarlas posteriormente para continuar su trabajo en diferentes sesiones sin perder datos.

P: ¿Cuánto cuesta actualmente el acceso a software de simulación de reactores y cómo afecta esto a los usuarios?
R: El software de simulación de reactores suele ser costoso, lo que restringe su accesibilidad. Los usuarios desean que el software sea de acceso abierto y gratuito para poder utilizarlo sin restricciones económicas.

P: ¿Qué recursos adicionales necesitan los usuarios para maximizar el uso del software?
R: Los usuarios necesitan acceso a documentación y tutoriales detallados para comprender mejor cómo usar el software y los conceptos teóricos detrás de las simulaciones.

ANALISIS DE RESULTADOS

Partiendo de la entrevista, se obtienen varios elementos importantes, tales como que el simulador de reactores debe ser capaz de generar diferentes tipos de reactores, como CSTR, PFR, PBR y batch, según las necesidades del usuario. Además, es importante que pueda manejar ecuaciones adaptadas para calcular la velocidad de reacción y otros aspectos importantes. En cuanto a los datos de entrada, es esencial que el usuario pueda especificar todas las condiciones iniciales relevantes para la simulación, como el tipo de reactor, las ecuaciones de reacción y las condiciones del medio. Por otro lado, los datos de salida deben incluir gráficos que muestren la evolución de la reacción y la distribución de variables en el reactor, así como información sobre las condiciones físicas y químicas a lo largo del proceso. Finalmente, en el proceso de validación, es importante que el modelo matemático esté correctamente estructurado y que se minimice el factor humano en el diseño, asegurando así una mayor precisión y confiabilidad de los resultados de la simulación.

ID DE HISTORIA DE USUARIO	Como <tipo de usuario>	Quiero <realizar alguna tarea>	**para que pueda <el logro algún objetivo>**
1	Usuario del software	Poder simular diferentes tipos de reactores y personalizar las ecuaciones de diseño para cada uno	Modelar con precisión las reacciones, en condiciones especificas
2	Usuario del software	Poder ingresar todas las condiciones de inicio relevantes para la simulación	Garantizar la precisión de los resultados de la simulación
3	Usuario del software	Obtener gráficos que representen la evolución de la reacción y la	Analizar el comportamiento del sistema

		distribución de las variables en el reactor	
4	Usuario del software	Obtener datos que describan cómo cambian las condiciones físicas y químicas a lo largo del reactor	Tomar decisiones informadas sobre el diseño y operación de un reactor
5	Usuario del software	Tener precisión y confiabilidad en los datos de entrada y salida del simulador	Utilizar la información para la creación de reactores de buena calidad y características
6	Estudiante de Ingeniería Química	Quiero ingresar parámetros de entrada como temperatura, presión y condiciones de la reacción	Para poder simular el comportamiento de un reactor bajo diferentes condiciones
7	Estudiante de Ingeniería Química	Quiero simular una reacción química única en condiciones ideales	Para estudiar el comportamiento del reactor sin complicaciones adicionales
8	Usuario novato	Quiero una interfaz fácil de usar y navegar	Para poder aprender a usar el software rápidamente sin mucha dificultad
9	Estudiante de Ingeniería Química	Quiero ver gráficos y tablas que representen los resultados de la simulación	Para analizar cómo varían las variables clave del reactor
10	Usuario del software	Quiero que el software sea de acceso abierto y gratuito	Para poder utilizarlo sin restricciones económicas

11	Estudiante de Ingeniería Química	Quiero tener acceso a documentación detallada	Para comprender mejor cómo usar el software y los conceptos teóricos detrás de Desalas simulaciones
12	Ingeniero Químico	Quiero comparar los resultados de diferentes simulaciones	Para evaluar cómo cambios en los parámetros de entrada afectan el comportamiento del reactor
13	Usuario Avanzado	Quiero personalizar las ecuaciones de diseño y los modelos utilizados en las simulaciones	Para adaptar el software a necesidades específicas de investigación o enseñanza

Solución - Desarrollo de solución:

Contexto del Sistema:

El diagrama de contexto proporciona una visión general del sistema y su entorno. En este, se identifica los actores principales que interactúan con el sistema y las relaciones entre ellos. Como se puede ver en la figura 8, los actores principales son el usuario que utilizara el software, siendo este un usuario final cualquiera, o usuarios más especializados tales como estudiantes de campos relacionados con la química, investigadores e ingenieros químicos. Estos interactúan con el simulador web, proveyéndole datos que el simulador enviara al backend, para que este transforme los datos y los convierta a gráficas, que serán mostradas al usuario por medio del simulador para que este los pueda utilizar.

