**Universidad de la Cuenca del Plata**

****

**Ingeniería en Sistemas de Información**

**Ingeniería en Software II**

**Profesor: Kutz, Rene Gabriel**

**Trabajo Práctico Nro 2**

**Integrantes**

Ø Agusti Lautaro

Ø Navarro Valentino

Contents

[**Introducción:**](#_2gq7vwy1n5tl) **5**

[**Patrón Singleton:**](#_6ceahzu54cc0) **5**

[Aplicabilidad](#_dl59tm562ik0) 5

[Pros:](#_t0s657tejjzq) 6

[Contras:](#_ofv1scaqelc8) 6

[**Patrón Factory (Fábrica):**](#_wqtvnmupanbh) **7**

[Implementación:](#_n6w6p2q6y12d) 7

[Aplicabilidad](#_2r67zbkwdszj) 8

[Pros](#_h6bj9g8l3zws) 8

[Contras:](#_qm26ly2rfto0) 9

[12.1](#_dcm5bqqoylii) 10

[Especificación de requerimientos dirigida por riesgos](#_r5k9gdgwuaae) 10

[**12.2**](#_cwgcafwvl3u2) **12**

[**Especificación de protección**](#_v0uh1tlpswpz) **12**

[**12.2.1**](#_11krves45lbr) **12**

[**Identificación de peligro**](#_abgzeh420b7m) **12**

[12.2.2](#_hz2j5catanav) 13

[Valoración del peligro](#_4fc9kd2t59g) 13

[12.2.3](#_obh9mcmrs71b) 15

[Análisis del peligro](#_6kiwkcxe5mj) 15

[12.2.4](#_g4jdit9ylj3o) 16

[Reducción del riesgo](#_vz37mzn8l71l) 16

[**12.3**](#_fwaq7ndixgq) **17**

[**Especificación de fiabilidad**](#_nzhtq1336ae9) **17**

[12.3.2](#_1cptkefej842) 18

[Requerimientos de fiabilidad no funcionales](#_wrxph0u89t3j) 18

[12.3.3](#_n9fqc9hyk1fb) 18

[Especificación de fiabilidad funcional](#_ljs1z7g4x2we) 18

[**12.4**](#_emmjiyg18mb5) **19**

[**Especificación de seguridad**](#_vku1jbuo2h3z) **19**

[**12.5**](#_d5s2232ypnuk) **20**

[**Especificación formal**](#_o67s63b3pt10) **20**

[**Conclusión:**](#_fwvzcl62rhaa) **21**

**Trabajo Práctico N° 2:** **Investigación y Presentación de Patrones de diseño de Software**

**04/05/2024 Fecha de entrega**

**Ejes Temáticos :** Nro 2 Diseño Orientado a Objetos (Patrones de Software)

Nro 3 Diseño Ul/interfaz Centrado en el Usuario

**Tema**: Patrones de Diseño Orientado de Objetos

**Objetivos:** investigar, analizar, resumir, ejemplificar y presentar dos patrones de diseño de software.

**Actividades:** Estableceremos en clases una lista de Patrones de Diseño de Software.

Los alumnos se dividirán en grupos de dos alumnos con dos patrones de diseño. Seleccionar el material bibliográfico de la cátedra, o presentados por los alumnos como referencia.

· En un segundo punto investigar sobre la Especificación de confiabilidad y seguridad en el Capítulo 11 del Libro Ingeniería el software (Ian Sommerville o PRESSMAN) novena Edición teniendo en cuenta los siguientes ítems:

12.1 Especificación de requerimientos dirigida por riesgos

12.2 Especificación de protección

12.3 Especificación de fiabilidad

12.4 Especificación de seguridad

12.5 Especificación formal

Leer comprensiva del material seleccionado. Elaborar la presentación y generar un informe escrito en formato presentación y ejemplo aplicación sobre los conceptos. Socialización y presentación de la producción realizada.

**Tipo de actividad:** Grupal (grupos de dos alumnos)

**Modalidad:** Investigación grupal, desarrollo de ejemplo y presentación oral. Entrega de los artefactos de la investigación en un repositorio de la materia en GitHub.

Tiempo: 2 Semanas

# Introducción:

El diseño de software es un aspecto fundamental en el desarrollo de sistemas que busca crear soluciones eficientes, mantenibles y escalables. Dos áreas importantes dentro del diseño de software son el Diseño Orientado a Objetos (DOO) y el Diseño de Interfaz de Usuario (UI/UX). En este trabajo, nos enfocaremos en ambos temas, comenzando con una exploración de los Patrones de Diseño Orientado a Objetos.

Los patrones de diseño son soluciones probadas y documentadas a problemas comunes que surgen durante el desarrollo de software. Estos patrones ofrecen un enfoque estructurado para resolver problemas de diseño y promueven la reutilización de código y la creación de sistemas más flexibles. Exploraremos dos patrones específicos, seleccionados en base a su relevancia y aplicabilidad en diversas situaciones de desarrollo de software.

# 

# Patrón Singleton:

En el contexto de la ingeniería de software, el patrón Singleton se puede considerar como una técnica para gestionar recursos compartidos o únicos. En un sistema de información, puede haber casos en los que sea necesario garantizar que solo exista una instancia de una determinada clase. Esto puede aplicarse, por ejemplo, a una clase que gestiona la conexión a una base de datos centralizada en un sistema distribuido. Garantizar que solo exista una instancia de esta clase Singleton puede ayudar a evitar problemas de concurrencia y a gestionar eficientemente los recursos del sistema.

