INSTITUTO SUPERIOR TECNOLÓGICO

JAPÓN

Amor al conocimiento

GUÍA METODOLÓGICA

DESARROLLO DE SOFTWARE

DESARROLLO DE SOFTWARE

AUTOR: ING. EDDY VARGAS

2020

1. IDENTIFICACIÓN DE

Nombre de la Asignatura: METODOLOGÍA DE DESARROLLO DE SOFTWARE	**Componentes del Aprendizaje**	Componente docencia: 54 Componente de prácticas de aprendizaje: 53 Componente de aprendizaje autónomo: 35
Resultado del Aprendizaje: Identifica metodologías describiendo sus características, seleccionando la adecuada. Aplica una metodología en el proyecto de desarrollo de software. Interpreta metodologías utilizadas en proyectos de desarrollo de software Conocer los paradigmas de desarrollo. Reconocer las metodologías pesadas de desarrollo de software, reconocer la metodología de desarrollo agiles de software		
Docente de Implementación:		
Ing. Eddy Vargas	**Duración:** 142 horas	

Unidades	Competencia	Resultados de Aprendizaje	Actividades	Tiempo de Ejecución

1. ANALIZAR METODOLOGÍAS DE DESARROLLO DE SOFTWARE	1.1 Introducción a metodología de desarrollo de software 1.2 Selección del sistema de información para el desarrollo del software 1.3 Recopilación de la información del desarrollo de software 1.4 Procesamiento de la información recopilada	COGNITIVO: Define procesamiento automatizado de datos Identifica estrategia de búsqueda de información Identifica métodos para la recolección de requerimientos Identifica requerimientos funcionales, no funcionales y restricciones Define etapas del ciclo de vida del software ACTITUDINAL: Analiza características del software Reconoce procesos de software Define etapas del ciclo de vida del software	Exposición visual con diapositivas de conceptos básicos de ingeniería de software Lluvia de ideas sobre procesos de software. Exposición visual de importancia y técnicas de búsqueda y levantamiento de requerimientos	**10h**

	1.5 Análisis de la problemática del sistema actual.	COGNITIVO: Identifica las condiciones iniciales del proyecto ACTITUDINAL: Especifica el alcance del proyecto de software	Ejemplificación especificación de requerimientos en un sistema informático Lectura comentada importancia de este proceso.	**4h**
2. MODELAR LA ARQUITECTURA DE SOFTWARE	2.1 Aplicación de la metodología del desarrollo de software 2.2 Aplicación de los diagramas de casos de uso	COGNITIVO: Identifica las diferentes metodologías describiendo sus características PROCEDIMENTAL: Diseña los requerimientos funcionales del producto de software mediante un diagrama	Técnicas expositivas: Clases, teóricas, metodologías Agiles y Pesadas. Resumen tipos de metodologías de desarrollo de software. Discusión del tema, preguntas por parte de los estudiantes	**5h**

	2.3 Aplicación de los diagramas de comportamiento 2.4 Aplicación de las clases	COGNITIVO: Identifica los elementos de un diagrama de estado Define clases en un proyecto de software PROCEDIMENTAL: Diseña los procesos a programar en el proyecto de software. ACTITUDINAL: Describe la relación entre las clases de un proyecto mediante diagrama de clases	Técnicas expositivas: Clases, teóricas. Ejemplos de diagramas de Interacción y de clases. Taller de Aplicación de Actividades de Diseño de Software	**7h**
	2.5 Aplicación de los diagramas de estados y actividades 2.6 Aplicación de los diagramas de despliegue	PROCEDIMENTAL: Diseña el comportamiento de los objetos y operaciones mediante diagramas Diseña la arquitectura del sistema como el despliegue de los artefactos de software en nodos	Técnicas expositivas: Clases, teóricas. Ejemplos de diagramas de estados y actividades. Exposición Metodologías: XP y Crystal por grupos de estudiantes	**9h**

	2.7 Diseño de la Base de Datos del Software	COGNITIVO: Identifica todos los requisitos de datos y transacciones en una base de datos PROCEDIMENTAL Diseña los modelos conceptual, lógico y físico de una base de datos ACTITUDINAL: Determina metodología para el diseño de una base de datos	Técnicas expositivas: clases teóricas. Estudio y el trabajo autónomo Exposición Metodología: Scrum por los estudiantes. Publicación de video: Diseño lógico de una base de datos Entidad Relación sistema de una ferretería. Técnicas de discusión en clase	**4h**

	2.8 Diseño de Interfaces del software 2.9 Diseño del menú del Software 2.10 Diseño de la interfaz acceso del software	COGNITIVO: Identifica modelos de descripción de la interfaz Identifica etapas del proceso de diseño de una buena interfaz PROCEDIMENTAL Utiliza herramientas CASE como apoyo en el diseño de interfaces y menús ACTITUDINAL: Selecciona objetos de la interfaz según su tipo en concordancia con la funcionalidad del sistema	Exposición visual Criterios y modelos de una buena interfaz. Dialogo con los estudiantes sobre los componentes que se deben incluir en una interfaz. Estudio de casos Ejemplificación generación de menús de aplicaciones	**9h**
	2.11 Prueba demostrativa del Software 2.12 Proceso de recuperación	COGNITIVO: Identifica metodologías para desarrollo de pruebas de software Identifica técnicas de validación y verificación de software PROCEDIMENTAL Planifica políticas de seguridad en el ciclo de vida de desarrollo del software	Exposición con estimulación verbal y multimedios de conceptos y técnicas de testeo o pruebas de software. Lectura comentada de un plan de pruebas y de los roles de los miembros del equipo de testeo.	**7h**

2. CONOCIMIENTOS PREVIOS Y RELACIONAD

Co-requisitos

3. UNIDADES TEÓRICAS

- **Desarrollo de las Unidades de Aprendizaje (contenidos)**

A. Base Teórica

UNIDAD 1

1. METODOLOGÍA DEL DESARROLLO

1.1 Introducción a metodología de desarrollo de software

¿Qué es la ingeniería de software?

La ingeniería de software se ha definido por varios autores. Según Ian Sommerville, considerado uno de los padres de la ingeniería de software, la ingeniería de software "es una disciplina de la ingeniería que comprende todos los aspectos de la producción del software" [Somerville, 2004].

La IEEE define a la ingeniería de software como "la aplicación de un enfoque sistemático, disciplinado y cuantificable al desarrollo, operación y mantenimiento del software".

Una de las definiciones más interesantes es la de Bohem (1976): "ingeniería de software es la aplicación práctica del conocimiento científico al diseño y construcción de programas de computadora y a la documentación asociada requerida para desarrollar, operar y mantenerlos. Se conoce también como desarrollo de software o producción de software". Definimos la ingeniería de software como:

Una disciplina en la que se aplican técnicas y principios de forma sistemática en el desarrollo de sistemas de software para garantizar su calidad.

¿Qué es una metodología de desarrollo de software?

Una metodología de desarrollo de software no es mas que una serie de pasos que se realizan de forma rigurosa tal que su resultado a partir de unos requisitos nuevos o modificados sea un software nuevo o modificado. Se puede ver como una caja negra, como muestra la siguiente imagen:

Esta nos permite responder a cuatro preguntas básicas:

¿Qué hacen los participantes del proyecto?

¿Quién participa en su desarrollo?

¿Cómo participan los anteriores?

¿Cuándo se realizan sus partes?

Personas realizan actividades según sus roles a partir de resultados parciales (conocidos como artefactos) que son partes necesarias para realizarlas o resultados parciales o totales de éstas. Estas actividades toman un orden y una estructura temporal entrelazada definida por la metodología en cuestión.

2. ¿Qué nos aporta una metodología de desarrollo de software?

Garantía de calidad.

Calidad = requisitos satisfechos.

Medir la calidad de un producto basándonos en los requisitos iniciales.

Forma de estimar y controlar costos. Así podemos saber cuanto vamos a tardar en realizarlo y si nos sale o no rentable llevarlo a cabo antes de realizar la inversión completa de tiempo, dinero y esfuerzo.

También evita una gran parte de los esfuerzos perdidos en rectificar fallos que se pueden evitar utilizando una metodología adecuada.

Proceso estructurado nos organiza la forma en el que el proyecto va a ser realizado, obligado a revisar los resultados sean los correctos antes de proseguir y marcando metas intermedias para controlar el avance del proyecto. Así pues, se logra una mayor eficiencia de recursos, es decir, se invierte lo mínimo para obtener lo máximo a cambio. Para que el proceso sea efectivo, éste debe ser aplicado con rigor.

3. ¿Qué metodología escoger?

Existen dos tipos principales de metodologías, las Ligeras y las Pesadas.

Metodologías Ligeras: Son extremadamente prácticas que generalmente obvian gran parte de la documentación y están más preparadas para utilizarse en proyectos cuyos requisitos cambiarán constantemente durante todo el proceso.