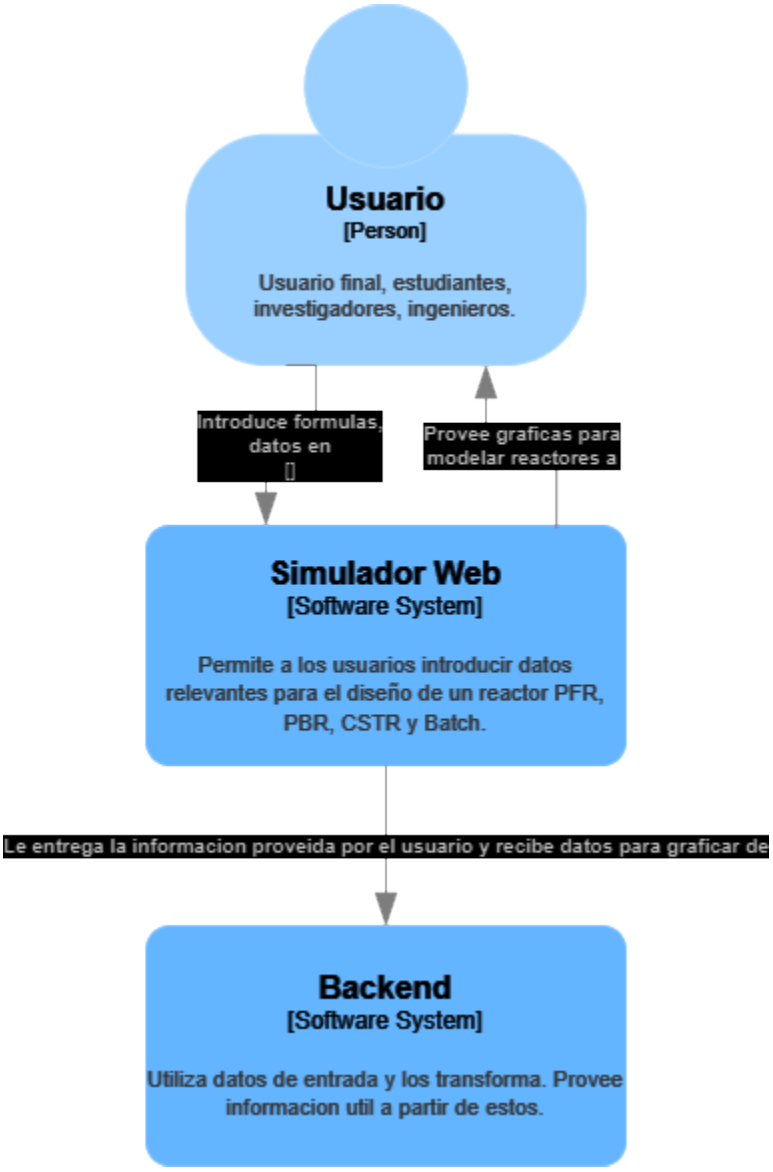


Figura 8. Contexto del sistema del software, a partir del modelo C4.

Diagrama de Contenedores:

En este diagrama se muestra más a detalle como el usuario interactúa con el software. Este ingresa a la aplicación web por medio de un enlace o, inicialmente, por medio de una dirección ip propia asignada al programa. Este lo dirige a la aplicación y a la interfaz de una sola página, donde le es posible introducir distintos valores, incluyendo formulas personalizadas. Esto

se conecta por medio de una API al backend, que se encarga de realizar las transformaciones de estos datos de entrada para mostrarlos posteriormente.