Implementación: Supongamos que estamos diseñando un sistema de gestión de inventario para una cadena de tiendas minoristas. Podríamos implementar un Singleton para la clase InventoryManager, que sería responsable de gestionar el inventario de todas las tiendas. De esta manera, garantizamos que solo exista una instancia de InventoryManager en todo el sistema, lo que nos permite controlar de manera centralizada la gestión del inventario y evitar inconsistencias en los datos.

## Aplicabilidad

Utiliza el patrón Singleton cuando una clase de tu programa tan solo deba tener una instancia disponible para todos los clientes; por ejemplo, un único objeto de base de datos compartido por distintas partes del programa.

El patrón Singleton deshabilita el resto de las maneras de crear objetos de una clase, excepto el método especial de creación. Este método crea un nuevo objeto, o bien devuelve uno existente si ya ha sido creado.

Utiliza el patrón Singleton cuando necesites un control más estricto de las variables globales.

Al contrario que las variables globales, el patrón Singleton garantiza que haya una única instancia de una clase. A excepción de la propia clase Singleton, nada puede sustituir la instancia en caché.

Ten en cuenta que siempre podrás ajustar esta limitación y permitir la creación de cierto número de instancias Singleton. La única parte del código que requiere cambios es el cuerpo del método getInstance.

## Pros:

Puedes tener la certeza de que una clase tiene una única instancia.

Obtienes un punto de acceso global a dicha instancia.

El objeto Singleton solo se inicializa cuando se requiere por primera vez.

## Contras:

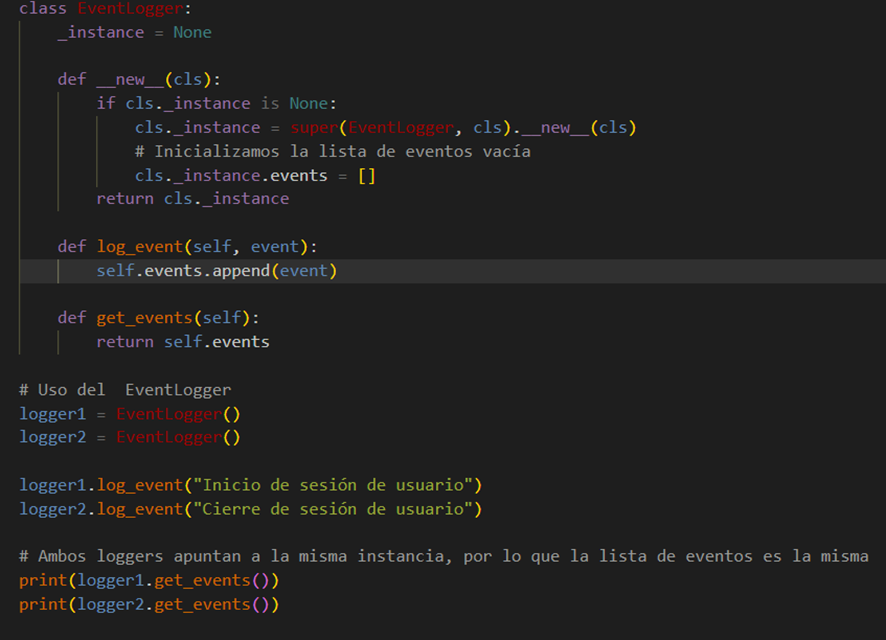
Vulnera el Principio de responsabilidad única. El patrón resuelve dos problemas al mismo tiempo.

El patrón Singleton puede enmascarar un mal diseño, por ejemplo, cuando los componentes del programa saben demasiado los unos sobre los otros.

El patrón requiere de un tratamiento especial en un entorno con múltiples hilos de ejecución, para que varios hilos no creen un objeto Singleton varias veces.

Puede resultar complicado realizar la prueba unitaria del código cliente del Singleton porque muchos frameworks de prueba dependen de la herencia a la hora de crear objetos simulados (mock objects). Debido a que la clase Singleton es privada y en la mayoría de los lenguajes resulta imposible sobrescribir métodos estáticos, tendrás que pensar en una manera original de simular el Singleton. O, simplemente, no escribas las pruebas. O no utilices el patrón Singleton.

Ejemplo de Singelton implementado a un registro de eventos:



# Patrón Factory (Fábrica):

En el contexto del desarrollo de sistemas de información, el patrón Factory se puede ver como una técnica para manejar la creación de objetos de manera flexible y desacoplada. En lugar de que una clase cree directamente instancias de otras clases, utiliza un método de fábrica para crear estas instancias. Esto permite cambiar fácilmente la implementación de los objetos creados sin modificar el código que los utiliza.

## Implementación:

Continuando con el ejemplo del sistema de gestión de inventario, podríamos tener diferentes tipos de productos en nuestras tiendas (por ejemplo, electrónica, ropa, alimentos, etc.). Podríamos implementar un Factory para la creación de objetos Product, donde cada subclase de Product representa un tipo diferente de producto (por ejemplo, ElectronicProduct, ClothingProduct, FoodProduct, etc.). De esta manera, podemos utilizar el Factory para crear instancias de productos según sea necesario en diferentes partes del sistema, sin tener que preocuparnos por los detalles de implementación de cada tipo de producto.

## Aplicabilidad

Utiliza el Método Fábrica cuando no conozcas de antemano las dependencias y los tipos exactos de los objetos con los que deba funcionar tu código.

El patrón Factory Method separa el código de construcción de producto del código que hace uso del producto. Por ello, es más fácil extender el código de construcción de producto de forma independiente al resto del código.