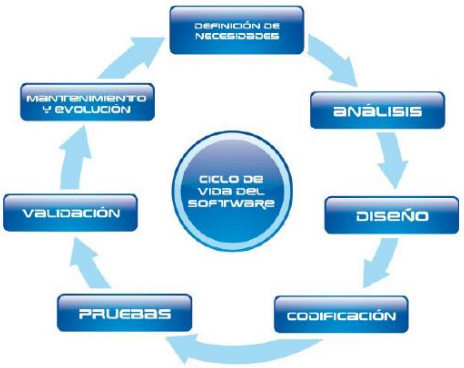
Ejemplos: eXtreme Programming (XP), SCRUM y cristal

Metodologías Pesadas: son metodologías donde todo está mucho más controlado y se genera muchísima documentación antes de proceder a implementar el proyecto, con mucho mayor peso del análisis y el diseño sobre el proyecto. Estas últimas son más indicadas para proyectos grandes o cuyo rendimiento y nivel de calidad son críticos para el éxito de éste.

Ejemplos: Rational Unified Process (RUP), ICONIX y Métrica 3.

4. Ciclo de vida del Software

Es el conjunto de etapas que sigue un proyecto de software desde su concepción hasta su finalización y cierre, inclusive los mantenimientos (cambios o ajustes que puedan producirse una vez está implementado, nuevas versiones, etc.).



Ejemplo de ciclo de vida del software. Éste se inicia con la definición de necesidades y sigue un flujo cíclico hasta retornar al punto de origen.

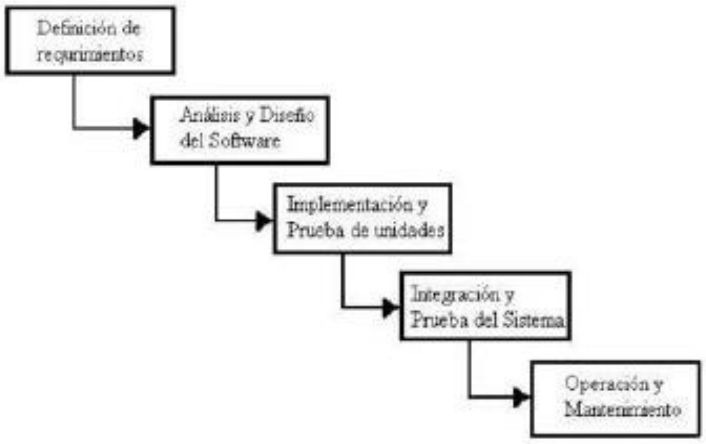
5. Modelos de Desarrollo de Software

Estos modelos de desarrollo son: Cascada, Prototipaje, Espiral, en V, Incremental.

Modelo Cascada:

Como se puede ver, se refiere a un enfoque secuencial

En este caso, cualquier fallo de las fases anteriores serán arreglados en la fase actual, y se procederá siempre hacia adelante, sin volver a pasar por ninguna de las fases anteriores.



Ejemplos: Empresas que utilizan la Metodología Cascada

En la industria del desarrollo de juegos, todos los requisitos suelen desarrollarse y perfeccionarse antes de que comience el proyecto, lo que permite que el equipo de desarrollo trabaje en el modelo de cascada pura. Una empresa que utiliza esta metodología aún es la empresa Capcom.

También se puede pensar en sistemas de salud en hospitales, sistemas bancarios, sistemas de control para procesos industriales que involucran químicos peligrosos, materiales nucleares o condiciones extremas, sistemas militares. Por ejemplo, el software de control para un transbordador espacial (no sería factible aplicar metodologías ágiles a tales sistemas, ya que ninguna de sus ventajas se puede aprovechar en dichos casos y todas sus desventajas pueden causar accidentes letales).

Otra empresa que implementa la metodología Cascada es BAE System. una compañía internacional de defensa aeroespacial y seguridad que ofrece una gama completa de productos y servicios para fuerzas aéreas, terrestres y navales, así como soluciones avanzadas de electrónica, seguridad y tecnología de la información y servicios de soporte al cliente. Empresa lider en capacidades de ciberinteligencia y seguridad para agencias gubernamentales, y un proveedor en crecimiento de capacidades de seguridad cibernética y de red para clientes comerciales.

Modelo Prototipaje:

En este caso el enfoque es iterativo.

Se basa en realizar pequeños prototipos finales de la aplicación de forma que sus funcionalidades se construyen encima de la versión anterior, hasta llegar al producto definitivo y su entrega al cliente.

Este modelo principalmente se lo aplica cuando un cliente define un conjunto de objetivos generales para el software a desarrollarse sin delimitar detalladamente los requisitos de entrada procesamiento y salida.

MODELO DE CONSTRUCCIÓN DE PROTOTIPOS

Características:

El prototipo es una aplicación que funciona

Los prototipos se crean con rapidez

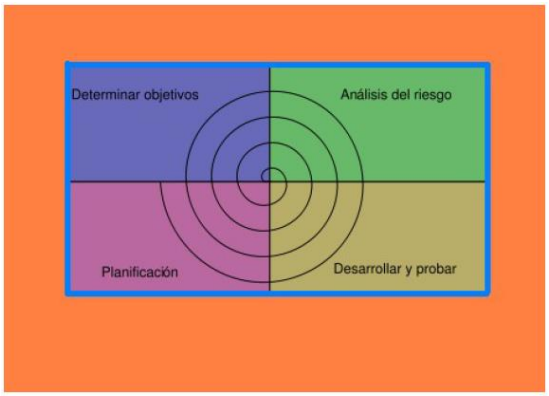
Los prototipos evolucionan a través de un proceso iterativo.

Los prototipos tienen un costo bajo de desarrollo

Ejemplo:

Sistema de inventarios para un centro de Turismo, las etapas son definición del problema, construcción del prototipo inicial, verificación y requerimientos, modificación del prototipo, diseño detallado y documentación para programación y mantenimiento, etc.

Modelo Espiral:



Se refiere a un enfoque combinado mas complejo que los anteriores.

Se visualiza el proceso como una espiral. Cada rotación representa una pequeña cascada y la distancia radial representa el volumen del proyecto. A mas avanzado, mayor volumen.

El costo es bastante impredecible debido al volumen del proyecto, con lo cual no suele ser un enfoque viable económicamente. Sin embargo, tiene sus usos cuando se realizan proyectos críticos. Como un gran sistema operativo, temas de control aéreo, militares o espaciales, ya que prima la calidad sobre el coste principalmente. Un solo fallo puede ser motivo de su completo fracaso.

1.2 Selección de sistema de información para el desarrollo del software

Causas por las cuales las organizaciones toman la decisión estratégica de abordar proyectos sobre sistemas de información en función de los parámetros mejorables de ésta.

Capacidad	Control	Comunicación	Costos	Ventajas Competitivas
Mayor velocidad de procesamiento	Mayor exactitud y mejora de la consistencia	Mejoras en la comunicación	Monitorización de costos	Atraer clientes
Incremento en el volumen		Integración de áreas de la empresa	Reducción de costos	Dejar fuera a la competencia
Recuperación más rápida de información				Mejores acuerdos con los proveedores
				Desarrollo de nuevos productos

Por estas causas es importante conocer como se deben iniciar este tipo de proyectos, así como las distintas formas de adquirir la información necesaria para su posterior realización

Inicio de Proyectos

Proceso de solicitud de proyecto

Es un formato no establecido dependiendo de la organización que contendrá información mínima para ser analizada por el comité.

El contenido de información será el siguiente:

• ¿Cúal es el problema?
• Detalles del problema.
• Importancia del problema.
• ¿Cúal es la solución aportada por el usuario?
• ¿En qué medida será de ayuda un sistema de información?
• ¿Qué otras personas conocen el problema y se puede contactar?

Fuentes de solicitud del Proyecto

Existen cuatro fuentes primarias de solicitudes de proyectos. Los ejecutivos, los jefes de departamento, los analistas de sistema y entes externos a la Organización.

Los jefes de departamento buscan mejorar la eficiencia del trabajo o reducir costes en su departamento, implantando para ello un sistema informatizado, sin considerar la interacción con otros departamentos.

Los directivos plantean proyectos globales a toda la Organización, normalmente multidepartamentales con un periodo de desarrollo más amplio, y normalmente asociado a políticas de empresa.

Los analistas de sistemas buscan áreas donde desarrollar proyectos, normalmente para la mejora de un departamento. El hecho de no partir la propuesta de proyecto por el jefe del departamento, obedece a un mejor conocimiento de la tecnología y las posibilidades de los equipos por parte del analista de sistemas.

Manejo de proceso de selección y revisión de proyectos

Se generan muchas solicitudes para el desarrollo de sistemas de las que las Organizaciones pueden emprender, obliga a un proceso de selección y priorización.

Uno de los métodos más comunes para revisar y seleccionar proyectos para su desarrollo es por medio de un comité.

Podemos hablar de varios tipos de comité:

• Comité directivo

Formado por ejecutivos, jefes de departamento y analista de sistemas. Normalmente corresponde con el personal con mayor responsabilidad, autoridad y con pocos miembros especialistas en sistemas. Reciben y evaluan las propuestas. Para la toma de decisión en firme necesitan mayor información que la contenida en la propuesta, por lo que deciden realizar un estudio preliminar.

• Comité de sistemas de información

El comité formado por profesionales del departamento de sistemas de información. Este comité aprueba o rechaza proyectos y fija las propiedades y también indica qué proyectos son más

importantes, dándoles atención inmediata. Esta composición del comité se puede utilizar para servicios rutinarios o mantenimiento de aplicaciones existentes.