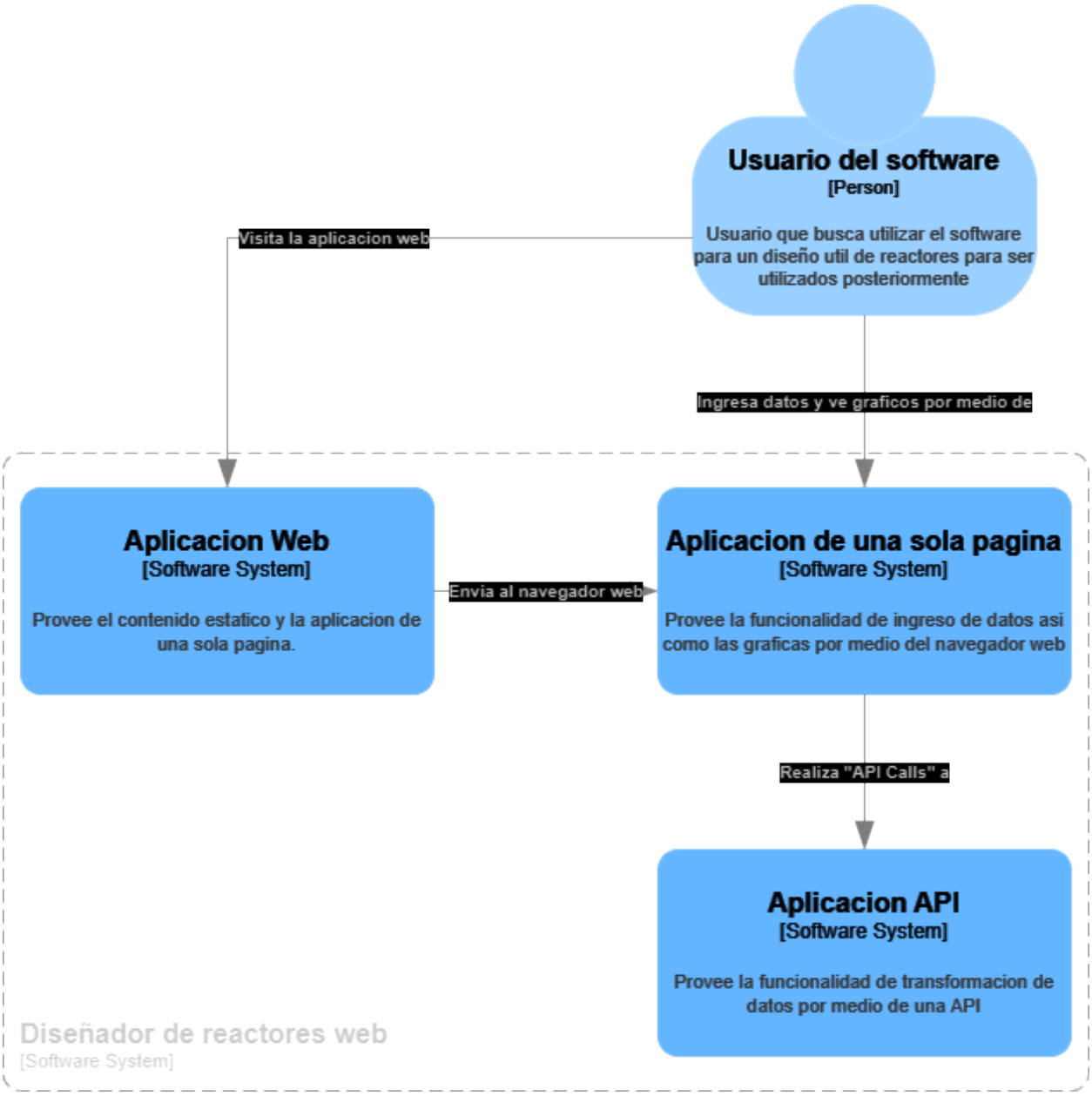


Figura 9. Diagrama de contenedores.

Diagrama de Componentes:

En este diagrama se desglosan los componentes de la aplicación que se va a crear, los cuales se separan en el frontend y el backend. El frontend se encarga de toda la parte visual como

se puede ver en la figura 10. Este es el que muestra la interfaz donde el usuario introduce los datos, así como donde se muestran las gráficas al usuario. Cuando se introducen los datos y se le da a graficar, estos son enviados al backend, el cual se encarga de elegir el modelo correcto, con las ecuaciones correctas para luego enviar la información de manera correcta al frontend, con el fin de que este pueda crear las gráficas apropiadas.

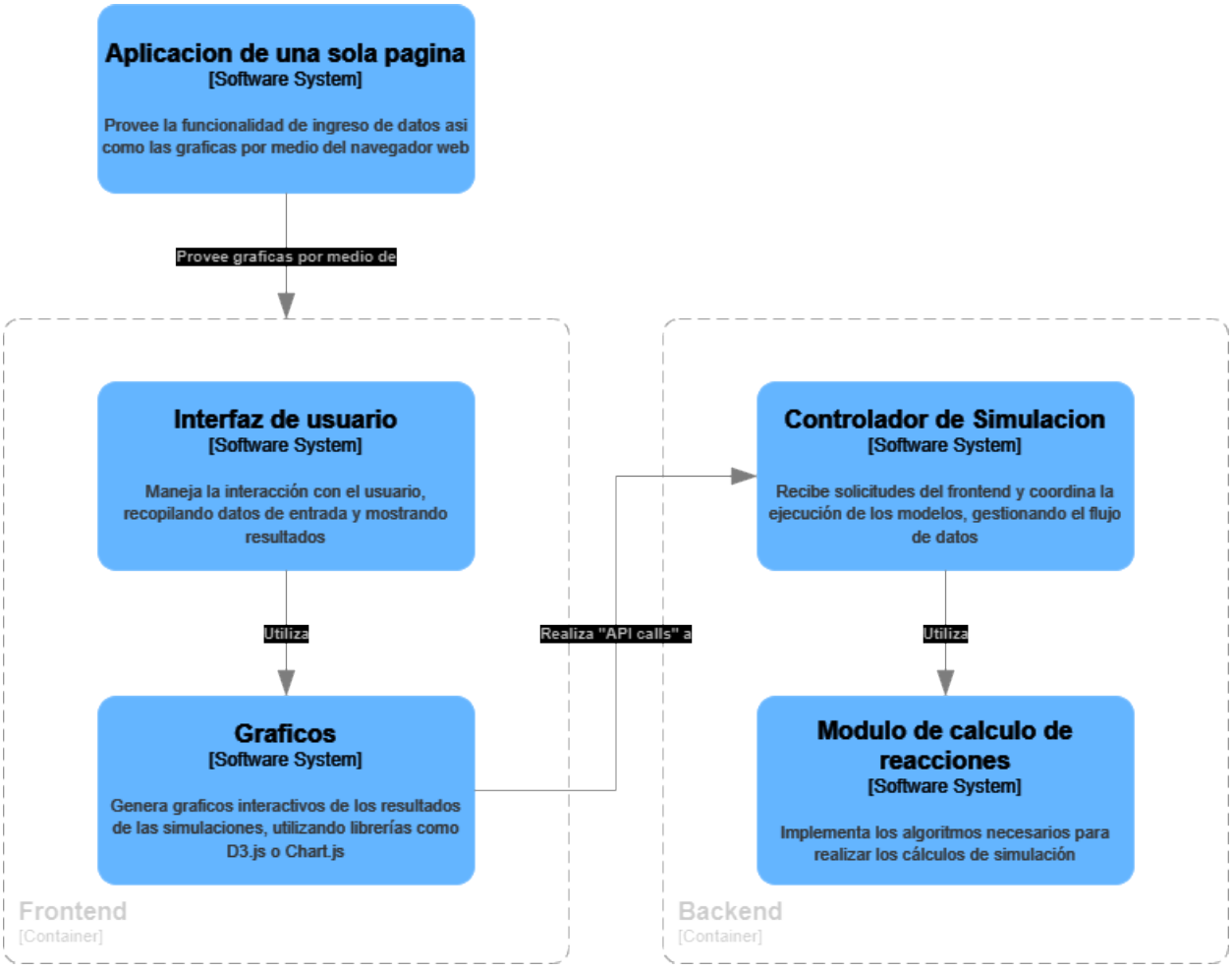


Figura 10. Diagrama de componentes.