Por ejemplo, para añadir un nuevo tipo de producto a la aplicación, sólo tendrás que crear una nueva subclase creadora y sobrescribir el Factory Method que contiene.

Utiliza el Factory Method cuando quieras ofrecer a los usuarios de tu biblioteca o framework, una forma de extender sus componentes internos.

La herencia es probablemente la forma más sencilla de extender el comportamiento por defecto de una biblioteca o un framework. Pero, ¿cómo reconoce el framework si debe utilizar tu subclase en lugar de un componente estándar?

La solución es reducir el código que construye componentes en todo el framework a un único patrón Factory Method y permitir que cualquiera sobrescriba este método además de extender el propio componente.

Veamos cómo funcionaría. Imagina que escribes una aplicación utilizando un framework de UI de código abierto. Tu aplicación debe tener botones redondos, pero el framework sólo proporciona botones cuadrados. Extiendes la clase estándar Botón con una maravillosa subclase BotónRedondo, pero ahora tienes que decirle a la clase principal FrameworkUI que utilice la nueva subclase de botón en lugar de la clase por defecto.

Para conseguirlo, creamos una subclase UIConBotonesRedondos a partir de una clase base del framework y sobrescribimos su método crearBotón. Si bien este método devuelve objetos Botón en la clase base, haces que tu subclase devuelva objetos BotónRedondo. Ahora, utiliza la clase UIConBotonesRedondos en lugar de FrameworkUI.

## Pros

Evitas un acoplamiento fuerte entre el creador y los productos concretos.

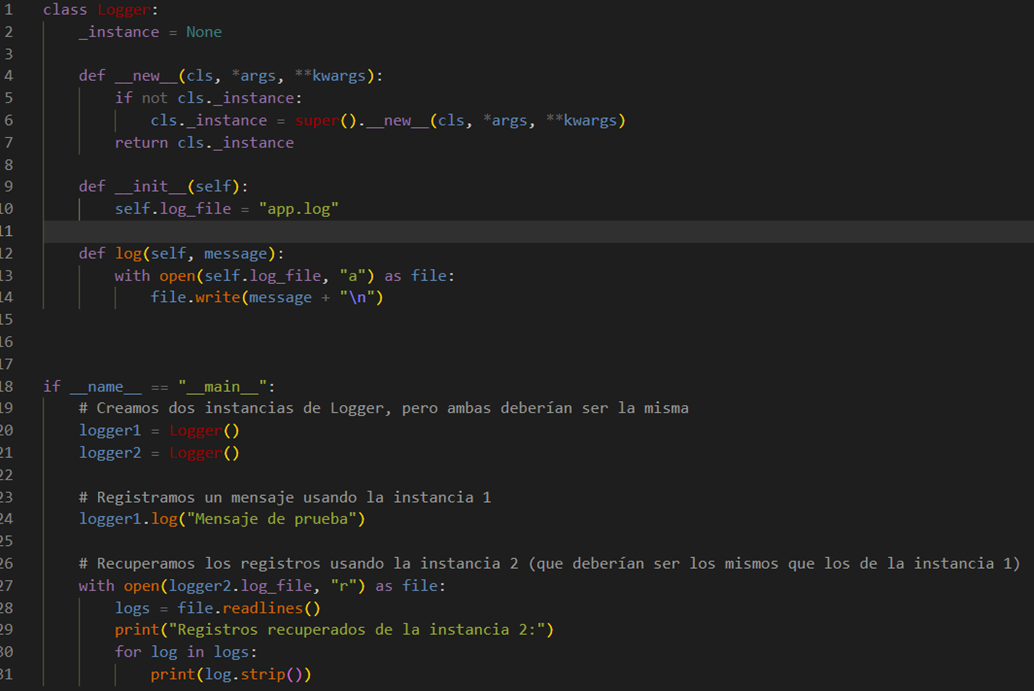
Principio de responsabilidad única. Puedes mover el código de creación de producto a un lugar del programa, haciendo que el código sea más fácil de mantener.

Principio de abierto/cerrado. Puedes incorporar nuevos tipos de productos en el programa sin descomponer el código cliente existente.

## Contras:

Puede ser que el código se complique, ya que debes incorporar una multitud de nuevas subclases para implementar el patrón. La situación ideal sería introducir el patrón en una jerarquía existente de clases creadoras.

Ejemplo de Factory Method aplicado a la automatización de un almacén:



**Actividad N 2**

## **12.1**

## **Especificación de requerimientos dirigida por riesgo**s

La especificación de requerimientos dirigida por riesgos es un enfoque crucial para garantizar la confiabilidad y seguridad de un sistema de software. Este enfoque se centra en comprender y abordar los riesgos potenciales que enfrenta el sistema y su entorno operativo. Aquí hay un resumen de los principales puntos relacionados con este enfoque:

Identificación del riesgo: Se identifican los riesgos potenciales al sistema, considerando eventos peligrosos que podrían ocurrir y la probabilidad de que sucedan.

Análisis y clasificación del riesgo: Cada riesgo se considera individualmente y se seleccionan aquellos que son potencialmente serios y no improbables para un análisis más detallado.

Descomposición del riesgo: Se analizan las posibles causas raíz de cada riesgo identificado, que pueden ser errores de software, hardware o vulnerabilidades inherentes al diseño del sistema.

Reducción del riesgo: Se proponen formas de reducir o eliminar los riesgos identificados, lo que contribuye a definir los requerimientos de confiabilidad del sistema y cómo se manejarán los riesgos.

