

Ingeniería de software

Sistemas	6
Sistemas software complejos	9
Evolución del software	9
Características del software	9
El software se desarrolla, no se fabrica en sentido estricto.	9
El software no se estropea.	10
Curva de fallos del hardware	10
Curva ideal de fallos del software	10
Curva real de fallos del software	11
La mayoría del software se construye a medida	11
Aplicación del software	11
Problemas con el software	12
El proceso - Metodologías	13
Definición de Ingeniería de Software	13
Producto – Proyecto – Proceso	14
Ciclo de vida del proceso de construcción de software	15
Objetivos - Alcance – Límite	15
Metodologías	16
Cascada clásica (Waterfall):	16
Desarrollo en espiral (Spiral):	17
Modelo de desarrollo iterativo e incremental:	18
Metodologías ágiles:	19
Desarrollo orientado a prototipos:	19
Ingeniería de requerimientos	21
Importancia de los requerimientos	22
Impactos de los requerimientos	22
Impacto positivo y negativo de la práctica de requerimientos de software	22
Impacto de los requerimientos	23
Requerimientos	25
Orientados al mercado - Específicos para un cliente	25
Requerimientos verdaderos y falsos	25
Tipos de Requerimientos según las capacidades o características	26
Requerimientos funcionales y no funcionales	27
Métricas para especificar las propiedades no funcionales del Sistema	29
Rol de los requerimientos	29
Análisis de requerimientos	29
Proceso de ingeniería de requerimientos	30
Adquisición - representación - evaluación	30
Especificación de requerimientos de software (SRS)	30
Beneficios de la SRS	30
Características de una buena SRS	30
Cada requerimiento debería tener estas características :	30

Índice del Standard IEEE 830	32
Proceso de ingeniería de requerimientos	36
Procesos - Elicitación -Especificación - validación	36
Gestión de requerimientos	38
Requerimientos estables y volátiles	38
Gestión del cambio	39
Rastreabilidad	39
Técnicas de elicitation	40
Entrevistas	40
Cuestionarios	40
Cuestionarios Escalas	40
Surveys	41
Brainstorming (Tormenta de ideas)	42
Focus Group	42
Rapid Application Development /Joint Applications Development	43
Escenarios	43
Análisis de formularios	43
Lenguaje natural	44
Prototipos	44
StoryBoarding	44
Proceso de análisis	45
Casos de uso	46
¿Que son los casos de Uso ?	46
Diferencias entre Casos de uso y eventos	46
Actores	46
Diferencias entre usuarios y actores	47
Descripción de los casos de uso	48
Relaciones entre casos de uso	48
Extensión	48
Uso	49
Actores y Casos de Uso Abstractos	50
Diferencia entre casos de uso de trazo grueso o esenciales vs de implementacion o de trazo fino	52
Casos de uso temporales	52
Modelo orientado a objetos	53
Deficiencias del análisis estructurado	53
Ventajas	53
Objetos de interface	54
Objeto entidades	56
Objetos de control	57
Preguntas Primer Parcial	59
Gestión de riesgos	66
Procesos de la Gestión de Riesgos	66
Estrategias frente al riesgo	67

Dificultades accidentales y dificultades esenciales	71
Diseño	72
Qué es diseño definición:	72
Objetivos del diseño	74
Criterios de Diseño	75
El acoplamiento	75
La cohesión	80
Niveles de cohesión	81
Factorización	86
Complejidad y accesibilidad	90
Relación entre accesibilidad y complejidad	92
Diagramas de secuencia	92
Ejemplo de diagramas de secuencia	93
Calidad del software	96
Calidad del software definiciones	98
Factores que determinan la calidad del software	98
Estándares y modelos de evaluación y mejora de los procesos software	98
ISO 9000	99
ISO 9001	100
CMM (Capability Maturity Model)	101
(SGC) Sistema de gestión de calidad	101
Ciclo de deming	102
Testing	103
1. Verificación:	103
2. Validación:	103
Plan de Pruebas	104
V-Model	105
Pruebas Unitarias	105
Pruebas Integrales	106
Pruebas de Homologación o de Sistemas o de validación	106
Pruebas de Regresión	106
Pruebas de Aceptación (UAT, por sus siglas en inglés)	106
Caja blanca (sistemas)	107
Caja negra (sistemas)	108
Pruebas alfa y beta	108
Pruebas escala total	108
Pruebas de performance	108
Pruebas de mutación	109
Pruebas de Stress	109
Pruebas de Penetración (y análisis de seguridad)	109
Plan de pruebas	110
Casos de prueba	110
Proceso de Implementación	111
Preguntas segundo parcial	117

Sistemas

Un sistema es un conjunto organizado de cosas o partes que se relacionan entre sí formando un todo unitario y complejo

1. Qué motivó a Ludwig von Bertalanffy postular la Teoría General de Sistemas?

Ludwig von Bertalanffy se sintió motivado a postular su teoría porque en su época, la tendencia predominante era analizar los fenómenos dividiéndolos en partes y tratándose de manera aislada, sin considerar cómo esas partes interactúan y se relacionan entre sí como un sistema integrado. En otras palabras, la visión reduccionista predominante no estaba capturando la complejidad y la interconexión inherente de los sistemas naturales y sociales.

2. Qué características tienen los Sistemas?

Los sistemas se caracterizan por tener :

- Entrada: Son los recursos, ya sean materiales, humanos o de información, que ingresan al sistema para su procesamiento.
- Salida: Son los resultados que se obtienen como producto del procesamiento de las entradas
- Proceso: Es la actividad mediante la cual las entradas son transformadas en salidas dentro del sistema.
- Caja Negra: Es una representación de los sistemas cuando no se conocen sus componentes internos o procesos.
- Relaciones: Son los enlaces que conectan a los objetos o subsistemas dentro de un sistema complejo. Se clasifican en simbióticas, sinérgicas y superfluas.
- Atributos: Definen al sistema tal como se observa o se conoce.
- Contexto: El sistema está siempre relacionado con su entorno
- Rango: Se refiere a la jerarquía de un sistema respecto a otros sistemas o subsistemas
- Variables: Son los elementos que forman parte del proceso interno de un sistema y que experimentan diferentes comportamientos según las circunstancias y el momento en el que se encuentren.
- Retroalimentación: Se refiere a cómo el sistema utiliza la información o los recursos generados por su propia salida .
- Permeabilidad: Describe la medida en que el sistema interactúa con su entorno. Un sistema más permeable está más abierto a influencias externas.
- Integración e independencia: Un sistema es independiente cuando los cambios dentro de él no afectan a otros sistemas. La integración se refiere a cómo diferentes partes del sistema trabajan juntas de manera coordinada.
- Adaptabilidad: Es la capacidad del sistema para ajustarse y modificar su funcionamiento en respuesta a cambios en su entorno.
- Mantenibilidad: Se refiere a la capacidad del sistema para mantenerse en funcionamiento de manera continua y equilibrada.
- Estabilidad: Indica la capacidad del sistema para mantenerse en equilibrio a través del flujo constante de materiales, energía e información.
- Armonía: Describe el nivel de compatibilidad del sistema con su entorno o contexto.
- Optimización: Se refiere a cómo el sistema se ajusta y mejora para alcanzar sus objetivos de manera más eficiente.

- Éxito: Es la medida en que el sistema logra cumplir sus objetivos de manera efectiva y satisfactoria.
- Parámetro: Cuando una variable se comporta como un parámetro, significa que permanece constante o inalterada ante circunstancias específicas
- Operadores: Son variables que tienen el poder de activar o influir significativamente en el proceso para que este se inicie o desarrolle

3. Qué relación encuentra entre la Entropía, la Información y los Sistemas Abiertos?

Estos conceptos están relacionados, ya que los **sistemas abiertos** pueden procesar **información** y mantener su equilibrio interno gracias al intercambio constante de energía y materia con su entorno. Este intercambio, a su vez, tiene un efecto en la **entropía** del sistema, ya que puede influir en el nivel de orden o desorden dentro de él .

4. Ventajas y desventajas de un sistema cerrado

Ventajas	Desventajas
Al no tener que adaptarse continuamente a cambios externos, puede ser más eficiente en el uso de recursos y en la ejecución de procesos.	Falta de adaptabilidad . Al no recibir retroalimentación del entorno, el sistema puede volverse obsoleto
Al no interactuar con el entorno, es más fácil controlar y predecir su comportamiento.	Incapacidad para incorporar nuevas ideas, tecnologías o información del entorno

5. Explique Sinergia y caracteriza con un ejemplo concreto

La sinergia es el efecto que se produce cuando varios elementos , se combinan para crear un resultado mas grande. Un ejemplo de esto podría ser un equipo de trabajo ya que cada miembro del grupo tiene habilidades y conocimientos únicos . Cuando estos miembros trabajan juntos aprovechando el intelecto de cada uno pueden llegar a un objetivo en común,

6. Dada la Jerarquía de los sistemas de Kenneth Boulding, proponga un ejemplo para cada caso.

- **Primer nivel :** Esta es la geografía y la anatomía del universo . Ejemplo : Una montaña
- **Segundo nivel:** sistema dinámico simple. Considera movimientos necesarios y predeterminados. Se puede denominar reloj de trabajo. Ejemplo : un reloj , un péndulo
- **Tercer nivel :** mecanismo de control o sistema cibernetico. El sistema se autorregula para mantener su equilibrio . Por ejemplo un termostato
- **Cuarto nivel:** "sistema abierto". En este nivel se comienza a diferenciar la vida. Por ejemplo : una célula , un ecosistema , un árbol.

- **Quinto nivel:** genético-social. Está caracterizado por las plantas ya que estas están genéticamente programadas para interactuar con su entorno y otros seres vivos .
- **Sexto nivel :** sistema animal. Por ejemplo un perro , un gato etc.
- **Séptimo nivel :** sistema humano. Es el nivel del ser individual, considerado como un sistema con conciencia y habilidad para utilizar el lenguaje y símbolos .
- **Octavo nivel:** sistema social o sistema de organizaciones humanas. Por ejemplo una empresa o una sociedad .
- **Noveno nivel,** sistemas trascendentales. Por ejemplo : la religión el arte etc

7. Relacione la definición de SISTEMA con los Sistemas Informáticos

Un sistema es un conjunto de elementos relacionados entre sí que funcionan como un todo. Entonces la relación es que un sistema informático es un conjunto de partes o recursos formados por el hardware , software y las personas que lo emplean , que se relacionan entre sí para almacenar y procesar información con un objetivo en común.

Sistemas software complejos

Evolución del software

El hardware era más importante que el software, ya que era costoso y limitaba el rendimiento. El desarrollo de software carecía de metodología y era realizado de forma informal. Las empresas se enfocaban en mejorar el hardware, lo que llevó a la complejidad de los sistemas informáticos y la necesidad de software más sofisticado. Sin embargo, la falta de metodología en el desarrollo de software resultó en programas con errores y problemas de mantenimiento costosos. Esto condujo a la llamada "**crisis del software**". Hoy en día, el software es la principal consideración en términos de costos y desafíos en el desarrollo de sistemas informáticos, lo que ha llevado a la adopción de la **Ingeniería de Software** para abordar estos problemas.

Características del software

Software:

- (1) instrucciones de ordenador que cuando se ejecutan proporcionan la función y el comportamiento deseado,
- (2) estructuras de datos que facilitan a los programas manipular adecuadamente la información,
- (3) documentos que describen la operación y el uso de los programas

El software incluye no sólo los programas de ordenador, sino también las estructuras de datos que manejan esos programas y toda la documentación que debe acompañar al proceso de desarrollo, mantenimiento y uso de dichos programas

El software se desarrolla, no se fabrica en sentido estricto.

El software presenta desafíos únicos que pueden afectar su calidad y costo . Para reducir el costo del producto , es importante enfocarse en la ingeniería y en las etapas de desarrollo , en lugar de centrarse únicamente en la producción

El software no se estropea.

Curva de fallos del hardware

Podemos comprar las curvas de índices del hardware y el software en función del tiempo. En el caso del hardware , se tiene la llamada “curva de bañera” , que indica que el software presenta relativamente muchos fallos al principio de su vida . Estos fallos pueden ser defectos de diseño o a la baja calidad inicial de la fase de producción . Una vez corregidos estos defectos , la tasa de fallos cae hasta un nivel estacionario y se mantiene así durante un cierto periodo de tiempo . Luego la tasa de fallos vuelve a incrementarse debido al deterioro de los componentes , que van siendo afectados por la suciedad , vibraciones y la influencia de muchos otros factores externos . Llegado a este punto podemos sustituir los componentes o todo el sistema y la tasa de fallos vuelve a situación en el nivel estacionario .

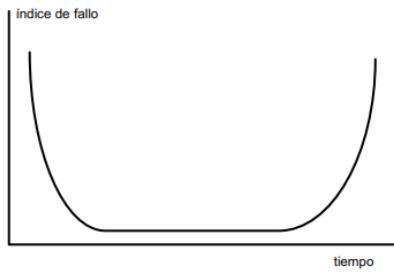


Figura. 1.1. Curva de fallos del hardware

Curva ideal de fallos del software

El software no es susceptible a los factores externos que hace el hardware se estropee. Inicialmente la tasa de fallos es alta debido a la presencia de errores no detectados durante el desarrollo, los denominados errores latentes. Una vez corregidos estos errores , la tasa de fallos debería alcanzar el nivel estacionario y mantenerse ahí indefinidamente.

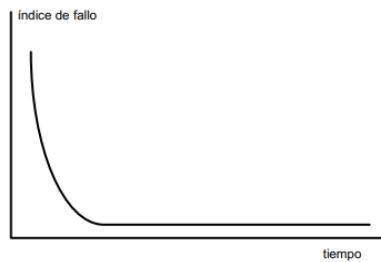


Figura 1.2. Curva ideal de fallos del software

Curva real de fallos del software

Esto no es mas que la simplificación del modelo real de fallos de un proceso de software .Durante su vida el software sufre cambios debido al mantenimiento. El mantenimiento puede deberse a la corrección de errores latentes o a cambios en los requisitos iniciales del producto. Estos errores pueden corregirse , pero el efecto de los sucesivos cambios hace que el producto se aleje cada vez mas de las especificaciones iniciales de acuerdo a las cuales fue desarrollado

Por estas razones ,el nivel estacionario que se consigue después de un cambio es algo superior al que había antes de efectuarlo , degradándose poco a poco al funcionamiento del sistema . Podemos decir entonces que el software no se estropea , pero se deteriora

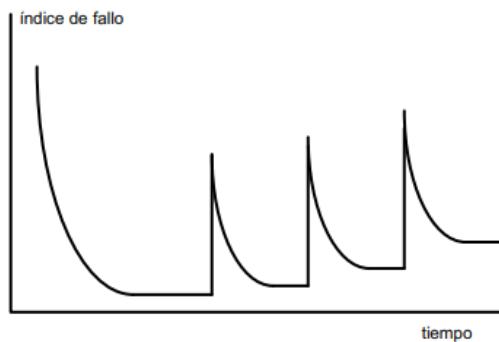


Figura 1.3. Curva real de fallos del software

La mayoría del software se construye a medida

El diseño de hardware se basa en componentes digitales probados y catalogados, mientras que el software se desarrolla principalmente a medida, con baja reutilización y altos costos de ingeniería. Aunque las bibliotecas de funciones y la programación estructurada han intentado fomentar la reutilización, la falta de modularidad y documentación limita su efectividad. Las técnicas orientadas a objetos ofrecen una nueva esperanza al permitir la programación por especialización, encapsulando datos y procedimientos en objetos con interfaces claras y la capacidad de heredar y reescribir partes, lo que facilita la reutilización incluso en nuevos contextos.

Aplicación del software

El software puede aplicarse a una variedad de situaciones del mundo real. Se utilizan lenguajes procedimentales para implementar algoritmos establecidos, lenguajes declarativos para problemas que pueden describirse formalmente, sistemas expertos para problemas con reglas heurísticas contradictorias, y redes neuronales para problemas con soluciones conocidas pero métodos de resolución no claros. Sin embargo, categorizar las aplicaciones de software se vuelve difícil con la creciente complejidad, aunque tradicionalmente se clasifican en lenguajes procedimentales, declarativos, sistemas expertos y redes neuronales.

- **Software de sistemas** :Son programas que sirven al desarrollo o funcionamiento de otros programas, como editores, compiladores, sistemas operativos, etc. Se caracterizan por estar cerca del hardware, ser utilizados por muchos usuarios y tener una amplia difusión. Deben ser fiables y cumplir estrictamente las especificaciones
- **Software en tiempo real**: Diseñado para medir, analizar y controlar eventos del mundo real en tiempo real, con requisitos temporales estrictos. Deben ser fiables y tolerantes a fallos, con poca interacción con el usuario.
- **Software de gestión**: Se utiliza para procesar grandes cantidades de información almacenada en bases de datos para facilitar transacciones comerciales o toma de decisiones. Puede incluir programas interactivos para soportar transacciones comerciales.
- **Software científico y de ingeniería**: Realiza cálculos complejos sobre datos numéricos, enfocándose en la corrección y exactitud de las operaciones. Incluye sistemas de diseño, ingeniería y fabricación asistida por ordenador, así como simuladores gráficos.
- **Software de ordenadores personales**: Aplicaciones como procesadores de texto, hojas de cálculo, bases de datos, juegos, etc., orientadas a usuarios no profesionales con requisitos de facilidad de uso y bajo costo.
- **Software empotrado**: Instalado en productos industriales para dotarlos de inteligencia, desde electrónica de consumo hasta sistemas de control en automóviles. Puede realizar desde cálculos en tiempo real hasta interacciones simples con el usuario.
- **Software de inteligencia artificial**: Se centra en problemas que requieren funciones intelectuales más elevadas, utilizando lenguajes declarativos, sistemas expertos y redes neuronales para abordarlos de manera eficiente.

Problemas con el software

Crisis del software es un estado pasajero de inestabilidad , que tiene como resultado un cambio de estado del sistema o una vuelta al estado inicial , en caso de que se tomen las medidas para superarla . Teniendo en cuenta eso, el software , más que padecer una crisis padece una enfermedad crónica

Estos problemas son causados por las propias características del software y por los errores cometidos por quienes intervienen en su producción. Entre ellos podemos citar:.

- **La planificación y la estimación de costos son muy imprecisas**: Los proyectos de software tienden a durar más de lo esperado debido a especificaciones ambiguas o incorrectas y falta de comunicación con el cliente. Cambios de última hora sin análisis de impacto aumentan la complejidad y los costos.
- **La calidad es mala** : Es común que el software entregado contenga errores debido a especificaciones ambiguas y falta de pruebas exhaustivas. Esto resulta en altos costos de mantenimiento y la aparición de conceptos como fiabilidad y garantía de calidad solo recientemente.
- **El cliente queda insatisfecho** : La falta de análisis de requisitos, comunicación durante el desarrollo y presencia de errores en el producto final deja a menudo a los clientes insatisfechos. Esto puede llevar a la necesidad de rediseñar o desarrollar nuevamente las aplicaciones, o cambiar de proveedor con frecuencia.
- **La productividad es baja** : Los proyectos de software suelen tener una duración más larga de lo esperado, lo que resulta en costos elevados y una disminución en la productividad y los beneficios. Esto se debe a la falta de propósitos claros al inicio del proyecto y a especificaciones ambiguas o incorrectas. La falta de comunicación con el cliente conduce a modificaciones de especificaciones de último minuto, sin un análisis detallado de su impacto. La falta de documentación y las modificaciones no documentadas hacen que el mantenimiento del software sea difícil, a menudo llevando más tiempo modificar el software existente que desarrollarlo desde cero.

El proceso - Metodologías

Definición de Ingeniería de Software

La Ingeniería de Software es una disciplina que se encarga de abordar los desafíos en el desarrollo de sistemas de software complejos. No hay una solución mágica para estos problemas, pero la combinación de métodos, herramientas y procedimientos puede ayudar a gestionar el proceso de desarrollo y mejorar la calidad del producto final.

Métodos: Se refieren a cómo construir técnicamente el software, abarcando tareas como la planificación, análisis de requisitos, diseño, codificación, pruebas y mantenimiento. Introducen notaciones específicas y criterios de calidad.

Herramientas: Proporcionan soporte automático o semiautomático para utilizar los métodos. Existen herramientas para cada fase del desarrollo y sistemas integrados conocidos como CASE (Computer Assisted Software Engineering).

Procedimientos: Definen la secuencia en que se aplican los métodos, los documentos necesarios, los controles de calidad y las directrices para evaluar el progreso del proyecto.

En conclusión, la Ingeniería de Software busca establecer y aplicar principios de ingeniería robustos para desarrollar software económico, fiable y eficiente. A través de métodos, herramientas y procedimientos, se busca mejorar la gestión del desarrollo de software y la calidad del producto final.

Otra definición :

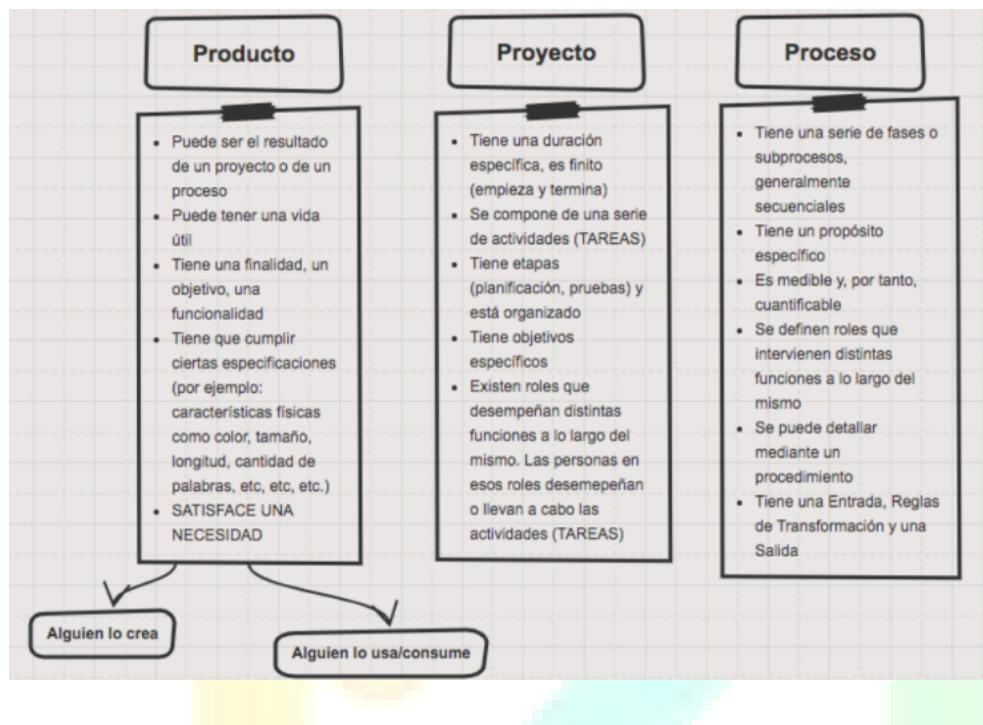
Refiere al campo de estudio y práctica que se enfoca en el desarrollo de software de alta calidad. Se basa en principios de ingeniería para analizar, diseñar, construir, probar y mantener software de manera sistemática y eficiente.

La Ingeniería de Software implica la aplicación de métodos, técnicas y herramientas específicas para gestionar el ciclo de vida completo del software, desde la concepción y especificación de requisitos, pasando por el diseño, la implementación y las pruebas, hasta el despliegue y mantenimiento del software en producción.

El objetivo principal de la Ingeniería de Software es desarrollar software confiable, eficiente, seguro y escalable, que cumpla con los requisitos y expectativas de los usuarios finales.

La Ingeniería de Software considera aspectos como la gestión de proyectos de desarrollo de software, el control de calidad, la reutilización de componentes y la gestión del cambio.

Producto – Proyecto – Proceso



Producto

- Puede ser el resultado de un proyecto o de un proceso
- Puede tener una vida útil
- Tiene una finalidad , un objetivo , una funcionalidad
- Tiene que cumplir ciertas especificaciones (Ejemplo : color , tamaño etc)
- SATISFACE UNA NECESIDAD.

Proyecto :

- Tiene una duración específica es finito (empieza y termina)
- Se compone de una serie de actividades(tareas)
- Tiene etapas (planificación , pruebas) y esta organizado
- Tiene objetivos específicos
- Existen roles que desempeñan distintas funciones a lo largo del mismo . Las personas en esos roles desempeñan o llevan a cabo las actividades (tareas)

Procesos :

- Tiene una serie de fases o subprocessos generalmente secuenciales .
- Tiene un proceso específico
- Es medible y por lo tanto cuantificable
- Se definen roles que intervienen distintas funciones a lo largo del mismo
- Se puede detallar mediante un procedimiento
- Tiene entrada ,reglas de transformación y una salida

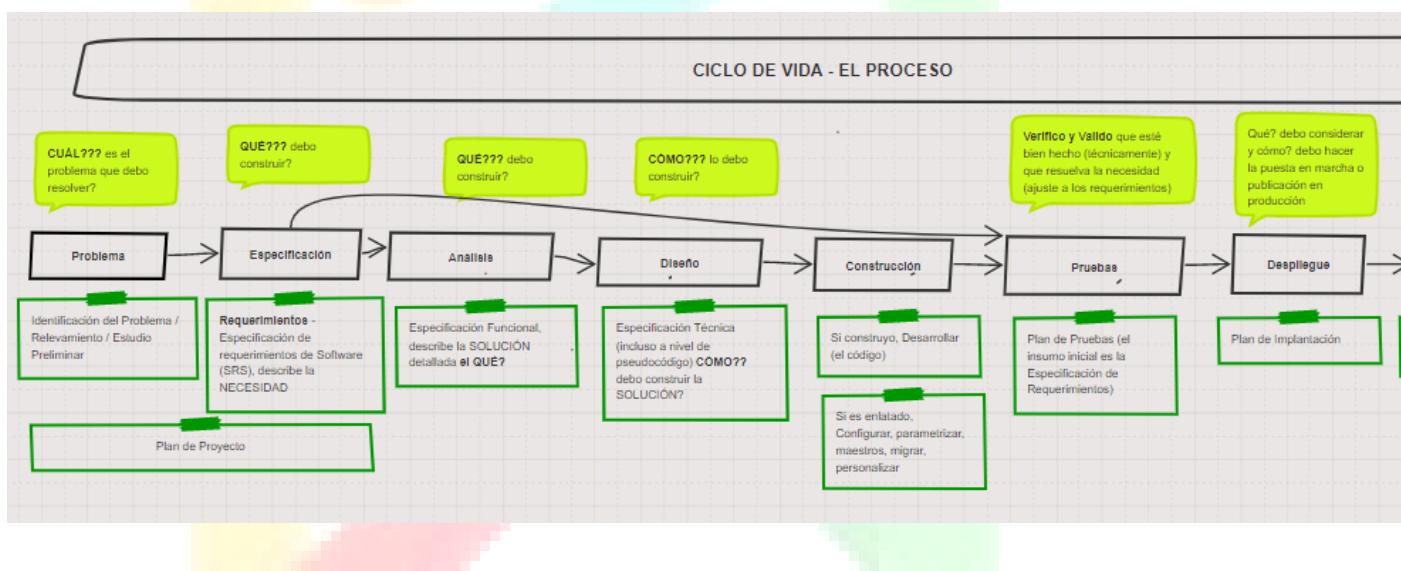
Producto: Un producto es el resultado tangible o intangible de un proceso o proyecto. Es el resultado final que se entrega o se pone a disposición de los usuarios o clientes. Puede ser un bien, como un dispositivo electrónico o un servicio, como la creación de un sitio web. El producto está diseñado para satisfacer las necesidades y expectativas de los usuarios finales.

Proyecto: Un proyecto es una iniciativa temporal que se realiza para alcanzar un objetivo específico. Tiene un inicio y un fin definidos, y se lleva a cabo para crear un producto o un resultado único. Los

proyectos suelen tener restricciones de tiempo, presupuesto y recursos. En el contexto de la ingeniería de software, un proyecto puede referirse al desarrollo de un nuevo software, una actualización importante de un sistema existente o la implementación de una solución tecnológica específica. Los proyectos, además, se gestionan utilizando metodologías de gestión de proyectos.

Proceso: Un proceso es una secuencia de actividades o pasos que se llevan a cabo para lograr un objetivo específico. Dada una entrada se aplican una serie de transformaciones para producir una salida. En el contexto de la ingeniería de software, un proceso refiere a las actividades que deben ser llevadas a cabo, metodológicamente (modelo en cascada, el modelo ágil o los enfoques evolutivos o incrementales o prototípicos) para desarrollar o construir una solución software. Esas actividades abarcan desde la concepción y especificación de requisitos, pasando por el diseño, la implementación y las pruebas, hasta el despliegue y mantenimiento del software en producción.

Ciclo de vida del proceso de construcción de software



Objetivos - Alcance – Límite

Alcance :

- Del proyecto : Define qué actividades y resultados están incluidos y excluidos dentro del proyecto . El alcance del proyecto se establece durante la etapa de planificación y ayuda a definir las metas y los límites claros del proyecto
- De la solución (que hace el programa).

Objetivo :

- Del proyecto : Puede ser la creación de un nuevo producto , la mejora de un proceso existente , la implementación de un sistema etc . Este debe ser claro y alcanzable
- De la solución : Son los resultados específicos que se pretenden lograr con la implementación de la

solución

Límite :

- Del proyecto : Este se refiere a las fronteras o restricciones que delimitan el alcance general del proyecto
- De la solución :Estos límites ayudan a definir las características y funcionalidades de la solución y establecen lo que la solución puede y no puede hacer

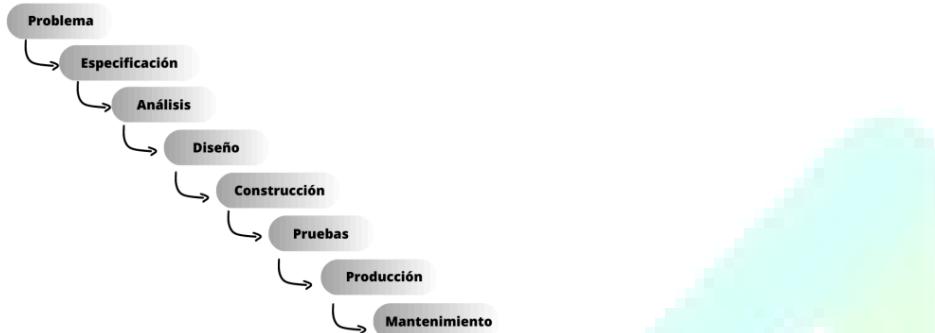
Metodologías

Una metodología es un conjunto de principios, prácticas, técnicas y herramientas estructuradas y sistematizadas que se utilizan para abordar de manera ordenada y eficiente la realización de un proyecto, la resolución de un problema o la ejecución de una tarea específica. Proporciona un marco de trabajo que guía el proceso y las actividades necesarias para lograr los objetivos establecidos.

Cascada clásica (Waterfall):

Es una metodología secuencial y lineal en la que se avanza a la siguiente fase del ciclo de vida del software sólo cuando se ha culminado la anterior. Comenzando por los requisitos, el análisis, el diseño, la implementación, las pruebas y finalmente la entrega. Como cada fase debe completarse antes de pasar a la siguiente, se impone la dificultad para realizar cambios una vez que se ha avanzado a una etapa posterior. En la actualidad es difícil encontrar soluciones que puedan construirse a “requerimiento cerrado”; es decir, que no habrá cambios en las necesidades mientras se construye la solución.