Como parte del proceso de diseño y desarrollo del software, se elaboró un diagrama de casos de uso que describe las diferentes interacciones entre los usuarios y el sistema. Este diagrama permitió identificar y estructurar las funcionalidades clave que el software debe ofrecer.

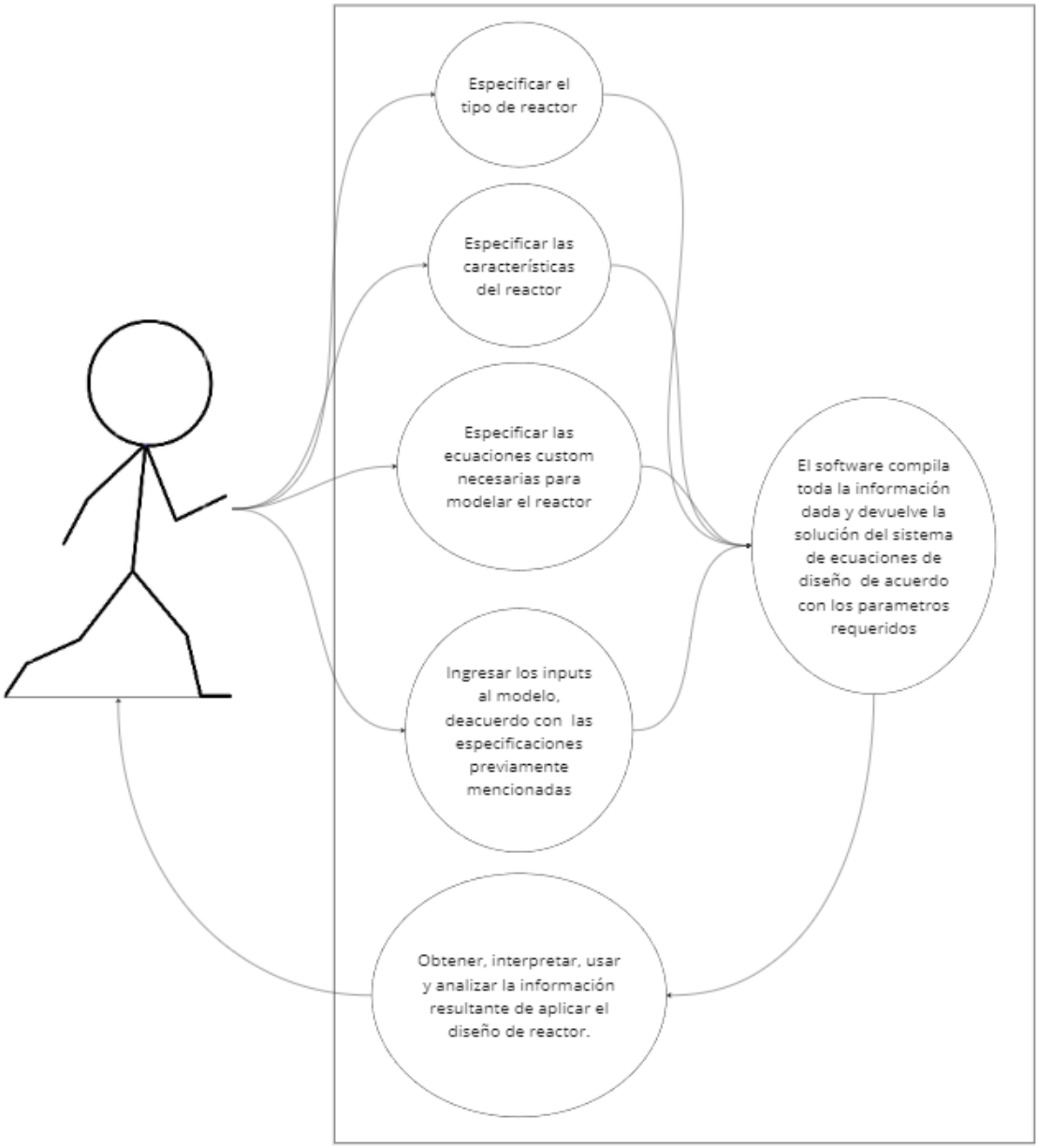


Figura 11. Diagrama de casos de uso en el software.

Contexto del Sistema

- **Usuarios finales:** Estudiantes de campos relacionados con la química, investigadores e ingenieros químicos que utilizan el software para realizar simulaciones de reactores.

- **Simulador web:** Interactúa con los usuarios, transformando datos ingresados en gráficos y resultados útiles.

Historias de Usuario y Diagrama de Contexto

- **HU1:** "Como usuario del software, quiero poder simular diferentes tipos de reactores y personalizar las ecuaciones de diseño para cada uno, para modelar con precisión las reacciones en condiciones específicas".
- **Contexto:** El simulador debe permitir la configuración de diferentes tipos de reactores (CSTR, PFR, PBR y batch) y personalización de ecuaciones.
- **HU8:** "Como usuario novato, quiero una interfaz fácil de usar y navegar, para poder aprender a usar el software rápidamente sin mucha dificultad".
- **Contexto:** La interfaz del simulador debe ser intuitiva y accesible, facilitando el aprendizaje rápido del usuario novato.

Diagrama de Contenedores

- **Aplicación web:** Los usuarios ingresan datos a través de una interfaz de una sola página que se conecta al backend mediante una API.
- **Backend:** Realiza transformaciones de los datos de entrada y devuelve los resultados al frontend para ser presentados en gráficos.