Para sistemas grandes, el análisis de riesgo puede estructurarse en fases, que incluyen:

Análisis preliminar del riesgo: Identificación de los principales riesgos del entorno del sistema y desarrollo de un conjunto inicial de requerimientos de seguridad y confiabilidad.

Análisis de riesgo de ciclo de vida: Dirigido a los riesgos surgidos por decisiones de diseño del sistema durante su desarrollo.

Análisis de riesgo operativo: Preocupado por la interfaz de usuario del sistema y los riesgos de error del operador.

Es fundamental tener en cuenta que los requerimientos de seguridad y confiabilidad están influenciados por la tecnología utilizada y las decisiones de diseño, y pueden requerir modificaciones a lo largo del ciclo de vida del sistema. Además, es necesario considerar la interoperabilidad con componentes de terceros y las limitaciones de los sistemas comerciales existentes al definir estos requerimientos.

# **12.2**

# **Especificación de protección**

La especificación de protección en sistemas críticos se centra en identificar y abordar los riesgos que podrían causar lesiones o muerte a las personas en el entorno del sistema. Los requerimientos de protección son esencialmente requerimientos de seguridad y se centran en reducir la probabilidad de fallas del sistema que puedan tener consecuencias graves. Aquí hay un resumen de los puntos clave relacionados con la especificación de protección:

Identificación del riesgo: En este proceso, se identifican los peligros que podrían amenazar al sistema y a las personas en su entorno.

Análisis de riesgo: Se evalúan y priorizan los riesgos identificados para determinar cuáles son los más peligrosos o los que tienen mayor probabilidad de ocurrir. Esta priorización ayuda a derivar los requerimientos de protección más críticos.

Descomposición del riesgo: Se centra en identificar los eventos que podrían conducir a un peligro. En la especificación de protección, este proceso se conoce como análisis de peligro.

Reducción del riesgo: Basado en el análisis de peligro, este proceso lleva a la identificación de requerimientos de protección. Estos requerimientos pueden enfocarse en prevenir que ocurran los peligros o en minimizar el daño asociado si ocurren.

Es importante encontrar un equilibrio adecuado entre seguridad y funcionalidad al derivar los requerimientos de protección, evitando la sobreprotección que podría resultar en sistemas costosos pero ineficaces. La terminología utilizada en la especificación de protección distingue entre "peligro" (algo que podría resultar en lesiones o muerte) y "riesgo" (la probabilidad de que el sistema entre en un estado peligroso), lo que permite enfocarse en los eventos que conducen a los peligros y en cómo mitigarlos de manera efectiva.

# **12.2.1**

# **Identificación de peligro**

En los sistemas críticos de protección, los riesgos principales están vinculados a los peligros que podrían desencadenar un accidente. Para abordar este problema, es fundamental identificar los diferentes tipos de riesgos, como físicos, eléctricos, biológicos, de radiación, de falla de servicio, entre otros. Cada una de estas clases de riesgos puede ser analizada para descubrir peligros específicos que podrían surgir. Además, es importante identificar posibles combinaciones de riesgos que puedan ser potencialmente peligrosas.

Tomando como ejemplo el sistema de bomba de insulina, considerado un sistema crítico de protección, una falla en su funcionamiento podría provocar lesiones o incluso la muerte del usuario. Algunos de los peligros asociados con este sistema incluyen cálculos incorrectos de dosis de insulina, fallos en el sistema de monitorización del hardware, fallas en la energía debido a batería baja, interferencia eléctrica con otros equipos médicos, entre otros.

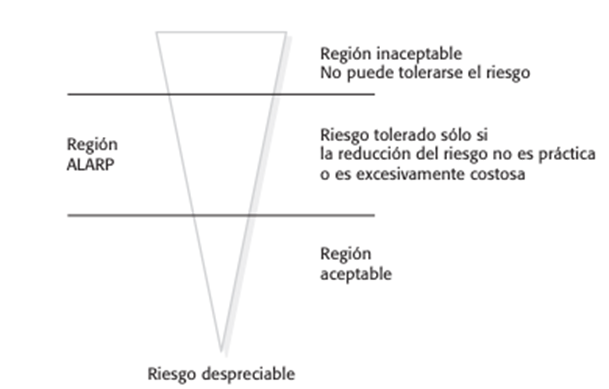
La identificación de estos peligros suele llevarse a cabo por ingenieros con experiencia, en colaboración con expertos del dominio y asesores de seguridad profesional. Se pueden utilizar técnicas como lluvias de ideas, donde un grupo de individuos intercambia ideas y experiencias. En el caso del sistema de bomba de insulina, los participantes podrían incluir médicos, físicos médicos, ingenieros y diseñadores de software.

Los peligros relacionados con el software suelen estar asociados con fallos en la entrega del servicio del sistema o con fallos en los sistemas de monitorización y protección. Estos últimos están diseñados para detectar condiciones críticas, como niveles bajos de batería, que podrían desencadenar una falla en el dispositivo.

## **12.2.2**

## **Valoración del peligro**

El proceso de valoración del peligro se enfoca en comprender la probabilidad de que sobrevenga un peligro y las consecuencias de que ocurriera un accidente o incidente asociado con dicho peligro. Es necesario hacer este análisis para comprender si un peligro



El análisis de riesgo proporciona una base para evaluar y decidir cómo manejar las amenazas identificadas al sistema o su entorno. Para cada peligro identificado, el análisis y la clasificación del riesgo conducen a un enunciado de aceptabilidad basado en la probabilidad de un accidente y sus consecuencias. Aquí hay un resumen de las tres categorías utilizadas en la valoración del riesgo:

Riesgos intolerables: Estos son aquellos que representan una amenaza para la vida humana y deben evitarse a toda costa. El sistema debe diseñarse de manera que estos peligros no puedan surgir, o que si lo hacen, sean detectados y mitigados antes de que ocurra un accidente grave. Por ejemplo, en el caso de una bomba de insulina, una sobredosis sería considerada un riesgo intolerable.