- No podes empezar otra etapa sin haber terminado la anterior
- El producto final va estar al final de la prueba
- No tiene gestión de riesgos
- Enfoque línea secuencial y poco flexible
- Etapas bien definidas pruebas y revisión al final de cada una
- Esta metodología es rígida a cambios en los requerimientos el cliente no tiene contacto ni puede probar el software hasta que el software se termina



Ventajas y desventajas : La metodología de cascada posee una estructura sencilla, fácil de comprender, lo que facilita el desarrollo en equipo, además, las fases están bien definidas, por lo que hay oportunidades de realizar pruebas y revisiones en cada etapa. Sin embargo, esta metodología de desarrollo es débil frente a 3 circunstancias cambiantes o variadas, al tener un enfoque tan rígido de sensible a cambios en los requerimientos o errores tardíos. Esto también puede deberse a que el cliente no puede ver el producto final hasta que esté completamente desarrollado, lo que aumenta el riesgo de que no cumpla las expectativas.

Desarrollo en espiral (Spiral):

Esta metodología combina elementos del enfoque en cascada con la gestión de riesgos. Se divide el desarrollo en ciclos que incluyen actividades de planificación, análisis de riesgos, desarrollo y evaluación. Cada ciclo permite la retroalimentación y la adaptación en función de los riesgos identificados. Además de introducir en la metodología la gestión de riesgos, su característica cíclica permite re-adaptarse o ajustarse a eventuales cambios durante la construcción de la solución.

El modelo en espiral fue ideado por el ingeniero informático estadounidense Barry Boehm. Este modelo es de tipo evolutivo del proceso de software y acopla la naturaleza iterativa de hacer prototipos con los aspectos controlados y sistemáticos del modelo de cascada, generando el desarrollo de versiones más completas con mayor rapidez. Las características principales de este modelo son:

- Enfoque cíclico, que asegura un mayor grado de definición e implementación con un riesgo menor
- Conjunto de puntos de referencia de anclaje puntual, que aseguran el compromiso de los participantes con soluciones factibles y satisfactorias para todos



El primer circuito alrededor del eje espiral nos da como resultado el desarrollo de una sola especificación del producto. Cada paso por planeamiento es para desarrollar prototipos y luego versiones más sofisticadas. La cantidad de vueltas que se dan puede ser determinada previamente por el gerente de proyecto o también se puede extender a lo largo de toda la vida del software, hasta su retiro.

El costo y la programación de las actividades del producto, se ajustan en base a la retroalimentación del cliente a lo largo de las entregas.

Este tipo de metodologías, si bien parece no tener desventajas, la más grande es convencer al cliente que este proceso evolutivo es controlable. Además, hay que tener una gran experiencia en la evaluación de los riesgos y requerimientos, ya que luego podemos tener graves problemas si alguno de ellos no es evaluado correctamente.

Flexibilidad ante los cambios se evalúan ante los riesgos . Tiene gestión de riesgos

Modelo de desarrollo iterativo e incremental:

Esta metodología se basa en la idea de construir el software en etapas pequeñas y manejables. El desarrollo se realiza en iteraciones, donde cada iteración incluye actividades de planificación, diseño, implementación y pruebas. Cada iteración entrega un incremento funcional del software y se basa en los aprendizajes y la retroalimentación obtenida en las iteraciones anteriores.

Es una combinación entre dos metodologías :

Metodología iterativa:

- Se basa en ciclos repetitivos de desarrollo del software .
- El proyecto se divide en pequeñas iteraciones como semanas o meses.
- Cada iteración incluye actividades de planificación, diseño, implementación, testing y retroalimentación del cliente.
- Después de cada iteración, el software funcional se entrega al cliente para su opinión.
- Los cambios se incorporan a medida que avanza el proyecto.

Evolución de un producto

Usualmente es una espiral (cuando termina cada espira se analiza y se procesa) En cada espiral se le va agregando lo que le falta

El producto va creciendo

Tiene un núcleo funcional

Metodología incremental :

- El proyecto está dividido en módulos que se desarrollan y se entregan en etapas sucesivas.
- Cada incremento se basa en el trabajo realizado en los incrementos anteriores, lo que permite una evolución gradual.
- Cada incremento agrega características y funcionalidades adicionales al producto. Esto hace que cada incremento sean versiones del producto que cumplan las necesidades del cliente de mejor forma.
- Los incrementos se basan en los incrementos realizados anteriormente.
- A medida que aumentan los incrementos, aumentan la cantidad de empleados necesarios por lo que se permite no disponer de una gran cantidad de personal para la implementación completa del proyecto.

Metodologías ágiles:

Las metodologías ágiles se centran en la flexibilidad, la colaboración y la entrega de valor de manera incremental. Estas metodologías fomentan la autoorganización del equipo, las iteraciones cortas y frecuentes, la comunicación constante con el cliente y la capacidad de respuesta a los cambios. Se prioriza la entrega temprana de funcionalidades y la adaptación continua a medida que se obtiene feedback del cliente. Algunas son Scrum, Kanban y Extreme Programming (XP).

- El destinatario de la solución es parte del equipo de trabajo
- Se termina una funcionalidad y se prueba
- El cliente mediante un historial de usuario indica por qué módulo empezar
- Cada entrega es un sprint. Estos entregas son de 1 a 4 semanas y se entrega un software funcional
- Se analiza se diseña se prueba y se pone en marcha todo esto dentro de un sprint

Desarrollo orientado a prototipos:

Esta metodología se centra en la construcción rápida de prototipos del software para obtener una comprensión temprana de los requisitos y las necesidades del cliente. Los prototipos se utilizan para obtener feedback y refinar los requisitos antes de pasar a la implementación final. Debe quedar muy claro que el prototipo es desecharlo y NO debería evolucionar desde su estado de prototipo en el producto final

- Cada vez que se termina una etapa se muestra y se prueba
- Refinar el procedimiento
- No es eficiente , no es un buen algoritmo . Su objetivo es mostrar algo hecho al cliente (no terminado)
- Requiere conocer los requerimientos previamente
- Diseño rápido del modelo
- Desechado : En caso de que el prototipo al cliente no le sirve se desecha y se vuelve a empezar
- Incremental : en este caso el programa va evolucionando a medida que avanza .
- La desventaja es que podés arrastrar errores desde el comienzo .

Consiste en la construcción, prueba y reelaboración de un prototipo hasta que se logra una versión aceptable del mismo. Así, mediante la construcción de esta versión preliminar del sistema que provee una funcionalidad simulada es posible identificar los requerimientos del cliente. De forma tal que si el prototipo no se ajusta a las expectativas del mismo, se diseña y construye otro prototipo rápidamente, refinado a partir del feedback de los usuarios. Gracias a esta técnica, el diseño va evolucionando ajustándose cada vez más a lo que necesita el cliente y a los requerimientos del sistema, hasta que se logran definir claramente y en su totalidad las especificaciones del mismo, desarrollando un prototipo final.

El modelo de desarrollo de software de prototipos tiene seis fases dentro del Ciclo de Vida del Desarrollo de Software que se distribuyen de la siguiente manera:

- Recopilacion y analisis de requisitos
- Diseño rápido
- Construir un prototipo
- Evaluación del usuario
- Refinar el prototipo
- Implementar el producto y mantener

Existen dos tipos de metodologías basadas en prototipo :

- Prototipo rápido y desecharable : Se realiza un desarrollo rápido para mostrar cómo se verá visualmente el requisito sin un desarrollo profundo. De esta manera, los comentarios del cliente ayudan a impulsar cambios en las especificaciones y el prototipo es desecharido y creado nuevamente hasta que se logran especificar los requerimientos en su totalidad. Como cada prototipo desarrollado se descarta, ninguno de estos se utilizará para el desarrollo del prototipo final. Conviene usar esta técnica para explorar ideas y obtener comentarios instantáneos sobre los requisitos del cliente. Este es el punto de vista de Brooks, el cual señala que cada prototipo, al no ser el sistema funcional y no ajustarse a las especificaciones del sistema, es conveniente desecharlo y empezar de cero con más conocimiento de las necesidades del cliente y mayor inteligencia para construir una versión refinada del sistema en la que se resuelvan los inconvenientes del prototipo anterior.
- Prototipos evolutivos : En los prototipos evolutivos, el prototipo desarrollado se perfecciona de forma paulatina en función de los comentarios de los clientes hasta que finalmente se acepta. A diferencia del anterior, no se desechan los prototipos si no que se desarrolla uno sobre el mismo previamente construido. Esto ayuda a ahorrar tiempo y esfuerzo ya que desarrollar un prototipo desde cero para cada iteración del proceso a veces puede resultar muy frustrante. Suele ser útil para proyectos donde cada funcionalidad debe verificarse una única vez permitiendo no tener que volver sobre lo que ya se desarrolló y validó con el cliente. Si bien este tipo de modelo permite un desarrollo más veloz, es importante aclarar que en caso de que se produzcan muchos cambios en los requerimientos del sistema, posiblemente se llegue a un punto donde sea más conveniente volver a empezar que seguir evolucionando el prototipo
- Creación de prototipos incrementales : En la creación de prototipos incrementales, el producto final se divide en diferentes prototipos o módulos más pequeños y de forma tal que es posible desarrollar cada uno individualmente. Finalmente, los diferentes prototipos se combinan en un solo producto final. Este método es útil para reducir el tiempo de retroalimentación entre el usuario y el equipo de desarrollo de aplicaciones.

Ventajas

- Útil cuando los requerimientos son cambiantes o no están detallados.

- En situaciones cuando no se conoce bien la aplicación.
- Funciona bien cuando el usuario no se quiere comprometer con los requerimientos.
- Cuando se necesita probar una arquitectura o tecnología que no se conoce con precisión.
- Cuando se requiere rapidez en el desarrollo.

Desventajas

- Es difícil estimar cuándo se tendrá un producto aceptable.
- En la misma línea no se sabe cuántas iteraciones serán necesarias.
- Da una falsa ilusión al usuario sobre la velocidad del desarrollo, ya que este cree que la aplicación está más avanzada de lo que realmente está.
- Existe el riesgo de que el producto pueda ser entregado incluso si no cumple con los estándares requeridos.
- Requiere de una participación activa del usuario que no siempre es posible

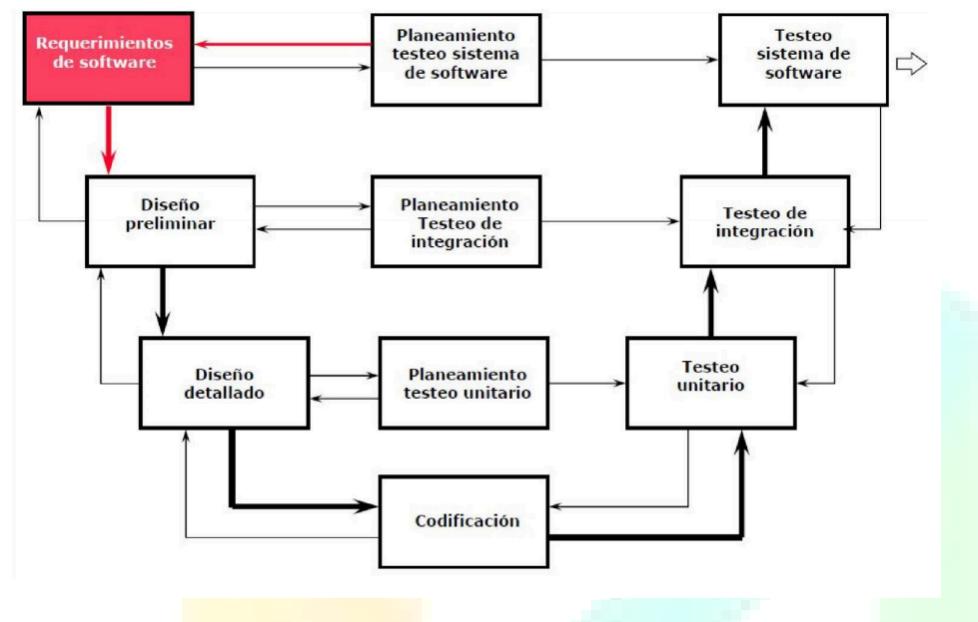
Ingeniería de requerimientos

- Se necesita comprender las necesidades del cliente(Estas necesidades pueden ser estratégicas , internas y externas) las cuales se hacen difíciles de entender dependiendo de la capacidad tanto del cliente como el equipo de trabajo. Esto resulta en documentar, diseñar la solución que tratamos de entender .
- La ingeniería de requerimientos usa un proceso de validación , documentación y lo que resulta en una especificación de requerimientos (srs) .Para documentar las especificaciones de requerimientos se usa una plantilla : 830 IEEE.

Definición :

Hemos analizado la ingeniería de requerimientos y hemos llegado a la conclusión de que es un proceso mediante el cual analizamos, especificamos y validamos los requerimientos de los clientes. Su objetivo es asegurar que el sistema o solución proporcionada satisfaga las necesidades de los usuarios de manera eficiente. Al profundizar en este proceso, podemos afirmar que sus beneficios incluyen la comprensión completa de las necesidades de los usuarios, lo que permite al equipo de desarrolladores implementar la solución de manera efectiva y reducir al mínimo la cantidad de errores. Además, facilita la posibilidad de realizar cambios en el software de forma gradual, lo que a su vez contribuye a la reducción de costos y tiempos durante el proceso de desarrollo.

Importancia de los requerimientos



Impactos de los requerimientos

Impacto positivo y negativo de la práctica de requerimientos de software

Impacto positivo:

- 1. Mejora la comunicación:** La práctica de requerimientos de software ayuda a establecer una comunicación clara y efectiva entre los diferentes actores involucrados en el desarrollo del software, como los clientes, los usuarios finales y los desarrolladores. Esto reduce la posibilidad de malentendidos y asegura que todos tengan una comprensión común de lo que se espera del software.
 - 2. Aumenta la satisfacción del cliente:** Al involucrar a los clientes y usuarios finales en el proceso de definición de requerimientos, se asegura que el software cumpla con sus necesidades y expectativas. Esto aumenta la satisfacción del cliente y reduce la posibilidad de que se realicen cambios significativos en etapas posteriores del desarrollo.
 - 3. Reduce los costos y el tiempo de desarrollo:** Al definir claramente los requerimientos desde el principio, se evitan retrabajos y cambios de último momento, lo que puede resultar en ahorros significativos en términos de tiempo y costos de desarrollo.
 - 4. Facilita la planificación y el seguimiento:** Los requerimientos bien definidos permiten una mejor planificación y seguimiento del proyecto. Los equipos de desarrollo pueden estimar de manera más precisa los recursos necesarios y el tiempo requerido para completar el proyecto.

Impacto negativo:

1. Complejidad y ambigüedad: La definición de requerimientos puede ser un proceso complejo y propenso a la ambigüedad. Los requerimientos mal definidos o ambiguos pueden llevar a malentendidos y a la entrega de un software que no cumple con las expectativas del cliente.

2. Cambios constantes: A medida que el proyecto avanza, es posible que los requerimientos cambien debido a cambios en las necesidades del cliente o a nuevas ideas que surgen durante el desarrollo. Estos cambios pueden afectar la planificación y el alcance del proyecto.

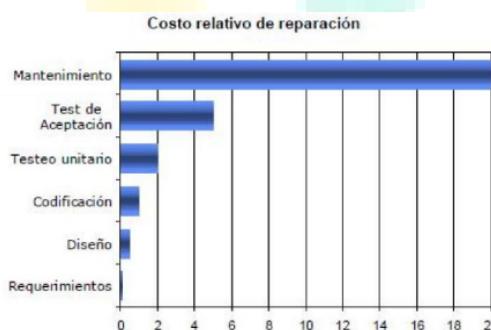
3. Falta de participación del cliente: Si los clientes o usuarios finales no participan activamente en la definición de requerimientos, existe el riesgo de que el software no cumpla con sus necesidades y expectativas. Esto puede resultar en una baja satisfacción del cliente y en la necesidad de realizar cambios significativos en etapas posteriores del desarrollo.

4. Costos adicionales: La práctica de requerimientos de software puede requerir recursos adicionales, como tiempo y personal, para llevar a cabo entrevistas, reuniones y documentación. Estos costos adicionales deben ser considerados en la planificación del proyecto.

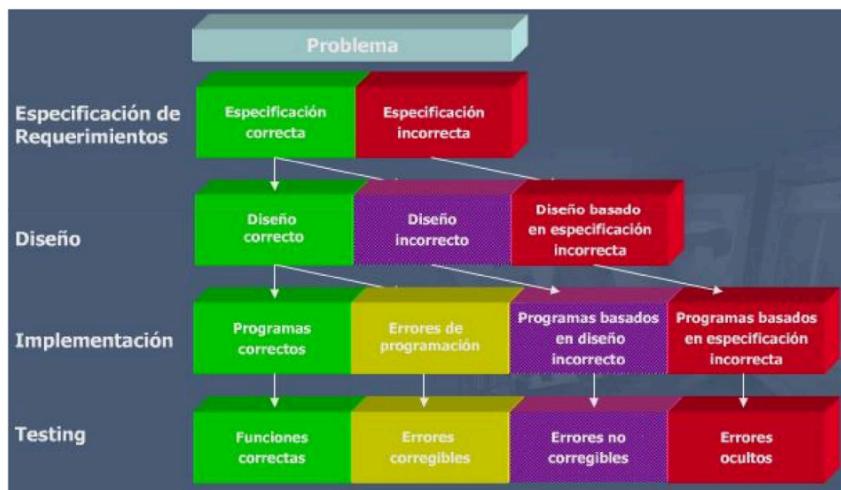
Impacto de los requerimientos

- ❖ Cuanto más tarde en el ciclo de vida se detecta un error mas cuesta repararlo .

Evidencia : incremento del costo de reparación



Explicaciones : Modelo de Mizuno



Conclusión :

- Cuanto más tarde en el ciclo de vida se detecta un error mas cuesta repararlo
- Muchas errores permanecen latentes y no son detectados hasta bastante después de la etapa en que se cometieron
- Los errores en los requerimientos se pueden detectar

Posibles tipos de fallas y sus causas :

Causa posible \ Tipo de Falla	Falta de proceso sistemático	Comunicación pobre entre la gente	Falta de conocimiento apropiado o comprensión compartida	Documentación inapropiada, incompleta o imprecisa	Gestión pobre de gente o recursos
Del proceso de desarrollo Beneficio negativo o el sistema no se usa	✓	✓			✓
De interacción Bajo nivel de uso del sistema		✓	✓	✓	
De expectativas No alcanzó las expectativas de ningún stakeholder		✓	✓	✓	
Tipo de correspondencia Los objetivos no han sido alcanzados	No se analiza porque es un tema que se relaciona con la esencia del desarrollo de software				

❖ Los requerimientos en el desarrollo

Son el punto de partida y de llegada del proceso de desarrollo
 están estrechamente ligados a la calidad del software
 contribuyen al éxito de los proyectos

- ❖ Son importantes para estimar riesgos
- ❖ La mala gestión de requerimientos tiene impactos negativos en los proyectos
- ❖ Realizar una buena gestión de requerimientos es posible y ventajoso en términos de costos

Requerimientos

1. Condición o capacidad que necesita el usuario para resolver un problema o alcanzar un objetivo
2. Condición o capacidad que debe satisfacer o poseer un sistema o un componente de un sistema para satisfacer un contrato, un Standard, una especificación u otro documento formalmente impuesto.
3. Representación documentada de una condición o capacidad como en 1 ó 2.

Se clasifican en :

Orientados al mercado - Específicos para un cliente

Orientados al Mercado	Específicos para un cliente
<ul style="list-style-type: none">• Bocetados e informales• Técnicas más de manufactura que de SE• Especificación en forma de presentación comercial• "Cliente" no fácilmente identificable• Consultores para aspectos deseables• Enfoque poco estructurado.	<ul style="list-style-type: none">• Voluminosos y más "formales"• Uso de técnicas de Ingeniería de Software• Largas especificaciones• Uso del conocimiento del dominio• Proyectos basados en personal propio• Enfoque estructurado con un enfoque particular

Requerimientos verdaderos y falsos

Los requisitos son declaraciones o especificaciones de funcionalidades y características deseadas de un sistema

Ejemplo :

Validos

1. El sistema debe permitir a los usuarios enviar mensajes privados entre ellos
2. El sistema debe generar un informe mensual de ventas
3. El sistema debe ser compatible con los navegadores

Etc.

No validos

1. El sistema debe ser capaz de volar como un avión
2. El sistema debe ser capaz de predecir el futuro

Etc.

Tipos de Requerimientos según las capacidades o características

Necesidades : Característica o capacidad requeridas al sistema software para resolver el problema

- Funcionalidad
- Comportamiento del sistema
- Performance, tiempo de respuesta
- Necesidades operacionales
- Características de las interfaces

Deseos : Característica o capacidad de un sistema de software deseadas por los stakeholders , no son imprescindibles para resolver el problema

- Funcionalidad extra
- Comportamientos específicos del sistema
- Características de la interface del usuario
- Algoritmos particulares

Expectativas : Las características o capacidades de un sistema software esperadas por los stakeholders. A menudo no explícitas y por ello se pierden fácilmente

- Aspectos específicos del dominio , funcionalidades
- Confiabilidad
- Características de la interface con el usuario
- Características de performance

Ejercicio : Considere un sistema de Biblioteca

Dar Ejemplos de :

Necesidades:

- Una base de datos del Stock
- Una base de datos del Usuario
- Funcionalidad para Retirar y devolver
- Funcionalidad de búsqueda

Expectativas:

- Cargue rápido la búsqueda del libro.
- Guardar el inicio de usuario.
- Interfaz amigable, moderna y en web.
- Fácil uso y navegación
- Mantenimiento mensual y actualización

Deseos:

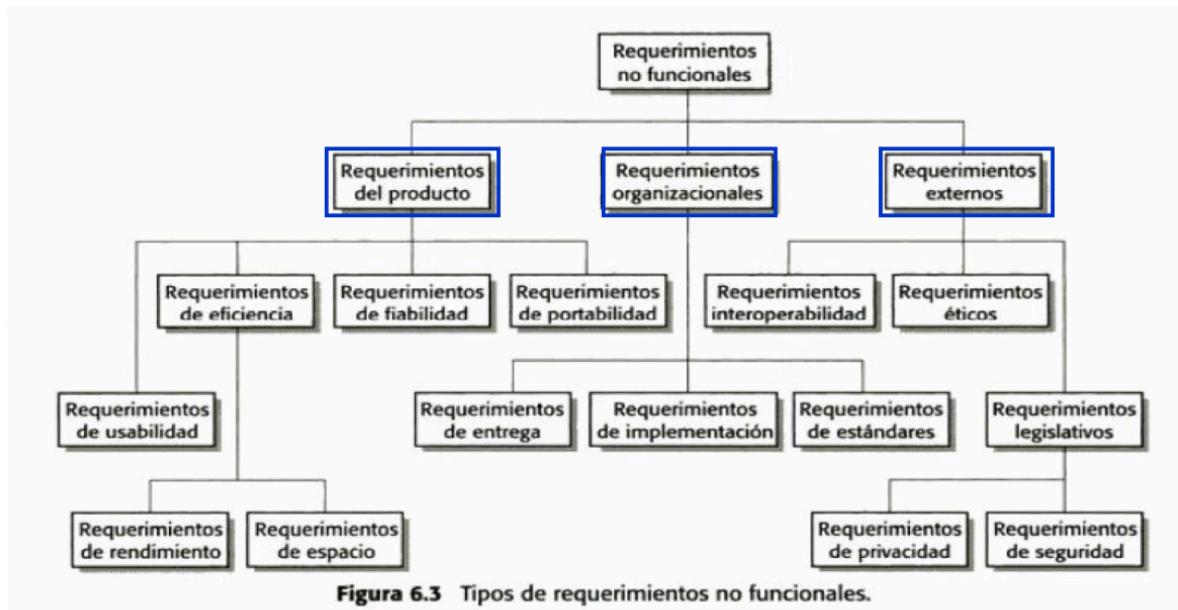
- Cartel que avise a los deudores de libros.
- Sugerencias relacionadas.
- Pedidos del mes.
- Una lista de sugerencias dada por los usuarios
- Interfaz amigable , moderna y en web

Requerimientos funcionales y no funcionales

Requerimientos no funcionales : Restricciones globales sobre cómo debe construirse y funcionar el sistema

Se deben redactar los requerimientos de manera cuantitativa para que se puedan probar de manera objetiva.

- ✓ Performance o Eficiencia
- ✓ Precisión
- ✓ Seguridad
- ✓ Amigabilidad o Usabilidad
- ✓ Mantenibilidad
- ✓ Portabilidad
- ✓ Interoperabilidad
- ✓ Confiabilidad o fiabilidad
- ✓ Restricciones particulares del dominio / Requerimientos de estándares
- ✓ Restricciones particulares de la tecnología de implementación / Requerimientos de implementación
- ✓ Restricciones Operacionales / Requerimientos de entrega
- ✓ Restricciones físicas / Requerimientos de entrega



Requerimientos del Producto

Especifican el comportamiento del producto.

- Rendimiento en la rapidez de ejecución del sistema y cuanta memoria requiere
- Requerimientos de fiabilidad que fijan la tasa de fallos para que el sistema sea aceptable
- Requerimientos de portabilidad
- Requerimientos de usabilidad

Requerimientos organizacionales

Se derivan de políticas y procedimientos existentes en la organización del cliente y en la del desarrollador

- Estándares en los procesos que deben utilizarse
- Requerimientos de implementación, como leng de programación o el método de diseño a utilizar
- Requerimientos de entrega que especifican cuándo se entregará el producto y su documentación

Requerimientos externos

Incluye todos los requerimientos que se derivan de los factores externos al sistema y de su proceso de desarrollo.

- Requerimientos de interoperabilidad que definen la manera en que el sistema interactúa con sistemas de otras organizaciones
- Requerimientos legislativos que deben seguirse para asegurar que el sistema funcione dentro de la ley
- Requerimientos éticos (puestos para asegurar que el sistema sera aceptado por susu usuarios y por el público en general).

Ejemplos :

Requerimiento del producto

8.1 La interfaz de usuario del LIBSYS se implementará como HTML simple sin marcos o applets Java.

Requerimiento organizacional

9.3.2 El proceso de desarrollo del sistema y los documentos a entregar deberán ajustarse al proceso y a los productos a entregar definidos en XYZCo-SP-STAN-95.

Requerimiento externo

10.6 El sistema no deberá revelar al personal de la biblioteca que lo utilice ninguna información personal de los usuarios del sistema aparte de su nombre y número de referencia de la biblioteca.

Métricas para especificar las propiedades no funcionales del Sistema

Propiedad	Medida
Rapidez	Transacciones procesadas por segundo. Tiempo de respuesta al usuario y a eventos Tiempo de actualización de la pantalla
Tamaño	K Bytes Número de chips de RAM
Facilidad de uso	Tiempo de formación Número de cuadros de ayuda
Fiabilidad	Tiempo medio entre fallos Probabilidad de no disponibilidad Tasa de ocurrencia de fallos Disponibilidad
Robustez	Tiempo de reinicio después de fallos Porcentaje de eventos que provocan fallos Probabilidad de corrupción de los datos después de fallos
Portabilidad	Porcentaje de declaraciones dependientes del objetivo Número de sistemas objetivo

Requerimientos funcionales :

- Descripción de lo que el sistema hace
- Qué información necesita ser mantenida ?
- ¿Qué necesita ser procesado ?

Rol de los requerimientos

- Acuerdo desarrolladores- clientes - usuarios finales
- Base para el diseño
- Soporte para verificación y validación (V y V)
- Soporte a la evolución del sistema
- Modela lo que se necesita y formula el problema
- Variedad de medios de expresión
- Diversas organizaciones de la estructura
- Razones para desarrollar una SRS
 - Proceso de comunicación
 - Arreglo contractual
 - Evaluar el producto final y las pruebas
- Tradicionalmente fue considerada una especificación funcional dado que respondía al concepto de entrada proceso y salida

Análisis de requerimientos

- Problemas no estructurados
- Los requerimientos se encuentran en un contexto organizacional
- Los problemas de análisis son dinámicos
- Se requiere de habilidades y conocimientos interdisciplinarios
- El conocimiento de analista evoluciona continuamente

- Utiliza información del ambiente
- Inicia con un modelo mental del problema en un nivel muy abstracto
- El proceso de análisis es cognitivo

Proceso de ingeniería de requerimientos

Adquisición - representación - evaluación

Una especificación:

- Describe objetos, reglas de negocio, agentes, componentes del sistema de información
- Representa un conjunto de proposiciones que se consideran válidas hasta que se demuestre lo contrario.
- Involucra actividades técnicas y de administración

Especificación de requerimientos de software (SRS)

- Documentación de los requerimientos
- Herramienta de comunicación de la comprensión del dominio , el negocio y el sistema
- Es parte de un acuerdo contractual
- Tiene un papel clave en la etapa de PRUEBAS
- Usuarios de la srs : Ejecutivos , gerentes , expertos del dominio , analistas , arquitectos , diseñadores , implementadores y testers.

Beneficios de la SRS

- Establece las bases del acuerdo desarrollador-cliente sobre qué debe hacer el producto
- Reduce el esfuerzo de desarrollo
- Provee una base de estimaciones
- Provee una baseline para validacion y verificación
- Sirve como base para el mantenimiento

Características de una buena SRS

MOVERA COCORANO

Cada requerimiento debería tener estas características :

Correcta: El sistema no debe tener ningún requerimiento que no esté representado en la sra o viceversa

No ambigua : Mitiga la ambigüedad con la velocidad , no puede ser la velocidad “ buena” tiene que ser cuantificable .

Una SRS es no ambigua si, y sólo si cualquier sentencia de requerimiento incluida en ella tiene una sola interpretación

Completa : si todos los requerimientos significativos que impone el software están en la srs ,

el usuario debe ingresar todos los datos de entrada y conseguir los datos de salida que necesite para cumplir las necesidades y todas las etiquetas y referencias a las figuras, tablas y diagramas de la SRS y las definiciones de los términos y unidades de medida

Consistente : consiste en los no conflictos entre requerimientos .

Ejemplo : Podría ser que no se puedan modificar la planilla de datos . El requerimiento es consistente y no es posible modificarlo .

Rankear : Un requerimiento podría ser más relevante para el cliente .Por ellos los requerimientos deben ser rankeables . Ayuda a priorizar qué requerimiento empezar a construir

Verificable : Que el requerimiento pueda verificarse . Que el sistema pueda chequear que satisfaga el requerimiento .

Modificable : Que cualquier cambio pueda hacerse fácilmente , manteniendo la completitud y consistentemente

Rastreable : Pueda ser rastreable hacia adelante y hacia atrás . Que se pueda verificar y buscar en la srs que el requerimiento está hecho en el sistema .

Índice del Standard IEEE 830

1. Introducción

Provee una visión global de la SRS

1.1 Propósito

Delinear el propósito de la SRS y especificar a quién se dirige

1.2 Alcance

Identificar los productos de SW , explicar que hará y que no hará cada uno , describir la aplicación

1.3 Definiciones , acrónimos y abreviaturas

Incluir las definiciones de los términos , acrónimos y abreviaturas requeridas para interpretar la SRS

1.4 Referencias

Proveer una lista completa de todos los documentos referenciados

1.5 Overview

Describir que contiene el resto de la SRS y explicar como esta organizada la SRS

2. Descripción general

Describe los factores generales que afectan al producto y a los requerimientos facilita su comprensión

2.1 Perspectiva del producto

Relación con otros productos o proyectos

Productos independientes

Componentes de un sistema de un proyecto

2.2 Funciones del producto

Resumen de las funciones que ejecutará el software

Comprendibilidad

Diagrama de bloques

No establece requerimientos específicos

2.3 Características del usuario

Características generales del usuario

Restricciones impuestas por los usuarios

2.4 Restricciones Generales

Límites al desarrollador

Requerimientos específicos o restricciones sobre la solución

Límites a las opciones para diseñar el sistema :

- Políticas regulatorias
- Limitaciones de hardware
- Interfaces con otras aplicaciones
- Operaciones paralelas
- Funciones de auditoría
- Funcionales de control
- Consideraciones de seguridad

2.5 Supuestos y dependencias

Factores que afectan los requerimientos

Restricciones de diseño

Cambios que pueden afectar los requerimientos en la SRS

3. Requerimientos específicos

El sector más importante de la SRS

Presentación y desarrollo de los requerimientos

3.1- Requerimientos Funcionales.