Historias de Usuario y Diagrama de Contenedores

- **HU2:** "Como usuario del software, quiero poder ingresar todas las condiciones de inicio relevantes para la simulación, para garantizar la precisión de los resultados".
- **Contenedores:** El frontend debe proporcionar formularios adecuados para la entrada de condiciones iniciales, y el backend debe procesar estos datos para simulaciones precisas.
- **HU9:** "Como estudiante de Ingeniería Química, quiero ver gráficos y tablas que representen los resultados de la simulación, para analizar cómo varían las variables clave del reactor".

- **Contenedores:** El backend debe enviar los datos procesados al frontend para generar gráficos y tablas comprensibles.

Diagrama de Componentes

- **Frontend:** Maneja la interfaz de usuario y la visualización de gráficos.
- **Backend:** Selecciona y ejecuta los modelos matemáticos apropiados basados en las entradas del usuario.
- **HU6:** "Como estudiante de Ingeniería Química, quiero ingresar parámetros de entrada como temperatura, presión y condiciones de la reacción, para poder simular el comportamiento de un reactor bajo diferentes condiciones".
- **Componentes:** El frontend debe incluir campos de entrada para estos parámetros y el backend debe ajustar los modelos de simulación en consecuencia.
- **HU12:** "Como ingeniero químico, quiero comparar los resultados de diferentes simulaciones, para evaluar cómo cambios en los parámetros de entrada afectan el comportamiento del reactor".
- **Componentes:** El backend debe soportar la comparación de múltiples simulaciones y el frontend debe presentar estas comparaciones de manera clara.

Ejemplos de casos de uso

Caso 1: Reactor PBR (Conversión y flujo)

Para probar una simulación en PFR, se planteó el siguiente modelo especificando los inputs de entrada y ecuaciones personalizadas:

FT0 (mol/min)	40
yA0	0.33
yB0	0.67
P0 (atm)	5
T0(°C)	170
T0(K)	443.15

ka' (molA/KgCat min atm)	1.74

Figura 12. Inputs caso 1.

$$rA' = k_a' \cdot PB$$

$$a + 2b \rightarrow c$$

$$PB = \frac{FB}{FT} \cdot P$$

Figura 13. Ecuaciones para caso 1.

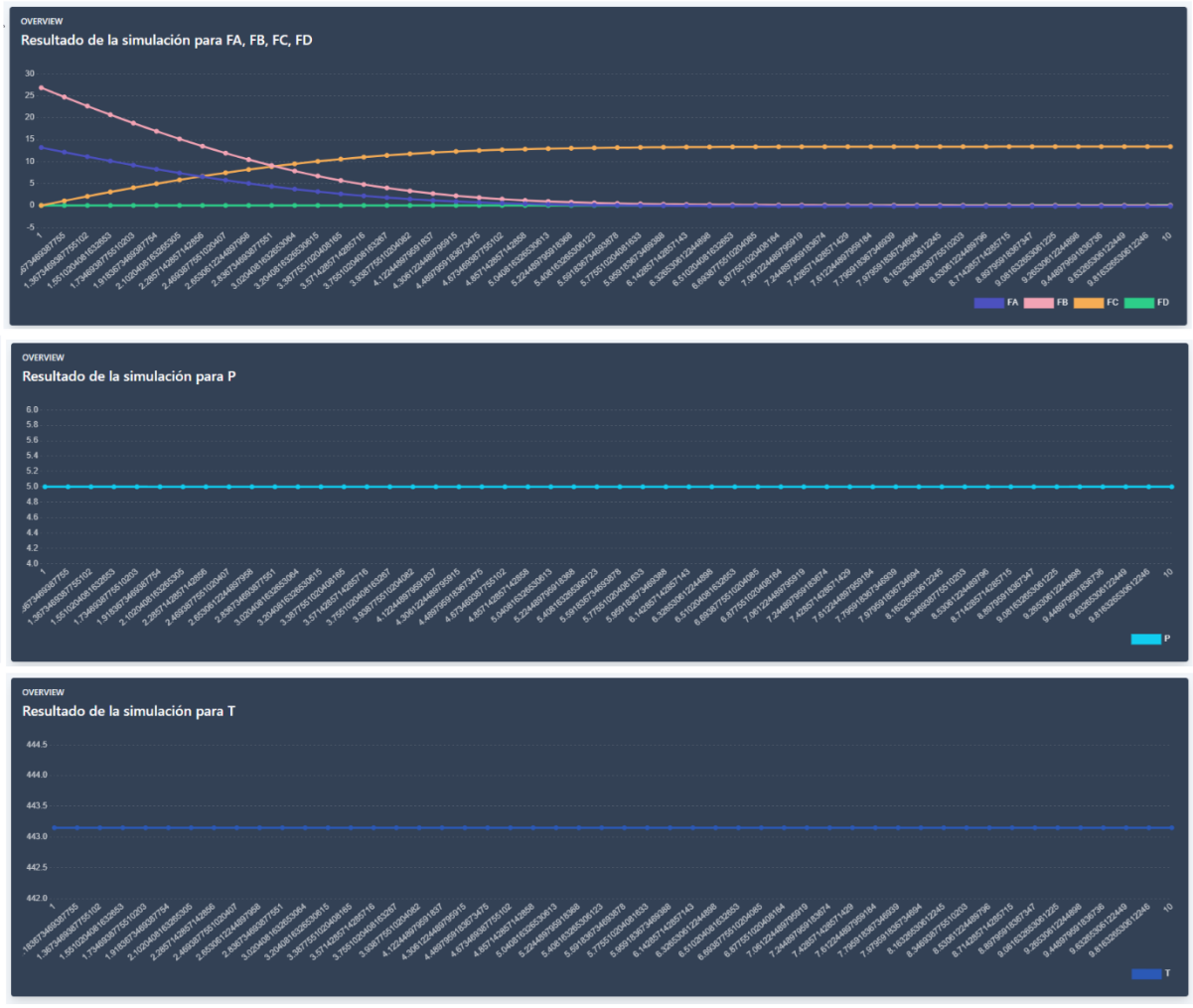


Figura 14. Resultados para caso 1 flujos y conversión.