• Comité de grupos de usuarios

En algunas organizaciones la responsabilidad de la toma de decisiones con respecto a los usuarios se deja en manos de éstos. Algunos departamentos contratan sus propios analistas y diseñadores. Pero puede ocurrir que varios departamentos pequeños que trabajan de forma independiente pasra alcanzar la misma meta pueden estar, de manera inconsciente, desperdiciando recursos y perdiendo la oportunidad para coordinar la planificación de un sistema de información compartido e integrado que podría beneficiar a toda la empresa

1.3 Recopilación de la información del desarrollo de software

¿Qué es determinar requerimientos?

Determinar requerimientos consiste en estudiar un sistema para conocer como trabaja y donde es necesario efectuar mejoras.

Un requerimiento es una característica que debe incluirse en el nuevo sistema. Esta puede ser la inclusión de determinada forma para capturar o procesar datos, producir información, controlar una actividad de la empresa o brindar soporte a los directivos.

Los analistas de sistemas no trabajan como directivos o empleados de los departamentos de usuarios, no tiene los mismos conocimientos, hechos y detalles que los usuarios y directivos de estas áreas. Por lo tanto el primer paso del analista es comprender la situación.

El primer paso del analista es comprender la situación actual.

Actividades de la determinación de Requerimientos

Anticipación de Requerimientos

Investigación de Requerimientos

Especificación de Requerimientos

La experiencia permite anticipar requerimientos para el nuevo sistema.

Es la actividad más importante, implica un riesgo ya que dependiendo de la calidad de requerimientos afectará al nuevo sistema.

Anticipación de Requerimientos	Investigación de Requerimientos	Especificación de Requerimientos
Preveer las características de sistema con base a la experiencia previa. Esto puede llevar al analista a investigar áreas y aspectos que de otra forma no serían tomados en cuenta.	Estudio y documentación del sistema actual utilizando para ello técnicas para hallar hechos, análisis de flujo de datos y análisis de decisión.	Análisis de los datos que describen el sistema para determinar qué tan bueno es su rendimiento, qué requerimientos deben satisfacer y las estrategias para alcanzarlos.
La experiencia permite anticipar requerimientos para el nuevo sistema.	Es la actividad más importante	Implica un riesgo ya que dependiendo de la calidad de requerimientos afectará al nuevo sistema.

Actividades en la determinación de Requerimientos

Los analistas estructuran su investigación en base a 4 preguntas:

1. ¿Cuál es el proceso básico de la empresa?
2. ¿Qué datos utiliza o produce este proceso?
3. ¿Qué frecuencia y volumen del proceso existe?
4. ¿Qué controles utiliza para su realización?

Estudiamos por separado cada una de estas cuatro preguntas:

¿Cuál es el proceso básico de la empresa?

Las siguientes preguntas se utilizan para adquirir la comprensión necesaria:

- ¿Cuál es la finalidad de esta actividad en la empresa?
- ¿Qué pasos se siguen para llevarla a cabo?
- ¿Donde se realizan estos pasos?
- ¿Quienes los realizan?
- ¿Cuánto tiempo tardan en efectuarlos?
- ¿Con cuanta frecuencia lo hacen?
- ¿Quienes emplean la información resultante?

¿Qué frecuencia y volumen de proceso existe?

Los analistas deben investigar con cuanta frecuencia se repite una actividad. Esto cambia mucho dependiendo de la actividad ya que por ejemplo el pago de la nómina se repite mensualmente o semanalmente pero el pago de impuestos es anualmente.

La manera más fácil de obtener esta información es identificar el objetivo de la actividad, es decir, cuál es la causa de la actividad.

El volumen de los procesos puede aumentar el tiempo de realización de las actividades, es decir la cantidad total de pasos que puede constar una actividad puede generar problemas aún ocurriendo con poca frecuencia.

¿Qué controles utiliza para su realización?

La falta o debilidad de los controles es un descubrimiento importante en cualquier investigación del sistema.

El analista debe examinar los métodos de control preguntando:

¿Quién se encarga de comparar lo realizado con los estándares?

¿Cómo se detectan los errores?

¿Cómo se corrigen los errores?

Ejemplo Práctico: Investigación de un sistema para reabastecer inventarios

La clase de preguntas que un analista debe hacer para adquirir la comprensión necesaria son:

¿Cuál es el proceso básico de la empresa? y para ello

• ¿cuál es la finalidad del sistema de reabastecimiento de inventarios?

asegurar la existencia de cantidades adecuadas de materiales y artículos en el almacén sin que sean excesivas y costosas.

• ¿qué pasos se siguen para reabastecer el inventario?

comprobar las existencias actuales y determinar las necesidades futuras y los tiempos óptimos para solicitar los pedidos.

• ¿dónde se realiza esta actividad?

el departamento de compras utiliza la información proporcionada por el personal de producción, ventas y almacén para hacer los pedidos y poder tomar decisiones anticipadas.

• ¿quienes realizan esta actividad?

los directores de compras aprueban todos los pedidos. Los directores de almacén escriben solicitudes de pedidos.

• ¿cuánto tiempo tarda esta actividad?

para pedidos simples tarda unos minutos y para pedidos de artículos nuevos o de determinadas características puede tardar un par de horas.

• ¿con cuánta frecuencia se realiza esta actividad? de forma continua.

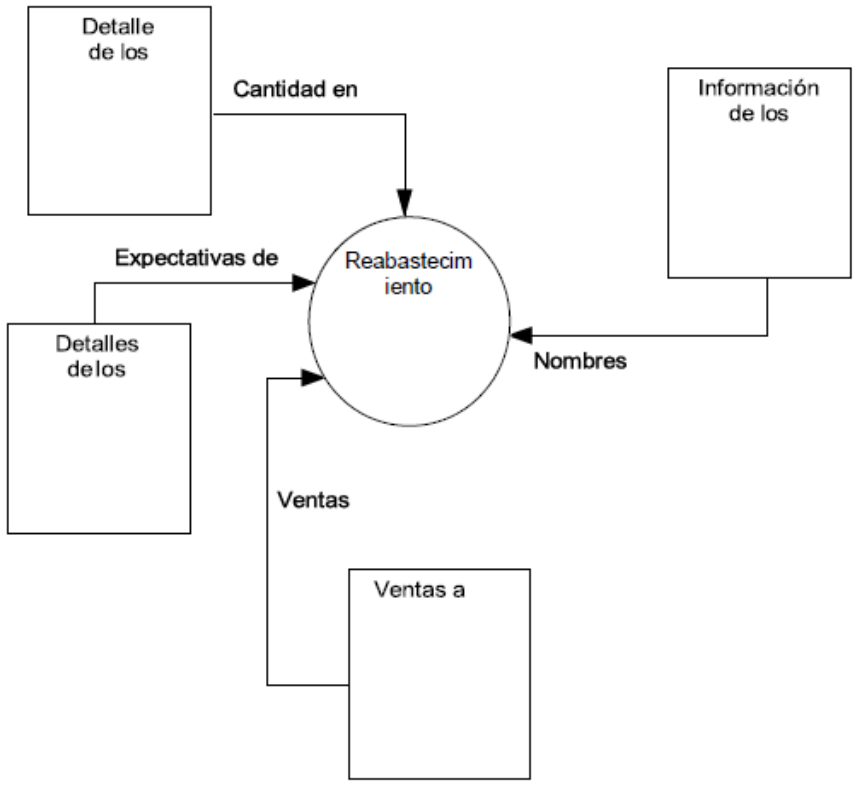
• ¿para qué utilizan la información resultante?

la información se emplea para administrar inventarios, hacer el seguimiento de las compras y pagos a proveedores.

¿Qué datos utiliza o produce este proceso?

Para reabastecer el inventario el flujo de información es:

Ejemplo de Reabastecimiento de inventario



¿Qué frecuencia y volumen de proceso existe?

Su frecuencia es de forma continuada pero el volumen de artículos manejados puede ser que aumente el tiempo necesario para completar la actividad.

¿Qué controles utiliza?

Tanto dirección de almacén como el personal del mismo llevan un seguimiento del proceso por lo tanto el control es bueno.

1.4 Procesamiento de la información recopilada

Obtención de Requerimientos

Se refiere a definir formalmente los requerimientos de la solución. Puede ser difícil debido a que:

Hay problemas de definición de alcances

Hay problemas de entendimiento entre los involucrados

Hay problemas de volatilidad (los requerimientos cambian con el tiempo)

Elaboración

Esta actividad expande y refina la información obtenida en la tarea de iniciación

Se enfoca en realizar modelos técnicos refinados de las funciones del software, características y limitantes.

Es básicamente una función de modelado. Se conduce a través de la definición de escenarios del usuario que describen la interacción del usuario final con el sistema

Se define el dominio del problema desde varios puntos de vista: información, funciones y comportamiento.

Especificación

Significa diferentes cosas para diferentes personas en el área de Ing. de software. Este es el producto de trabajo final de la ingeniería de requerimientos. Sirve como base para actividades subsecuentes.