Definen las acciones del sistema al aceptar y procesar las entradas y en generar las salidas. En general tienen la forma “el sistema DEBE”

Control de validez de las entradas • Secuencias de operaciones • Respuestas a situaciones anormales (overflow, manejo de errores) • Relaciones entre entradas y salidas

3.1.1- Requerimiento Funcional n+1

3.1.1.1- Introducción.

3.1.1.2- Inputs.

3.1.1.3- Procesamiento.

3.1.1.4- Salidas.

3.2- Requerimientos de Interfase Externa.

Descripción detallada de todas las entradas y salidas del sistema. Refleja contenidos y formatos:

- Nombre del ítem • Propósito • fuente/destino de la entrada/salida • Rango de validez, precisión, tolerancia • Unidad de medida • Timing • Relaciones con otras entradas/salidas • Formatos y organización de pantallas y ventanas • Formatos de datos y comandos • Mensajes de finalización

3.2.1- Interfaces de Usuario.

3.2.2- Interfaces de Hardware.

3.2.3- Interfaces de Software.

3.2.4- Interfaces de Comunicaciones.

3.3- Requerimientos de Performance.

Requerimientos de performance: Requerimientos estáticos y dinámicos del software o de la interacción humana del software como un todo

- Requerimientos estáticos incluyen: • Cantidad de terminales • Usuarios simultáneos • Cantidad y tipo de información a manejar • Requerimientos dinámicos incluyen: • Número de tareas y transacciones a procesar por unidad de tiempo, carga normal y picos

3.4- Pautas de Diseño.

Restricciones de diseño: se imponen por estándares, hardware, software disponible, etc.

3.4.1- Cumplimiento de Standards.

3.4.2- Limitaciones de Hardware.

.....

3.5- Atributos.

Hay atributos del software que pueden documentarse como requerimientos.

Requerimientos no funcionales • Confiabilidad • Disponibilidad • Mantenibilidad • Portabilidad •

Es necesario organizar los requerimientos . Diferentes tipos de sistemas requieren diferentes tipos de organizaciones de requerimientos • Por clase de usuario • por objetos • Por modo •

Especifique los requerimientos cuantitativamente

Voz activa: “el sistema debe hacer algo” y no “algo debe hacerse”

vitar el debería, podría y los condicionales

En “el usuario debe” especifique al usuario: “el cajero debe”

El sistema debe + verbo

3.5.1- Seguridad.

3.5.2- Mantenibilidad.

.....

3.6- Otros Requerimientos.

3.6.1- Data Base.

3.6.2- Operaciones.

3.6.3- Adaptaciones al Medio

4. Apéndices



Proceso de ingeniería de requerimientos

“Es el proceso sistemático de desarrollar requerimientos a través de un proceso cooperativo e iterativo de analizar el problema, documentar las observaciones resultantes de una variedad de formatos de representación y chequear la precisión de la comprensión obtenida”

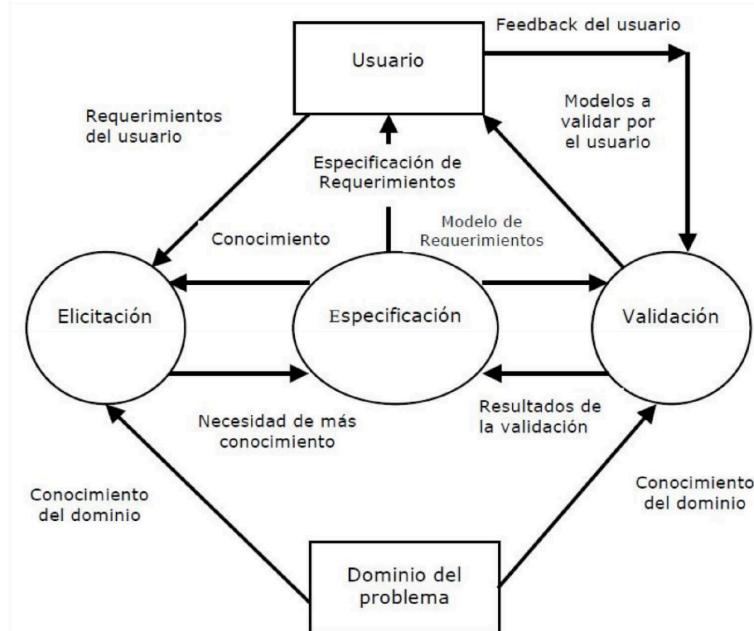
Aspectos fundamentales de la IR :

- Comprender el problema
- Describir el problema
- Acordar sobre la naturaleza del problema

Procesos - Elicitación -Especificación - validación

Cada requerimiento se define en términos de :

- Propósito del proceso
- El input del proceso y sus orígenes
- Las actividades que tienen lugar durante la ejecución del proceso y las técnicas usadas
- Los entregables final e intermedios
- La interacción con otros procesos de la Ing de Req



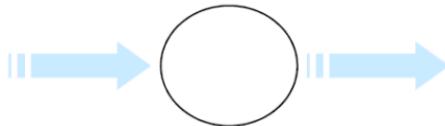
Proceso de ELICITACIÓN

Propósito

- Ganar conocimiento relevante del problema, para producir una especificación rigurosa del software necesario para resolver el problema
- Al final de la fase de RE el analista podría ser un “experto” en el dominio del problema

• Input

- Expertos del dominio
- Literatura sobre el dominio
- Software existente en el dominio
- Software similar en otros dominios
- Standards nacionales e internacionales
- Otros stakeholders



• Actividades

- Identificar fuentes de conocimiento
- Adquirir y decidir sobre la relevancia de un conocimiento
- Comprender el significado e impacto del conocimiento
- Técnicas utilizadas: entrevistas y cuestionarios, prototipos, ...

• Productos

- Es un proceso de creación de modelos:
 - Modelos conceptuales hasta modelos de especificación de req
 - El analista comienza con modelos mentales con conocimiento dependiente del dominio
 - Al avanzar los modelos se van orientando al software
 - No se produce ningún modelo formal
 - Sucesión de modelos mentales del dominio del problema

Una especificación : es un contrato entre cliente y desarrollador – define el comportamiento funcional deseado del software y otras propiedades de este, tales como performance, seguridad, ...

No indica CÓMO se va a lograr todo lo que se enumera

Proceso de ESPECIFICACION

• Propósito

- Acuerdo usuario-desarrollador sobre el problema a resolver
- Pauta para el desarrollo de un sistema de software

• Input

- Conocimiento sobre el dominio del problema
- Lo provee el proceso de **ELICITACION**

• Actividades

- Análisis y asimilación del conocimiento de los requerimientos
- Síntesis y organización del conocimiento en un modelo de requerimientos

• Productos

- Se producen una variedad de modelos:
 - Modelos orientados al usuario, que especifican comportamiento, características no funcionales
 - Modelos orientados al desarrollador, que especifican propiedades funcionales y no funcionales del software y restricciones

• Interacción con otros procesos

- Proceso central de la IR

Validación : Proceso que certifica que se resolverá el problema correcto

Proceso de VALIDACION

- **Propósito**
 - Certifica la consistencia del modelo de requerimientos con las intenciones del cliente y usuario
 - Visión más general que el concepto común de validación
 - Se aplica también a modelos intermedios
- **Input**
 - Todo modelo está sujeto a validación. Cada modelo es una entrada
 - El conocimiento sobre el dominio del problema debe ser validado
 - Algunas partes del modelo formal
- **Actividades**
 - Interacción entre analistas, clientes y usuarios en el dominio del problema
- **Productos**
 - Modelo de requerimientos
- **Interacción con otros procesos**
 - La Validación está presente en todos los procesos de la IR, y la dispara:
 - Nuevo conocimiento sobre el dominio del problema (ELICITACION)
 - Formulación de un modelo de requerimientos (ESPECIFICACION)

Gestión de requerimientos

- Administración de los cambios en los requerimientos
- Los requerimientos no pueden administrarse si no son rastreables (trazables)
- Un requerimiento es rastreable si se conoce: Quien lo sugirió ,por qué existe y Dependencia con otros requerimientos

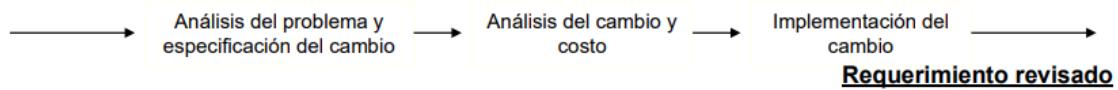
Requerimientos estables y volátiles

- Los cambios en los requerimientos ocurren desde el comienzo del desarrollo, durante el desarrollo y luego de la puesta en marcha
- Algunos requerimientos están más sujetos al cambio que otro:
Los más estables : vinculados con la esencia del sistema
Los más volátiles :son específicos de la instancia del sistema en un ambiente
- Razones por las que los requerimientos cambian:
 - ❖ Errores, conflictos, inconsistencias
 - ❖ Evolución del conocimiento del cliente/usuarios finales
 - ❖ Problemas técnicos, de calendarios, de presupuesto
 - ❖ Cambios en las prioridades de los usuarios
 - ❖ Cambios del contexto u organizacionales
 - ❖ Política

Gestión del cambio

- La identificación del problema de requerimientos
- El análisis del cambio propuesto
- La implementación del cambio

Problema identificado



Rastreabilidad

- Permite evaluar el impacto de los cambios de requerimientos
- Tipos de rastreabilidad – HACIA DELANTE – HACIA ATRÁS

Elicitación

Es el proceso de adquirir (elidar) todo el conocimiento relevante necesario para producir un modelo de requerimientos de un dominio de problema

el objetivo es entender el dominio del problema particular y tener conocimiento del dominio del problema

Problemas en el proceso de elicitation :

- Dónde encontrar la información
- Cómo obtener el conocimiento
- Conocimiento distribuido en diferentes fuentes, en muchos casos humanos

Problemas en la transmisión del conocimiento del dominio :

- El conocimiento no está disponible en un formato necesitado por los analistas
- Usuarios sesgados por sistemas actuales
- Dificultad al elicitar información de expertos humanos
- Sesgo que introduce el analista

Técnicas de elicitation

Entrevistas

Una entrevista puede tener diferentes estructuras y preguntas ya sean abiertas o cerradas . El objetivo de esta es registrar información fácilmente utilizable en los procesos de la ingeniería de requerimiento . Además de descubrir información proveniente del entrevistado precisa y eficiente Asegura al entrevistado que su comprensión del tema ha sido explorado , entendido y valorado.

Proceso de entrevistas :

1. Planificación y preparación
 - 1.1 Establecer los objetivos
 - 1.2 Preparar la preguntas
 - 1.3 Adquirir información sobre el tema de la entrevista
 - 1.4 Organizar el ambiente para tener una entrevista efectiva
2. Conducción de la entrevista
 - 2.1. Conversar en modo profesional usando técnicas adecuadas de preguntas
 - 2.2 Tomar notas sin imponer ideas subjetivas ni prejuicios
3. Consolidación y representar la información obtenida
 - 3.1. Elaborar minutas
 - 3.2. Consolidar notas provenientes de los diferentes entrevistados
4. Validación de la información obtenida de los entrevistados
 - 4.1. Entrevistas de seguimiento para validar supuestos
 - 4.2. Dar feedback a los entrevistados de los conversados

Cuestionarios

- Permiten cuantificaciones por el uso de preguntas cerradas
- Permiten evaluar la difusión de un concepto
- Convenientes cuando: – Dispersión geográfica de los entrevistados – Gran cantidad de involucrados y se necesita establecer % de aprobación – Identificar problemas del sistema actual
- La relación de las preguntas no es trivial y se deben seguir técnicas de preparación de formularios • Las escalas utilizadas requieren de mucho cuidado

Cuestionarios Escalas

- Escalamiento proceso de asignar nros. u otros símbolos a un atributo o característica con objeto de medir ese atributo o característica.
- Las escalas son arbitrarias y pueden no ser únicas

Diferentes formas de medir las escalas

- Nominal: usadas para clasificar cosas. Forma más débil de medición – QUÉ TIPO DE PROGRAMA UTILIZA PRINCIPALMENTE?
 1. Un procesador de palabras
 2. Una hoja de cálculo
 3. Una base de datos
 4. Un programa de graficación
- Ordinal: permiten clasificación, pero también implica ordenamiento de rango
 1. Excesivamente útil
 2. Muy útil
 3. Moderadamente útil
 4. No muy útil
 5. Inútil
- De intervalo: los intervalos entre cada uno de los números son iguales. Ejemplo: escalas Celsius INUTIL EXTREMADAMENTE ÚTIL
 - 1
 - 2
 - 3
 - 4
 - 5
- De relación: similar a la escala de intervalo, pero el intervalo coincide con los nros

Surveys

- Válidos cuando el tamaño del universo hace imposible interactuar con todos los usuarios
- Tener en cuenta mismos criterios de formulación de cuestionarios
- Se agrega el componente de diseño de la muestra y análisis estadístico de los datos
- Un survey permite obtener información y analizarla. Los entrevistados responden preguntas que fueron desarrolladas con anterioridad.

El proceso de preparación está formado por 7 pasos:

1. Identify the research objectives.
2. Identify and characterize the target audience.
3. Design the sampling plan.
4. Design and write the questionnaire.
5. Pilot test the questionnaire.
6. Distribute the questionnaire.
7. Analyze the results and write a report.

- Surveys difiere de otros métodos debido a que permiten generalizar acerca de creencias y opiniones de varias personas mediante el estudio de un conjunto de ellas.
- Sólo se puede generalizar si se sigue el proceso estrictamente y se hace una buena selección de la muestra.

Brainstorming (Tormenta de ideas)

- Introducida en el año 1930 para generar ideas • Todos los miembros del grupo contribuyen a una lista de problemas a resolver o soluciones a un problema
- Discutir muchas ideas en un corto período de tiempo
- Requiere seguir ciertas reglas:
 - Establecer un tiempo para la reunión
 - No se pueden criticar ideas que otro proponga
 - Asegurar que todos puedan participar libremente
 - Asegurar que todos contribuyan
 - Dejar hablar a los participantes, no al moderador •
- Ventajas
- – Útil para resolver falta de consenso entre participantes
- – Ayuda a entender el dominio del problema
- – Disminuye la dificultad del usuario para transmitir
- – Ayuda a entender: al usuario y al analista

Focus Group

- Se obtienen datos cualitativos. Valiosos para conocer opiniones y actitudes.
- Entrevistas cualitativas a un grupo reducido y seleccionado cuidadosamente
- La composición se basa en la homogeneidad
- La participación es voluntaria y confidencial
- No es una entrevista grupal • No es comparable el resultado obtenido de diferentes grupos
- Un moderador mantiene el grupo enfocado para generar una discusión productiva. Guía la discusión.
- Las preguntas deben ser:
 - Abiertas-cerradas. El moderador no debe sugerir una respuesta en la pregunta
 - Preguntas claramente formuladas, fáciles de entender
 - Ordenadas

Ventajas

- Se puede obtener un amplio rango de información en poco tiempo
- No requieren técnicas complejas de muestreo

Desventajas

- La muestra no es seleccionada al azar ni representa una población, por lo tanto, los resultados no pueden ser generalizados ni tratados estadísticamente
- La calidad de los datos se puede ver influenciada por las habilidades de conducción del moderador

Rapid Application Development /Joint Applications Development

- Sesión estructurada en la que los stakeholders y otros expertos planifican y entienden el sistema a construir. • Requiere claridad del propósito y objetivos en la reunión
- Un facilitador mantiene a los participantes en foco, puede haber observadores pero no pueden participar

Ventajas

- Asegura que los usuarios se involucren temprano, facilita la obtención de los requerimientos del usuario y puede ayudar en el diseño
- Gana compromiso del usuario – Permite resolver conflictos
- El facilitador es imparcial a los requerimientos
- Los proyectos se ejecutan mejor con decisiones grupales
- Reduce los desacuerdos organizativos
- Permite obtener requerimientos rápidos

Desventajas:

- Puede no alcanzarse consenso y algunas cuestiones pueden llevar demasiado tiempo
- Se requiere un buen facilitador para involucrar a los participantes del usuario

Escenarios

- Escenario: historia que ilustra cómo un sistema puede satisfacer las necesidades del usuario
- Descripción detallada de una interacción hombre-máquina
- Diversos medios (gráfica, texto)
- Estructurados en diálogos o narrativas

Ventajas: – Los usuarios encuentran más fácil transmitir su experiencia o conocimiento a través de “contar una historia”

Análisis de formularios

- Formulario: colección estructurada de variables que están formateadas para soportar ingreso de datos y su recuperación
- Es una fuente importante porque:
 - Es un modelo formal
 - Es un modelo de datos
 - Contiene información sobre la empresa
 - Sus instrucciones de uso presentan información del dominio

Lenguaje natural

- Forma más habitual de representación del conocimiento
- Informalidad
- Fuente importante del conocimiento
- Dos limitaciones:
 - El lenguaje natural es muy completo
 - Ambigüedad

Prototipos

- A software requirements prototype is a partial implementation of a software system, built to help developers, users, and customers better understand system requirements.
- Prototype the “fuzzy” requirements: those that, although known or implied, are poorly defined and poorly understood.

StoryBoarding

- Un proceso para ordenar ideas. Combina brainstorming
- Puede ser usado para mostrar casos en grupos, en secuencia o en detalle.
- Se presenta una pregunta guía
- Cada participante en silencio hace una tormenta de ideas y crea una lista de ítems que respondan a la pregunta
- Cada participante enumera el top 3 de su lista
- Se dividen los participantes en subgrupos
- Cada subgrupo revisa los ítems de cada lista y se agrega cualquier ítem que aparezca en la discusión
- Los subgrupos seleccionan el top 5-6 y los escriben en una tarjeta que se les entregó con las indicaciones correspondientes: Solo se pueden escribir de 3 a 5 palabras en letra imprenta.
- Solicitar una carta por subgrupo utilizando algún criterio (el más fácil de implementar, el menos controversial) Se ubica la carta en el pizarrón
- Leer la carta y aclarar cualquier pregunta • Continuar hasta que todas cartas estén en el pizarrón, siguiendo el mismo procedimiento.
- Agrupar las cartas (algunas podrían no tener grupo)
- Pedir a todos que vuelvan a sus listados originales y verifiquen por cualquier ítem no presente en el storyboard. Escribir una carta en caso de que algún ítem no esté presente y ubicarlo en el grupo correspondiente.

Proceso de análisis

Un proceso de análisis en el desarrollo de software es una etapa relevante en el Proceso de Desarrollo en la que se recopilan y documentan los requisitos y objetivos del sistema a desarrollar. Una descripción general del proceso de análisis sería :

1. **Identificación de requerimientos** : El análisis comienza identificando y documentando los requisitos del software . Esto implica la recopilación de información de diversas fuentes , como usuario , cliente y otras partes interesadas relevantes. Los requerimientos pueden ser funcionales (que debe hacer el software) y no funcionales (restricciones y características de calidad)
2. **Análisis de requerimientos** : En esta etapa , se examinan los requisitos recopilados para comprenderlos en profundidad .Se realiza un análisis detallado para asegurarse de que sean coherentes, completos, no ambiguos y factibles. Además, se identifican las dependencias y las relaciones entre los requisitos.
3. **Modelado** : Se utiliza el modelado para representar visualmente los requisitos y las interacciones del sistema. Esto puede incluir diagramas de casos de uso , diagramas de flujo de datos u otros modelados específicos según la metodología utilizada
4. **Validación de requerimientos** : Los requisitos analizados se validan con las partes interesadas para garantizar que reflejen con precisión las necesidades y expectativas del sistema.Esto puede implicar revisiones, discusiones y acuerdos con los stakeholders para asegurar que todos estén alineados en cuanto a lo que se espera del software.
5. **Documentación de requerimientos** : Los requisitos y los resultados del análisis se documenta en un formato claro y comprensible .Esto puede incluir especificaciones de requisitos , documentos de casos de uso, historias de usuarios u otros documentos relevantes que se utilizan en las etapas posteriores del desarrollo (SRS)

El proceso de análisis es esencial para establecer una base sólida para el desarrollo de software. Un análisis adecuado ayuda a comprender las necesidades del sistema, evita malentendidos y cambios costosos más adelante en el proceso. Además, proporciona una referencia clara para el diseño, implementación y prueba de la solución.

Es importante destacar que los detalles y las técnicas específicas utilizadas en el proceso de análisis pueden variar según la metodología o paradigma utilizado, como el enfoque en cascada, evolutivo, ágil o cualquier otro marco de trabajo específico.

Casos de uso

Es un modelo de análisis que por una serie de interacciones entre el sistema y un actor (una entidad externa , ejerciendo un rol determinado) , una determinada forma de utilizar el sistema . Cada interacción comienza con un evento que el actor envía al sistema y continua con una serie de eventos entre el actor , el sistema y posteriormente otro actores involucrados.

¿Que son los casos de Uso ?

- Los casos de uso son una técnica para especificar el comportamiento de un sistema
- “*Un caso de uso es una secuencia de interacciones entre un sistema y alguien o algo que usa alguno de sus servicios.*”
- Por ejemplo, un sistema de ventas, si pretende tener éxito, debe ofrecer un servicio para ingresar un nuevo pedido de un cliente. Cuando un usuario accede a este servicio, podemos decir que está “ejecutando” el caso de uso ingresando pedido
- El conjunto de casos de uso especifica la funcionalidad completa del sistema

Diferencias entre Casos de uso y eventos

- Los eventos se centran en describir qué hace el sistema cuando el evento ocurre, mientras que los casos de uso se centran en describir cómo es el diálogo entre el usuario y el sistema
- Los eventos son “atómicos”: se recibe una entrada, se la procesa, y se genera una salida, mientras que los casos de uso se prolongan a lo largo del tiempo mientras dure la interacción del usuario con el sistema. De esta forma, un caso de uso puede agrupar a varios eventos
- Para los eventos, lo importante es qué datos ingresan al sistema o salen de él cuando ocurre el evento (estos datos se llaman datos esenciales), mientras que para los casos de uso la importancia del detalle sobre la información que se intercambia es secundaria. Según esta técnica, ya habrá tiempo más adelante en el desarrollo del sistema para ocuparse de este tema.

Actores

- **Un actor** es una agrupación uniforme de personas, sistemas o máquinas que interactúan con el sistema que estamos construyendo de la misma forma. Por ejemplo, para una empresa que recibe pedidos en forma telefónica, todos los operadores que reciban pedidos y los ingresen en un sistema de ventas, si pueden hacer las mismas cosas con el sistema, son considerados un único actor: *Empleado de Ventas*.
- *Cada actor puede realizar un número de casos sobre el sistema, recorriendo todos los actores identificados, podemos entonces derivar la totalidad de la funcionalidad del sistema.*

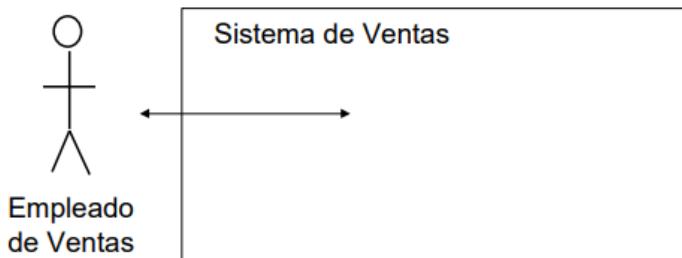


Figura 1 – El empleado de ventas es un actor del sistema de ventas

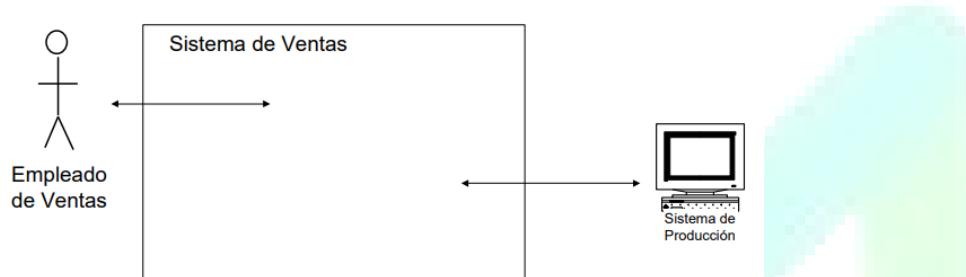


Figura 2 – Cuando el actor es otro sistema se puede cambiar la notación

De estos ejemplos podemos decir que :

- El grupo de usuarios que haga otras operaciones con los pedidos , como por ejemplo autorizarlos , cancelarlos y modificar sus plazos de entrega será un actor
- Todo grupo de usuarios que reciba ciertos informes del sistema , como por ejemplo estadísticas de ventas , será un actor
- El grupo de usuarios que ingrese pedidos al sistema será un actor

Diferencias entre usuarios y actores

Se realiza una diferenciación entre actores y usuarios. El usuario es la persona que está usando el sistema, mientras que los actores representan a los usuarios de los casos, un actor representa un cierto rol que un usuario puede tener. Una persona puede instanciar varios actores diferentes. Se considera al actor como una clase y al usuario como instancias de esa clase. Estas instancias existen solamente cuando el usuario hace algo con el sistema.

En resumen : El usuario es la persona que esta usando el sistema , mientras que los actores representan a los usuarios de los casos , un actor representa un cierto rol que un usuario puede tener

- Un caso de uso es **iniciado por un actor**. A partir de ese momento, ese actor, junto con otros actores, intercambian datos o control con el sistema, participando en ese caso de uso.
- El nombre de un caso de uso se expresa con un verbo en gerundio, seguido generalmente por el principal objeto o entidad del sistema que es afectado por el caso

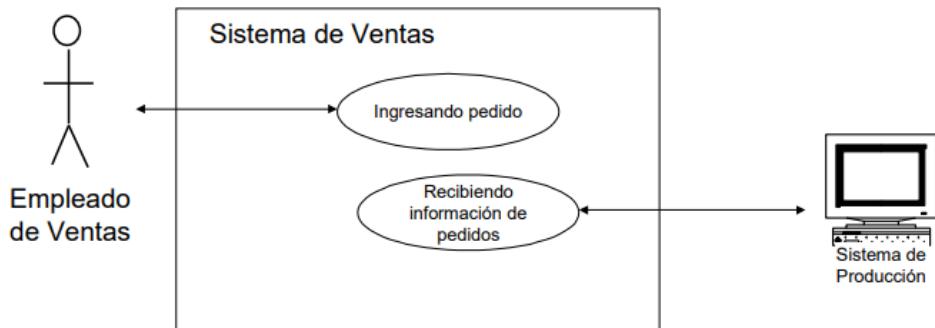


Figura 3 – Los casos de uso se representan gráficamente con óvalos

Descripción de los casos de uso

Los casos de uso se documentan con texto informal. En general, se usa una lista numerada de los pasos que sigue el actor para interactuar con el sistema. A continuación se muestra una parte simplificada de la descripción del caso de uso “Ingresando Pedido”

Caso de Uso : Ingresando Pedido.
Actor : Empleado de Ventas.
1) El cliente se comunica con la oficina de ventas, e informa su número de cliente
2) El oficial de ventas ingresa el número de cliente en el sistema
3) El sistema obtiene la información básica sobre el cliente
4) El cliente informa el producto que quiere comprar, indicando la cantidad
5) El sistema obtiene la información sobre el producto solicitado, y confirma su disponibilidad.
6) Se repite el paso 4) hasta que el cliente no informa más productos
7) ...

Figura 4 – Descripción simplificada del caso de uso “Ingresando Pedido”

Relaciones entre casos de uso

Extensión

Las extensiones tienen algunas características

- Representan una parte de la funcionalidad del caso que no siempre ocurre.
- Son un caso de uso en sí mismas.
- No necesariamente provienen de un error o excepción .

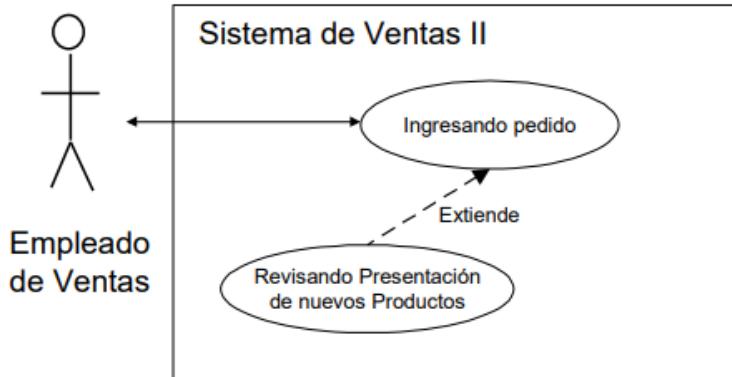


Figura 6 – Una relación de extensión entre dos casos de uso

Uso

Las características de la relaciones de uso son :

- Aparecen como funcionalidad común , luego de haber especificado varios casos de uso.
- Los casos usados son casos de uso en si mismos
- El caso es usado siempre que el caso que lo usa es ejecutado . Esto marca la diferencia con las extensiones , que son opcionales

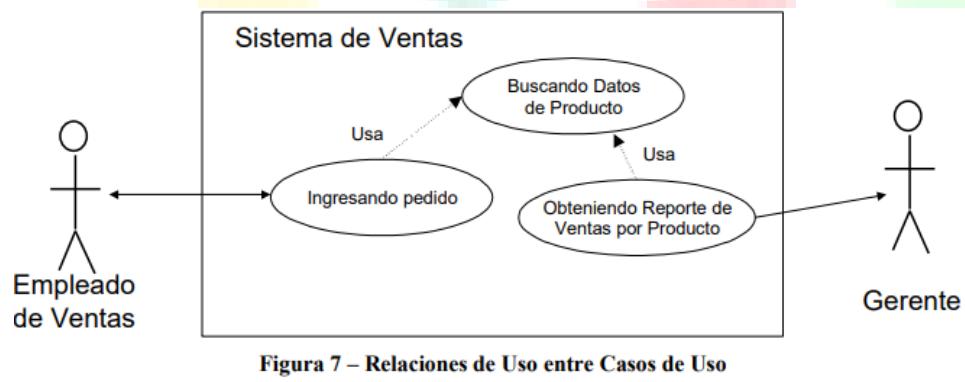


Figura 7 – Relaciones de Uso entre Casos de Uso

¿Qué pasa con la funcionalidad que es común a varios casos de uso, pero al mismo tiempo es opcional? Por ejemplo, pensemos en la impresión de un comprobante, algo que el usuario de un sistema puede o no hacer en distintos casos de uso. Si uno se guía por la funcionalidad común a varios casos, piensa que el caso de uso imprimiendo comprobante es usado por otros casos, pero si se guía por la optionalidad, piensa que extiende a otros casos. Como esto no queda claro a partir de la bibliografía, creemos conveniente que este tipo de situaciones se especifiquen como extensiones, ya que de esta forma podemos remarcar gráficamente la optionalidad de la relación.