Riesgos "tan bajos como sea razonablemente práctico" (ALARP): Estos son riesgos que tienen consecuencias menos serias o que tienen una baja probabilidad de ocurrencia, pero que aún deben ser minimizados dentro de lo razonable. El diseño del sistema debe considerar factores como costo y entrega para reducir estos riesgos al nivel más bajo posible. Por ejemplo, la falla del sistema de monitorización del hardware en una bomba de insulina podría considerarse un riesgo ALARP, ya que podría resultar en una subdosis de insulina a corto plazo, pero no representaría un riesgo grave de accidente.

Riesgos aceptables: Estos son aquellos en los que los accidentes asociados generalmente causan daños menores. Los diseñadores del sistema deben tomar medidas para reducir estos riesgos siempre que sea posible, pero sin aumentar significativamente los costos, el tiempo de entrega u otros atributos no funcionales del sistema. Por ejemplo, una reacción alérgica en el usuario de la bomba de insulina podría considerarse un riesgo aceptable, ya que generalmente solo causa irritación menor en la piel, y no justificaría el uso de materiales más costosos en el dispositivo para reducir este riesgo.



La evaluación del riesgo implica estimar la probabilidad de un peligro y la gravedad de sus consecuencias. Esto puede ser complicado, ya que los peligros y los accidentes son eventos poco comunes, y los ingenieros pueden carecer de experiencia directa en incidentes previos. Por lo tanto, se utilizan términos relativos como "probable", "improbable", "raro", "alto", "medio" y "bajo" para asignar probabilidades y gravedad a los riesgos.

La figura 12.3 muestra una clasificación de riesgo para los peligros identificados en el sistema de entrega de insulina. Se separaron los peligros relacionados con el cálculo incorrecto de la insulina en sobredosis y subdosis de insulina, siendo la sobredosis potencialmente más grave que la subdosis a corto plazo. Por ejemplo, una sobredosis de insulina podría conducir a disfunción cognitiva, coma e incluso la muerte, mientras que una subdosis puede causar fatiga a corto plazo y problemas cardiacos, renales y oculares a largo plazo.

Aunque algunos peligros identificados en la figura 12.3 no están relacionados con el software, el software desempeña un papel importante en la detección de peligros. El software de monitoreo del hardware debe estar diseñado para advertir sobre posibles problemas, lo que permite detectar el peligro antes de que ocurra un accidente. Por ejemplo, la falla de energía se puede detectar monitoreando la batería, y la ubicación incorrecta de la máquina se puede detectar monitoreando las señales del sensor de azúcar en la sangre.

Es importante destacar que el software de monitoreo está relacionado con la protección del sistema. La falla en detectar un peligro podría conducir a un accidente. Si el sistema de monitoreo falla y no detecta una falla en el hardware, esto podría tener consecuencias más graves que si la falla del hardware se detectara adecuadamente.

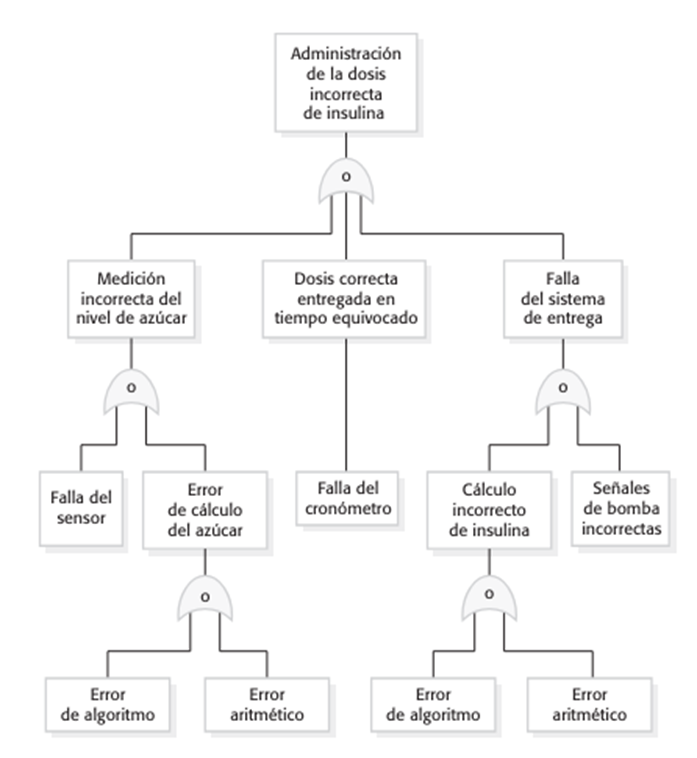
## **12.2.3**

## **Análisis del peligro**

El análisis del peligro es un proceso que busca identificar las causas raíz de los peligros en un sistema de protección crítico, con el objetivo de detectar qué eventos o combinaciones de eventos podrían provocar una falla del sistema que resulte en un peligro. Este análisis puede seguir un enfoque descendente, partiendo del peligro y retrocediendo hasta la posible falla del sistema, o un enfoque ascendente, comenzando con una falla de sistema propuesta e identificando los peligros resultantes.