Describe la función y desempeño de un sistema y las restricción que tiene.
Hay muchas técnicas para escribir especificaciones: diagramas, narraciones en prosa, modelos matemáticos, dibujos, etc.

Validación

Los requerimientos levantados deben ser evaluados en términos de congruencia y calidad. Se debe asegurar que la especificación concuerda con las expectativas del usuario y que no es ambigua.

Deben detectarse y corregirse errores, omisiones e inconsistencias con respecto a los estándares establecidos en el proyecto. El mecanismo común de validación es la revisión técnica formal.
Una lista de requerimientos, de preferencia organizados por función, y las restricciones de dominio que los afectan. Un conjunto de escenarios de uso que dan idea del uso del producto en diferentes condiciones operativas prototipos desarrollados

Administración de requerimientos
Actividades que ayudan al equipo de trabajo a identificar, controlar y seguir los requerimientos y cambios que ocurren en ellos a través de todo el proceso de desarrollo. La administración empieza con la identificación de cada requerimiento. Posteriormente se generan tablas que permitirán darles seguimiento. Algunas de éstas son:
Tablas de características
Tablas de fuentes
Tablas de dependencias
Tablas de subsistemas
Tablas de interfaces

Levantamiento de requerimientos

Incluye juntas colaborativas de definición, despliegue de funciones y descripción de escenarios
Los productos que genera son:
Una declaración de necesidades y factibilidades
Una declaración delimitada del alcance del sistema o producto
Una lista de consumidores, usuarios y otros involucrados (stakeholders) que participaron en la definición del documento
Una descripción del medio ambiente técnico del sistema
Una lista de requerimientos, de preferencia organizados por función, y las restricciones de dominio que los afecta.
Un conjunto de escenarios de uso que dan idea del uso del producto en diferentes condiciones operativas prototipos desarrollados.

Tipos de Requerimientos

Restricciones globales

Afectan a todo el producto y son determinadas por el usuario y los que administran el proyecto/producto.

1. Propósito del sistema
2. El cliente
3. El usuario
4. Convenciones para la nomenclatura y las definiciones
5. Hechos relevantes
6. Restricciones del proyecto
7. Suposiciones

Requerimientos Funcionales

Lo que el producto debe hacer

8. Alcance del sistema
9. Requerimientos Funcionales y de datos

Requerimientos No-Funcionales

Apoyan a las funciones, son las propiedades que el producto debe tener.

10. Apariencia y sensación
11. Usabilidad
12. Performance
13. Operabilidad
14. Mantenibilidad
15. Seguridad
16. Requerimientos Políticas
17. Requerimientos legales

5. Análisis de la problemática del sistema actual.

Los procedimientos de búsqueda para sistemas existentes deben incluir las numerosas fuentes de información de los sistemas para establecer una secuencia de búsqueda en esas fuentes. Comienzan con una serie de entrevistas iniciales para saber de qué se trata el problema. Estas entrevistas identifican funciones y a veces establecen los límites del problema. En este punto, generalmente los analistas estudian algunos de los informes. y documentos más importantes. Entonces diseñan un modelo de nivel superior y lo verifican durante el siguiente conjunto de entrevistas.

Una vez establecido un modelo de nivel superior satisfactorio, el analista comienza operaciones más detalladas. La búsqueda de esa información, más detallada empieza normalmente con entrevistas al personal de operación. Estas entrevistas establecen las fuentes detalladas de información, incluyendo programas, informes y manuales.

La clase de datos solicitados en el nivel inferior depende del sistema. Si se

perfecciona un sistema de ordenador, el analista de be examinar los sistema de ordenador y programas existentes. Si se van a mejorarlos procedimientos manuales, el analista realizará un examen detallado de los procedimientos actuales. En la mayoría de casos se debe examinar tanto el sistema de ordenador como el procedimiento de usuario.

El analista utiliza esta nueva información detallada para expandir el modelo de nivel superior que será verificado con el usuario.

Este procedimiento puede repetirse varias veces mientras se busca información cada vez más detallada. Esta subsecuencia de entrevistas y búsquedas se hará más crítica según el analista vaya aprendiendo cosas sobre el sistema. El analista comienza la identificación de los problemas del sistema y junto con el usuario establece los objetivos del nuevo sistema.

La iteración continuará hasta que el analista esté conforme con el modelo. Este pasa entonces por una serie de revisiones, comenzando con una revisión técnica para establecer formalmente la exactitud del modelo. finalmente, pasa a revisión por la dirección para que dé la conformidad a los objetivos del sistema y obtener los recursos para el desarrollo del nuevo sistema.

UNIDAD 2

2. MODELAR LA ARQUITECTURA DEL SOFTWARE

2.1 Aplicación de la metodología del desarrollo de software

Las metodologías de desarrollo de software son decisivas en el éxito o fracaso de un proyecto. En general las metodologías ponen en práctica una serie de procesos comunes, que son buenas prácticas para lograr los objetivos de negocio, costes, funcionalidad, sencillez, etc.

La elección de una metodología inadecuada o su mala aplicación pueden conducir a que el proyecto no llegue a su fin.

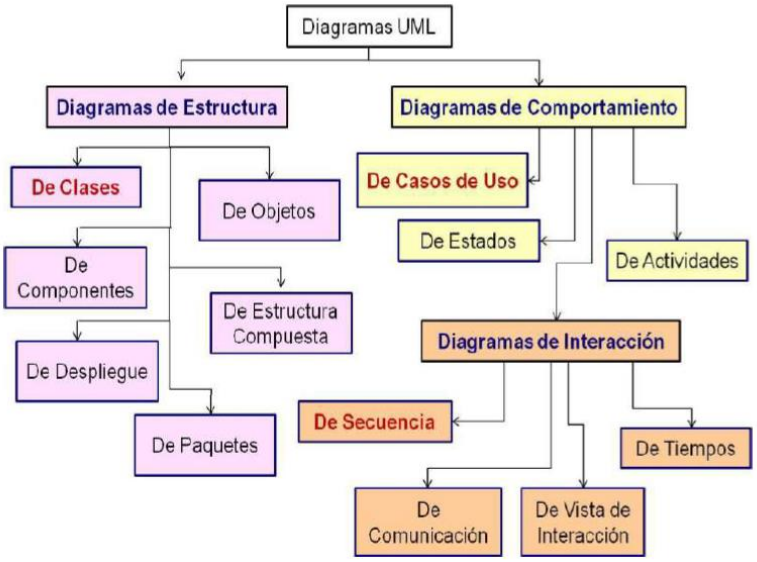
Para la selección e implantación de una metodología existe una importante labor de documentación previa y, a partir de ahí, escoger alguna de las metodologías vistas para aplicar en el día a día del desarrollo de software.

Usar las herramientas adecuadas ayuda a triunfar, pero no garantiza el éxito. Es fácil confundir el éxito/fracaso del proyecto, con el éxito/fracaso de la herramienta.

Del mismo modo, el hecho de no aplicar todas y cada una de las recomendaciones de la herramienta no conduce al fracaso. No es requisito indispensable para el éxito.

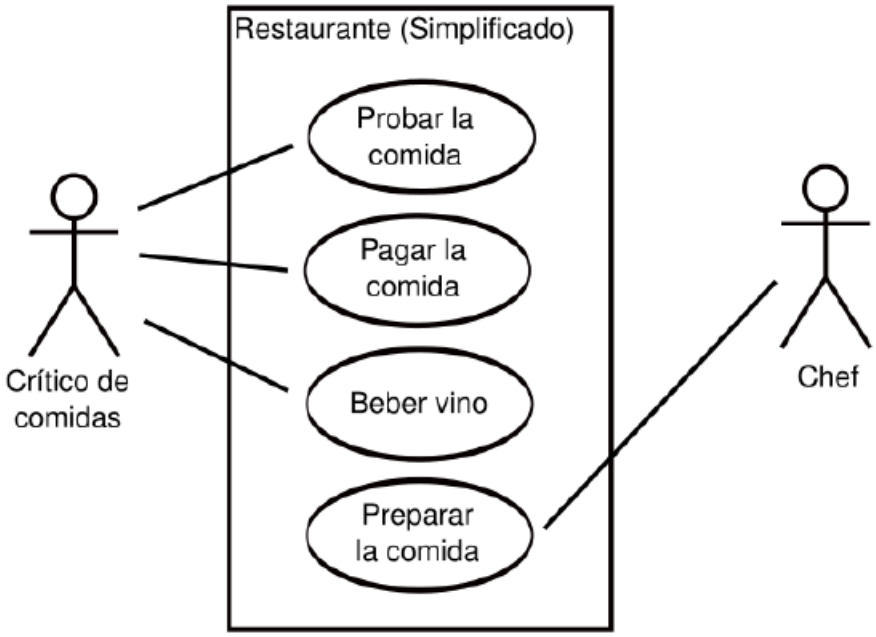
- Un proyecto puede triunfar debido a una gran herramienta.
- Un proyecto puede triunfar a pesar de una pésima herramienta.
- Un proyecto puede fallar debido a una pésima herramienta.
- Un proyecto puede fallar a pesar de una gran herramienta.

Los tipos de diagramas UML



2.2 Aplicación de los diagramas de casos de uso

Los diagramas de casos de uso describen las relaciones y las dependencias entre un grupo de casos de uso y los actores participantes en cada uno de los casos de uso.