Actores y Casos de Uso Abstractos

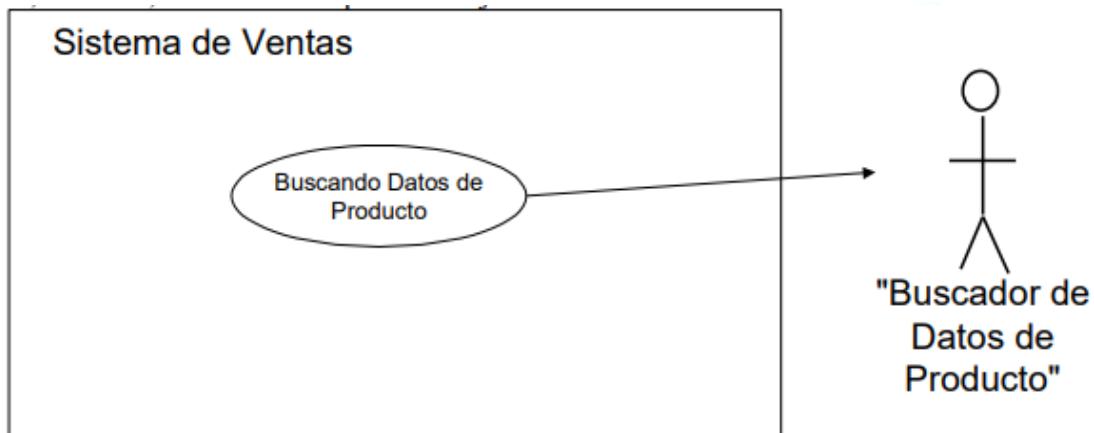


Figura 8 – El nombre de los actores abstractos suele no ser feliz

Por ejemplo, el caso de uso buscando datos de producto es accedido por muchos actores (el empleado de ventas que ingresa un pedido, el gerente que quiere obtener estadísticas por producto, el supervisor que quiere consultar la información de algún producto, etc.). Ahora bien, como el caso de uso nunca se ejecuta fuera del contexto de otro caso de uso, decimos que es un caso de uso abstracto. Lo llamamos abstracto porque no es implementable por sí mismo: sólo tiene sentido como parte de otros casos.

Cómo relacionar este actor abstracto con los actores concretos: los que sí existen en la realidad y ejecutan casos de uso concretos, como ingresando pedido y obteniendo estadísticas de ventas

Para esto podemos usar el concepto de herencia, uno de los conceptos básicos de la orientación a objetos. Como todos los actores concretos también ejecutan el caso buscando datos de producto, a través de la relación de uso, podemos decir que los actores concretos heredan al actor abstracto

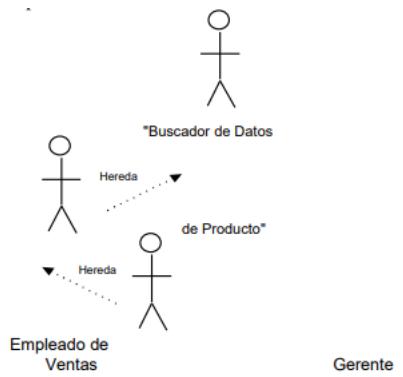


Figura 9 – Herencia de Actores

La relación de herencia no necesariamente implica la existencia de un caso abstracto. Puede ocurrir que un actor ejecute todos los casos que ejecuta otro actor, y algunos más. En nuestro sistema, el supervisor de ventas puede hacer todo lo que hace el empleado de ventas, pero además puede autorizar pedidos. En este caso, podemos decir que el Supervisor de Ventas hereda al Empleado de Ventas, aunque el Empleado de Ventas no sea un actor abstracto. De esta forma, toda la funcionalidad que está habilitada para el Empleado de Ventas también lo está para el Supervisor.

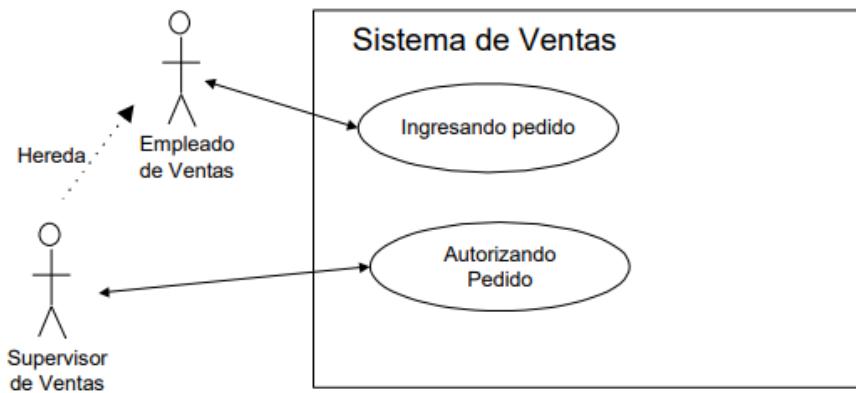


Figura 10 – Relación de herencia entre actores concretos

Diferencia entre casos de uso de trazo grueso o esenciales vs de implementacion o de trazo fino

Para aplicar los casos de uso a desarrollos incrementales, empezamos por identificar todos los casos de uso del sistema, sólo al nivel de su nombre. Una vez que los identificamos, los expresamos en “trazo grueso”, esto es:

- ignoramos detalles sobre la forma de la interacción entre el actor y el sistema
- Sólo incluimos las alternativas más relevantes, ignorando la mayoría de los errores que aparecen en el uso del sistema
- No entramos en detalle sobre las acciones que realiza el sistema cuando el usuario interactúa con él. Por ejemplo, si la empresa tuviera una política de descuentos para sus clientes, no es necesario especificar cómo es esa política: nos alcanza con saber que existe una y que debe ser tenida en cuenta.

Los casos de uso de trazo fino son aquellos que se especifican una vez que se ha tomado la decisión de implementarlos. En este momento debemos completar todos los detalles que dejamos pasar:

- A medida que vamos haciendo prototipos de las interfaces con los usuarios, incluimos detalles sobre la forma de la interfaz en la descripción del caso. Por ejemplo, podemos incluir detalles como: “el operador puede en cualquier momento pasar de la ventana de datos del cliente a la ventana de datos del pedido”. Si bien esto implica anticiparse al diseño, esto no es negativo, ya que es prácticamente imposible –y perjudicial– hablar con los usuarios siempre en términos de un sistema abstracto.
- Incluimos otras alternativas. En particular especificamos todos los errores o excepciones que provienen de requerimientos de los usuarios. Para esto debemos tener en cuenta que un sistema tiene dos tipos de errores o excepciones: las que provienen de las definiciones del negocio y las que provienen del procesamiento interno del sistema. Por ejemplo, pensemos en un requerimiento del tipo: “si un cliente hace un pedido por un monto mayor al autorizado, se debe rechazar el pedido”. Esta excepción es claramente un requerimiento, y debe ser incluida en las alternativas de los casos de uso. Por el contrario, una excepción del tipo: “Si el usuario ingresa una letra en el lugar del código del producto se le informa que el código de producto debe ser numérico” no debe ser incluida en esta etapa del análisis.
- Especificamos con más detalle el comportamiento interno del sistema. En nuestro ejemplo de los descuentos, deberíamos especificar cómo es esa política, en un nivel de detalle suficiente para luego poder diseñar una forma de implementarla dentro del sistema.

Casos de uso temporales

Los casos de uso tienen un actor que los inicia, y uno o más actores que participan de él. En muchos casos, el inicio de una determinada funcionalidad del sistema es provocado exclusivamente por el paso del tiempo. Supongamos que nuestro sistema de ventas debe generar en forma automática un conjunto de estadísticas para ser entregadas al directorio de la empresa el último día hábil de cada mes. En este caso, el paso del tiempo es el que inicia el caso de uso, y el directorio es el actor del sistema. Sin embargo, para expresar claramente que es el paso del tiempo el que inicia el caso, podemos incluir un símbolo representando un reloj en el gráfico de casos de uso, o usar una línea punteada en el borde del óvalo del caso.

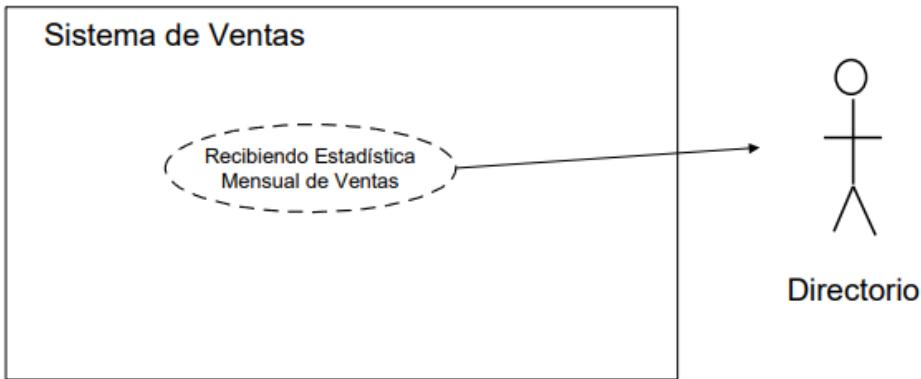


Figura 11 – Un caso de uso temporal

Es importante que cuando se especifican los casos de uso de trazo fino, se exprese claramente cuál es el momento del tiempo en el que se inicia el caso. Es común que se indiquen cosas como “una vez al mes” cuando se habla de casos iniciados por el paso del tiempo.

Modelo orientado a objetos

Deficiencias del análisis estructurado

- **Descomposición funcional :** No existe un mecanismo para comprobar si la especificación del sistema expresa con exactitud los requerimientos del sistema
- **Flujo de datos:** El análisis estructurado muestra cómo fluye la información a través del sistema.no es nuestra forma habitual de pensar
- **Modelo de datos** En la práctica, los modelos de procesos y de datos de un mismo sistema se parecen muy poco

Ventajas

- Dominio del problema
- Comunicación : Mejor comunicación entre el analista y el experto en el dominio del problema (cliente)
- Resistencia al cambio
- reutilización

Este modelo de análisis se describe al sistema utilizando tres tipos de objetos :

Objeto de interface : Modela el comportamiento e información dependiendo de la interface del sistema

Objeto entidad : Modela información en el sistema que se mantiene por un tiempo.

Objeto de control : Modela la funcionalidad que no es naturalmente asignada a cualquier objeto. Tal comportamiento consiste de operaciones sobre varios objetos entidad . haciendo algunas operaciones y retornando el resultado a un objeto interfaz .



Interfase



Entidad



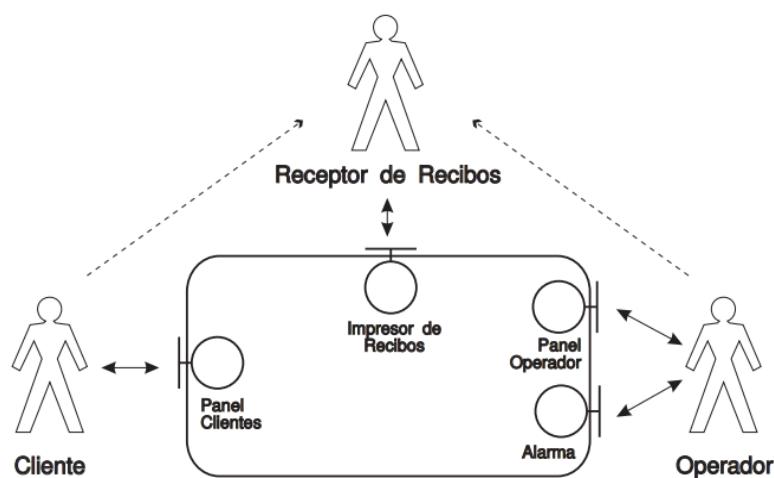
Control

Los casos de uso encapsulan información (identidad), comportamiento (control) y interfaz (funcionalidad)

Objetos de interface

- Toda la funcionalidad especificada en los casos de uso que es directamente dependiente del ambiente en donde se desenvolverá el sistema, es colocado en los objetos de interface. Es a través de estos objetos que los actores se comunican con el sistema
- Transcribir las acciones de los actores del sistema en eventos dentro del sistema y traducir los eventos del sistema que el actor está interesado en conocer, en información que pueda ser presentada al actor.
- Describen comunicaciones bidimensionales entre el sistema y los actores

En el ejemplo, si se utiliza la estrategia de obtener los objetos de interface desde los actores, se observa que cada actor concreto, Cliente y Operador, necesita su propio objeto de interface para el sistema. El Cliente necesita el Panel de Cliente, el cual contiene los botones y la ranura para insertar los elementos, y el Operador necesita su interface, Panel de Operador, para poder cambiar la información y para generar los informes diarios.



Descripción de los objetos de interface

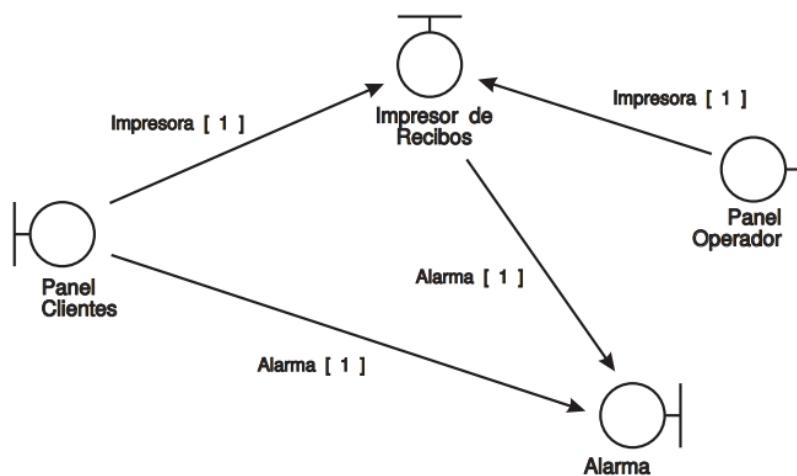
Panel de Cliente: Es la interface que usa el cliente para sus acciones con la maquina de reciclado. El panel tiene un botón de comienzo y otro para requerir el recibo. El panel, tiene también una ranura para el ingreso de los elementos, donde hay sensores para identificar qué clase de elementos colocó el cliente. Los sensores también miden el tamaño de los elementos.

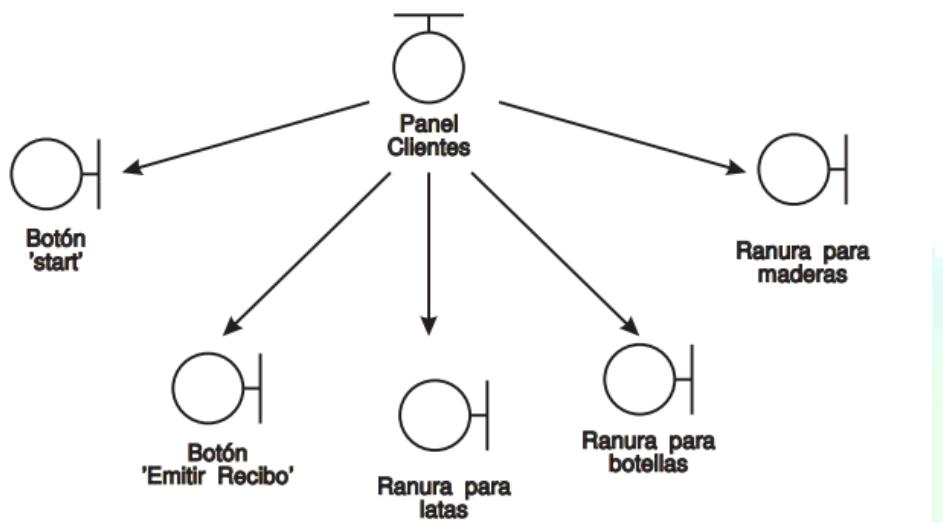
Panel del Operador: Es el panel que usa el operador para cambiar la información del sistema. Por medio de este panel, el operador puede ordenar el informe del día.

Dispositivo de Alarma: Control el parlante, el cual emite un sonido cuando fuera necesario.

Impresor de Recibos: Controla una pequeña impresora, la cual escribe texto sobre un papel. Luego que termina la impresión, el papel es cortado. Cuando el papel esta por terminarse se debe llamar al operador

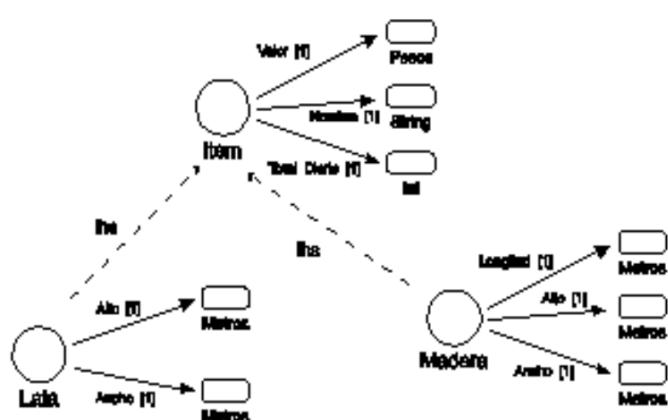
- Los objetos de interfaces no son enteramente independientes unos de otros, y deben conocerse mutuamente para resolver algunas tareas, esto se resuelve con la introducción de **asociaciones de conocimiento mutuo** entre objetos. Tales asociaciones son estáticas y significa que una instancia conoce la existencia de otra, no permitiendo intercambiar información entre ellas.
- Un objeto se puede asociar con muchas instancias de la misma clase por lo que a las asociaciones se les agrega cardinalidad indicando las asociaciones de conocimiento mutuo con flechas llenas y **cardinalidad** entre corchetes a continuación del nombre de la asociación. Las asociaciones son “solo” unidireccionales
- Un tipo especial de asociación de conocimiento mutuo son las **asociaciones de composición** que indica que un objeto esta compuesto por varios otros (aggregates). Este caso se presenta en las interfaces hombre-maquina (tal como sistemas de ventanas), el resultado es generalmente descrito con una estructura arborescente





Objeto entidades

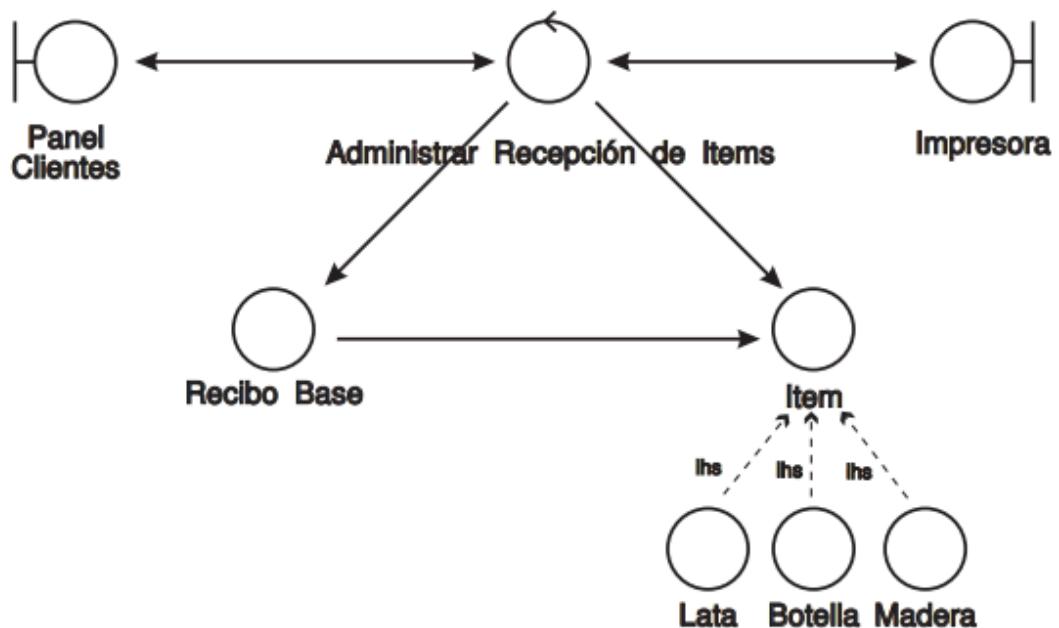
- Para modelar la información que el sistema debe manejar a lo largo del tiempo, se usan “objetos entidades”.
- Para almacenar información utilizamos “atributos”. Para cada tipo de entidades podemos encontrar varios atributos, cada atributo tiene un “tipo” que puede ser un tipo primitivo o un tipo de dato compuesto que sea especialmente definido. Un atributo se gráfica con un nombre y una cardinalidad indicando el tipo de atributo. Los atributos se grafican como rectángulos con bordes redondeados.. Los atributos deben ser usados en todos los tipos de objetos para la información que debe ser almacenada.



- Conjunto de operaciones que debe ofrecer un objeto entidad
 - Almacenar y recuperar información interna
 - Cursos que deben ser cambiados si el objeto entidad cambia
 - Creación y remoción de un objeto entidad

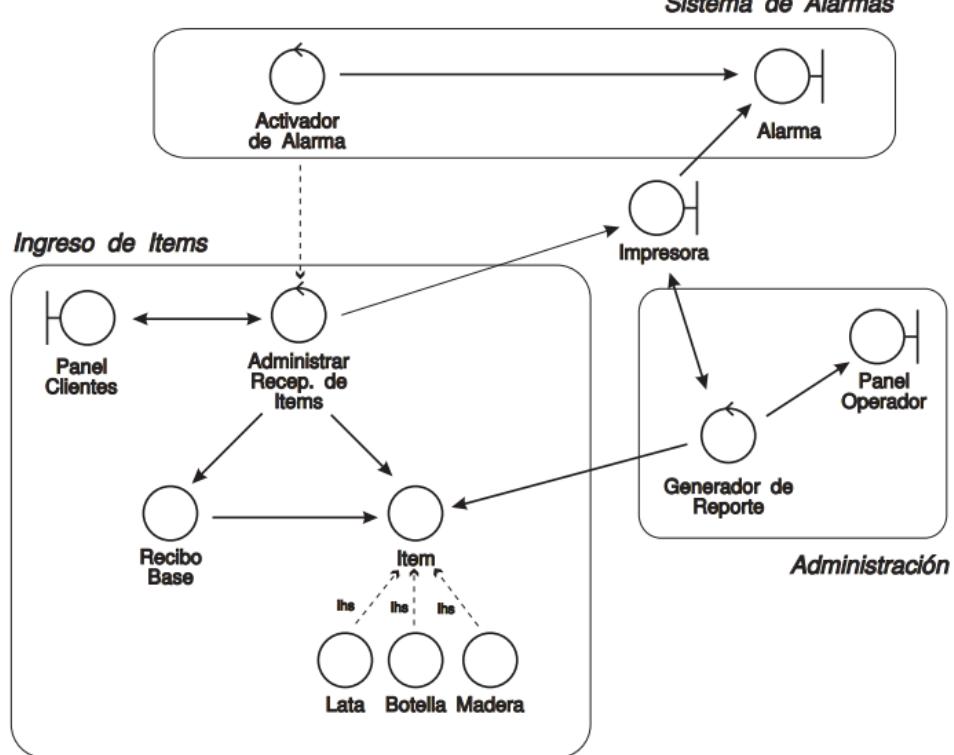
Objetos de control

- El comportamiento es difícil de ubicar en uno u otro tipo de objetos ya que no forma parte ni de la interface ni de la información a ser manejada
- Los objetos de control son los más efímeros de los objetos y existen durante la “ejecución” de un caso
- Cada caso de uso involucra objetos entidades y objetos de interface, el comportamiento que se mantiene luego de que los objetos han sido obtenidos es ubicado en los objetos de control.
- La funcionalidad típica encontrada en los objetos de control incluye: comportamiento relativo a transacciones, secuencias de control específicas de uno o varios casos de uso; o funcionalidad que oculta los objetos entidad de los objetos de interface.



“El caso comienza en ‘Panel de Clientes’ cuando el ‘Cliente’ presiona el botón de ‘start’, entonces, los sensores son activados. El ‘Cliente’ puede ahora depositar los elemento usando el ‘Panel de Clientes’. ‘Panel de Clientes’ dirá al objeto activo ‘Receptor de Elementos en deposito’ que un elemento ha sido depositado, como también las medidas del elemento. ‘Receptor de elementos en deposito’ entonces usa los objetos entidad ‘Lata’, ‘Botella’ o ‘Maderas’, junto con los valores medidos

del ‘Panel de Clientes’, para determinar que tipo de elemento ha sido recibido. El total diario del objeto recibido será enumerado en el objeto. Por medio del objeto entidad ‘Base de Recibo’, el ‘Receptor de Elementos en deposito’ mantiene traza de cuantos elementos del tipo corriente el cliente ha introducido. Después de retornar todos los elementos, el ‘Cliente’ solicita su recibo presionando el botón ‘Emitir Recibo’ en el ‘Panel de Clientes’. El ‘Receptor de elementos en deposito’ busca la información a ser impresa en el recibo. Una línea es impresa para cada elemento depositado, manteniendo el numero de elementos de cada tipo y el valor de retorno para ese numero. Finalmente, la suma de los valores es impresa en el recibo. La información del recibo es tomada de ‘Base de Recibo’. La información es impresa sobre el objeto de interface ‘Impresora de Recibos’ que dice cuando la impresión esta lista”



Preguntas Primer Parcial

1. a) Explica la curva IDEAL de fallos de software

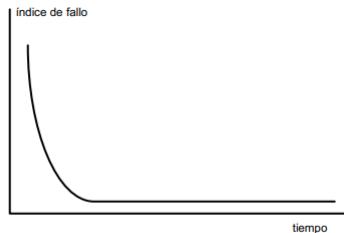


Figura 1.2. Curva ideal de fallos del software

b) Dibuje y explique la curva REAL de fallos de software

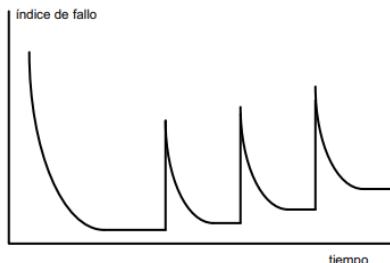


Figura 1.3. Curva real de fallos del software

2. En el contexto de la IR , caracterice los atributos de NO ambigüedad y verificabilidad de una SRS y explique qué relación hay entre ambos.

3. Explique que modela un caso de uso ? y explique que modela cada una de las asociaciones entre casos de uso?

Ejemplifica según corresponda

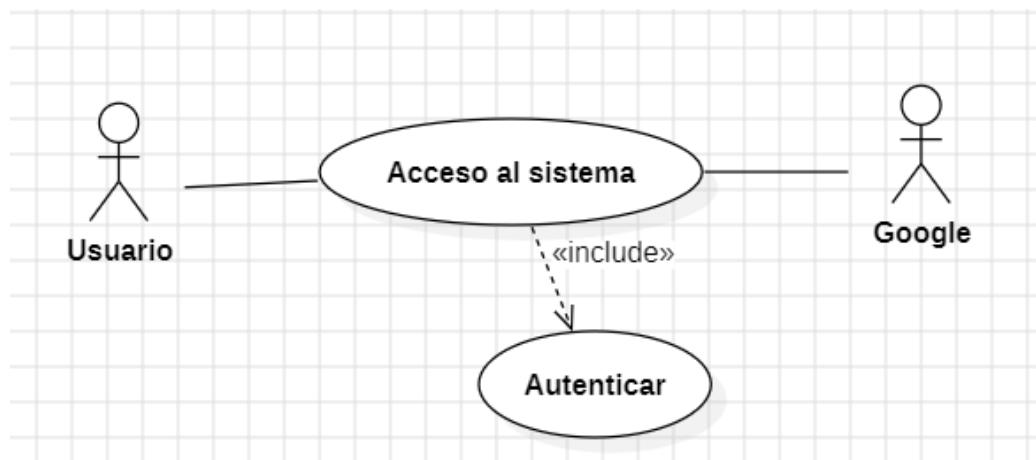
4. ¿Qué técnicas/Herramientas se emplean para el proceso de elicitation de Requerimientos ?Explique en qué circunstancias se recomienda usar cada una de ellas (o alguna combinación)

5. Práctica . Se ha reconocido la necesidad de que TASK MANAGER utilice Google Account como medio de identificación , acceso al sistema y control de sesión. De este modelo funcionalidad en cuestión quedará fuera del alcance del desarrollo del sistema , más siendo una necesidad por resolver.

a)Especificiar convenientemente el/los Requerimientos Funcionales necesarios

b) Modelar y especificar en forma completa el / los Casos de Uso según corresponda

Possible solution :



Otras preguntas de parciales :

1. ¿Qué es la ingeniería en requerimientos ?
2. Curva de fallos del hardware . Explicar

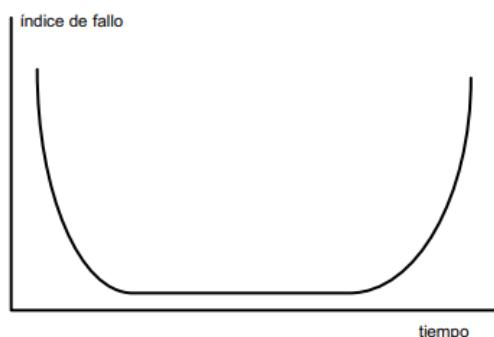


Figura. 1.1. Curva de fallos del hardware

3. ¿Cuáles son los problemas del software ?
4. ¿Cuáles son los atributos deseables de una SRS ? Explicar cada uno

.Los atributos de una SRS son :

Correcta : Significa que no debe haber ningún requerimiento en la implementación del sistema que no esté representado en la SRS y viceversa . Esto asegura que todos los aspectos funcionales y no funcionales del sistema estén claramente definidos en el documento de requisitos

No ambigua : Especificar claramente los requisitos de manera que no haya interpretaciones múltiples . Cada requerimiento debe ser cuantificable y tener una única interpretación para evitar discrepancias en la implementación.

Completa : Garantiza que todos los requerimientos significativos del software están incluidos en la SRS .

Además de cubrir todas las funciones requeridas , debe contener todas las entradas necesarias para el sistema , las salidas esperadas y todas las referencias necesarias como figuras, tablas diagramas y definiciones de términos.

Consistente : Asegura que no haya conflictos entre los diferentes requisitos especificados en la SRS. Los requisitos deben ser coherentes entre sí y no pueden entrar en conflicto directo en términos de funcionalidad o comportamiento esperado del sistema.

Rankear : Permite que los requisitos sean clasificados por su importancia relativa para los stakeholders . Esto ayuda a priorizar qué requisitos deben ser implementados primero o recibir más atención durante el desarrollo del software

Verificable : Implica que cada requerimiento debe ser verificable , es decir , debe poder establecer un método para determinar si el sistema cumple con ese requisito o no . Esto generalmente implica criterios de aceptación claros y medibles para cada requisito

Modificable : Se refiere a la capacidad de realizar cambios en los requisitos sin perder la integridad, completitud y consistencia de la SRS. Debe ser posible actualizar los requisitos cuando sea necesario para adaptarse a cambios en los requisitos del negocio o nuevos descubrimientos durante el desarrollo

rastreable : Cada requerimiento debe ser rastreable tanto hacia adelante como hacia atrás. Esto significa que se debe poder seguir desde la SRS hasta su implementación en el sistema (trazabilidad hacia adelante) y desde el sistema hasta su origen en la SRS (trazabilidad hacia atrás). Esto facilita la gestión de cambios, la verificación y la validación de los requisitos a lo largo del ciclo de vida del desarrollo de software.

5. ¿Cuál es el proceso de análisis?

El **proceso de análisis** es el desarrollo del software es una etapa relevante en el proceso de Desarrollo en la que se recopilan , comprenden y documentan los requisitos y objetivos del sistema a desarrollar

1. Se identifican los requerimientos : El análisis comienza identificando y documentando los requisitos del software . Esto implica la recopilación de información de diversas fuentes , como usuarios, clientes y otros interesados . Los requisitos pueden ser funcionales (que debe hacer el software) o no funcionales (restricciones y características)

2. Análisis de requerimientos : Estos requerimientos se analizan para que sean coherentes , no ambiguos , completos y verificables .