Entre las técnicas utilizadas para el análisis del peligro se encuentran las revisiones y listas de verificación, así como técnicas más formales como el análisis de red de Petri, la lógica formal y el análisis de árbol de fallas. Este último es ampliamente utilizado y relativamente sencillo de entender sin requerir un conocimiento especializado del dominio.

En un análisis de árbol de fallas, se parte de los peligros identificados y se retrocede para identificar las posibles causas de cada peligro. Cada peligro se coloca en la raíz del árbol y se identifican los estados del sistema que podrían conducir a dicho peligro. Se continúa descomponiendo estos estados hasta llegar a las causas raíz del riesgo. Los peligros que solo pueden surgir de una combinación de causas raíz suelen ser menos probables de conducir a un accidente que aquellos con una sola causa raíz.



En el árbol de fallas mostrado en la figura 12.4, se observan tres condiciones que podrían conducir a la administración de una dosis incorrecta de insulina en el sistema de entrega de insulina:

Medición incorrecta del nivel de azúcar en la sangre: Esta condición puede ocurrir debido a la falla del sensor que proporciona la entrada para calcular el nivel de azúcar o a un error en el cálculo del nivel de azúcar en la sangre. Un cálculo incorrecto podría derivar de un algoritmo incorrecto o de errores aritméticos.

Administración de la dosis correcta en el momento equivocado: Esta condición se refiere a situaciones en las que la dosis correcta de insulina se calcula, pero se administra demasiado pronto o demasiado tarde debido a una falla en el cronómetro del sistema.

Falla del sistema de entrega: Esta condición podría surgir de un cálculo incorrecto del requerimiento de insulina o de una falla al enviar las señales correctas a la bomba que entrega la insulina. Al igual que en el caso anterior, un cálculo inexacto puede provocar una falla en el algoritmo o errores aritméticos.

Los árboles de fallas no solo se utilizan para identificar problemas potenciales de software, sino también de hardware. Estos árboles pueden proporcionar una comprensión de los requisitos para que el software detecte y, posiblemente, corrija dichos problemas. Por ejemplo, la capacidad del procesador está disponible para operar programas de diagnóstico y autoverificación, lo que puede ayudar a descubrir errores de hardware, como los del sensor, la bomba o el cronómetro, y emitir advertencias antes de que tengan un efecto grave en el paciente.

## **12.2.4**

## **Reducción del riesgo**

Una vez identificados los riesgos potenciales y sus causas raíz, se pueden derivar requerimientos de seguridad para administrar los riesgos y garantizar que no ocurran incidentes o accidentes. Hay tres posibles estrategias para abordar estos requerimientos:

Evitar el peligro: Diseñar el sistema de modo que el peligro no pueda ocurrir en absoluto.

Detectar y eliminar el peligro: Diseñar el sistema para detectar y neutralizar los peligros antes de que ocurra un accidente.

Limitar el daño: Diseñar el sistema para minimizar las consecuencias de un accidente si ocurre.

Normalmente, en los sistemas críticos de seguridad, se combinan estos enfoques. Por ejemplo, en un sistema de control de una planta química, se puede intentar detectar y evitar una presión excesiva en el reactor, pero también puede haber un sistema de protección independiente que monitorea la presión y abre una válvula de alivio si se detecta una presión alta.

En el caso específico del sistema de entrega de insulina, se pueden establecer requisitos de seguridad como los siguientes:

RS1: El sistema no debe entregar una dosis individual de insulina mayor que una dosis máxima especificada para un usuario.

RS2: El sistema no debe entregar una dosis diaria almacenada de insulina mayor que una dosis diaria máxima especificada para un usuario.

RS3: El sistema debe incluir una instalación de diagnóstico de hardware que se ejecute al menos cuatro veces por hora.

RS4: El sistema debe incluir un manejador de excepciones para todas las excepciones identificadas.

RS5: La alarma audible debe activarse cuando se detecte cualquier anomalía de hardware o software, y se debe mostrar un mensaje diagnóstico correspondiente.

RS6: En caso de una alarma, la entrega de insulina debe suspenderse hasta que el usuario restablezca el sistema y desactive las alarmas.

Para abordar las fallas del software que podrían conducir a una dosis incorrecta de insulina, se pueden considerar soluciones como la gestión de errores aritméticos y algorítmicos, incluyendo manipuladores de excepciones y mecanismos de advertencia.

Los requerimientos de seguridad se derivan de los riesgos identificados y sus posibles causas raíz, y se expresan detalladamente en la especificación de requerimientos del sistema. Estos requerimientos ayudan a garantizar la seguridad y confiabilidad del sistema de entrega de insulina.

# **12.3**

# **Especificación de fiabilidad**

la fiabilidad se puede especificar como la probabilidad de que ocurra una falla del sistema en un entorno operacional dado. Se pueden usar diferentes métricas dependiendo del sistema y los requisitos de la aplicación:

Probabilidad de falla a pedido (POFOD): Es la probabilidad de que una demanda de servicio del sistema resulte en una falla. Por ejemplo, una POFOD de 0.001 significa una probabilidad de 1/1,000 de falla por solicitud.

Tasa de ocurrencia de fallas (ROCOF): Indica el número probable de fallas del sistema en relación con el tiempo o el número de ejecuciones del sistema. Su recíproco es el tiempo medio para la falla (MTTF), que representa el promedio de tiempo entre fallas observadas.

Disponibilidad (AVAIL): Refleja la probabilidad de que el sistema esté operativo cuando se solicita un servicio. Por ejemplo, una disponibilidad del 99.99% significa que el sistema estará disponible el 99.99% del tiempo de operación.