Actor, acciones y sistema

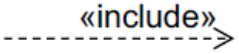
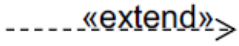
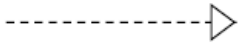
Los actores son los diferentes participantes en nuestro escenario. Son los que llevan a cabo las acciones que describimos en los casos de uso. Los actores se representan como estereotipos figuras de palo, conocido como “Stick man” o “monigote”.

¡Los actores no son necesariamente personas! Pueden representar organizaciones o incluso componentes del software.

Las acciones son aquello que hacen los actores con el sistema. Son funcionalidades que deberán ser luego implementadas.

En el centro, dibujamos el sistema. Es el escenario dónde nuestros actores interactúan. Lo representaremos con un rectángulo, y dibujaremos los casos de uso dentro del sistema.

Relaciones de los casos de Uso

Relación	Descripción	Notación
Asociación	Línea de comunicación entre un actor y un caso de uso en el que participa	
Generalización	Una relación entre un caso de uso general y un caso de uso más específico, que hereda y añade propiedades al caso de uso base	
Inclusión	Inserción de comportamiento adicional en un caso de uso base, que describe explícitamente la inserción	
Extensión	Inserción de comportamiento adicional en un caso de uso base que no tiene conocimiento sobre él	
Realización	Establece una relación entre el caso de uso y los diagramas que describen la funcionalidad del caso de uso	

Ejemplo: Casos de usos para una tienda On Line

Vamos a ver algunos casos de uso para una tienda online

1. Un vendedor de libros se conecta a la web para subir el artículo que tiene a la venta.

2. Un cliente se conecta a la página web de una tienda online para comprar un libro. Escribe el título del libro en la barra de búsqueda y busca el mejor precio entre los resultados . Cuando encuentra una oferta que le interesa, le añade al carrito de la compra. Tiene que haber iniciado una sesión para poder hacerlo.

Los casos de uso siempre deben comportar un verbo.

Este es un caso de uso muy sencillo. Muchas veces, la descripción será más larga si el proyecto es más complejo.

Ahora tratemos de traducir nuestro caso de uso a un diagrama de casos de uso.

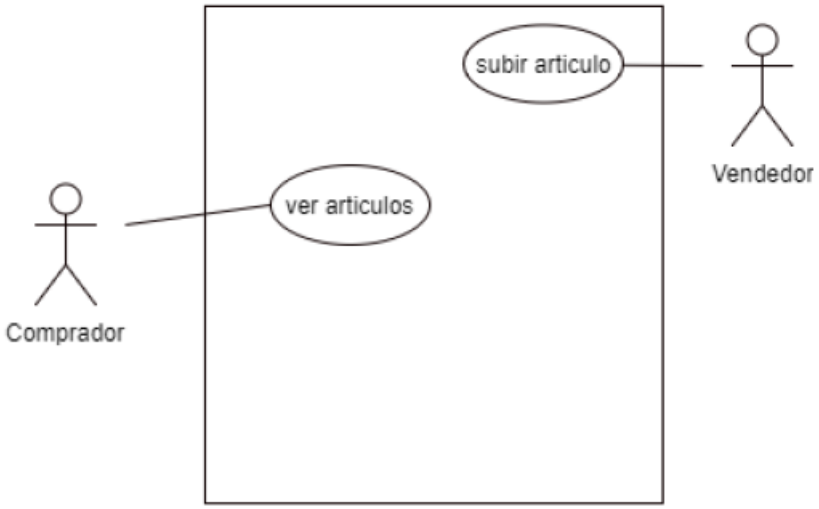
Este diagrama incluye lo siguiente:

Un actor o varios actores.

Acciones.

Primera etapa del diseño de un diagrama de casos de uso

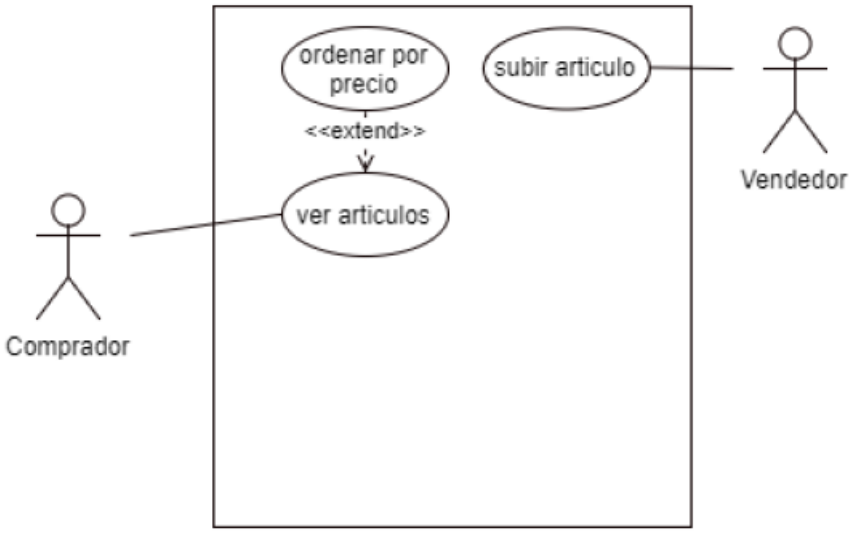
Empecemos con las primeras acciones descritas en el caso de uso textual: el comprador buscando un artículo. Podemos añadir “ver artículos” en el sistema. Dibujamos una línea entre el comprador y la acción para significar quien lleva a cabo la acción. También añadimos la acción “subir artículo” y dibujamos una línea entre el vendedor y la acción.



Actores y acciones

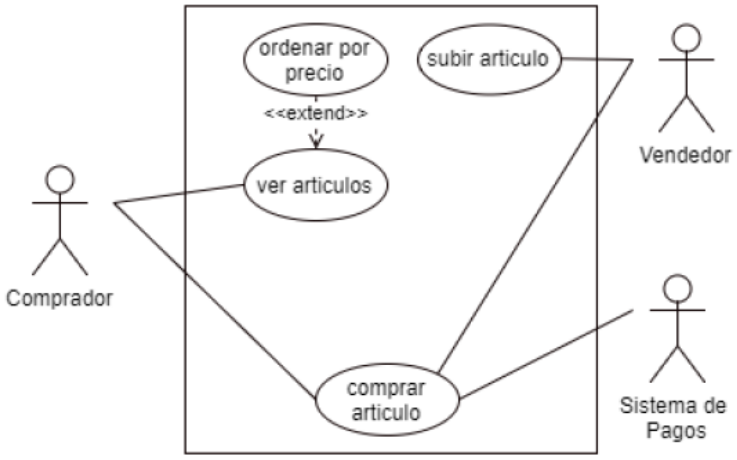
Para poder elegir la mejor oferta, el comprador tiene que ser capaz de ordenar los artículos por precio.

Usaremos la palabra clave <<extend>> para designar una acción que puede derivar de otra, dibujando una línea punteada entre los casos de uso con la palabra<<extend>>en la línea.

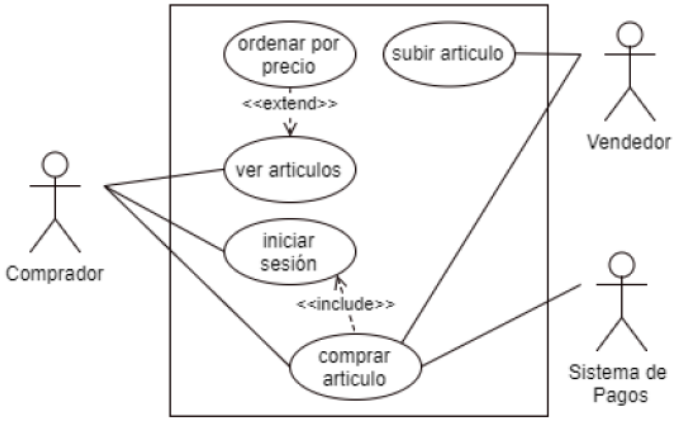


Añadimos la palabra clave <<extend>>

Ahora pasamos a la acción "**comprar artículo**". Esta acción involucra el **comprador** así como el **vendedor**. También hemos de añadir otro actor, ya que para efectuar el pago, necesitamos usar el **sistema de pagos**. Lo añadiremos como otro actor (recordar que los actores no son necesariamente humanos!). Dibujamos una línea entre cada actor y la acción.



Añadimos la acción "comprar artículo"



Añadimos una acción con la palabra clave <>

Finalmente, hemos de asegurarnos de que para poder comprar, un usuario esté registrado en la plataforma. Este requerimiento lo modelamos con la palabra clave <<include>> con una flecha desde "**comprar artículo**" hacía "**iniciar sesión**".