Caso 2: Reactor PFR (Conversión y flujo)

Para probar una simulación en PFR, igualmente se planteó el siguiente modelo especificando los inputs de entrada y ecuaciones personalizadas:

FA0 (mol/s)	1
yA0	0.72
yB0	0.0035
yI0	0.2765
yC0	0
T0 (°C)	190
T0(K)	463.15
P0 (atm)	1.3
CA0 (mol/L)	0.024645653
CA0 (mol/cm3)	2.46457E-05
R (cal/molK)	1.9872
R (atm L/mol K)	0.082
k (cm3/mol)^3/2 1/s	22010.41894

Figura 15. Inputs caso 2.

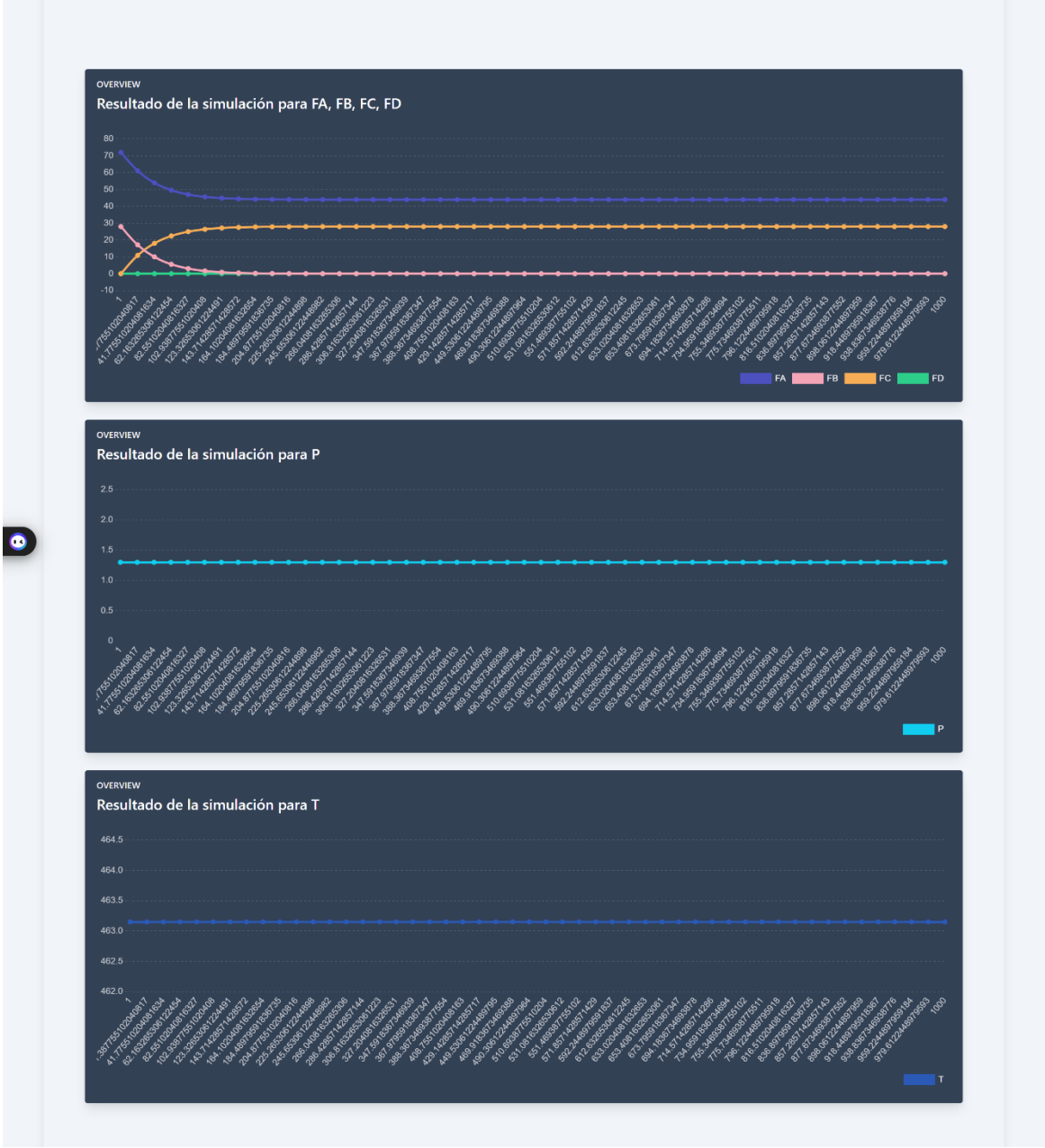


Figura 16. Resultados caso 2.

Caso 3: Reactor Batch (Conversión y flujo)

Para probar una simulación en Batch, se planteó el siguiente modelo especificando los inputs de entrada y ecuaciones personalizadas:

FT0 (mol/min)	40
yA0	0.33
yB0	0.67
P0 (atm)	5
T0(°C)	170
T0(K)	443.15
ka' (molA/KgCat min atm)	1.74

Figura 17. Inputs caso 3.