La elección de la métrica depende de si las fallas son esporádicas o regulares, y de si se valora el tiempo absoluto entre fallas. Para evaluar la fiabilidad, se necesitan datos sobre el número de fallas del sistema, el tiempo entre fallas y el tiempo de recuperación después de una falla.

Las unidades de tiempo pueden variar según el tipo de sistema, como tiempo de calendario, tiempo de procesador o número de transacciones. Por ejemplo, los sistemas de conmutación telefónica pueden usar tiempo de procesador, mientras que los sistemas de monitorización continua pueden usar tiempo de calendario. Los sistemas de transacciones variables, como cajeros automáticos, pueden usar el número de transacciones como unidad de tiempo.

## **12.3.2**

## **Requerimientos de fiabilidad no funcionales**

Los requerimientos de fiabilidad no funcionales son cruciales para determinar la confiabilidad y disponibilidad necesarias de un sistema. Aunque históricamente se han utilizado principalmente en sistemas críticos de seguridad, su aplicación se está extendiendo a medida que más empresas demandan servicios continuos. Establecer estos requerimientos con precisión conlleva varias ventajas:

Clarificación de necesidades: Determinar el nivel de fiabilidad necesario ayuda a los participantes a comprender las distintas fallas del sistema y los costos asociados con niveles más altos de fiabilidad.

Base para pruebas: Establecer criterios claros permite detener las pruebas una vez que se alcanza el nivel deseado de fiabilidad.

Evaluación de estrategias de diseño: Permite comparar diferentes enfoques de diseño para mejorar la fiabilidad del sistema.

Cumplimiento regulatorio: Es crucial para sistemas críticos obtener la aprobación de los reguladores, lo que requiere evidencia de cumplimiento con los estándares de fiabilidad.

Sin embargo, existe el riesgo de sobreespecificar la fiabilidad, lo que puede resultar en altos costos de desarrollo y validación. Esto ocurre cuando es difícil traducir la experiencia práctica en métricas cuantitativas. Además, la sobreespecificación puede llevar a costos exorbitantes de pruebas y ser casi imposible de validar.

Para evitar la sobreespecificación, se sugieren varios pasos:

Especificar diferentes niveles de fiabilidad para diferentes tipos de fallas, priorizando aquellas que tienen consecuencias más graves.

Establecer requerimientos separados para la disponibilidad y la fiabilidad de diferentes servicios, priorizando los más críticos.

Evaluar si la fiabilidad cuantitativa es necesaria o si se pueden lograr metas de confiabilidad de otras formas, como mediante mecanismos de detección de errores.

## **12.3.3**

## **Especificación de fiabilidad funcional**

La especificación de fiabilidad funcional implica identificar requerimientos que definan restricciones y características contribuyentes a la fiabilidad del sistema. Estos pueden incluir:

Requerimientos de comprobación: Garantizan la detección de entradas incorrectas antes de su procesamiento.

Requerimientos de recuperación: Establecen procedimientos para la restauración del sistema después de una falla.

Requerimientos de redundancia: Especifican características redundantes para prevenir la pérdida total del servicio ante la falla de un componente.

Además, pueden existir requerimientos de proceso que aseguren prácticas que reduzcan el número de fallas durante el desarrollo del sistema.

No hay reglas fijas para derivar estos requerimientos, pero en organizaciones con experiencia en sistemas críticos, suele haber conocimiento acumulado sobre los requisitos de fiabilidad y su impacto real en el sistema. Este conocimiento puede ser reutilizado entre proyectos similares.

# **12.4**

# **Especificación de seguridad**

La especificación de requerimientos de seguridad para sistemas comparte similitudes con los requerimientos de protección, pero la seguridad implica un mayor esfuerzo debido a varias razones:

Naturaleza de los ataques: Mientras que la protección considera principalmente errores accidentales, la seguridad implica ataques deliberados por parte de personas que pueden conocer las debilidades del sistema.

Detección de fallas: En la protección, encontrar las causas de las fallas suele ser más simple, pero en seguridad, los ataques pueden ser encubiertos, lo que dificulta la identificación de la causa raíz.

Ataques de negación de servicio: A diferencia de la protección, donde es aceptable desactivar servicios para evitar fallas, en seguridad, los ataques pueden tener la intención de apagar completamente el sistema.

Adversarios inteligentes: Los ataques en seguridad provienen de adversarios que pueden aprender y adaptarse, lo que complica la defensa del sistema.

Los requerimientos de seguridad suelen ser más detallados que los de protección y pueden incluir aspectos como identificación, autenticación, autorización, inmunidad contra amenazas, integridad de datos, detección de intrusiones, no repudio, privacidad y auditoría de seguridad, entre otros.

El proceso de análisis y valoración del riesgo se utiliza para identificar estos requerimientos de seguridad, en etapas que incluyen el análisis preliminar del riesgo, el análisis del ciclo de vida y el análisis del riesgo operativo. Este proceso implica identificar activos, estimar su valor, evaluar la exposición al riesgo, identificar amenazas, valorar los posibles ataques, identificar controles, evaluar su factibilidad y definir los requerimientos de seguridad.

La política de seguridad de la organización es una importante entrada en este proceso, estableciendo las condiciones que deben mantenerse para garantizar la seguridad del sistema. Las políticas de seguridad pueden expresarse de manera formal y ser verificadas automáticamente para garantizar su cumplimiento.