Adicionalmente, puede comportar

• relaciones <<include>> entre acciones cuando una acción es prerrequisito de otra

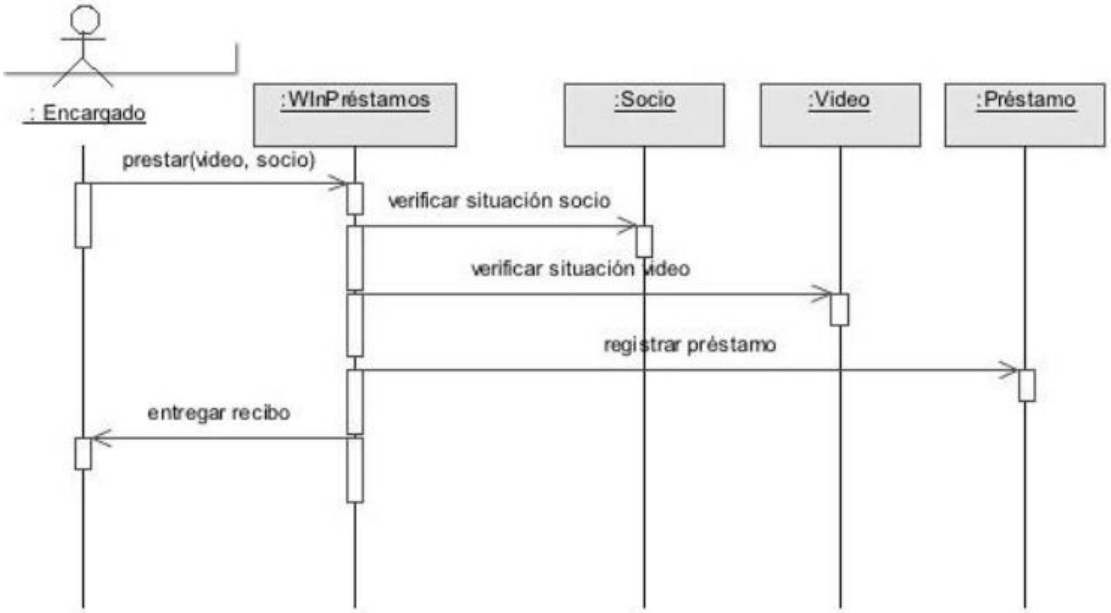
• relaciones <<extend>> entre acciones cuando una acción es una opción después de realizar otra.

2.3 Aplicación de los diagramas de comportamiento

Los Diagramas de Comportamiento se enfocan en describir qué tiene que pasar dentro del sistema modelado. Dado que estos diagramas ilustran el comportamiento de un sistema, se usan a menudo para describir la funcionalidad de un sistema.

El diagrama de secuencia describe con detalle el orden de ejecución de varias funciones del sistema para llevar a cabo acciones específicas.

Los componentes del diagrama de secuencia son los objetos, la línea de vida de los objetos y los mensajes.



2.4 Aplicación de las clases

El diagrama de clase entra en la categoría de los diagramas estructurales, y es sin duda el más común de ellos. Son diagramas que permiten describir la arquitectura de un sistema con bastante detalle. Es muy recomendable tener un diagrama de clase antes de empezar a escribir código para una base de datos o una aplicación.

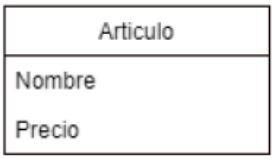
Escribiendo un diagrama de clase, podemos describir cómo se relacionan los elementos de un sistema, qué atributos les caracterizan.

A continuación, vamos a escribir el diagrama de clase para el ejemplo con el que estamos trabajando.

En el apartado anterior, hemos identificado los elementos principales de nuestro sistema: compradores, vendedores, y artículos.

Identificar las clases del sistema

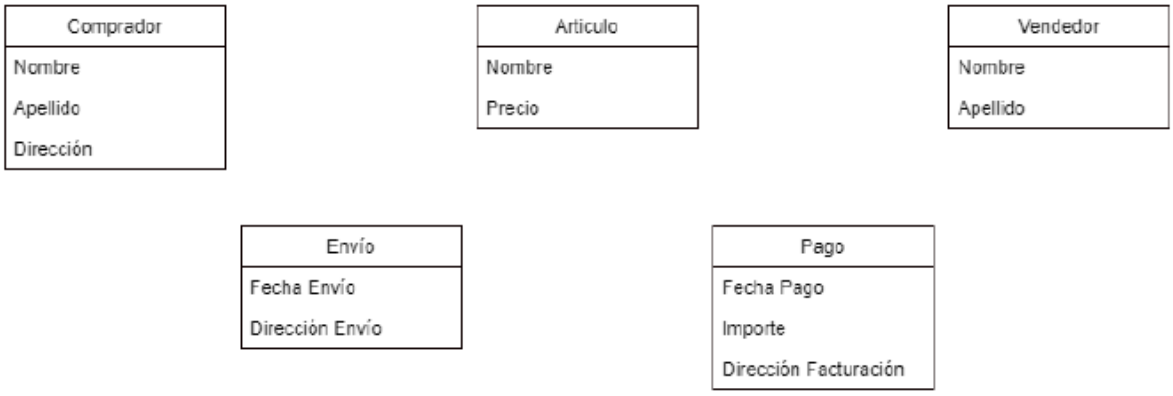
Añadiremos los artículos como el primer elemento en nuestro diagrama de clase. Es tan sencillo como dibujar un cuadrado y escribir “articulo” arriba del todo



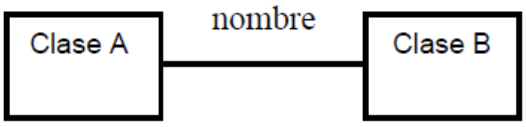
La clase "artículo"

Recuerda que adoptar el proceso unificado nos permite refinar el modelo de forma iterativa y que entonces podemos añadir detalles a medida que vayamos mejorando la aplicación o recibamos feedback de nuestros clientes u otros miembros de nuestro equipo.

A continuación detallamos las otras clases de nuestro sistema. Hemos añadido el comprador, el vendedor, el envío y el pago de un artículo.

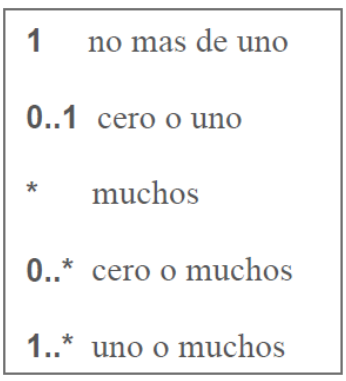


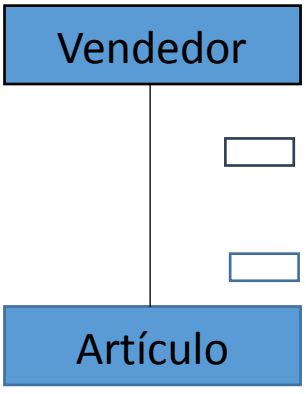
Estas serían las clases del diagrama de clases



Ahora hay que averiguar cómo se relacionan los elementos entre ellos. Una pregunta que tenemos que hacer cada vez que estudiamos la relación entre dos elementos es: ¿Cuántos elementos A por cada elemento B? ¿Y Cuántos B por cada A?

Este proceso lo llamamos establecer la cardinalidad de las relaciones. Además es recomendado añadir verbos a las relaciones para ayudar a cubrir mejor la semántica de las relaciones.





Relación de varios a varios

¿Cuántos artículos puede vender un vendedor?

Relaciones de uno a uno

Finalmente, vamos a mirar la relación entre Envíos y Pagos. En este caso, consideramos que a cada envío le corresponde un solo pago y vice-versa. Un envío se creará en cuánto se realice el pago del artículo.

Es casi como si fuesen la misma entidad. Los separamos para mantener la semántica de lo que representan. Así podemos diferenciar más fácilmente la fecha de envío de la fecha de pago, y la dirección de envío de la dirección de pago.

Entonces, lo que aquí acabamos de definir es una relación de uno a uno.

Establecer el diagrama completo del sistema

Este es el diagrama completo. Hemos añadido las relaciones comprador-envío (uno a varios) y vendedor-envío (uno a varios). Añadimos unos verbos en las líneas entre entidades.

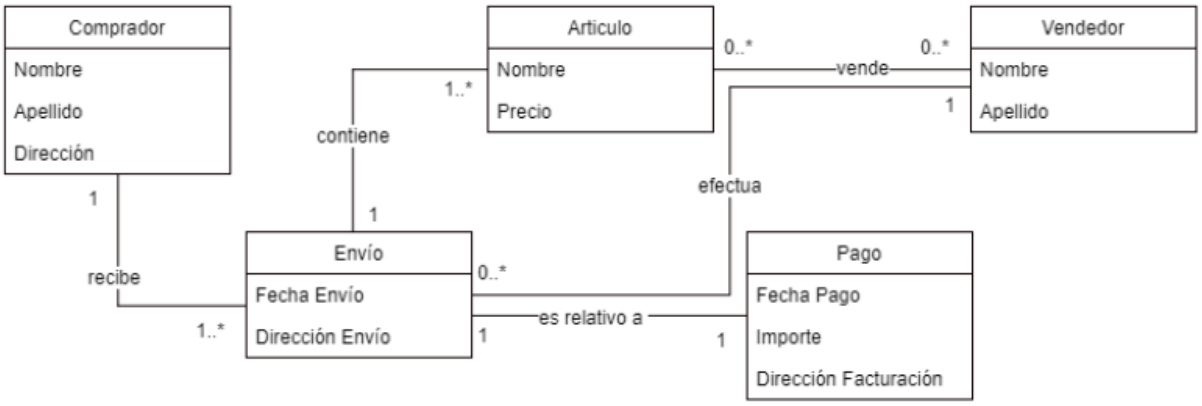


Diagrama de clase con las cardinalidades de las relaciones

Ahora que se ha visto los elementos básicos de un diagrama de clase, vamos a introducir elementos del análisis orientado a objetos, que es la filosofía subyacente al lenguaje UML.