3 . Modelado : Se pueden hacer con casos de uso , diagrama de flujo de datos u otro modelo específico según la metodología utilizada

4. Validación de requerimientos : Estos se validan con las partes interesadas para garantizar que reflejen las necesidades del sistema .

5. Documentación de requerimientos : Se documentaran en un formato claro y comprensible . Esto puede incluir especificaciones de requerimientos (SRS)

6. Diferencias entre Casos de uso y eventos

Los eventos se centran en describir que hace el sistema cuando el evento ocurre mientras que los casos de uso se centran en describir como es el diálogo entre el usuario y el sistema.

Los eventos reciben una entrada , se procesa y se genera una salida en cambio los casos de uso se prolonga a lo largo del tiempo mientras dure la interacción del usuario con el sistema

7. Diferenciar un actor de un usuario

El usuario es la persona que está usando el sistema , mientras que los actores representan a los usuarios de los casos , un actor representa un cierto rol que un usuario puede tener

8. Explicar en qué consiste el proceso de elicitation , especificación y validación

El proceso de elicitation consiste en adquirir todo el conocimiento necesario para hacer los requerimientos de un determinado problema .Su objetivo es entender el dominio del problema y tener conocimiento de este . Esto se hace mediante técnicas de elicitation

El proceso de especificación es un contrato entre el cliente y el desarrollador .Este define el comportamiento funcional deseado del software y otras propiedades de este . No indica COMO se va a lograr lo que se enuncia.

El proceso de validación es el proceso que certifica que lo que se resolvio es correcto

9. Explicar qué son los objetos entidad, control y interfaz



Interfase



Entidad



Control

El objeto control : El comportamiento que es difícil ubicarlo en uno u otro tipo de objeto ya que no forma parte de la interface ni de la información que se maneja

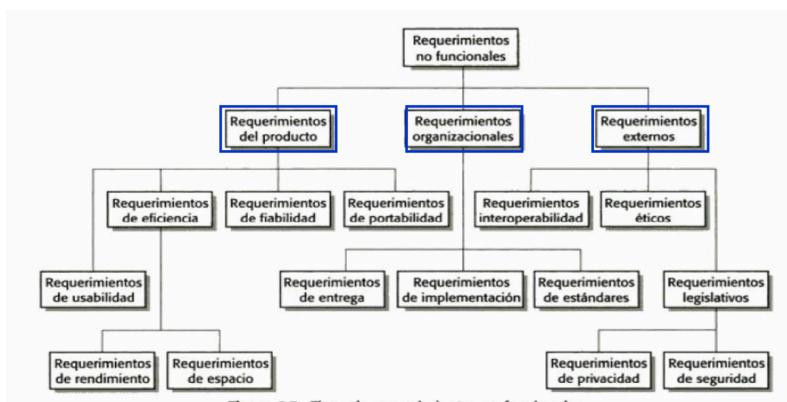
Este objeto generalmente se refiere a una parte del software que gestiona el flujo de la aplicación y coordina las interacciones entre otros objetos . Se encarga de recibir las entradas del usuario o de otros sistemas, procesarlas y coordinar las salidas correspondientes.

El objeto entidades : Este objeto modela la información que el sistema debe manejar a o largo del tiempo. Representan conceptos del mundo real que tienen relevancia para la aplicación.Su principal función es almacenar y estructurar la información que el sistema necesita gestionar

El objeto Interface: Este objeto se refiere a la capa de presentación o la parte del sistema que permite la interacción con los usuarios o con otros sistemas externos.Incluye elementos visuales y de interacción como formularios, botones, menús, etc., que facilitan la comunicación entre el usuario y la aplicación

10. Explicar que es un requerimiento no funcional . Ejemplificar cada uno ¿Cómo se identifican?

Los requerimientos no funcionales son las restricciones del sistema . Estas se deben redactar de manera cuantitativa para que se puedan probar de manera objetiva.



- ✓ Performance
- ✓ Precisión
- ✓ Seguridad
- ✓ Amigabilidad
- ✓ Mantenibilidad
- ✓ Portabilidad
- ✓ Interoperabilidad
- ✓ Confiabilidad
- ✓ Restricciones particulares del dominio
- ✓ Restricciones particulares de la tecnología de implementación
- ✓ Restricciones Operacionales
- ✓ Restricciones físicas

Requerimiento del producto

8.1 La interfaz de usuario del LIBSYS se implementará como HTML simple sin marcos o applets Java.

Requerimiento organizacional

9.3.2 El proceso de desarrollo del sistema y los documentos a entregar deberán ajustarse al proceso y a los productos a entregar definidos en XYZCo-SP-STAN-95.

Requerimiento externo

10.6 El sistema no deberá revelar al personal de la biblioteca que lo utilice ninguna información personal de los usuarios del sistema aparte de su nombre y número de referencia de la biblioteca.

Se deben redactar los requerimientos de manera cuantitativa

Estos se dividen en tres grandes grupos . Requerimientos del producto , requerimientos organizacionales y requerimientos externos . Dentro de estos se pueden observar algunos requerimientos como :

Performance : Se refiere a cómo responde el sistema bajo diferentes cargas de trabajo y la eficiencia con la que maneja recursos como memoria y procesamiento .

Por ejemplo : El sistema debe ser capaz de procesar al menos 1000 transacciones por segundo durante las horas pico de uso

Precisión : Define la exactitud de los resultados generados por el sistema y la mínima tolerancia a errores

Por ejemplo : El sistema de calculo de impuestos debe calcular los montos con un margen de error maximo al 1%

Seguridad : Se refiere a la protección de los datos sensibles y la prevención de acceso no autorizados

Por ejemplo : Todos los accesos al sistema deben ser autenticados mediante un protocolo de autenticación de dos factores

Amigabilidad : Define la facilidad de uso y la experiencia de usuario final al interactuar con el sistema

Por ejemplo : La interfaz de usuario debe ser intuitiva y amigable permitiendo a los usuarios completar tareas comunes sin necesidad de formación extensa

Mantenibilidad : Se refiere a la facilidad con la que el sistema puede ser mantenido y evolucionado a lo largo del tiempo

Por ejemplo : El código del sistema debe estar bien documentado y seguir estándares de codificación que faciliten la incorporación de nuevas funcionalidades

Portabilidad : Define la capacidad del sistema para ser ejecutado por diferentes aplicaciones y entornos

Por ejemplo : La aplicación debe ser compatible con los sistemas operativos Windows, macOS y Linux.

Interoperabilidad : Define la capacidad del sistema para interactuar con otros sistemas externos de manera eficiente y efectiva.

Por ejemplo : El sistema de gestión de inventarios debe integrarse sin problemas con los sistemas de contabilidad existentes en la empresa.

Confiabilidad : Se refiere a la capacidad del sistema para funcionar de manera consistente y predecible bajo diversas circunstancias.

Por ejemplo: El sistema de reservas de vuelos debe estar disponible el 99.9% del tiempo programado.

Restricciones Particulares del Dominio: Son restricciones específicas impuestas por el area o sector en el que se utiliza el sistema

Por ejemplo : Un sistema de gestión de datos médicos debe cumplir con las regulaciones HIPAA en cuanto a privacidad y seguridad de la información.

Restricciones Particulares de la Tecnología de Implementación: Son limitaciones impuestas por las tecnologías específicas que se utilizan para desarrollar el sistema.

Por ejemplo: El sistema debe estar desarrollado utilizando el framework Django en Python debido a las políticas de tecnología de la empresa

Restricciones Generales: Son restricciones que afectan al sistema en su conjunto, no específicamente a su dominio o tecnología.

Por ejemplo: El sistema debe cumplir con los estándares ISO 9001 de calidad de procesos y gestión.

Restricciones Físicas: Se refieren a limitaciones físicas que el sistema debe considerar, como el espacio, el consumo de energía, entre otros.

Por ejemplo: El sistema embebido en un dispositivo IoT debe operar en temperaturas entre -10°C y 50°C sin degradación en su rendimiento.

11. Describir qué es Proyecto Producto Proceso Alcance

Alcance : Es básicamente todas la funcionalidad del sistema o los requerimientos

Proceso : Un proceso es un conjunto de actividades, acciones y tareas que tiene un propósito específico . En este se definen roles que intervienen distintas funciones a lo largo del mismo . Este tiene una entrada , un procedimiento y una salida .

Proyecto : Un proyecto es por el cual se construye o elabora un producto con ciertas personas que cumplen un rol determinado . Este proyecto tiene como objetivo desarrollar o no un sistema . Este objetivo en si es llegar a la realización del producto . Este objetivo tiene que ser SMART : Específico , medible , alcanzable , realista , de duración limitada .

Entonces es un ámbito donde ciertas personas que elaboraron un rol diferente tienen como objetivo desarrollar un producto mediante un proceso (realizar actividades) divididas en diferentes etapas o manejada por un proceso de metodológico

Producto : Es el resultado de un proyecto o un proceso. Este tiene que satisfacer las necesidades específicas de quien lo crea o quien lo usa/consume

Su relación es que un proyecto se dedica a crear un producto de software mediante un conjunto de actividades organizadas y fases definidas. El producto de software final es el resultado tangible del proyecto y cumple con las características y especificaciones establecidas durante el proceso de desarrollo. Este proceso es la manera en que se llevan a cabo las tareas necesarias para transformar los requisitos del software en el producto deseado.

12. ¿Qué es la ingeniería de software ?



Gestión de riesgos

La gestión de riesgos en la Ingeniería de Software es un proceso relevante que se centra en identificar, evaluar y mitigar los riesgos que pueden afectar el éxito de un proyecto de software. Los riesgos son eventos o condiciones inciertas que pueden tener un impacto negativo en el proyecto si se materializan.

En este contexto, la gestión de riesgos se convierte en una parte integral de la planificación y ejecución de proyectos de software. Al identificar, evaluar y mitigar riesgos de manera proactiva, los equipos pueden minimizar el impacto de eventos adversos y mejorar la calidad del software entregado. Las mejores prácticas incluyen involucrar a todos los stakeholders en el proceso y revisar periódicamente el plan de gestión de riesgos para adaptarse a cambios y nuevas amenazas.

- La Gestión de los Riesgos del Proyecto incluye los procesos relacionados con la planificación de la gestión de riesgos, la identificación y el análisis de riesgos, las respuestas a los riesgos, y el seguimiento y control de riesgos de un proyecto
- La mayoría de estos procesos se actualizan durante el proyecto
- Los objetivos de la Gestión de los Riesgos del Proyecto son aumentar la probabilidad y el impacto de los eventos positivos, y disminuir la probabilidad y el impacto de los eventos adversos para el proyecto.

Procesos de la Gestión de Riesgos

1. **Planificación de la Gestión de Riesgos:** decidir cómo enfocar, planificar y ejecutar las actividades de gestión de riesgos para un proyecto.
2. **Identificación de Riesgos:** determinar qué riesgos pueden afectar al proyecto y documentar sus características
3. **Análisis Cualitativo de Riesgos:** priorizar los riesgos para realizar otros análisis o acciones posteriores, evaluando y combinando su probabilidad de ocurrencia y su impacto.
4. **Análisis Cuantitativo de Riesgos:** analizar numéricamente el efecto de los riesgos identificados en los objetivos generales del proyecto.
5. **Planificación de la Respuesta a los Riesgos:** desarrollar opciones y acciones para mejorar las oportunidades y reducir las amenazas a los objetivos del proyecto
6. **Seguimiento y Control de Riesgos:** realizar el seguimiento de los riesgos identificados, supervisar los riesgos residuales, identificar nuevos riesgos, ejecutar planes de respuesta a los riesgos y evaluar su efectividad a lo largo del ciclo de vida del proyecto.

Riesgo : Probabilidad de que ocurra un evento donde su impacto puede ser positivo o negativo

Riesgos conocidos son aquellos que han sido identificados y analizados, y es posible planificar

Los riesgos desconocidos no pueden gestionarse de forma proactiva, y una respuesta prudente del equipo del proyecto puede ser asignar una contingencia general contra dichos riesgos, así como contra los riesgos conocidos para los cuales quizás no sea rentable o posible desarrollar respuestas proactivas.

Estrategias frente al riesgo

Estrategias REACTIVAS

Método:

- Evaluar las consecuencias del riesgo cuando este ya se ha producido (ya no es un riesgo)
- Actuar en consecuencia Consecuencias de una estrategia reactiva
- Apagado de incendios Gabinetes de crisis
- Se pone el proyecto en peligro

Estrategias PROACTIVAS

Método

- Evaluación previa y sistemática de riesgos
- Evaluación de consecuencias • Plan para evitar y minimizar las consecuencias
- Plan de contingencias Consecuencias
- Evasión del riesgo
- Menor tiempo de reacción
- Justificación frente a los superiores

Clasificación del Riesgo

- Riesgos del proyecto
- Riesgos técnicos
- Riesgos del negocio
 - De mercado
 - De estrategia
 - De ventas
 - De gestión
 - De presupuesto

Identificación de riesgos

- **Asociados con el tamaño del producto**
 - Tamaño estimado del proyecto (LOC/PF)
 - Confianza en la estimación
 - Número de programas, archivos y transacciones
 - Tamaño relativo al resto de proyectos
 - Tamaño de la base de datos

- Número de usuarios
- Número de cambios de requerimientos previstos antes y después de la entrega
- Cantidad de software reutilizado

- **Impacto en la organización**

- Efecto del producto en la cifra de ventas
- Visibilidad desde la dirección de la organización
- Fecha límite de entrega razonable
- Número de clientes que usarán el producto
- Número de productos con los que deberá interaccionar
- Sofisticación del usuario final
- Cantidad y calidad de la documentación a entregar al cliente
- Límites legales y gubernamentales
- Costes asociados al retraso en la entrega
- Costes asociados a errores en el producto



- **Relacionados con el cliente**

- Hay experiencias anteriores con dicho cliente
- Tiene una idea clara de lo que precisa
- Está dispuesto a dedicar tiempo en la especificación formal de requerimientos
- Está dispuesto a relacionarse de forma ágil con el equipo de desarrollo
- Está dispuesto a participar en la revisiones
- Es un usuario experto
- Dejará trabajar al equipo de desarrollo sin dar consejos de experto informático
- Entiende el ciclo de vida de una aplicación



- **Riesgos del proceso de producción**

- Hay una política clara de normalización y seguimiento de una metodología
- Existe una metodología escrita para el proyecto
- Se ha utilizado en otros proyectos
- Están los gestores y desarrolladores formados
- Conoce todo el mundo los standards
- Existen plantillas y modelos para todos los documentos resultado del proceso
- Se aplican revisiones técnicas de la especificación de requerimientos diseño y codificación
- Se aplican revisiones técnicas de los procedimientos de revisión y prueba
- Se documentan los resultados de las revisiones técnicas
- Hay algún mecanismos para asegurar que un proceso de desarrollo sigue los standards
- Se realiza gestión de la configuración
- Hay mecanismos para controlar los cambios en los requerimientos que tienen impacto en el software
- Se documenta suficientemente cada subcontrato
- Se ha habilitado y se siguen mecanismos de seguimiento y evaluación técnica de cada subcontrato

- **Riesgos tecnológicos (Technical risks)**
 - Se trata de una tecnología nueva en la organización
 - Se requieren nuevos algoritmos o tecnología de I/O
 - Se debe interactuar con hardware nuevo
 - Se debe interactuar con software que no ha sido probado
 - Se debe interactuar con un B.D. cuya funcionalidad y rendimiento no ha sido probada
 - Es requerido un interface de usuario especializado
 - Se necesitan componentes de programa radicalmente diferentes a los hasta ahora desarrollado
- **Riesgos tecnológicos (cont.)**
 - Se deben utilizar métodos nuevos de análisis, diseño o pruebas
 - Se deben utilizar métodos de desarrollo no habituales, tales como métodos formales, Inteligencia Artificial o redes neuronales
 - Se aplican requisitos de rendimiento especialmente estrictos
 - Existen dudas de que el proyecto sea realizable
- **Respecto al entorno de desarrollo**
 - Hay herramientas de gestión de proyectos
 - Hay herramientas de gestión del proceso de desarrollo
 - Hay herramientas de análisis y diseño
 - Hay generadores de código apropiados para la aplicación
 - Hay herramientas de prueba apropiadas
 - Hay herramientas de gestión de configuración apropiadas
- **Respecto al entorno de desarrollo (cont.)**
 - Se hace uso de una base de datos o repositorio centralizado
 - Están todas las herramientas de desarrollo integradas
 - Se ha proporcionado formación a todos los miembros del equipo de desarrollo
 - Hay expertos a los cuales solicitar ayuda acerca de las herramientas
 - Hay ayuda en línea y documentación disponible
- **Asociados al equipo y la experiencia**
 - Es el mejor personal disponible
 - Tienen los miembros las técnicas apropiadas
 - Hay suficiente gente disponible
 - Está el personal comprometido en toda la duración del proyecto
 - Habrá parte del personal dedicado solamente en parte al proyecto
 - Tiene el personal las expectativas correctas del trabajo
 - Tiene el personal la necesaria formación

- Puede la rotación del personal perjudicar el proceso de desarrollo

Estimación de Riesgos

- Creación de una tabla de riesgos
- Ordenación y filtrado
 - Ordenación por probabilidad y prioridad
 - Despreciar riesgos poco probables y los medicamentos probables con poco impacto
- Factores que definen el impacto de la ocurrencia de un riesgo
 - Alcance del mismo: cuán serio y cuánto del proyecto se ve afectado
 - Temporalización de los efectos, cuándo y por cuánto tiempo

Gestión, monitoreo y Mitigación de Riesgos (RM)

Objetivo: Marcar las estrategias y formas de actuar del equipo de trabajo frente a los riesgos:

- Cómo mitigarlos
 1. Definir las estrategias necesarias para evitar que el riesgo se convierta en un problema
 2. Tomar las medidas encaminadas para que, aún cuando se produzca, se minimicen sus efectos
- Como monitorearlos
 1. Definir los indicadores que influyen en la probabilidad de que el riesgo se produzca
 2. Monitorear periódicamente dichos factores
 3. Monitorear la efectividad real de las acciones encaminadas a evitar el riesgo
- Como gestionarlos y plan de contingencia
 1. Se asume que la evitación y la monitoreo han fallado y el riesgo se ha producido
 2. Se definen las estrategias y acciones a tomar para evitar que los efectos se minimicen
 3. Nunca se podrá reducir a cero el coste del plan de contingencia.
 4. Dicho plan puede implicar unos costes en sí mismo, por lo cual se ha de valorar el beneficio que se espera obtener de éste

Para tener éxito, la organización debe estar comprometida a tratar la gestión de riesgos de forma proactiva y consistente durante todo el proyecto.

Dificultades accidentales y dificultades esenciales

Dificultades accidentales (como ?) : Son problemas relacionados con la implementación y la práctica del desarrollo del software .

Por ejemplo : lenguajes de programación , metodologías , cómo formó el equipo de trabajo , con que precisión fue escrita la srs

Dificultades esenciales (que ?): Son los problemas fundamentales que están en la esencia de la naturaleza del software . Están relacionados con la complejidad, la gestión del cambio , la comunicación entre diferentes partes del software y la comprensión de cómo diferentes partes del sistema interactúen

Diseño

Introducción Definición:

"Diseño es el proceso de aplicar distintas técnicas y principios con el propósito de definir un dispositivo, proceso, o sistema, con los suficientes detalles como para permitir su realización física" (E.S.Taylor, An Interim Report on Engineering Design, Massachusetts Institute of Technology)

Definición: "Diseño es el proceso de aplicar distintas técnicas y principios con el propósito de definir un dispositivo, proceso, o sistema, con los suficientes detalles como para permitir su realización física"

El objetivo del diseñador es producir un modelo de una entidad que se construirá más adelante. El proceso por el cual se desarrolla el modelo combina: la intuición y los criterios en base a la experiencia de construir entidades similares, un conjunto de principios y/o heurísticas que guían la forma en la que se desarrolla el modelo, un conjunto de criterios que permiten discernir sobre calidad y un proceso de iteración que conduce finalmente a una representación del diseño final.

Qué es diseño definición:

"Diseño es el proceso de decidir que componentes utilizar, bajo qué condiciones, y la interconexión entre los mismos, para solucionar un problema bien especificado".

El diseño es una actividad que comienza cuando el analista de sistemas ha producido un conjunto de requerimientos funcionales lógicos para un sistema, y finaliza cuando el diseñador ha especificado los componentes del sistema y las relaciones entre los mismos. Muchas veces, analista y diseñador son la misma persona, sin embargo es necesario que se realice un cambio de enfoque mental al pasar de una etapa a la otra. Al abordar la etapa de diseño, la persona debe quitarse el sombrero de analista y colocarse el sombrero de diseñador. Una vez que se han establecido los requisitos del software (en el análisis), el diseño del software es la primera de tres actividades técnicas: diseño, codificación, y prueba. Cada actividad transforma la información de forma que finalmente se obtiene un software para computadora válido. Los requisitos del sistema, establecidos mediante los modelos de información, funcional y de comportamiento, alimentan el proceso del diseño.

Mediante alguna metodología se realiza el diseño de arquitectura, procedimental, y de datos.

El **diseño de datos** transforma el modelo del campo de información, creado durante el análisis, en las estructuras de datos que se van a requerir para implementar el software.

El **diseño de arquitectura** define las relaciones entre los principales elementos estructurales del programa. El objetivo principal del diseño de arquitectura es desarrollar

una estructura de programa modular y representar las relaciones de control entre los módulos.

El **diseño procedimental** transforma los elementos estructurales en una descripción procedural del software. El diseño procedimental se realiza después de que se ha establecido la estructura del programa y de los datos. Define los algoritmos de procesamiento necesarios.

Es aquí donde se toman decisiones que afectarán finalmente al éxito de la implementación del programa y, con igual importancia, a la facilidad de mantenimiento que tendrá el software. Estas decisiones se llevan a cabo durante el diseño del software, haciendo que sea un paso fundamental de la fase de desarrollo. La importancia del diseño del software se puede sentar con una única palabra: calidad.

El diseño es el proceso en el que se asienta la calidad del desarrollo del software. El diseño produce las representaciones del software de las que puede evaluarse su calidad. El diseño sirve como base para todas las posteriores etapas del desarrollo y de la fase de mantenimiento. Sin diseño nos arriesgamos a construir un sistema inestable, un sistema que falle cuando se realicen pequeños cambios, un sistema que pueda ser difícil de probar, un sistema cuya calidad no pueda ser evaluada hasta más adelante en el proceso de ingeniería de software, cuando quede poco tiempo y se haya gastado ya mucho dinero.

Pienso la solución en el entorno real de ejecución, en términos del ambiente real de implantación, en qué lugar y en qué condición va ejecutarse.

Por ejemplo, definir la arquitectura, tecnologías, frameworks, hardware, conexiones, tipo de aplicación. Pensamos cómo vamos a modelar:

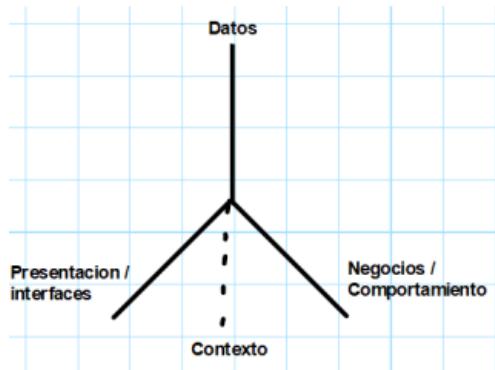
Por ejemplo, definir la arquitectura, tecnologías, frameworks, hardware, conexiones, tipo de aplicación. Pensamos cómo vamos a modelar:

- Los datos
- La arquitectura (procedimientos y estructura)
- Interfaces con otros sistemas (maquina - maquina)
- Interfaces de usuario (maquina - hombre)
- Las distintas capas

Presentación → Interfaces Permite la interacción con el sistema

Negocios → Comportamiento Proporciona la lógica necesaria para la interacción con los datos.

Datos



Objetivos del diseño

Los objetivos del diseño constituyen los principios que debe perseguir el ingeniero de software durante la etapa de Diseño en el desarrollo de soluciones informáticas cuyo cumplimiento está orientado a mejorar la calidad del sistema final. El Diseño determina cómo el sistema debe funcionar, no basta con que funcione, si no que lo haga de la mejor manera, cumpliendo con las expectativas de las partes interesadas. Si bien las mismas son subjetivas, se definen una serie de parámetros por convención que permiten decidir cuando un diseño es bueno objetivamente. Estos parámetros son los denominados objetivos del diseño e incluyen:

- **Eficiencia:** hace referencia a la optimización o uso inteligente de los recursos del sistema de computación por parte del software. Estos recursos consisten en la memoria, procesador, almacenamiento secundario, tiempo de periféricos de entrada salida, tiempo de líneas de teleproceso, tiempo de personal, etc. Se trata de solucionar el problema con el menor consumo de recursos posible.
- **Mantenibilidad:** un sistema se define como mantenible si permite la detección, análisis, rediseño, y corrección de errores fácilmente. De este modo, el tiempo entre fallas debe ser muy superior al tiempo que se tarda en corregir cada una de estas fallas para que un sistema sea muy mantenible. Es por eso que también es un concepto que se relaciona con la confiabilidad del software.
- **Modificabilidad:** se refiere a la capacidad del sistema de adaptarse fácilmente a cambios. Así, consiste en el costo de mantener un sistema viable en condiciones de cambio de requerimientos. A diferencia del mantenimiento, no se trata de la corrección de errores, si no de la posibilidad de agregar nuevas partes o realizar modificaciones a partes del sistema o con facilidad.
- **Flexibilidad:** este objetivo se funda principalmente en la posibilidad del sistema de adaptarse o realizar variaciones sobre una misma temática, sin que sea necesario hacer modificaciones. Así, un diseño flexible permite personalizaciones o ajustes rápidos sin necesidad de alterar significativamente la estructura base del sistema.
- **Generalidad:** se basa en el alcance sobre un determinado tema. Es decir, busca la creación de soluciones que no sean demasiado específicas para un solo problema o contexto. En definitiva, un diseño generalizado puede aplicarse a una variedad de casos similares, lo que aumenta su reutilización y reduce la necesidad de tener que desarrollar soluciones desde cero cada vez que se enfrenta un nuevo problema.

- **Utilidad:** consiste en la facilidad o amabilidad del sistema para que un usuario pueda satisfacer sus necesidades de una forma sencilla. Un sistema cuyo diseño de interfaces sea demasiado "duro" o difícil de usar tiende a ser resistido por los usuarios.

GUMMEF

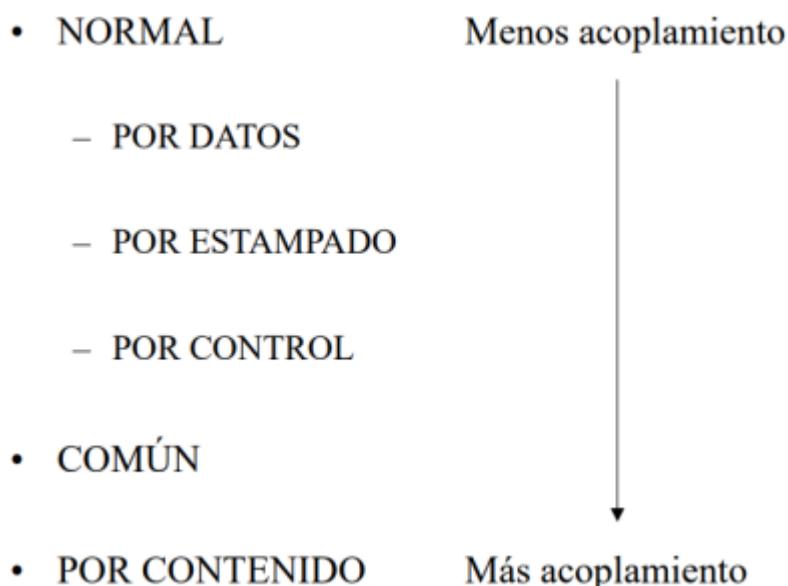
Criterios de Diseño

Un buen diseño es esencial para asegurar que un proyecto cumpla con su propósito de manera efectiva. Para que un diseño se considere bueno, debe cumplir ciertos **criterios de calidad**.

El acoplamiento

Evalúa la relación entre módulos. El acoplamiento se refiere al grado de dependencia entre módulos. Un bajo acoplamiento permite que los módulos sean modificados o reemplazados con poco o ningún impacto sobre otros módulos.

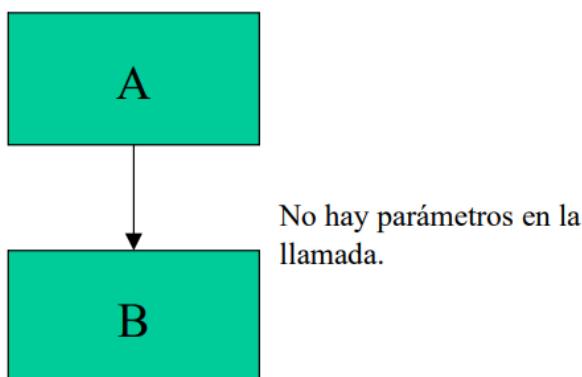
Esta relación entre módulos puede ser:



Acoplamiento normal

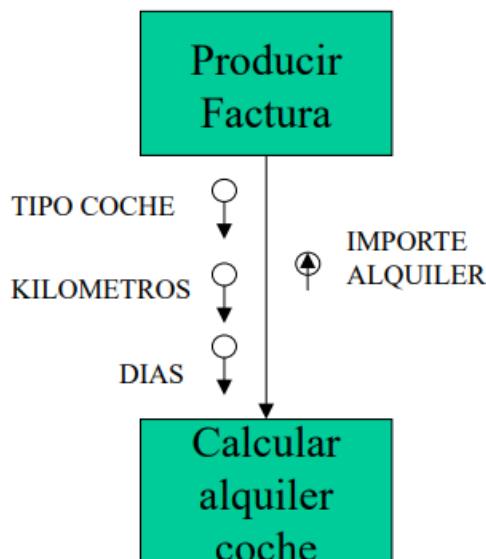
Se dice que dos módulos “A” y “B” están acoplados normalmente si:

- A invoca a B
- B realiza su función y retorna el control a A
- Toda la información que comparten la realizan a través de los parámetros presentes en la llamada.



Acoplamiento normal por datos

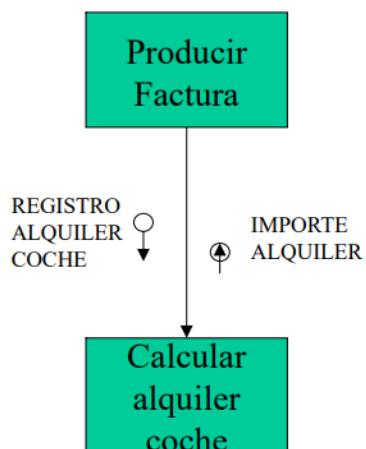
Dos módulos están acoplados por datos si están acoplados normalmente y además todos los parámetros que se intercambian son datos elementales sin estructura interna (tipos básicos).