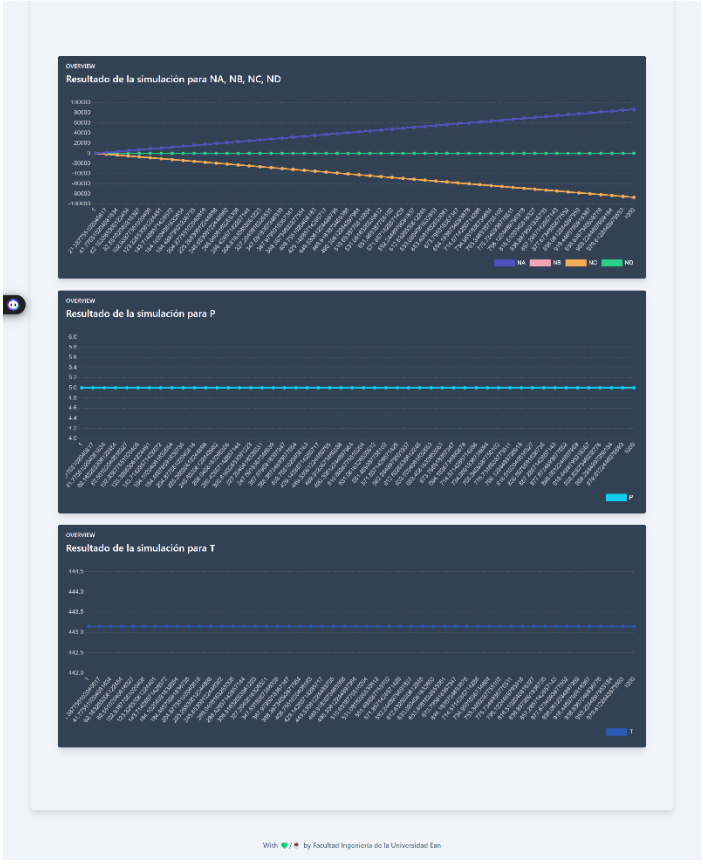


Figura 18. Ecuaciones para caso 3.

DISCUSIÓN

El desarrollo del prototipo de diseño de reactores, como producto mínimo viable (MVP), busca resolver varios desafíos en la enseñanza y simulación de reactores químicos. Tradicionalmente, los estudiantes y profesionales de la ingeniería química han dependido de herramientas como Excel, MATLAB y Aspen para la simulación de reactores. Sin embargo, estas herramientas presentan tanto ventajas como desventajas, por ejemplo que tiene costos elevados, accesibilidad restringida y una curva de aprendizaje poco pronunciada debido a interfaces poco intuitivas.

Nuestro software se centra la eficiencia y facilidad de uso, ofreciendo una solución accesible y gratuita. Al permitir a los usuarios definir condiciones de entrada y generar resultados basados en ecuaciones de diseño ajustadas a esas condiciones, se facilita el aprendizaje de conceptos complejos mediante una interfaz intuitiva y simplificada. Este enfoque, se pudo ver que el prototipo funcional dio los resultados esperados.

La implementación de un diagrama de casos de uso fue clave en el desarrollo del MVP. Este diagrama permitió identificar claramente las interacciones entre los usuarios y el sistema, asegurando que las funcionalidades clave estuvieran bien definidas y alineadas con las necesidades de los usuarios. Las historias de usuario derivadas de este análisis nos guiaron en el diseño de una interfaz que no solo es fácil de usar, sino también capaz de proporcionar resultados precisos y visualmente comprensibles.

En términos de accesibilidad, el software se diseñó para ser de código abierto y gratuito, eliminando las barreras económicas que a menudo impiden a los estudiantes y profesionales acceder a herramientas de simulación avanzadas. Este enfoque promueve la democratización del conocimiento y facilita el aprendizaje autónomo, permitiendo a los usuarios explorar y experimentar con simulaciones de reactores sin restricciones.

Este proyecto aborda eficazmente las limitaciones de las herramientas existentes y proporciona una solución viable que mejora significativamente la experiencia de aprendizaje y simulación de reactores químicos. Con la combinación de accesibilidad, facilidad de uso y precisión en la parametrización, este software puede convertirse en un recurso de alta utilidad para estudiantes y profesionales de la ingeniería química.

CONCLUSIONES

Durante el desarrollo de este proyecto, se han podido ver distintos puntos de vista gracias a que este es un equipo multidisciplinario. Es así que, gracias al conocimiento de cada área, se llegó a un proyecto que permitió el entrelazado de dicho conocimiento, donde se identificó un problema de parte del área de la Ingeniería Química, viable solucionar con la aplicación de la Ingeniería de Sistemas.

Se ha construido un Producto Mínimo Viable (MVP) en forma de un software con interfaz web para la simulación de reactores de reacción única, en condiciones ideales y sin intercambiador de calor. Se enfoca en ofrecer una herramienta de aprendizaje gratuito, de código abierto, enfatizando la parametrización, eficiencia y facilidad de uso para los usuarios.

Se examinaron las ecuaciones de diseño necesarias para simular cada tipo de reactor, pudiendo ejecutarse en un programa adaptado a los requerimientos indicados por el usuario, de forma que el programa también ha hecho cada cálculo algebraico de condiciones físicas y químicas internas del reactor con respecto al cambio de la variable independiente.