Conceptos del análisis orientado a objetos

Clase e Instancia

Unos elementos fundamentales del AOO son las nociones de objetos, clases, e instanciación.

La clase

Una clase es una plantilla para objetos, con una especificación sobre qué atributos deben comportar estos objetos para poder ser considerados como parte de esta clase. Una clase puede describir algo abstracto (el estado de un proceso, un comportamiento) tanto como algo físico (una mercancía, una persona). Las clases son el concepto que manipulamos diseñando nuestro diagrama de clase. Definiendo clases y sus atributos, intentamos anticipar los agentes de nuestro sistema y sus comportamientos, para poder clasificarlos mejor. Eligiendo las metáforas correctas podemos describir cualquier cosa con clases.

El objeto

Un objeto es un elemento único e identificable, que tiene que derivar de una clase. Se dice que es una instancia de esta clase, cuando posee todos los atributos de la clase y les atribuye un valor.

La instanciación

La instanciación es el proceso de creación de un objeto a partir de una clase. El resultado es un objeto con atributos definidos que es añadido a nuestra base de datos o a nuestro programa. El diagrama de objetos permite dar ejemplos de qué valor pueden recibir estos atributos, es decir que proporciona un ejemplo de instanciación de las clases que definimos. Se usan los términos instancia y objeto como equivalentes.

Herencia

El AOO nos permite definir una relación de herencia entre clases. La clase de la que se herede se dice que es una abstracción de la que herede.

Por ejemplo, si volvemos al ejemplo de nuestra plataforma de venta introducida en los capítulos anteriores.

Conceptos del análisis orientado a objetos

Clase e Instancia

Unos elementos fundamentales del AOO son las nociones de objetos, clases, e instanciación.

La clase

Una clase es una plantilla para objetos, con una especificación sobre qué atributos deben comportar estos objetos para poder ser considerados como parte de esta clase. Una clase puede describir algo abstracto (el estado de un proceso, un comportamiento) tanto como algo físico (una mercancía, una persona). Las clases son el concepto que manipulamos diseñando nuestro diagrama de clase. Definiendo clases y sus atributos, intentamos anticipar los agentes de nuestro sistema y sus

comportamientos, para poder clasificarlos mejor. Eligiendo las metáforas correctas podemos describir cualquier cosa con clases.

El objeto

Un objeto es un elemento único e identificable, que tiene que derivar de una clase. Se dice que es una instancia de esta clase, cuando posee todos los atributos de la clase y les atribuye un valor.

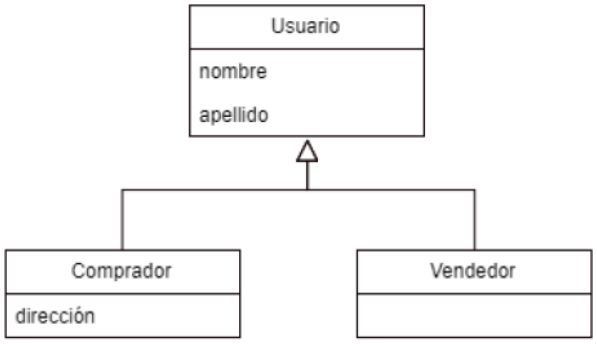
La instanciación

La instanciación es el proceso de creación de un objeto a partir de una clase. El resultado es un objeto con atributos definidos que es añadido a nuestra base de datos o a nuestro programa. El diagrama de objetos permite dar ejemplos de qué valor pueden recibir estos atributos, es decir que proporciona un ejemplo de instanciación de las clases que definimos. Se usan los términos instancia y objeto como equivalentes.

Herencia

El AOO nos permite definir una relación de herencia entre clases. La clase de la que se herede se dice que es una abstracción de la que herede.

Por ejemplo, si volvemos al ejemplo de nuestra plataforma de venta introducida en los capítulos anteriores.



Ejemplo de herencia

En el modelo relacional no hay relación de herencia, pero este concepto puede ser útil durante el diseño y es fácil de implementar en el modelo físico de datos. Pondremos una referencia de clave ajena en cada de las tablas que heredan de la clase Usuario

Composición y Agregación

Composición

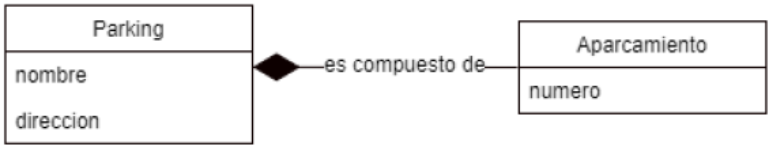
Cuando objetos de una clase están compuestos de elementos de otra clase se llama una relación de composición. Las composiciones son un subconjunto de las relaciones de uno a varios. Por ejemplo:

• un equipo está compuesto de varios miembros

• un avión está compuesto de asientos

• un parking está compuesto de aparcamientos

La composición se representa con un diamante negro del lado del contenedor. A continuación te mostramos el ejemplo de una relación de composición entre Parking y Aparcamiento.



Ejemplo de una relación de composición

Agregación

Cuando objetos de una clase son los componentes de otra clase pero los componentes de esa clase no tienen que formar parte de los objetos de la clase contenedora se trata de una relación de agregación.

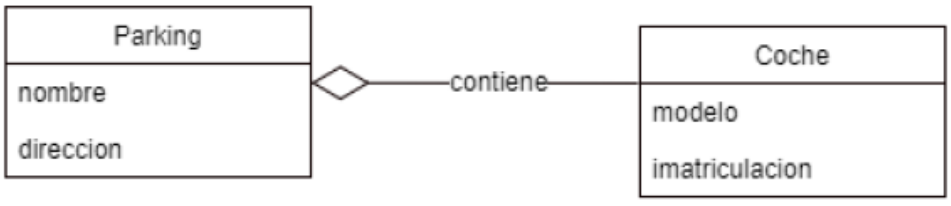
Por ejemplo:

• un equipo tiene un campo para entrenar

• un avión tiene pasajeros

• un parking contiene coches

La agregación se representa con un diamante blanco del lado del contenedor. Aquí tienes un ejemplo de una relación de agregación entre Parking y Coche.



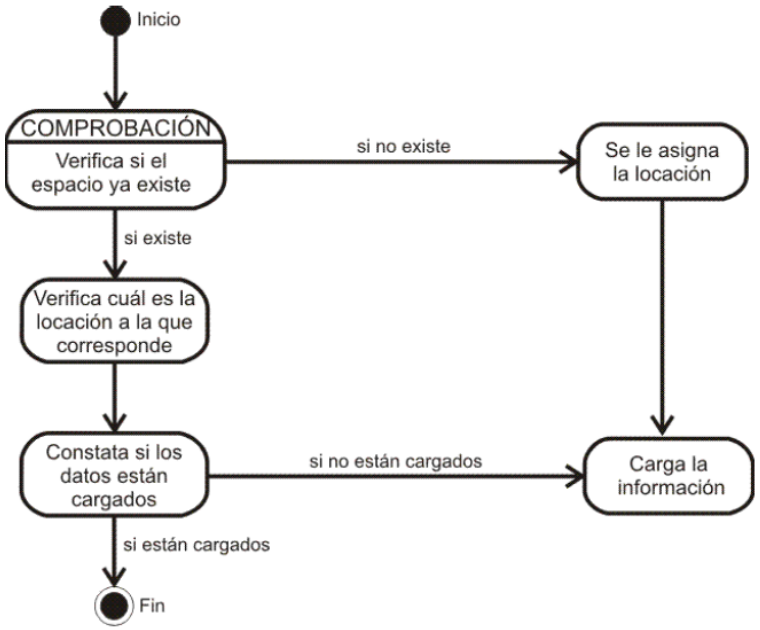
Ejemplo de una relación de agregación

Una forma de reconocer si se trata de una relación de agregación o de composición es evaluar el grado de integración de los elementos. Si el elemento que contiene a los otros no puede existir sin los subelementos, se trata de composición. Si no, hablamos de una agregación.

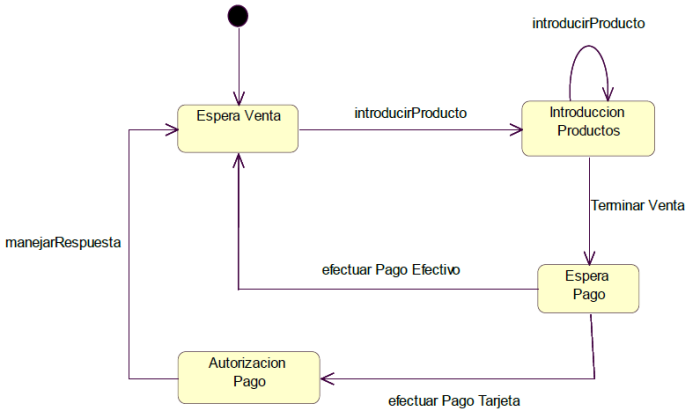
Las relaciones de agregación y composición son una forma de recoger semánticamente un tipo de relación uno a varios, con una cardinalidad 1..* del lado del contenido para la composición (sin contenido no hay contenedor) y una cardinalidad 0..* del lado del contenido para la agregación (sin contenido puede haber contenedor).