Los datos que se pasan en este ejemplo son básicos

Acoplamiento normal por estampado

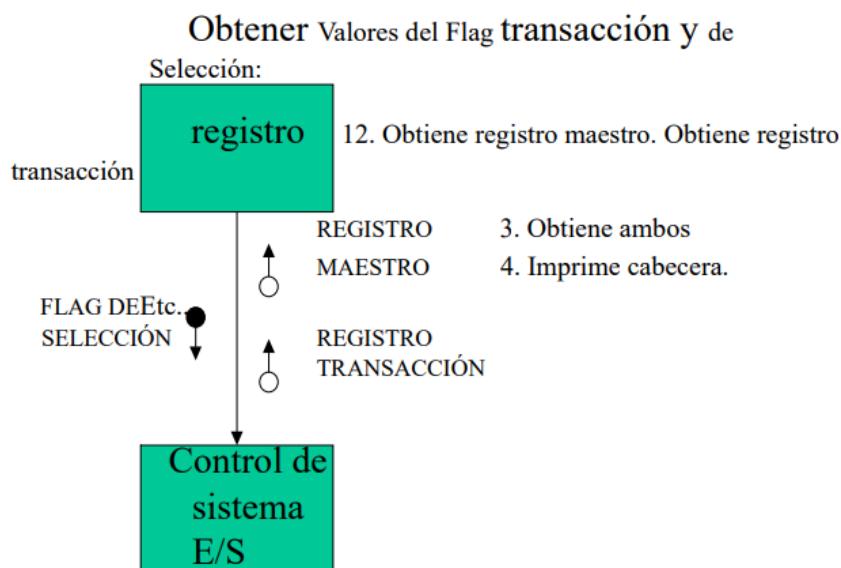
Dos módulos acoplados normalmente lo están por estampado si uno le pasa al otro un dato compuesto (con estructura interna).



Los datos que se pasan en este ejemplo son compuestos

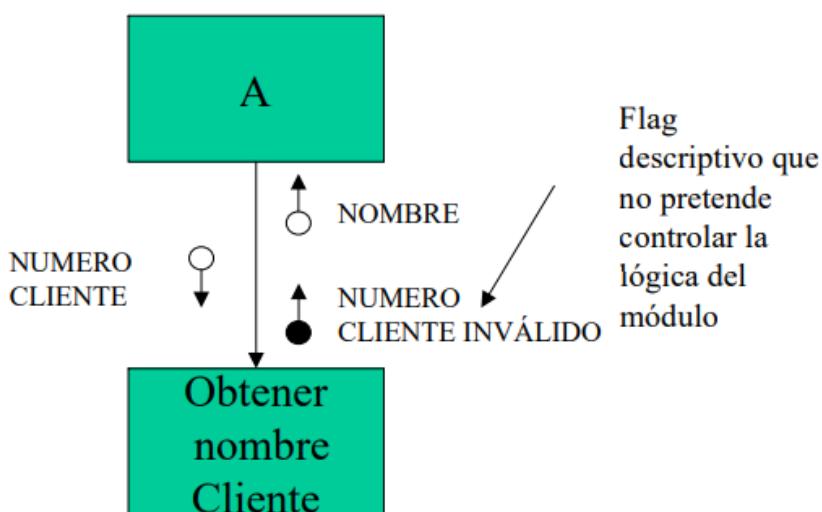
Acoplamiento normal por control

Dos módulos acoplados normalmente lo están por control si uno le pasa al otro un dato con la intención de controlar su lógica interna.



- El módulo jefe controla, a través del dato “Flag de selección”, la lógica del módulo subordinado. Si fuera al revés se dice que hay una inversión de autoridad.

- Con frecuencia el acoplamiento por control indica la presencia de algún error de diseño más grave (Que los módulos posean poca cohesión).
- Dentro del acoplamiento por control hay dos tipos de flag:
 - Flags de control: que intentan controlar la lógica del módulo que los recibe.
 - Flags descriptivos: que revelan cualidades relativas al dato.



TIPO DE INFORMACIÓN QUE INTERCAMBIAN MÓDULOS NORMALMENTE ACOPLADOS

TIPO DE INFORMACIÓN	CONTENIDO DEL NOMBRE	EJEMPLO
DATO SIMPLE O ESTRUCTURADO	SUSTANTIVO	Registro cliente Código Postal
FLAG DESCRIPTIVO	ADJETIVO	Registro es válido Código Postal numérico
FLAG DE CONTROL	VERBO	Leer siguiente registro Rechazar este cliente

Acoplamiento común por variables globales

- Dos módulos A y B están acoplados globalmente si se refieren a una misma zona global de datos o variable global.
- Las áreas de datos globales son desaconsejables: – Un error de programación que aparece en un módulo acoplado globalmente puede aparecer en otros módulos que comparten dicha área global.
- Estos módulos son menos reutilizables
- Es un tipo de acoplamiento en el cuál es difícil determinar la procedencia de la información depositada en el área global.
- Los sistemas con muchas áreas globales son muy difíciles de mantener

Acoplamiento por contenido

Se dice que dos módulos están acoplados por contenido si uno se refiere al interior del otro en alguna de las siguientes formas:

- Modificando o leyendo sus datos internos
- Saltando directamente al interior de su código

En este punto se pierde totalmente la modularidad

COMPARACIÓN DE LOS DISTINTOS TIPOS DE ACOPLAMIENTO

Dos módulos pueden presentar más de un tipo de acoplamiento. En ese caso siempre se considerará el peor de los dos.

Tipo de acoplamiento	Modificabilidad	Comprensión	Reusabilidad
Por datos	Buena	Buena	Buena
Por estampado	Buena	Media	Media
Por control	Pobre	Pobre	Pobre
Global	Media	Mala	Pobre
Contenido	Mala	Mala	Mala

La cohesión

Mide el grado de conexión funcional de los elementos de un módulo. En otras palabras, mide cuan estrechamente relacionadas están las partes de un diseño. Un alto nivel de cohesion significa que cada componente contribuye al objetivo general del sistema

La cohesión es una medida de la fuerza de la relación funcional entre los elementos de un módulo.

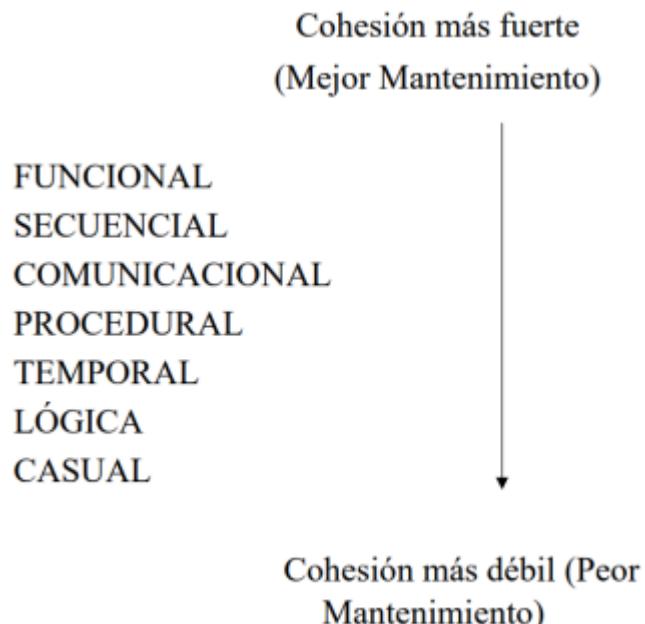
La cohesión es una medida de la fuerza de la relación funcional entre los elementos de un módulo.

Se entiende por elemento de un módulo:

- Una instrucción
- Un grupo de instrucciones
- Una definición de datos
- Una llamada a otro módulo

Lo ideal es disponer de módulos fuertemente cohesivos, cuyos elementos tengan poca relación con otros elementos de otros módulos.

Escala de cohesión



En los tres primeros niveles los módulos se comportan como cajas negras y por tanto representan los niveles más deseables de cohesión dentro de un sistema.

Niveles de cohesión

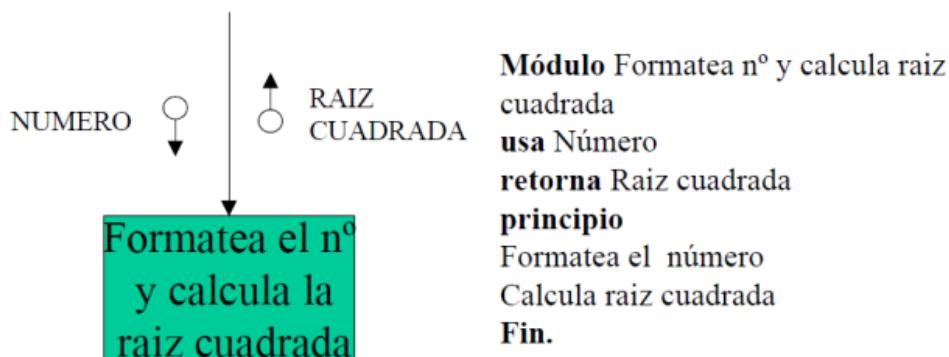
COHESIÓN FUNCIONAL

- Un módulo presenta cohesión funcional si contiene elementos que contribuyen a la realización de una sola función.
- Son los módulos más reutilizables ya que solo es necesario conocer la función que cumplen.
- El nombre de este tipo de módulos suele indicar claramente la función que realizan:
 - Calcular suma
 - Validar campo numérico
 - Calcular seno del ángulo

COHESIÓN SECUENCIAL

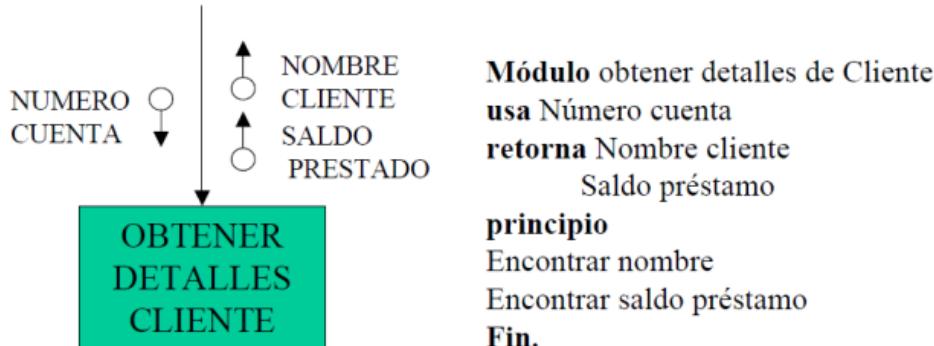
Un módulo presenta cohesión secuencial si sus elementos están envueltos en tareas, tal que la salida de una tarea sirve de entrada para la siguiente.

Como se puede ver el módulo implementa un conjunto de funciones relacionadas entre sí



COHESIÓN COMUNICACIONAL

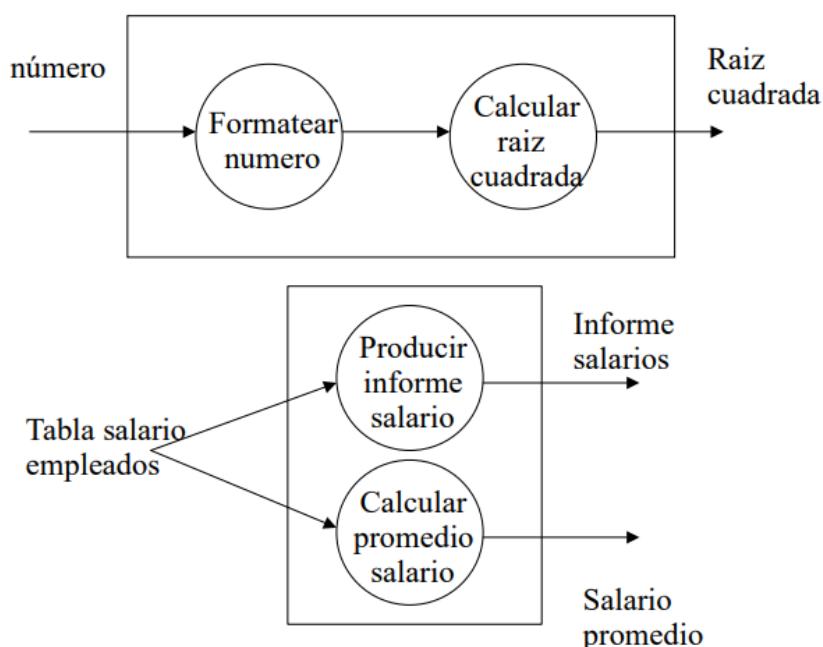
Un módulo presenta cohesión comunicacional si contiene actividades que comparten los mismo datos ya sean de entrada o de salida.



En algunos casos un módulo con cohesión comunicacional puede ser dividido en dos o más módulos con cohesión funcional.

DIFERENCIAS ENTRE COHESIÓN SECUENCIAL Y COMUNICACIONAL

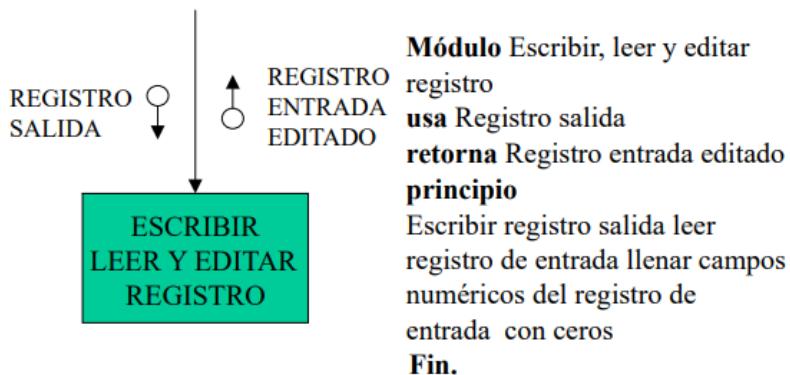
En el módulo con cohesión secuencial es relevante el orden mientras que en el comunicacional no.



COHESIÓN PROCEDURAL

En un módulo con cohesión procedural los elementos desarrollan actividades diferentes, posiblemente sin relación alguna, tal que el flujo de control fluye de una actividad a la siguiente.

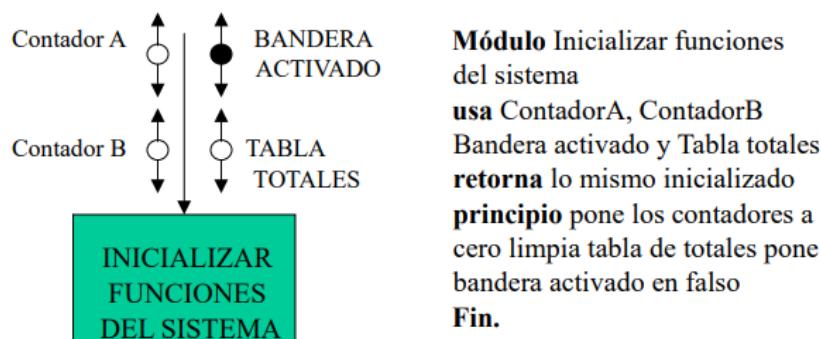
En estos módulos las diferentes actividades no comparten datos, lo único que las relaciona es el flujo de control



COHESIÓN TEMPORAL

En un módulo con cohesión temporal los elementos están envueltos en actividades que están relacionadas por el tiempo (se deben desarrollar al mismo tiempo). Normalmente cada una de estas actividades responde a una tarea diferente.

La cohesión procedural y la temporal son muy parecidas, salvo por el hecho de que en la procedural suele ser más importante el orden.

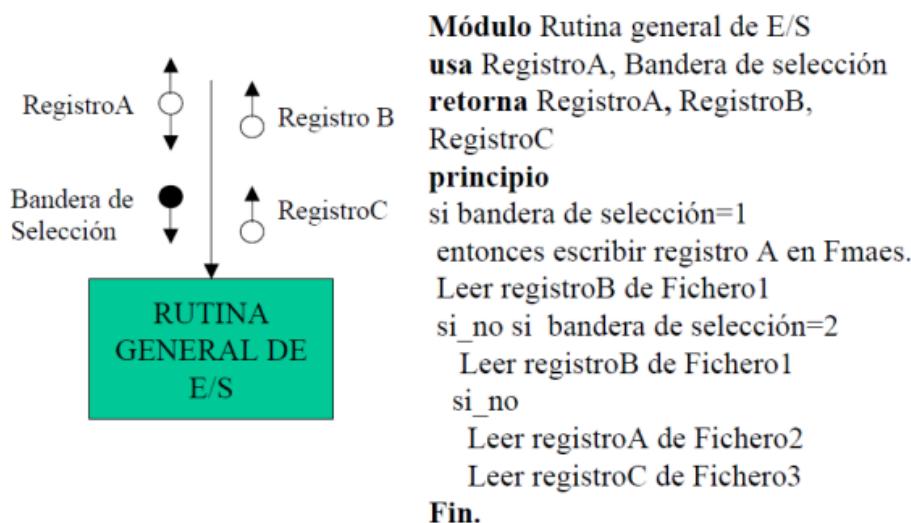


COHESIÓN LÓGICA

En un módulo con cohesión lógica los elementos contribuyen en tareas de la misma categoría general y las actividades a desarrollar se seleccionan fuera del módulo.

Estos módulos se caracterizan por:

- Necesitan una interfaz ancha que permita seleccionar la actividad a ejecutar.
- Son módulos difíciles de entender y mantener

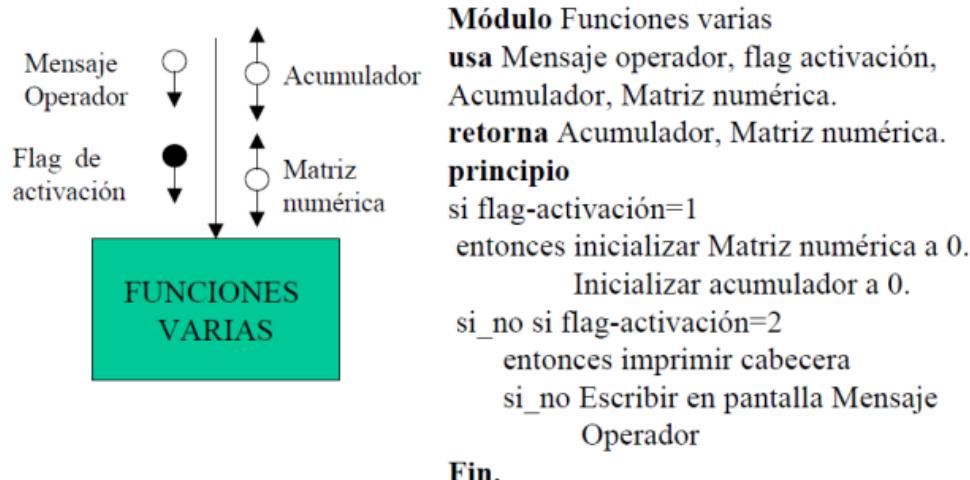


COHESIÓN CASUAL

Un módulo con cohesión casual es aquel cuyos elementos realizan tareas diferentes sin relación significativa entre ellas.

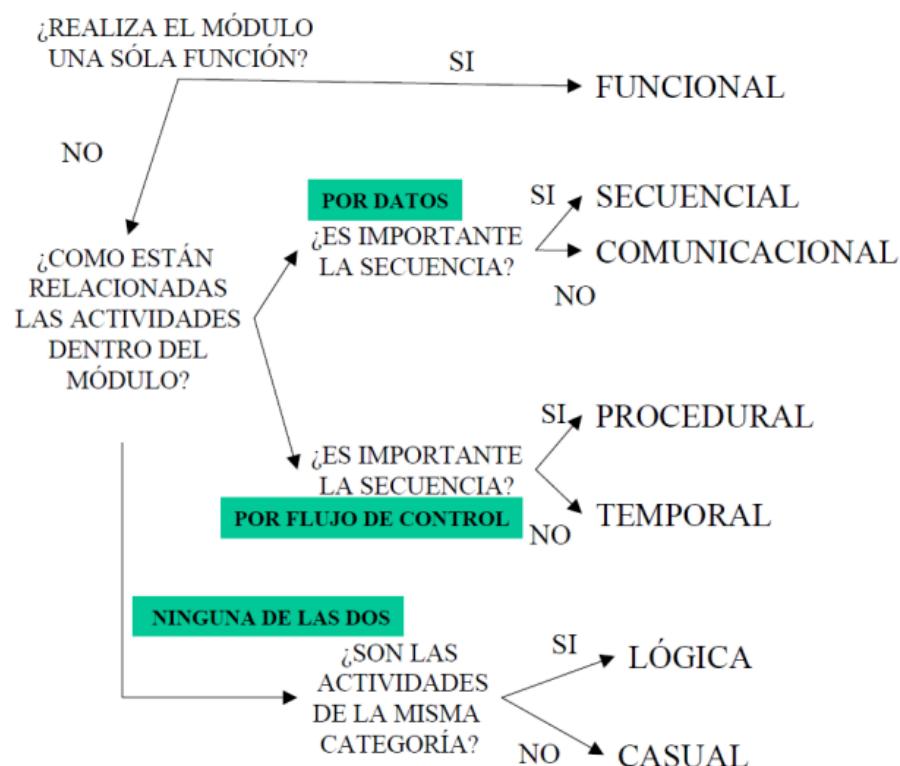
Estos módulos aparecen debido a:

- Intentos por ahorrar tiempo o memoria, situando dentro de un módulo trozos de código que se repite a lo largo del sistema.
- Cambios de mantenimiento mal hechos, en módulos con poca cohesión



DETERMINACIÓN DE LA COHESIÓN DE UN MÓDULO. ÁRBOL DE COHESIÓN

A partir de una serie de preguntas situadas en los nodos se determina la cohesión dominante del módulo.

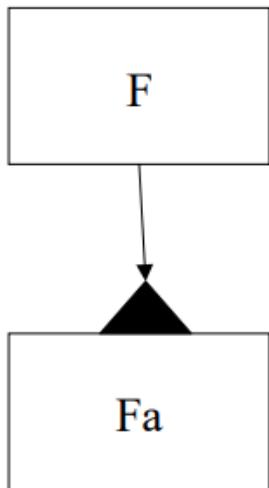


Factorización

Consiste en separar la funcionalidad de un módulo en otro, esto reduce el tamaño de los módulos, fomenta la reusabilidad y simplifica la implementación.

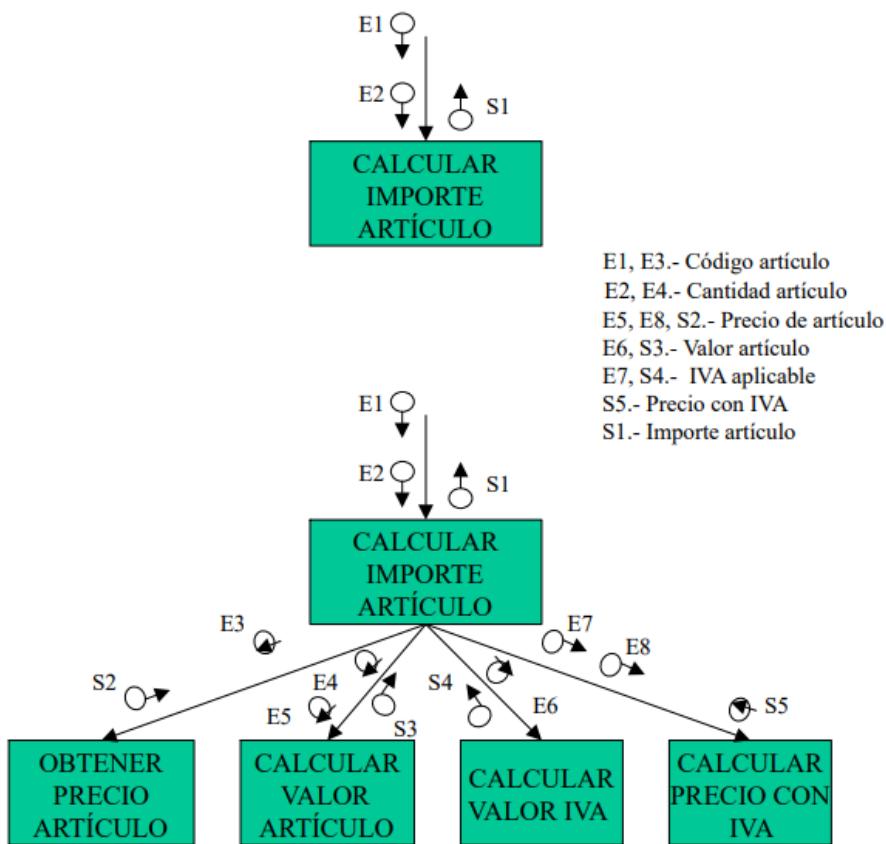
La factorización consiste en la separación de la función contenida como código dentro de un módulo, para formar un nuevo módulo.

El proceso inverso se denomina desfactorización y también puede representarse en el D.E.



Razones de uso de la factorización

- ❖ Reducir el tamaño de los módulos
- ❖ Clarificación del sistema obteniendo las ventajas del diseño descendente
- ❖ Evitar que una misma función este codificada en varios módulos
- ❖ Separar el cálculo y el procesamiento de las decisiones sobre los módulos.
- ❖ Crear módulos reusables.
- ❖ Simplificar la implementación.



CRITERIOS PARA DETENER LA FACTORIZACIÓN

- ❖ Cuando no se encuentre una función bien definida susceptible de ser factorizada.
- ❖ Cuando las interfaces comienzan a ser tan complejas como los propios módulos de manera que no se mejore la comprensión del módulo

CLASIFICACIÓN DE LOS MÓDULOS EN FUNCIÓN DEL FLUJO DE LOS DATOS

- ★ Aferente: el módulo envía información de abajo hacia arriba, es decir recoge información de algún módulo subordinado.
- ★ Eferente: el módulo envía información de arriba hacia abajo.
- ★ Transformación: Toma información del módulo jefe y la transforma en datos que le devuelve.
- ★ Coordinación: coordina la información entre los módulos subordinados.
- ★ Los módulos pueden presentar una combinación de estos tipos.

Fan-In

El "fan-in" de un módulo representa cuántos otros módulos utilizan o dependen de ese módulo en sus operaciones o funciones. Un alto "fan-in" indica que muchos módulos confían en las funciones proporcionadas por el módulo en cuestión. Esto puede ser beneficioso, ya que evita la duplicación de código y promueve la reutilización de funciones.

comunes. Sin embargo, un "fan-in" excesivamente alto puede hacer que el módulo sea demasiado complejo y difícil de mantener.

Fan-out

El "fan-out" o amplitud de control se refiere a la cantidad de subordinados o elementos que un módulo, administrador o unidad de un sistema supervisa o maneja directamente. En otras palabras, es el número de subordinados inmediatos que están directamente bajo la autoridad o gestión de un módulo o entidad superior. Para abordar una fan-out bajo, se sugiere descomponer el módulo en subfunciones o comprimirlo dentro de módulos superiores si tiene sentido en la estructura del problema. En el caso de un fan-out excesivamente alto, puede deberse a la falta de delegación de responsabilidades en submódulos o a una estructura de niveles mal definida. Para resolver este problema, se recomienda agrupar varios subordinados en una función combinada, asegurándose de que los nuevos módulos tengan una cohesión adecuada según los principios de diseño

FORMA DEL SISTEMA

- A partir de la forma del diagrama de estructura se puede evaluar la calidad del diseño.
- En un diagrama de estructura se distinguen tres áreas:
 - aferente: que trata con la información de entrada.
 - Transformación: que procesa los datos de entrada para producir la salida.
 - Eferente: que trata con la información de salida

A partir del desarrollo de cada una de estas áreas podremos evaluar el diseño

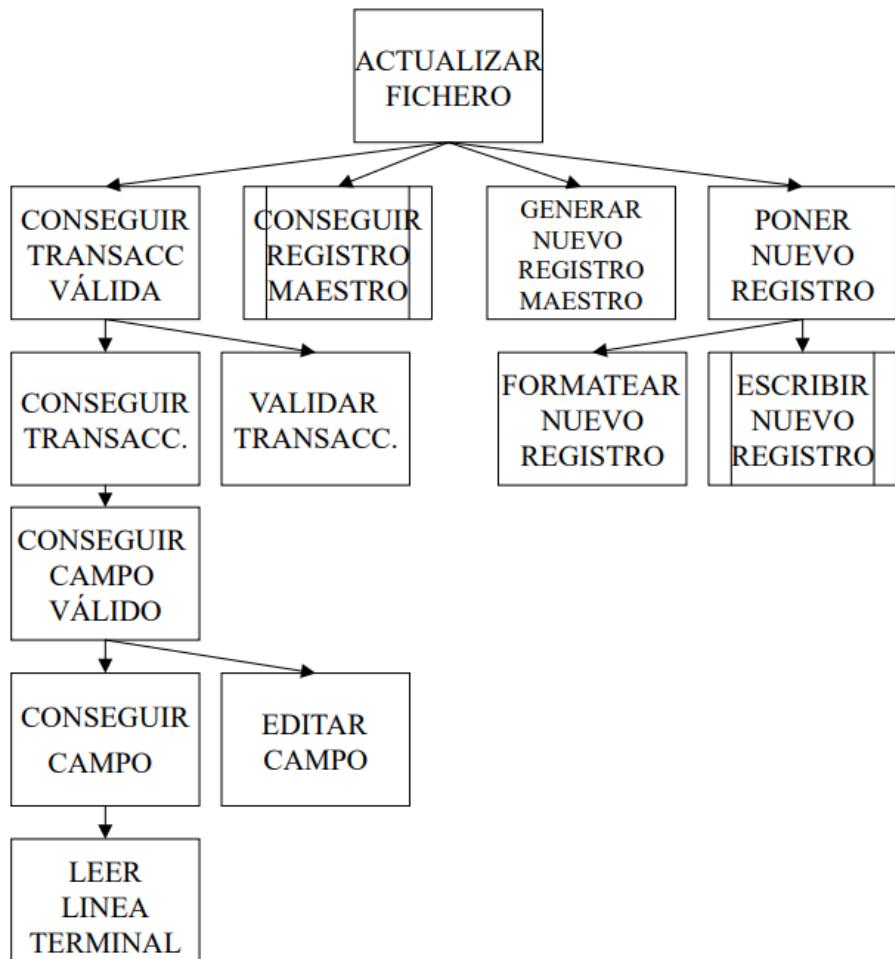
SISTEMAS BALANCEADOS

Un sistema balanceado es aquel en el que los módulos superiores manejan información de naturaleza lógica más que física.

Son los módulos de niveles inferiores de las ramas aferente y eferente los que tratan con los datos físicos de entrada y salida.

La ventaja de los sistemas balanceados es que se mejora el acoplamiento entre los módulos y a su mantenibilidad

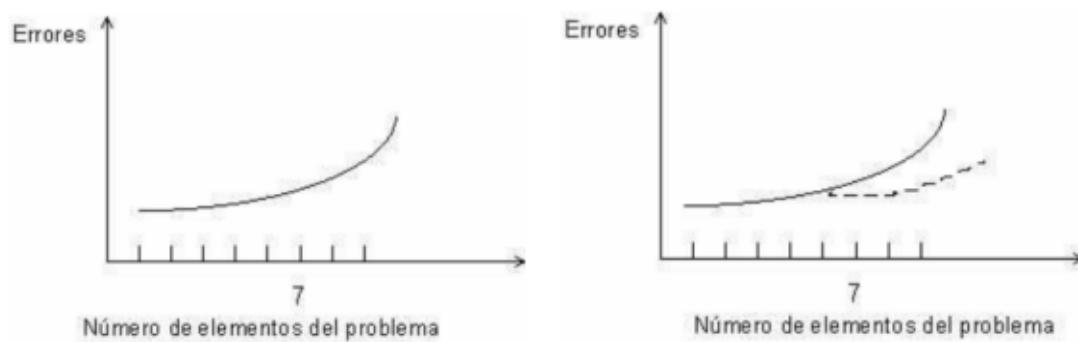
EJEMPLO DE SISTEMA BALANCEADO



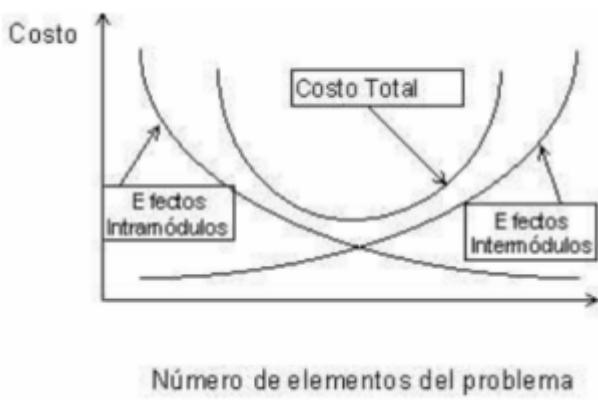
Complejidad y accesibilidad

Complejidad

El número de instrucciones de un programa no es una medida de complejidad ya que existe instrucciones más complejas que otras, y algoritmos más complejos que otros. El proceso de factorización de un problema en partes, puede introducir algunos errores, pero en general si se realiza correctamente tiende a aplastar la curva de errores.