Un ejemplo de este proceso aplicado a un sistema de información hospitalario para la atención a la salud mental podría incluir la identificación de activos, la estimación de su valor, la evaluación de exposición al riesgo, la identificación de amenazas, la valoración de posibles ataques, la identificación de controles y la definición de requerimientos de seguridad específicos para el sistema.

# **12.5**

# **Especificación formal**

Los métodos formales para el desarrollo de software han sido investigados durante más de 30 años y ofrecen una aproximación basada en matemáticas, donde se define un modelo formal del software. Esto permite analizar formalmente el modelo y utilizarlo como base para una especificación formal del sistema. Esta especificación formal describe sin ambigüedades qué debe hacer el sistema y facilita la detección temprana de problemas de requerimientos.

El desarrollo de una especificación formal generalmente se realiza como parte de un proceso de software basado en un plan, donde los requerimientos y el diseño se definen completamente antes de la implementación. Sin embargo, el uso de especificaciones formales puede limitarse a componentes críticos del sistema debido a su costo.

Aunque las especificaciones formales ofrecen ventajas como una comprensión profunda de los requerimientos, la reducción de costos de pruebas y la garantía de que un programa cumpla su especificación, hay varios argumentos en contra de su adopción generalizada:

Los propietarios y expertos de dominio pueden no entender las especificaciones formales, lo que dificulta la verificación de que representen con precisión los requerimientos del sistema.

Es difícil estimar el ahorro de costos resultante del uso de especificaciones formales, lo que hace que los administradores sean renuentes a adoptar este enfoque.

La falta de capacitación en el uso de lenguajes de especificación formal limita la aceptación de este enfoque entre los ingenieros de software.

Escalar los enfoques actuales a la especificación formal de sistemas muy grandes es complicado, lo que lleva a utilizar la especificación formal principalmente para el software crítico en lugar de sistemas completos.

La especificación formal no es compatible con los métodos de desarrollo ágiles, lo que puede dificultar su integración en entornos que utilizan estos métodos.

12.1 Especificación de requerimientos dirigida por riesgos:

Imagina que estamos automatizando un almacén que maneja productos farmacéuticos sensibles a la temperatura. Un riesgo importante es la posibilidad de que se interrumpa la cadena de frío durante el transporte o el almacenamiento, lo que podría comprometer la eficacia de los medicamentos. Para mitigar este riesgo, se especifica que todos los productos sensibles a la temperatura deben ser monitoreados continuamente durante su almacenamiento y transporte, y se establece un protocolo de respuesta rápida en caso de cualquier desviación de la temperatura especificada.

12.2 Especificación de protección:

Dado que el almacén maneja productos valiosos y sensibles, como dispositivos electrónicos, se establece una serie de medidas de protección. Esto incluye la instalación de sistemas de seguridad avanzados, como cámaras de vigilancia, sensores de movimiento y cerraduras biométricas en áreas restringidas. Además, se implementa un sistema de control de acceso que requiere autenticación de múltiples factores para garantizar que solo el personal autorizado pueda acceder a ciertas áreas del almacén.

12.3 Especificación de fiabilidad:

Para garantizar una operación fluida y confiable del almacén, se establece un objetivo de disponibilidad del sistema del 99.9%. Esto se logra mediante la implementación de redundancia en los sistemas críticos, como los sistemas de gestión de inventario y los sistemas de automatización de la cadena de suministro. Además, se establecen procedimientos de mantenimiento preventivo y se implementa un sistema de monitoreo continuo para detectar y abordar cualquier problema potencial antes de que afecte la operación del almacén.

12.4 Especificación de seguridad:

Considerando la importancia de proteger la integridad y la confidencialidad de los datos del almacén, se implementan medidas de seguridad robustas. Esto incluye la encriptación de datos sensibles almacenados en la base de datos del almacén, así como el establecimiento de políticas de seguridad de la información que regulen el acceso y el manejo de los datos por parte del personal. Además, se realizan auditorías de seguridad periódicas para identificar y mitigar cualquier vulnerabilidad en el sistema.

12.5 Especificación formal:

Para garantizar que el sistema de automatización del almacén cumpla con los requisitos operativos y de negocio, se desarrolla una especificación formal que describe detalladamente el comportamiento esperado del sistema. Esto incluye la definición de reglas de negocio, flujos de trabajo y protocolos de comunicación entre los diferentes sistemas y componentes del almacén. Esta especificación formal sirve como guía durante el desarrollo y la implementación del sistema, asegurando que se cumplan todas las funcionalidades requeridas de manera consistente y confiable.

# Conclusión:

En conclusión, el diseño de software es un proceso complejo que requiere una atención cuidadosa a múltiples aspectos, desde la arquitectura hasta la usabilidad y la seguridad. En este trabajo, hemos explorado dos áreas específicas dentro del diseño de software: los Patrones de Diseño Orientado a Objetos y la Especificación de Confiabilidad y Seguridad.

Los Patrones de Diseño Orientado a Objetos ofrecen soluciones probadas y documentadas a problemas comunes de diseño, promoviendo la reutilización de código y la creación de sistemas más flexibles y mantenibles. A través de la exploración de estos patrones, hemos visto cómo pueden aplicarse en una variedad de contextos para mejorar la estructura y la calidad del software.

Por otro lado, la Especificación de Confiabilidad y Seguridad es crucial para garantizar que el software cumpla con los estándares de calidad y seguridad necesarios para su correcto funcionamiento. Al centrarnos en los ítems de especificación dirigidos por riesgos, protección, fiabilidad, seguridad y formalidad, hemos destacado la importancia de abordar estos aspectos desde las primeras etapas del desarrollo del software.