Se determinó que la pertinencia del proyecto radica en la importancia de la adaptación del diseño de reactores químicos con las necesidades del usuario, ya que el programa permite solucionar las condiciones únicas de cada reacción, mejorando así la eficiencia, y efectividad de la planeación de reactores químicos pudiendo reducir costos operativos. El proyecto se posiciona en la intersección de la ingeniería química y la ingeniería de software, aprovechando los avances tecnológicos en el desarrollo de software para ofrecer una solución tangible a un problema complejo y multidimensional.

Se destaco la importancia de seguir un ciclo de vida de desarrollo de software estructurado, aplicando principios y patrones de diseño para garantizar la calidad, mantenibilidad y escalabilidad del producto final. En última instancia, el proyecto aspira a generar un recurso de aprendizaje, difundiendo el conocimiento y el acceso a tecnologías para el diseño y simulación de reactores químicos.

BIBLIOGRAFÍA

Altiok, M., & Melamed, B. (2007). *Simulation modeling and analysis with arena*. Elseiver Inc.

Bernd Bruegge, & Dutoit, A. H. (2002). Object-oriented software engineering: Conquering complex and changing systems (1st ed.). Pearson (Prentice Hall). https://www-ebooks7-24-com.bdbiblioteca.universidadean.edu.co/?il=4337&pg=1 (Original work published 2000)

Grady, J. O. (2014). *System requirements analysis* (2nd ed.). Elsevier Inc. https://www.sciencedirect.com/book/9780124171077/system-requirements-analysis

Holcombe, M. (2008). *Running an agile software development project*. John Wiley & Sons.

IEEE Standards Association. (2000). IEEE recommended practice for architectural description for software-intensive systems. In *Institute of Electrical and Electronics Engineers (IEE)*. Institute of Electrical and Electronics Engineers (IEE).

International Standards Organization. (2017). ISO 12207. https://www.iso.org/standard/63712.html

Papazoglou, M. (2008). *Web services: Principles and technology*. Prentice Hall.

Portnoy, M. (2016). *Virtualization essentials* (2nd ed.). John Wiley & Sons.

Pressman, R. S., & Maxim, B. R. (2021). *Ingeniería del software: Un enfoque práctico*. Mcgraw-Hill.

Shornikov, Y. V., & Popov, E. A. (2020). Modeling and simulation of chemical processes using the hybrid methodology. *IOP Conference Series: Materials Science and Engineering*, *734*(1), 012079–012079. https://doi.org/10.1088/1757-899x/734/1/012079

Spinellis, D. (2005). Version control systems. *IEEE Software*, *22*(5), 108–109. https://doi.org/10.1109/ms.2005.140

Agbebi, R., & Sandrock, C. (2015). Best of breed control of platinum precipitation reactors. In *Computer-aided chemical engineering* (pp. 1661–1666). https://doi.org/10.1016/b978-0-444-63577-8.50122-4

Anaya-Durand, A., Cauich-Segovia, G. I., Funabazama-Bárcenas, O., & Gracia-Medrano-Bravo, V. A. (2014). Evaluación de ecuaciones de factor de fricción explícito para tuberías. *Educación Química*, *25*(2), 128–134. https://doi.org/10.1016/s0187-893x(14)70535-x

Anculle-Arauco, V., Krüger-Malpartida, H., Arévalo-Flores, M., Correa-Cerdeño, L., Maß, R., Hoppe, W., & Pedraz-Petrozzi, B. (2022). Content validation using Aiken methodology through expert judgment of the first Spanish version of the Eppendorf Schizophrenia Inventory (ESI) in Peru: A brief qualitative report. *Revista De Psiquiatría Y Salud Mental*. https://doi.org/10.1016/j.rpsm.2022.11.004

Asale, R.-., & Rae. (n.d.). *reactor | Diccionario de la lengua española*. «Diccionario De La Lengua Española» - Edición Del Tricentenario. https://dle.rae.es/reactor

ECUACIÓN QUÍMICA. (n.d.). Encyclopedia Britannica. https://moderna-eb-com.bdbiblioteca.universidadean.edu.co/levels/academica/article/ECUACI%C3%93N-QU%C3%8DMICA/577265

Fogler, H. S. (2008). *Elements of Chemical Reaction Engineering* (4th ed.). Pearson Educación.

Green, D. W., & Southard, M. Z. (2019). *Perry's Chemical Engineers' Handbook, 9th Edition* (9th ed.). McGraw Hill Professional.

Jaibiba, P., Vignesh, S., & Hariharan, S. (2020). Working principle of typical bioreactors. In *Elsevier eBooks* (pp. 145–173). https://doi.org/10.1016/b978-0-12-821264-6.00010-3

Mathematics and Optimization - MATLAB & Simulink - MathWorks América Latina. (n.d.). https://la.mathworks.com/help/overview/mathematics-and-optimization.html?s_tid=hc_panel

Smith, R. L., Inomata, H., & Peters, C. J. (2013). Systems, devices and processes. In *Supercritical fluid science and technology* (pp. 55–119). https://doi.org/10.1016/b978-0-444-52215-3.00002-7

Trambouze, P., Euzen, J., & Bononno, R. (2005). Chemical reactors: from design to operation. *Choice Reviews Online*, *42*(05), 42–2834. https://doi.org/10.5860/choice.42-2834

Michaels, P. (2022). Software architecture by example. In *Apress eBooks*. Apress Berkeley. https://doi.org/10.1007/978-1-4842-7990-8

Richards, M., & Ford, N. (2020). *Fundamentals of software architecture: An engineering approach*. O'reilly.

Sommerville, I. (2011). *Ingeniería de Software* (9th ed.). Pearson Educación. https://www-ebooks7-24-com.bdbiblioteca.universidadean.edu.co/?il=3313&pg=1