A medida que los módulos sean más pequeños, podemos esperar que su complejidad (y costo) disminuyan, pero además, a mayor cantidad de módulos, tendremos mayor posibilidad de problemas debido a errores en las conexiones intermódulos. Estas son dos curvas en contraposición



Efectos intra módulos (Costos Internos)

Cuando un sistema tiene pocos módulos o es pequeño en tamaño, la complejidad interna de cada módulo tiende a ser mayor. Esto se debe a que los módulos más grandes y complejos tienen más funciones, más relaciones internas y, por lo tanto, son más difíciles de entender y mantener. A medida que se descomponen los módulos en partes más pequeñas y se reducen los elementos del problema dentro de cada módulo, la complejidad interna disminuye. Los módulos más pequeños son más fáciles de entender, probar y mantener. Esto lleva a una reducción en los costos internos asociados con la complejidad.

Efectos inter módulos (Costos Externos)

A medida que se agregan más elementos al problema y se aumenta el tamaño del sistema, la cantidad de interacciones entre módulos diferentes tiende a aumentar. Cada vez que se comunica o coopera entre módulos, existe un costo asociado a la comunicación, coordinación y administración de esas interacciones. Con un número creciente de módulos, la gestión de las relaciones entre ellos se vuelve más compleja y costosa. Se necesitan más esfuerzos para garantizar que los módulos funcionen juntos sin conflictos.

La complejidad se divide en tres aspectos principales: cantidad, accesibilidad y estructura.

Cantidad

La cantidad de información se refiere al número de datos o parámetros que un programador debe manejar para comprender la interfaz de un módulo. A medida que aumenta la cantidad de parámetros, aumenta la probabilidad de error. Programar con muchos parámetros puede ser propenso a confusiones y dificultades. Además, un módulo con demasiados parámetros puede indicar que realiza múltiples funciones y podría dividirse en módulos más simples y funcionales con menos argumentos.

Accesibilidad

La accesibilidad se refiere a la facilidad con la que un programador puede acceder y comprender la información necesaria para escribir o interpretar el código correctamente.

- La interface es menos compleja si la información puede ser accedida (por el programador, no por la computadora) directamente; es más compleja si la información referencia indirectamente otros elementos de datos.
- La interface es menos compleja si la información es presentada localmente dentro de la misma sentencia de llamada. La interface es más compleja si la información necesaria es remota a la sentencia.
- La interface es menos compleja si la información es presentada en forma standard que si se presenta de forma imprevista.
- La interface es menos compleja si su naturaleza es obvia es menos compleja que si su naturaleza es obscura.

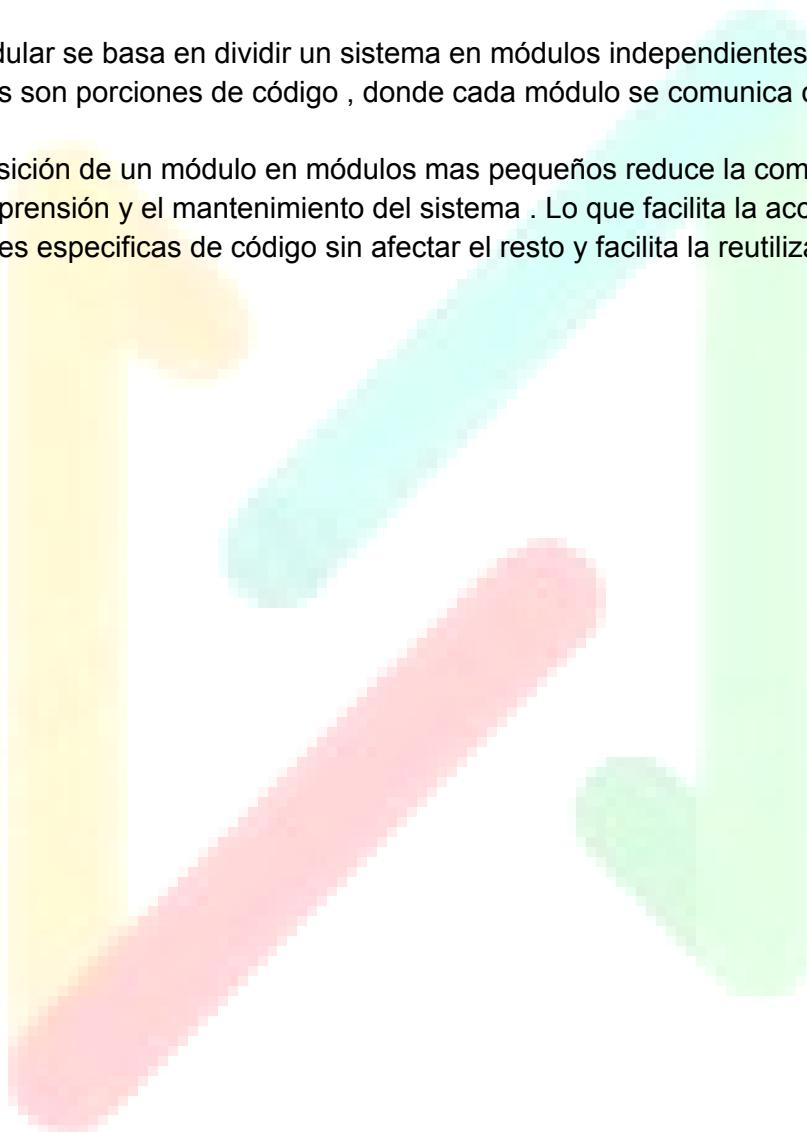
Estructura

La estructura de la información es otro factor clave en la complejidad. La información lineal y afirmativa es menos compleja de entender que la información anidada o negativa. Por ejemplo, las construcciones de sentencias IF anidadas o las expresiones lógicas que involucran negaciones (NOT) son más difíciles de comprender que las alternativas más simples y directas. La estructura también es relevante en las referencias intermodulares.

Relación entre accesibilidad y complejidad

El diseño modular se basa en dividir un sistema en módulos independientes y autónomos . Estos módulos son porciones de código , donde cada módulo se comunica con otro

La descomposición de un módulo en módulos mas pequeños reduce la complejidad , lo que facilita la comprensión y el mantenimiento del sistema . Lo que facilita la accesibilidad para modificar partes específicas de código sin afectar el resto y facilita la reutilización de módulos



Diagramas de secuencia

Parten de diagramas del análisis de los casos de uso, y muestran el diagrama de comunicación de objetos mediante mensajes, que ocurre mediante un método que está implementado y se ejecuta en el objeto que lo recibe. Cada objeto tiene que estar

diferenciado (si es de interfaz, control o datos). A diferencia del diagrama de colaboración de objetos, en este si se ve la secuencia de como suceden las cosas.

- Ejemplo de interfaz, sería por ejemplo un objeto que reciba un mensaje que representa un método de mostrar detalles de vista.
- Ejemplo de control sería que una interfaz le manda al mismo para que ejecute una lógica encapsulada.
- Ejemplo de datos, un objeto el cual sea alcanzado por un mensaje, que representa un método validar, que valide la existencia de ciertos datos.

Partiendo de los casos de uso, se partitionan, de acuerdo a los siguientes principios

La funcionalidad de los casos de uso que dependen directamente del ambiente del sistema, es ubicada en los objetos de interface.

La funcionalidad que se refiere con almacenamientos y manejo de información, la cual no es naturalmente ubicada en un objeto de interface, es ubicada en objetos entidad.

La funcionalidad específica a uno o pocos casos de uso y no naturalmente ubicado en otro objeto, es ubicado en objetos de control.

Realizando esta división, se obtiene una estructura, la cual, ayuda a entender el sistema desde una vista lógica y menos sensible a modificaciones. Como los objetos son identificados y especificados, se describe cómo se relacionan estos objetos a través de distintos tipos de asociaciones.

Ejemplo de diagramas de secuencia

Caso de uso iniciar sesión

Precondición: El usuario debe estar registrado.

Actor principal: Usuario: Este usuario tendrá un rol de **administrador, tecnovigilante, enfermero, director o mantenimiento**.

Flujo Principal:

4.2.1 El usuario ingresa su nombre de usuario y contraseña.

4.2.2 El sistema verifica si el usuario está registrado.

4.2.3 Si el usuario está registrado, lo lleva a la ventana principal.

Flujo alternativo:

4.2.4 Si el usuario no está registrado, se muestra un mensaje de error: "El usuario no está registrado en el sistema".

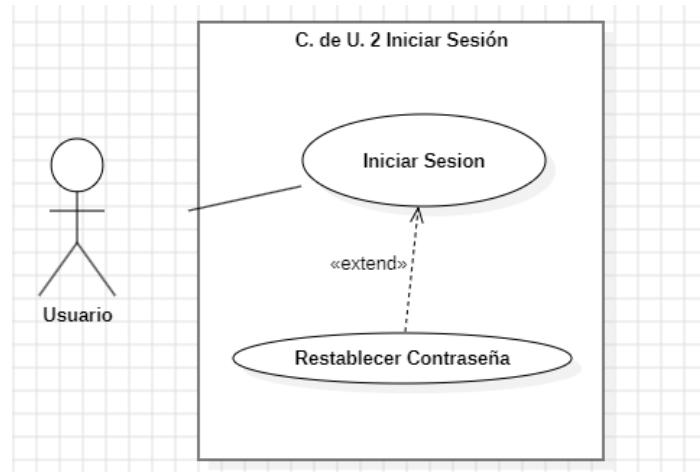


Diagrama de colaboración de objetos

- ❖ **Interfaz de login:**
 - ★ **Atributos:**
 - Usuario
 - Contrasenia
 - ★ **Métodos:**
 - showLogin(): void
 - showErrorMsg(): void
 - giveAccess(): void.
- ❖ **Verificar datos:**
 - ★ Métodos:
 - ValidarCampos(Usuario,Contrasenia): boolean
- ❖ **Usuario**
 - ★ **Métodos**
 - getUsuario(Usuario, Contrasenia): boolean

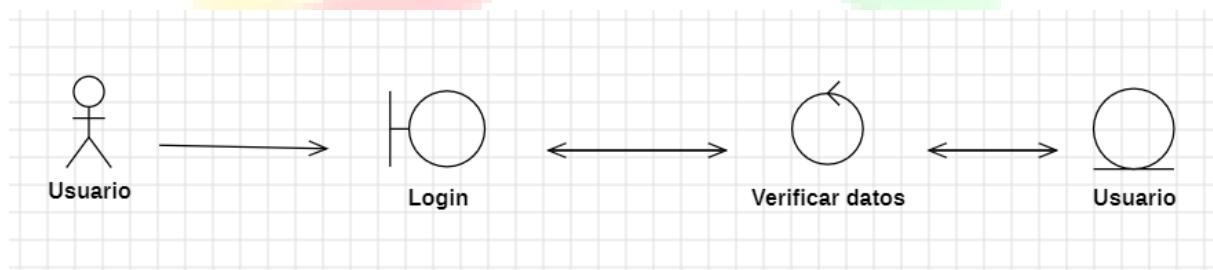
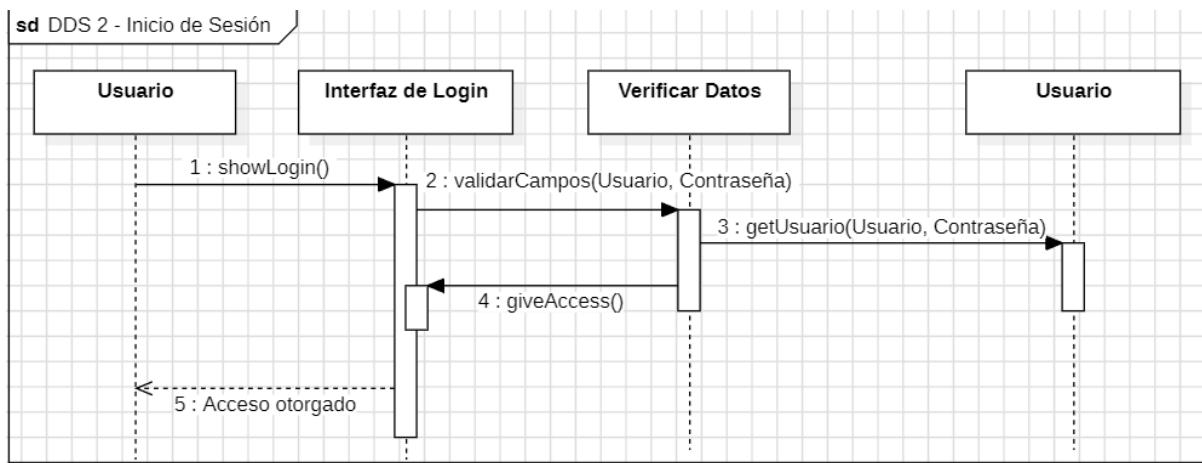


Diagrama de secuencia



Inicio:

Método IniciarSesion()

```

showLogin()

Usuario <- PedirInput("Ingrese su nombre de usuario")

Contrasenia <- PedirInput("Ingrese su contraseña")

Si ValidarCampos(Usuario, Contrasenia) Entonces

    Si getUsuario(Usuario, Contrasenia) Entonces

        ShowMsg("Acceso concedido. Bienvenido a la ventana principal.")

        giveAccess()

    Sino

        ShowErrorMsg("El usuario no está registrado en el sistema.")

    FinSi

    Sino

        ShowErrorMsg("Debe ingresar todos los campos correctamente.")

    FinSi
  
```

Método ShowMsg(mensaje: String)

```
Escribir(mensaje)
```

Método ValidarCampos(Usuario: String, Contrasenia: String): boolean

```
// Validar Usuario

Si (longitud(Usuario) = 0) Entonces
```

```

        Retornar Falso

    Sino

        // Validar Contraseña

        Si (longitud(Contraseña) = 0) Entonces

            Retornar Falso

        Sino

            Retornar Verdadero

        FinSi

    FinSi

Método getUsuario(Usuario: String, Contraseña: String): boolean

    // Consulta en la base de datos si el usuario con la contraseña existen

    // Retornar Verdadero si existe, Falso si no

    Retornar BuscarUsuarioEnDB(Usuario, Contraseña)

Método showLogin():void;

    // Muestra la interfaz gráfica de login

    Escribir("Mostrar interfaz de login con campos de usuario y contraseña.")

Método showErrorMsg(mensaje: String): void

    // Muestra un mensaje de error en la interfaz de login

    Escribir("Error: " + mensaje)

Método giveAccess(): void

    // Redirige al usuario a la ventana principal del sistema

    Escribir("Redirigiendo a la ventana principal...")

```

Fin

Calidad del software

La **calidad** es un concepto amplio y multidimensional que se utiliza en diversos contextos para describir el grado de excelencia, superioridad o satisfacción en un producto, servicio, proceso o resultado. Se refiere a la medida en que un producto o servicio cumple con las expectativas, requisitos o estándares establecidos por las partes interesadas.

La **calidad en el software** se refiere a la medida en que un programa o sistema cumple con los requisitos y expectativas establecidos, y a su capacidad para funcionar de manera confiable, eficiente y segura. La calidad del software es esencial para garantizar que el software cumpla con sus objetivos y sea satisfactorio para los usuarios finales. Algunos aspectos clave de la calidad en el software:

- **Satisfacción de los Requerimientos:** El software de calidad debe cumplir con los requerimientos funcionales y no funcionales establecidos. Esto incluye asegurarse de que todas las características y funcionalidades especificadas estén implementadas correctamente.
- **Fiabilidad:** El software debe ser confiable y consistente en su funcionamiento. Debe funcionar correctamente sin errores o fallos inesperados. La fiabilidad es esencial, especialmente en aplicaciones críticas.
- **Eficacia y Eficiencia:** La calidad puede estar relacionada con la eficiencia y la eficacia. Un producto o proceso de calidad debería lograr sus objetivos de manera eficaz (hacer lo correcto) y de manera eficiente (hacerlo de manera rentable).
- **Mejora Continua:** La búsqueda constante de la mejora es un principio fundamental de la gestión de la calidad. Las organizaciones se esfuerzan por identificar áreas de mejora y aplicar medidas para aumentar la calidad con el tiempo.
- **Contexto y Perspectiva:** La calidad es relativa y puede variar según el contexto y la perspectiva. Lo que se considera de alta calidad en un contexto puede no serlo en otro. La calidad puede ser subjetiva y depende de las necesidades y expectativas de las partes interesadas.
- **Normas y Estándares:** En muchos campos, existen normas y estándares que definen criterios específicos de calidad. Estas normas ayudan a establecer un marco de referencia objetivo para la calidad.
- **Usabilidad:** La usabilidad se refiere a la facilidad de uso del software. Debe ser intuitivo y amigable para el usuario, lo que significa que los usuarios pueden interactuar con él de manera efectiva y sin confusiones.
- **Mantenibilidad:** El software debe ser fácil de mantener y actualizar. Esto implica que el código esté bien estructurado, documentado y modularizado para facilitar cambios y correcciones.
- **Seguridad:** La seguridad del software es crítica, especialmente cuando se trata de aplicaciones que manejan datos sensibles o información personal. Debe proteger contra amenazas y vulnerabilidades, como ataques cibernéticos.
- **Portabilidad:** El software debe ser portátil, lo que significa que puede ejecutarse en diferentes plataformas y sistemas operativos sin problemas.
- Escalabilidad: En aplicaciones que deben manejar un aumento en la carga de trabajo, la escalabilidad es importante. El software debe poder adaptarse y manejar mayores volúmenes de datos o usuarios sin degradación del rendimiento.
- Documentación: La documentación clara y completa es esencial para ayudar a los desarrolladores y usuarios a comprender y utilizar el software de manera efectiva.
- Pruebas y Validación: La calidad del software se verifica a través de pruebas exhaustivas y validación. Se realizan pruebas para detectar errores, evaluar el rendimiento y garantizar que el software cumpla con los criterios de calidad.

- Ciclo de Vida y Procesos de Desarrollo: La calidad del software también está relacionada con la metodología de desarrollo utilizada. Los procesos de desarrollo y gestión de proyectos influyen en la calidad del producto final.
- Satisfacción del Cliente: En última instancia, la calidad del software se evalúa en función de la satisfacción del cliente. Si los usuarios finales están satisfechos y el software cumple con sus necesidades, se considera que tiene alta calidad.

Calidad del software definiciones

- La calidad del software es el grado con el que un sistema, componente o proceso cumple los requerimientos especificados y las necesidades o expectativas del cliente o usuario". (IEEE, Std. 610-1990).
- "Concordancia del software producido con los requerimientos explícitamente establecidos, con los estándares de desarrollo prefijados y con los requerimientos implícitos no establecidos formalmente, que desea el usuario" (Pressman, 1998)

Factores que determinan la calidad del software

Se pueden clasificar en dos grandes grupos (Pressman) :

- **Factores que pueden ser medidos directamente**
- **Factores que solo pueden ser medidos indirectamente**

Se centran en tres aspectos importantes de un producto software (McCall)

- **Características operativas**
 - ❖ Corrección. ¿Hace lo que quiero?
 - ❖ Fiabilidad. ¿Lo hace de forma fiable todo el tiempo?
 - ❖ Eficiencia. ¿Se ejecutará en mi hardware lo mejor que pueda?
 - ❖ Seguridad (Integridad). ¿Es seguro?
 - ❖ Facilidad de uso. ¿Está diseñado para ser usado?
- **Capacidad de soportar los cambios**
 - ❖ Facilidad de mantenimiento. ¿Puedo corregirlo?
 - ❖ Flexibilidad. ¿Puedo cambiarlo?
 - ❖ Facilidad de prueba. ¿Puedo probarlo?
- **Adaptabilidad a nuevos entornos**
 - ❖ Portabilidad. ¿Podré usarlo en otra máquina?
 - ❖ Reusabilidad. ¿Podré reutilizar alguna parte del software?
 - ❖ Interoperabilidad. ¿Podré hacerlo interactuar con otro sistema?

Estándares y modelos de evaluación y mejora de los procesos software

- ISO 9000 (ISO 9001:2000)
- (SPICE) ISO/IEC 15504
- CMM / CMMI
- Certificación. Organismos

ISO 9000

ISO : Organización Internacional para la Estandarización . Prefijo Griego que significa IGUAL.

“Di lo que haces, haz lo que dices”

Con el objetivo de estandarizar los sistemas de calidad de las diferentes empresas y sectores, se publican las normas ISO 9000, que son un conjunto de normas editadas y revisadas periódicamente por la Organización Internacional de Normalización (ISO) sobre la garantía de calidad de los procesos.

Así, se consolida a nivel internacional la normativa de la gestión y control de calidad

- Directrices para la gestión del sistema de calidad y modelos de garantía de calidad para la empresa
- Las directrices son genéricas y aplicables a cualquier sector
- Es un marco de trabajo para la mejora continua

Objetivos de ISO 9000 :

- Proporcionar una guía para la gestión de la calidad: diseño e implantación de sistemas de calidad. (ISO 9000 no normaliza el sistema de gestión de calidad, ya que esto depende del tipo de sector, tamaño de la empresa, organización interna, etc, sino que normaliza las verificaciones que se han de realizar sobre el sistema de calidad)
- Describir los requerimientos generales para garantizar la calidad (demostrar la idoneidad del sistema de calidad)

Aspectos positivos

- Es un factor competitivo para las empresas
- Proporciona confianza a los clientes
- Ahorra tiempo y dinero, evitando recertificar la calidad según los estándares locales o particulares de una empresa.
- Se ha adaptado a más de 90 países e implantado a todo tipo de organizaciones industriales y de servicios, tanto sector privado como público
- Proporciona una cierta garantía de que las cosas se hacen tal como se han dicho que se han de hacer

Aspectos Negativos

- Es costoso
- Muchas veces se hace por obligación.
- Es cuestión de tiempo que deje de ser un factor competitivo
- Hay diferencias de interpretación de las cláusulas del estándar
- No es indicativa de la calidad de los productos, procesos o servicio.
- Hay mucha publicidad engañosa.

ISO 9001

La ISO 9001 se centra en establecer aquellos requisitos certificables para garantizar la calidad. Para esto, establece mecanismos definidos para:

- identificación y comunicación de los requisitos del cliente
- identificación de los objetivos de los procesos
- evidencia del planificación de los procesos
- definición de responsabilidades y autoridades
- adecuación de personal competente
- adecuación de recursos y ambiente de trabajo
- adecuación de la documentación que describa los métodos operativos
- monitoreo de la performance del proceso y control de las NC
- aplicación de acciones correctivas/preventivas ✓ evidencia de mejora continua
- disponibilidad de registros

Los pilares de la ISO 9001 son los de garantizar la gestión por procesos , satisfacción al cliente y mejorar continua



CMM (Capability Maturity Model)

“CMM es una aplicación de sentido común de los conceptos de gestión de procesos y mejora de la calidad al desarrollo y mantenimiento del software

- Estudia los procesos de desarrollo de software de una organización y produce una evaluación de la madurez de la organización según una escala de cinco niveles
- La madurez de un proceso es un indicador de la capacidad para construir un software de calidad.
- Es un modelo para la mejora de las organizaciones
- Obliga a una revisión constante

(SGC) Sistema de gestión de calidad

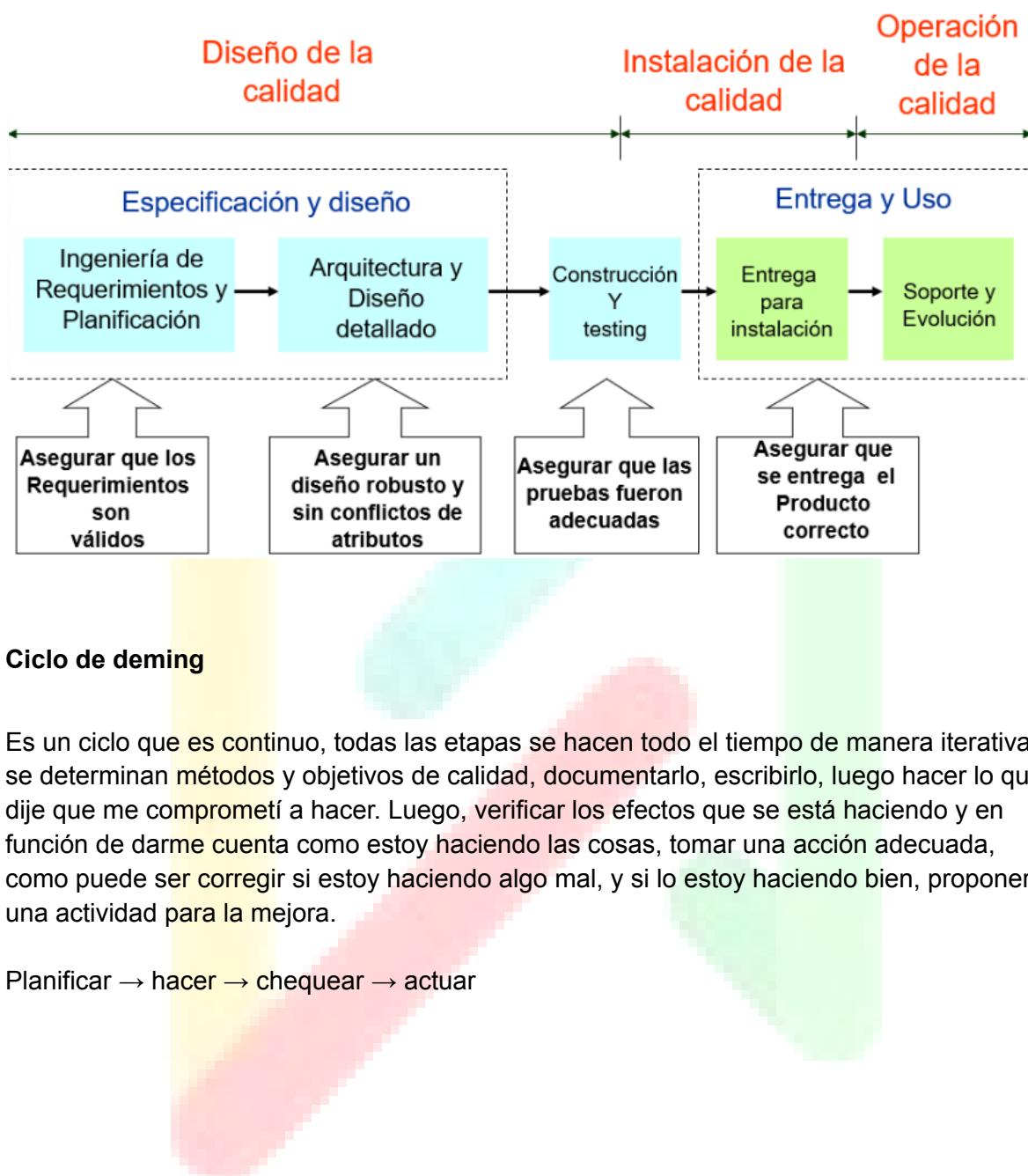
Un Sistema de Gestión de Calidad (SGC) es un conjunto de procesos, políticas, procedimientos y recursos que una organización implementa para garantizar que los productos o servicios que ofrece cumplan con ciertos estándares de calidad y satisfagan las necesidades y expectativas de sus clientes.

El objetivo principal de un SGC es asegurar que la calidad sea una parte integral de todas las actividades de la organización. Este permite:

- Reducir la necesidad de “apagar incendios”.
- Identificar tareas adecuadas.
- Documentar la experiencia de la organización de manera estructurada.
- Proporcionar los medios que documenten la experiencia.
- Proporcionar los medios para identificar y resolver problemas.
- Proporcionar los medios para que el personal realice las tareas bien desde el principio.
- Proporcionar pruebas para demostrar la calidad de los procedimientos y servicios.
- Proporcionar datos para detectar el funcionamiento de sus procesos, mejorarlos y lograr la satisfacción del cliente.
- Proporcionar datos para suministrar servicios que no sean “devueltos”.



En dónde “fabricamos” la calidad?





Testing

Se puede definir al testing, como el proceso de evaluación de un producto desde un punto de vista crítico para verificar que el producto hace lo que tiene que hacer, en el que el "tester" (persona que realiza las pruebas) somete el producto a una serie de acciones inquisitivas, y el producto responde con su comportamiento como reacción

En ingeniería de software, **verificación y validación** son procesos clave para garantizar la calidad del software. Aunque a menudo se usan juntos, tienen significados diferentes:

1. Verificación:

La verificación responde a la pregunta: "**¿El software se está construyendo correctamente?**"

Se refiere al proceso de asegurarse de que el software cumpla con los requisitos y especificaciones establecidas. Implica revisar el diseño, el código, y la implementación para garantizar que todo esté alineado con los planes iniciales.

Técnicas de verificación: Revisiones de código, inspecciones, pruebas estáticas, y análisis de requisitos.

2. Validación:

La validación responde a la pregunta: "**¿El software construido cumple con las necesidades del usuario?**"

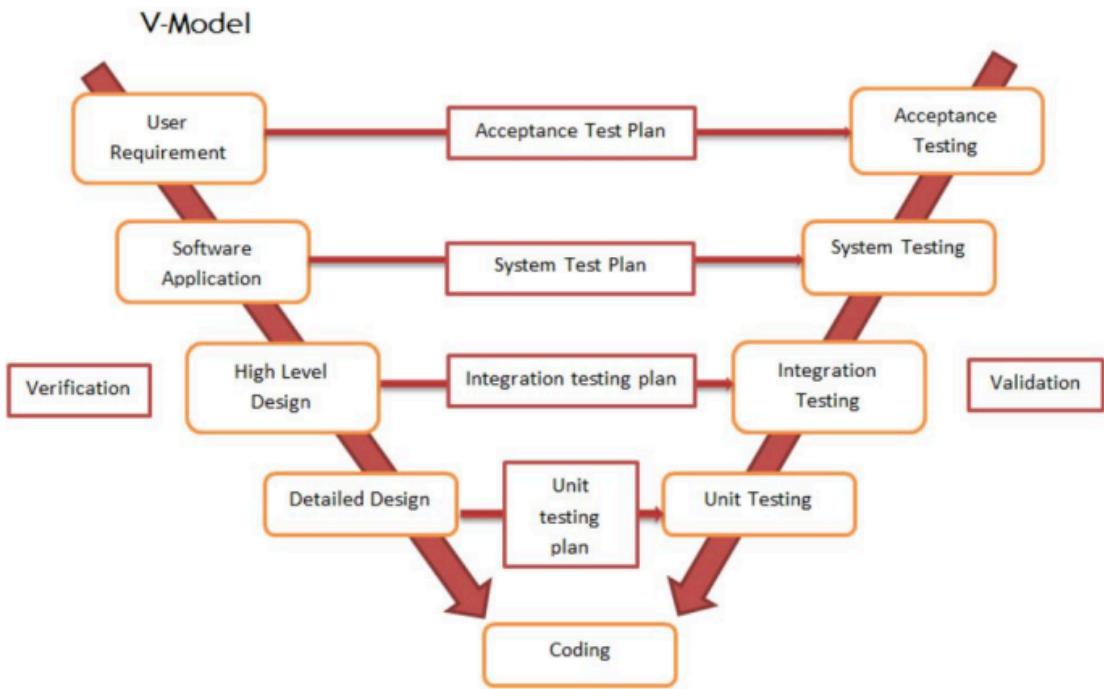
Se enfoca en comprobar que el software realmente satisface las necesidades y expectativas del cliente o usuario final. La validación asegura que el producto es el adecuado para su propósito.

Técnicas de validación: Pruebas dinámicas, pruebas de usuario, y pruebas de aceptación.

Plan de Pruebas

- El Plan de Pruebas (PP) es un guía para gestionar las distintas pruebas que se realizarán a lo largo del proyecto.
- Pressman, autor del libro "Ingeniería del Software. Un enfoque práctico", señala que "Las pruebas son el último bastión para la evaluación de la calidad y, de manera más pragmática, el descubrimiento de errores" (p.384)
- Todos los desarrollos de soluciones tecnológicas implican la realización de pruebas, ya sea para verificar el funcionamiento del producto desarrollado, validar si cumple con los requerimientos, o para validar/refutar alguna hipótesis de investigación. La Verificación se lleva a cabo para determinar que se esté construyendo correctamente la solución. Y la Validación para determinar que se esté construyendo la solución correcta.
- Las pruebas pueden ser de distintos tipos y, por lo tanto, deberán ser capaces de analizar cuáles aplican y de qué manera para su proyecto en particular. Cada uno de estos tipos de prueba se relaciona con alguna de las etapas, y por ende los entregables, del proyecto, y también se relaciona con la etapa de desarrollo del producto que estén desarrollando, si fuese el caso.

V-Model



Las pruebas pueden ser de distintos tipos y cada uno de estos tipos de prueba se relaciona con alguna de las etapas del proceso de desarrollo de software. Por tal motivo, el sub-proceso de testing recibe como entrada no solo el producto de software construido, sino también, la especificación de requerimientos modelada por los casos de prueba. De ese modo, durante todo el proceso se puede validar que se está construyendo el producto correcto y que se está construyendo correctamente. Secuenciadas y ordenadas según el momento en el que debieran realizarse, las pruebas se denominan (y pueden entenderse, conceptualmente, según el siguiente modelo en V)

Ordenadas según el momento en el que debieran realizarse, entendiendo una secuencia espiralada para su realización, las pruebas se denominan:

Pruebas Unitarias

Una prueba unitaria es una forma de probar el correcto funcionamiento de un módulo de código. Esto sirve para asegurar que cada uno de los módulos funcione correctamente por separado. Luego, con las Pruebas de Integración, se podrá asegurar el correcto funcionamiento del sistema o subsistema en cuestión. La idea es escribir casos de prueba para cada función no trivial o método en el módulo de forma que cada caso sea independiente del resto.

Se concentran en cada módulo o componente del dispositivo/sistema que se está construyendo, entendidos de manera aislada.

Pruebas Integrales

Pruebas integrales o pruebas de integración son aquellas que se realizan en el ámbito del desarrollo de software una vez que se han aprobado las pruebas unitarias. Únicamente se refieren a la prueba o pruebas de todos los elementos unitarios que componen un proceso, hecha en conjunto, de una sola vez. Consiste en realizar pruebas para verificar que un gran conjunto de partes de software funcionan juntos. Las pruebas de integración (algunas veces llamadas integración y testeo I&t) es la fase del testeo de software en la cual módulos individuales de software son combinados y testeados como un grupo. Son las pruebas posteriores a las pruebas unitarias y preceden el testeo de sistema.

Se realizan en la medida que los distintos módulos o componentes se comienzan a integrar entre sí.

Pruebas de Homologación o de Sistemas o de validación

Las pruebas de validación en la ingeniería de software son el proceso de revisión que el sistema de software producido cumple con las especificaciones y que cumple su cometido. Es normalmente una parte del proceso de pruebas de software de un proyecto, que también utiliza técnicas tales como evaluaciones, inspecciones, y tutoriales. La validación es el proceso de comprobar lo que se ha especificado es lo que el usuario realmente quería. Se trata de evaluar el sistema o parte de este durante o al final del desarrollo para determinar si satisface los requisitos iniciales

Se realizan sobre el sistema/dispositivo como un todo, validando los requerimientos planteados al inicio. Muchas veces participa de ellas el cliente/usuario/destinatario.

Pruebas de Regresión

Se denominan Pruebas de regresión a cualquier tipo de pruebas de software que intentan descubrir las causas de nuevos errores (bugs), carencias de funcionalidad, o divergencias funcionales con respecto al comportamiento esperado del software, inducidos por cambios recientemente realizados en partes de la aplicación que anteriormente al citado cambio no eran propensas a este tipo de error. Esto implica que el error tratado se reproduce como consecuencia inesperada del citado cambio en el programa.

Este tipo de cambio puede ser debido a prácticas no adecuadas de control de versiones, falta de consideración acerca del ámbito o contexto de producción final y extensibilidad del error que fue corregido (fragilidad de la corrección), o simplemente una consecuencia del rediseño de la aplicación.

Se realizan cada vez que se modifica, ajusta o agrega un módulo o componente en el marco de pruebas de integración u homologación para evitar que existan problemas con funcionalidades que antes funcionaban correctamente.

Pruebas de Aceptación (UAT, por sus siglas en inglés)

Son básicamente pruebas funcionales, sobre el sistema completo, y buscan una cobertura de la especificación de requisitos y del manual del usuario. Estas pruebas no se realizan

durante el desarrollo, pues sería impresentable al cliente; sino que se realizan sobre el producto terminado e integrado o pudiera ser una versión del producto o una iteración funcional pactada previamente con el cliente. También se la conoce como prueba de homologación o UAT (user acceptance testing). La experiencia muestra que aún después del más cuidadoso proceso de pruebas por parte del desarrollador y del equipo de testing luego, quedan una serie de errores que sólo aparecen cuando el cliente comienza a usarlo

Son las pruebas del cliente/destinatario/usuario final. Se enfocan en las características generales y la funcionalidad del sistema, los elementos visibles y revisables por parte del cliente.

De acuerdo al dispositivo o sistema del que se trate, serán aplicables otros tipos de pruebas específicas, tales como:

- de Recuperación
- de Documentación
- de Resistencia
- de Operación
- de Estrés
- Negativa.

Caja blanca (sistemas)

- En programación, se denomina caja blanca a un tipo de pruebas de software que se realiza sobre las funciones internas de un módulo. Así como las pruebas de caja negra ejercitan los requisitos funcionales desde el exterior del módulo, las de caja blanca están dirigidas a las funciones internas. Entre las técnicas usadas se encuentran; la cobertura de caminos (pruebas que hagan que se recorran todos los posibles caminos de ejecución), pruebas sobre las expresiones lógico-aritméticas, pruebas de camino de datos (definición-uso de variables), comprobación de bucles (se verifican los bucles para 0,1 y n iteraciones, y luego para las iteraciones máximas, máximas menos uno y más uno).
- Las pruebas de caja blanca se llevan a cabo en primer lugar, sobre un módulo concreto, para luego realizar las de caja negra sobre varios subsistemas (integración).
- En los sistemas orientados a objetos, las pruebas de caja blanca pueden aplicarse a los métodos de la clase, pero según varias opiniones, ese esfuerzo debería dedicarse a otro tipo de pruebas más especializadas (un argumento podría ser que los métodos de una clase suelen ser menos complejos que los de una función de programación estructurada).

Caja negra (sistemas)

- Se denomina caja negra a aquel elemento que es estudiado desde el punto de vista de las entradas que recibe y las salidas o respuestas que produce, sin tener en cuenta su funcionamiento interno. En otras palabras, de una caja negra nos interesaría su forma de interactuar con el medio que le rodea (en ocasiones, otros elementos que también podrían ser cajas negras) entendiendo qué es lo que hace, pero sin dar importancia a cómo lo hace. Por tanto, de una caja negra deben estar muy bien definidas sus entradas y salidas, es decir, su interfaz; en cambio, no se precisa definir ni conocer los detalles internos de su funcionamiento

Pruebas alfa y beta

- La prueba alfa se lleva a cabo en el ámbito de desarrollo por un grupo representativo de usuarios finales. El software se usa en un escenario natural con el desarrollador “mirando sobre el hombro” de los usuarios y registrando los errores y problemas de uso. Las pruebas alfa se realizan en un ambiente controlado (no liberado).
- La prueba beta se realiza en uno o más sitios del usuario final. La prueba beta es una aplicación “en vivo” del software en un ambiente que no puede controlar el desarrollador (productivo, pero restringido). El cliente registra todos los problemas (reales o imaginarios) que se encuentran durante la prueba beta y los reporta al equipo de desarrollo periódicamente. Como resultado de los problemas reportados durante las pruebas beta, es posible hacer modificaciones y luego preparar la liberación del producto de software a toda la base de clientes.

Pruebas escala total

El test de escala total pone el sistema a funcionar en condiciones críticas; esto incluye llevar todos los parámetros del sistema a sus límites máximos, conectar todo el equipamiento previsto, accesos simultáneos de usuarios y muchos casos de operación corriendo al mismo tiempo. Este test requiere la totalidad del sistema y una planta de escala total disponible, lo que lo hace muy caro; pero, normalmente es el que descubre los errores más difíciles de prever

Pruebas de performance

Las pruebas de performance son, hoy en día, cada vez más necesarias: los tiempos de respuesta por encima de lo aceptable, la excesiva variabilidad de los mismos en función de la carga del sistema y los problemas de fiabilidad o disponibilidad deben de considerarse errores tan graves como los de funcionalidad. Los problemas de rendimiento son provocados por causas que pueden clasificarse en dos categorías: predecibles e impredecibles. El test de performance tiene por objeto medir la capacidad de proceso del sistema. Este debe ser diseñado de manera que permita medir la capacidad del sistema a

diferentes cargas. Lo que se desea medir es ubicación de almacenamiento, utilización de CPU, velocidad de respuesta, etc. Los valores medidos son comparados con los valores requeridos

Pruebas de mutación

Las pruebas de mutación son pruebas para medir la calidad del código a largo plazo. Estas pruebas consisten en cambiar (mutar) ciertos puntos del código fuente y ver si se detectan estos cambios (errores) mediante las pruebas unitarias. Las pruebas de mutación son, por tanto, un sistema doble de calidad que nos permite comprobar tanto la eficacia de nuestros tests unitarios y respaldar la cobertura de código exigida en nuestros proyectos, como la robustez del código testeado mediante esos tests unitarios

Pruebas de Stress

La prueba de estrés es una actividad de prueba de software que determina la solidez del software al probar más allá de los límites de la operación normal con el objetivo de prever escenarios de riesgos ante cargas extremas o estacionales. Las pruebas de estrés son particularmente importantes para el software de "misión crítica", pero se utilizan para todo tipo de software. Un tipo especial de stress test es el de "sobrecarga" y se relaciona con el test de performance. El objetivo no es esperar que el sistema maneje estos casos, sino verificar que no se detiene ante estas circunstancias. El sistema debe superar picos ocasionales de carga; también es interesante verificar cuánto decae la performance en estos casos

Pruebas de Penetración (y análisis de seguridad)

Pruebas de penetración: Las pruebas de penetración, también conocidas como pruebas éticas de hacking, son una forma específica de Testing de Software que se centra en evaluar la seguridad de un sistema identificando y explotando sus vulnerabilidades. Estas pruebas ayudan a identificar las debilidades de seguridad y permiten a los ingenieros de software tomar medidas para mejorar la seguridad del sistema.

Análisis de seguridad estática y dinámica: El análisis de seguridad estática y dinámica es una parte integral del Testing de Software enfocado en la identificación de posibles vulnerabilidades en el código y en tiempo de ejecución. Estas técnicas pueden revelar problemas de seguridad como inyecciones de código, desbordamientos de búfer, fugas de información y muchos otros. Al llevar a cabo estas pruebas de seguridad, los ingenieros de software pueden detectar y corregir fallas que podrían ser aprovechadas por atacantes.

Plan de pruebas

El Plan de Pruebas es una guía para gestionar las distintas pruebas que se realizarán a lo largo del proyecto. Como tal permite definir cuál será la estrategia de trabajo para la realización de las pruebas, quiénes y cuándo estarán involucrados y también detallar una a una las pruebas a realizar. Además de los casos de prueba, debiera incluir información respecto de los tipos de prueba, las herramientas requeridas, el ambiente para la ejecución de pruebas, el plan de trabajo (cronograma con fechas y responsables), las políticas de trabajo y comunicación, estrategia y metodología de prueba, entre otros. Sobre estos últimos aspectos, algunas cuestiones a no olvidar definir en el Plan de Pruebas son:

1. Determinar cuáles son los niveles de pruebas que abarca el Plan, que justamente son aquellos tipos de pruebas y su relación (por eso se habla de "niveles" - ver la V de Test-).
2. Establecer cuál será la frecuencia de ejecución de pruebas.
3. Definir cuáles son las técnicas y modo de ejecución de las pruebas (manual o automática) y herramientas que se utilizarán.
4. Identificar claramente cuáles son los criterios de éxito y finalización de las pruebas.
5. Reconocer y establecer cuáles serán los esquemas de comunicación de los resultados de las pruebas y cuáles serán los mecanismos para el reporte de incidencias.
6. Especificar qué cantidad de personas ejecutarán las pruebas

Casos de prueba

Los Casos de Prueba se definen tempranamente en el proceso de desarrollo. Es útil hacerlo durante la etapa de definición y modelado de requerimientos. Luego, en la etapa específica de Testing pueden refinarse. Un caso de prueba debe contener como mínimo:

- id
- Nombre de la prueba (Identificación corta)
- Criterio de aceptación (paso a paso)
- Alcance (¿qué se prueba?)
- Tipo de prueba
- Resultado esperado
- Procedimiento
- Caso de uso relacionado
- Datos entrada

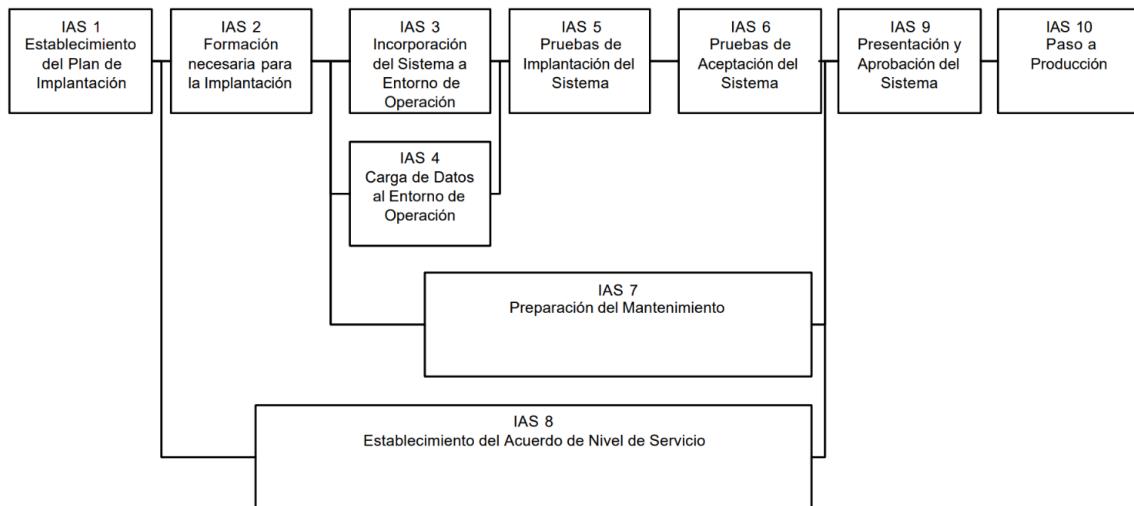
Un aspecto relevante a tener en cuenta es que las pruebas no sólo deben apuntar a mostrar que el dispositivo o sistema funciona o validar la su corrección, sino que el enfoque del tester debe pensar, describir y ejecutar también pruebas que sirvan para "romper". Es decir, probar qué pasaría si alguien "por error" "o por malicia" hace tal o cual cosa. Más adelante, al momento de la ejecución de la prueba, y como registro de las mismas, se deberá informar, como mínimo:

- Fecha de realización de la prueba
- Iteración (empezando en 1 y señalando la cantidad de veces que se ejecuta en caso de que haya errores)
- Resultado obtenido
- Responsable de la prueba
- Observaciones / Descripción / Conclusión

Proceso de Implantación

El proceso de implantación constituye el último eslabón en el Proceso de Desarrollo, luego de la construcción y las actividades de Testing. Es el momento de poner a funcionar la solución en su entorno real de uso y operación. Luego de todo el trabajo requerido para llegar a este punto, la fase de implantación puede ser compleja. Como con los procesos de desarrollo y prueba, la complejidad depende, entre otras cosas, de las características de la tecnología, el ámbito de aplicación y el tamaño del sistema y su distribución. La estrategia de implantación, que debe ser explicitada en el plan de implantación, que fijará los pasos necesarios para la prueba y aceptación de la solución construida. Se deben llevar a cabo actividades de preparación de la infraestructura, armado de los equipos y su capacitación para la puesta en marcha y transición hacia el lanzamiento productivo (personal de la empresa proveedora como los key users del lado cliente). Se especifican aquí los compromisos y condiciones del posterior monitoreo y mantenimiento. Se llevan a cabo las actividades de migración de datos y/o carga inicial de datos (dependiendo del caso). Se lleva a cabo la puesta en marcha en función de la estrategia definida y se hace seguimiento activo de la misma. Finalmente, se hacen las validaciones de homologación y aceptación y se llevan a cabo las actividades de transferencia de responsabilidades. Con el fin del proceso de implantación se consideran concluidas todas las actividades del ciclo de vida de construcción de la solución. A partir de ese momento, el producto de software está en modo productivo. Por supuesto, existen diferentes metodologías que ordenan y estructuran todas las actividades, materiales y recursos involucrados en este proceso.

El proceso de implantación de soluciones software refiere a la etapa en la cual se distribuye y pone en funcionamiento un sistema de software en un entorno de producción. Implica una serie de actividades planificadas y ejecutadas para asegurar que el sistema esté listo y operativo de manera efectiva. El proceso de implantación de sistemas de software es crucial para el éxito y la adopción efectiva de un nuevo sistema en una organización. Una planificación adecuada, la gestión de cambios eficiente y el enfoque en la satisfacción de los usuarios finales son elementos clave para lograr una implantación exitosa. A continuación se presentan las actividades comunes del proceso de Implantación:



1. ESTABLECIMIENTO DEL PLAN DE IMPLANTACIÓN

En esta actividad se revisa la estrategia de implantación para el sistema, establecida inicialmente en el proceso Una vez estudiado el alcance y los condicionantes de la implantación, se decide si ésta se puede llevar a cabo. Será preciso establecer, en su caso, la estrategia que se concretará de forma definitiva en el plan de implantación. Se constituye el equipo de implantación, determinando los recursos humanos necesarios para la propia instalación del sistema, para las pruebas de implantación y aceptación, y para la preparación del mantenimiento. Se identifican, para cada uno de ellos, sus perfiles y niveles de responsabilidad.

2. FORMACIÓN NECESARIA PARA LA IMPLANTACIÓN

Se determina la formación necesaria para el equipo de implantación, en función de los distintos perfiles y niveles de responsabilidad identificados en la actividad anterior. Para ello, se establece un plan de formación que incluye los esquemas de formación correspondientes, los recursos humanos y de infraestructura requeridos para llevarlo a cabo, así como una planificación que queda reflejada en el plan de formación.

3. INCORPORACIÓN DEL SISTEMA AL ENTORNO DE OPERACIÓN

En esta actividad se realizan todas las tareas necesarias para la incorporación del sistema al entorno de operación en el que se van a llevar a cabo las pruebas de implantación y aceptación del sistema

Mientras que las pruebas unitarias, de integración y del sistema se pueden ejecutar en un entorno distinto de aquél en el que finalmente se implantará, las pruebas de implantación y aceptación del sistema deben ejecutarse en el entorno real de operación. El propósito es comprobar que el sistema satisface todos los requisitos especificados por el usuario en las mismas condiciones que cuando se inicie la producción.

4. CARGA DE DATOS AL ENTORNO DE OPERACIÓN

Teniendo en cuenta que los sistemas de información que forman parte del sistema a implantar pueden mejorar, ampliar o sustituir a otros ya existentes en la organización, puede ser necesaria una carga inicial y/o una migración de datos cuyo alcance dependerá de las características y cobertura de cada sistema de información implicado

5. PRUEBAS DE IMPLANTACIÓN DEL SISTEMA

La finalidad de las pruebas de implantación es doble:

- Comprobar el funcionamiento correcto del mismo en el entorno de operación.
- Permitir que el usuario determine, desde el punto de vista de operación, la aceptación del sistema instalado en su entorno real, según el cumplimiento de los requisitos especificados.

Para ello, el responsable de implantación revisa el plan de pruebas de implantación y los criterios de aceptación del sistema, previamente elaborados. Las pruebas las realizan los técnicos de sistemas y de operación, que forman parte del grupo de usuarios técnicos que ha recibido la formación necesaria para llevarlas a cabo

6. PRUEBAS DE ACEPTACIÓN DEL SISTEMA

Las pruebas de aceptación tienen como fin validar que el sistema cumple los requisitos básicos de funcionamiento esperado y permitir que el usuario determine la aceptación del sistema. Por este motivo, estas pruebas son realizadas por el usuario final

7. PREPARACIÓN DEL MANTENIMIENTO DEL SISTEMA

El objetivo de esta actividad es permitir que el equipo que va a asumir el mantenimiento del sistema esté familiarizado con él antes de que el sistema pase a producción. Para conseguir este objetivo, se ha considerado al responsable de mantenimiento como parte integrante del equipo de implantación. Por lo tanto, se habrá tenido en cuenta su perfil al elaborar el esquema de formación correspondiente

8: ESTABLECIMIENTO DEL ACUERDO DE NIVEL DE SERVICIO

Antes de la aprobación definitiva del sistema por parte del Comité de Dirección es conveniente:

- Determinar los servicios que requiere el mismo.
- Especificar los niveles de servicio con los que se va a valorar la calidad de ese prestación.
- Definir qué compromisos se adquieren con la entrega del sistema.

Para ello, en primer lugar, se negocia entre los máximos responsables del usuario y de operación qué servicios y de qué tipo se van a prestar. Una vez acordados, se detallan los niveles de servicio definiendo sus propiedades funcionales y de calidad.

9. PRESENTACIÓN Y APROBACIÓN DEL SISTEMA

Una vez que se han efectuado las pruebas de implantación y de aceptación, y que se ha fijado el acuerdo de nivel de servicio, el Comité de Dirección debe formalizar la aprobación del sistema

10. PASO A PRODUCCIÓN

Esta actividad tiene como objetivo establecer el punto de inicio en que el sistema pasa a producción, se traspasa la responsabilidad al equipo de mantenimiento y se empiezan a dar los servicios establecidos en el acuerdo de nivel de servicio, una vez que el Comité de Dirección ha aprobado el sistema. Para ello es necesario que, después de haber realizado las pruebas de implantación y de aceptación del sistema, se disponga del entorno de producción perfectamente instalado en cuanto a hardware y software de base, componentes del nuevo sistema y procedimientos manuales y automáticos

Proceso de Implantación

Preparación del Entorno

Se realiza una planificación detallada que abarca los recursos necesarios, los plazos, la asignación de tareas y las responsabilidades del equipo de implantación. Se definen los objetivos y se establece una estrategia para la implementación.

Pruebas y Verificación (para la implantación)

Con independencia de las pruebas propias del Proceso de PRUEBAS que se llevan a cabo en un ambiente controlado, el Proceso de IMPLANTACIÓN también tiene algunas actividades de este tipo para asegurar, frente al destinatario de la solución, que el sistema funcione correctamente y cumpla con los requisitos especificados en el entorno real de operación (Producción). Esto implica, además, la participación activa de key users del lado del cliente.

Migración de Datos

Si fuera requerido, se debe realizar la migración de los datos existentes al nuevo sistema. Esto puede implicar la extracción, transformación y carga de datos desde sistemas antiguos o de origen a la nueva plataforma.

Formación y Capacitación

Se proporciona formación y capacitación adecuada a los key users del lado del cliente como a los usuarios finales para asegurar que estén familiarizados con el sistema y puedan utilizarlo de manera efectiva. Esto puede incluir sesiones de capacitación, materiales de referencia y soporte continuo durante la transición.

Estrategia de Implantación

Dependiendo del alcance y la complejidad del sistema, la implantación puede llevarse a cabo de forma gradual, en paralelo (cuando existe un sistema que debe ser reemplazado) o one-shot (o alguna definida específicamente ad-hoc). La implantación gradual es una puesta en marcha progresiva (por sectores, funciones, módulos, etc). En paralelo, se pone en funcionamiento el sistema mientras sigue operativo el sistema existente. Y one-shot implica poner a funcionar el nuevo sistema de una sola vez.

Soporte

Durante el proceso de Implantación debe acordarse cuál y cómo será el soporte luego de la puesta en marcha. Lo habitual es que después de la implantación, se brinde soporte continuo para resolver problemas, atender consultas y realizar ajustes necesarios. Esto puede incluir la resolución de errores, la aplicación de actualizaciones y la mejora del sistema según las necesidades y los comentarios de los usuarios.

Evaluación y Seguimiento

Se realiza una evaluación posterior a la implantación para verificar el cumplimiento de los objetivos establecidos, identificar áreas de mejora y recopilar retroalimentación de los usuarios finales. El proceso de implantación puede ajustarse en función de los resultados de esta evaluación.

Consideraciones de seguridad

La seguridad, asociada a la protección y acceso a la información y los recursos, se puede considerar desde dos enfoques, seguridad física y seguridad lógica. Estos son aspectos fundamentales a considerar durante la implantación de sistemas. Aquí se mencionan algunos aspectos relevantes para cada uno de estos ámbitos

Consideraciones de Seguridad Física

Acceso restringido: Limitar el acceso físico a las instalaciones donde se encuentran los equipos y servidores del sistema, utilizando medidas como cerraduras, tarjetas de acceso, cámaras de seguridad y guardias de seguridad.

Protección de equipos: Asegurar que los equipos y servidores estén ubicados en lugares seguros y protegidos contra robos, daños físicos y desastres naturales. Esto puede implicar el uso de sistemas de alarmas y sistemas de respaldo de energía. O considerar las recomendaciones para crear un Data Center.

Respaldo y almacenamiento seguro de datos: Definir políticas para realizar copias de seguridad (y su restauración) periódicas de los datos y almacenarlas en ubicaciones seguras, fuera del sitio, para protegerlos contra pérdidas o daños. También se pueden aplicar técnicas de cifrado para proteger la confidencialidad de los datos en caso de robo o acceso no autorizado.

Protección contra intrusiones físicas: Implementar sistemas de detección y prevención de intrusiones físicas, como alarmas de intrusión, sistemas de vigilancia por video y controles de acceso físico, para evitar el acceso no autorizado a las instalaciones.

Consideraciones de Seguridad Lógica

Autenticación y autorización: Implementar mecanismos sólidos de autenticación y autorización para garantizar que sólo los usuarios autorizados tengan acceso al sistema. Esto puede incluir el uso de contraseñas seguras, autenticación multifactor, certificados digitales y políticas de acceso basadas en roles, etc.

Cifrado de datos: Utilizar técnicas de cifrado para proteger la confidencialidad de los datos durante su almacenamiento y transmisión. Esto incluye el uso de protocolos seguros de comunicación, cifrado de bases de datos y cifrado de archivos sensibles.

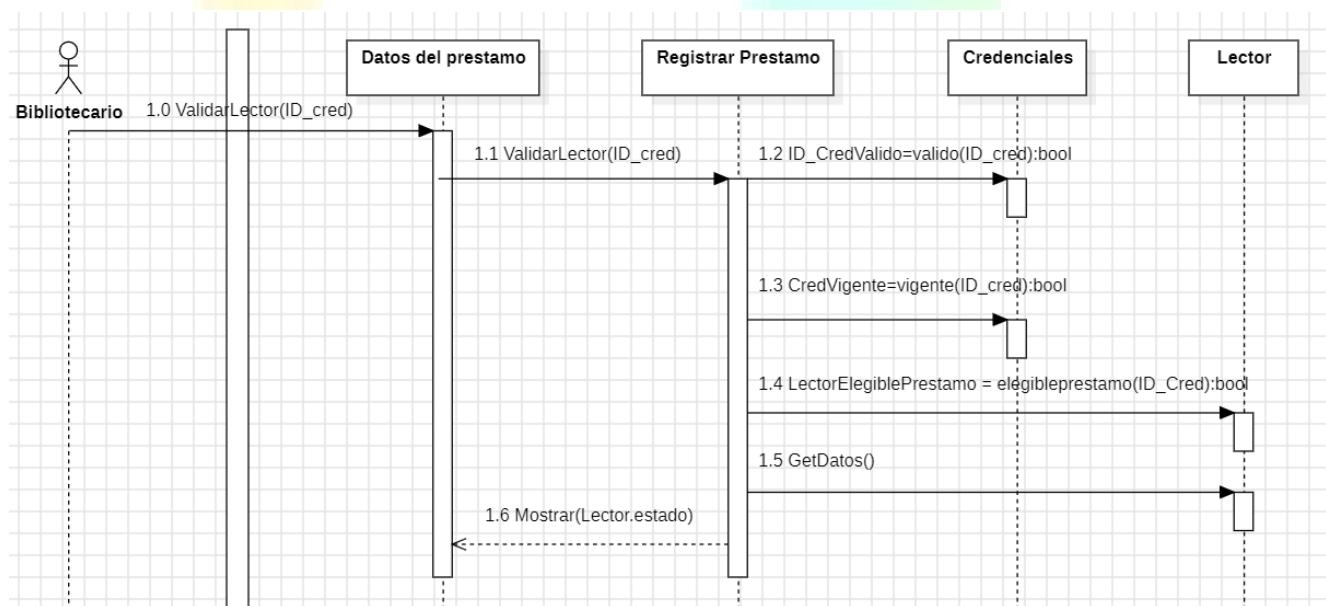
Auditoría y registro de actividades: Implementar un sistema de registro de actividades y auditoría que registre los eventos relevantes del sistema. Esto permite rastrear y monitorear las acciones realizadas por los usuarios y detectar posibles anomalías o intentos de acceso no autorizado.

Actualizaciones y parches de seguridad: Mantener los sistemas actualizados con los últimos parches de seguridad y actualizaciones de software para corregir posibles vulnerabilidades conocidas.

Firewalls y sistemas de detección de intrusiones: Implementar firewalls y sistemas de detección de intrusiones para monitorear y proteger el sistema contra posibles ataques externos o internos.

Preguntas segundo parcial

1. ¿Qué es V & V ? Explique . ¿Cuál es su relevancia en el proceso de desarrollo de software ?
2. Diseño modular. ¿Qué relación existe entre complejidad y accesibilidad ? Explique y ejemplifique según corresponda.
3. Describa el proceso de diseño y explique su impacto en la calidad de software
4. a) Enumere explique y ejemplifique al menos cinco tipos de TEST
 b) Si un componente funciona y pasa al testing está libre de errores ? Justifique
5. A partir del diagrama de secuencia de la figura . Determine
 - El requerimiento que modula
 - El caso de uso del cual deriva y el flujo que modela
 - El tipo de cada objeto representado por los bloques del diseño



Otras preguntas :

1. Explicar que es Modelo -V
2. Explicar que es la calidad de software (producto) y la calidad del proceso (ciclo de vida)
3. ¿Que es la gestión de riesgos y por que debe hacerse ? . Proponer 2 riesgos técnicos y uno de negocio
4. ¿Cuales son los objetivos del diseño ? Explicar los procesos de diseño de datos y diseño arquitectura
5. Explique la relación de testeo y calidad (QS) y su evolución
6. ¿Por que debe elaborarse un plan de implantación ?