2.5 Aplicación de los diagramas de estados y actividades

El diagrama de Máquina de estado describe el ciclo de vida de los objetos. El diagrama de estados se usa para modelar el comportamiento dinámico de un objeto en particular, o de una clase de objetos.



En el diagrama de estados se indica qué eventos hacen que se pase de un estado a otro y cuáles son las respuestas y acciones que genera. También ilustra qué eventos pueden cambiar el estado de los objetos de la clase



En cuanto a la representación, un diagrama de estados es un grafo cuyos nodos son estados y cuyos arcos dirigidos son transiciones etiquetadas con los nombres de los eventos.

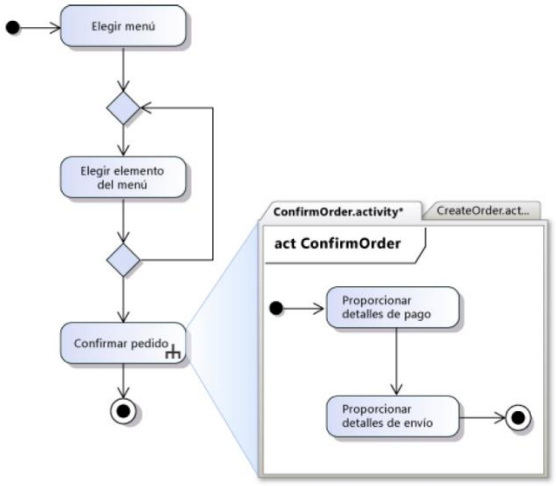
Normalmente contienen: estados y transiciones. Como los estados y las transiciones incluyen, a su vez, eventos, acciones y actividades.

Los diagramas de actividad representan el orden de las acciones, sin especificar qué objetos son necesarios para llevarlas a cabo.

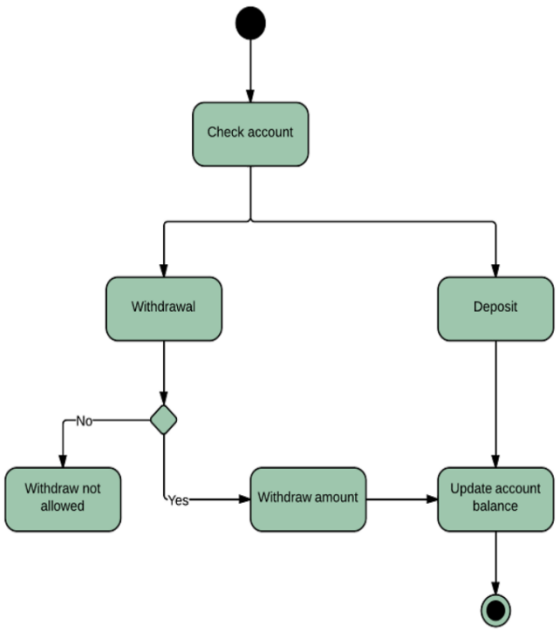
Los diagramas de actividad se pueden usar para modelar un caso de uso, o una clase, o un método complicado.

Son parecidos a un diagrama de flujo; la diferencia clave es que los diagramas de actividad pueden mostrar procesamiento paralelo

Es importante cuando se usan para modelar procesos que pueden actuar en paralelo, y para modelar varios hilos en los programas multihilo



- Diagrama de actividades para un sistema bancario
- Este diagrama muestra el proceso de retirar dinero o depositar dinero en una cuenta bancaria. Una ventaja de representar el flujo de trabajo visualmente en UML es la posibilidad de mostrar retiros y depósitos en un diagrama



2.6 Aplicación de los diagramas de despliegue

Llamados también de distribución, se utiliza para modelar el hardware utilizado en las implementaciones de sistemas y las relaciones entre sus componentes (enlaces de comunicación), mostrando las relaciones físicas de los distintos nodos que componen un sistema y el reparto de los componentes sobre dichos nodos.

El despliegue es el proceso de asignar artefactos a nodos o instancias de artefactos a instancias de nodos

“Los diagramas de despliegue muestran el hardware sobre el que se ejecutará el sistema y cómo el software se despliega en ese hardware”.

Muestra la configuración de los nodos que participan en la ejecución y de los artefactos que residen en los nodos.

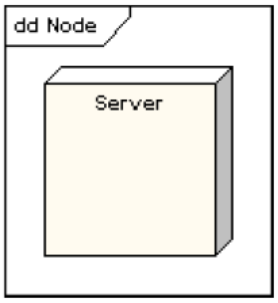
Forma de descriptor y forma de instancia

Incluye nodos y arcos que representan conexiones físicas entre nodos (o instancias de nodos y arcos).

Modelado de sistemas empotrados, sistemas cliente-servidor, sistemas distribuidos. Los elementos usados por este tipo de diagrama son nodos (representados como un prisma), componentes (representados como una caja rectangular con dos protuberancias del lado izquierdo) y asociaciones

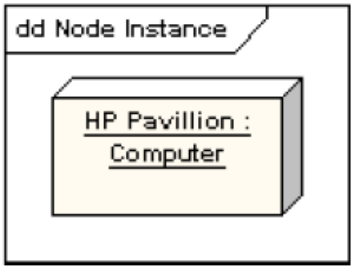
Nodo

Un Nodo es un elemento de hardware o software. Esto se muestra con la forma de una caja en tres dimensiones, como a continuación.



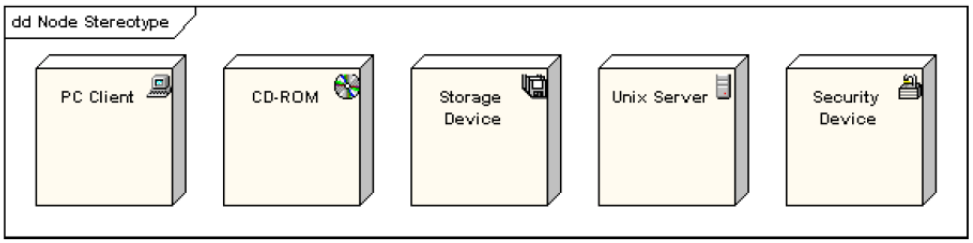
Instancia de Nodo

Una instancia de nodo se puede mostrar en un diagrama. Una instancia se puede distinguir desde un nodo por el hecho de que su nombre esta subrayado y tiene dos puntos antes del tipo de nodo base. Una instancia puede o no tener un nombre antes de los dos puntos. El siguiente diagrama muestra una instancia nombrada de una computadora.



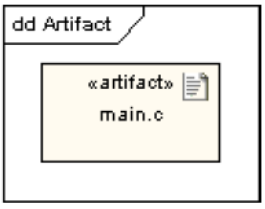
Estereotipo de Nodo

Un número de estereotipos estándar se proveen para los nodos, nombrados «cdrom», «cd-rom», «computer», «disk array», «pc», «pc client», «pc server», «secure», «server», «storage», «unix server», «user pc». Estos mostrarán un icono apropiado en la esquina derecha arriba del símbolo nodo.



Artefacto

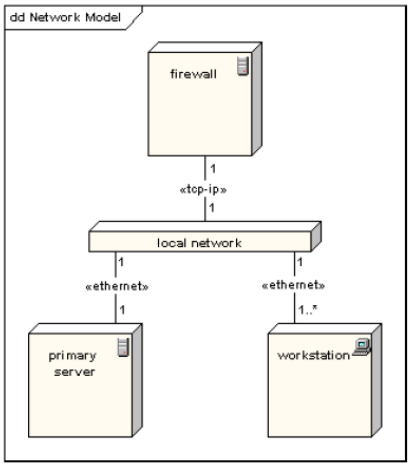
Un artefacto es un producto del proceso de desarrollo de software, que puede incluir los modelos del proceso (e.g. modelos de Casos de Uso, modelos de Diseño, etc.), archivos fuente, ejecutables, documentos de diseño, reportes de prueba, prototipos, manuales de usuario y más. Un artefacto se denota por un rectángulo mostrando el nombre del artefacto, el estereotipo «artifact» y un icono de documento, como a continuación.



Asociación

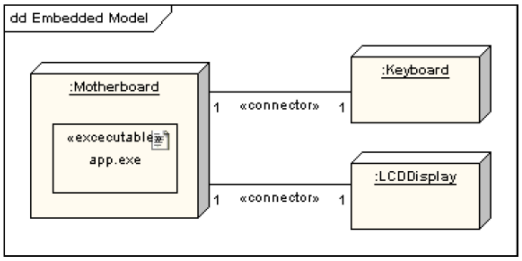
En el contexto del diagrama de despliegue, una asociación representa una ruta de comunicación entre los nodos.

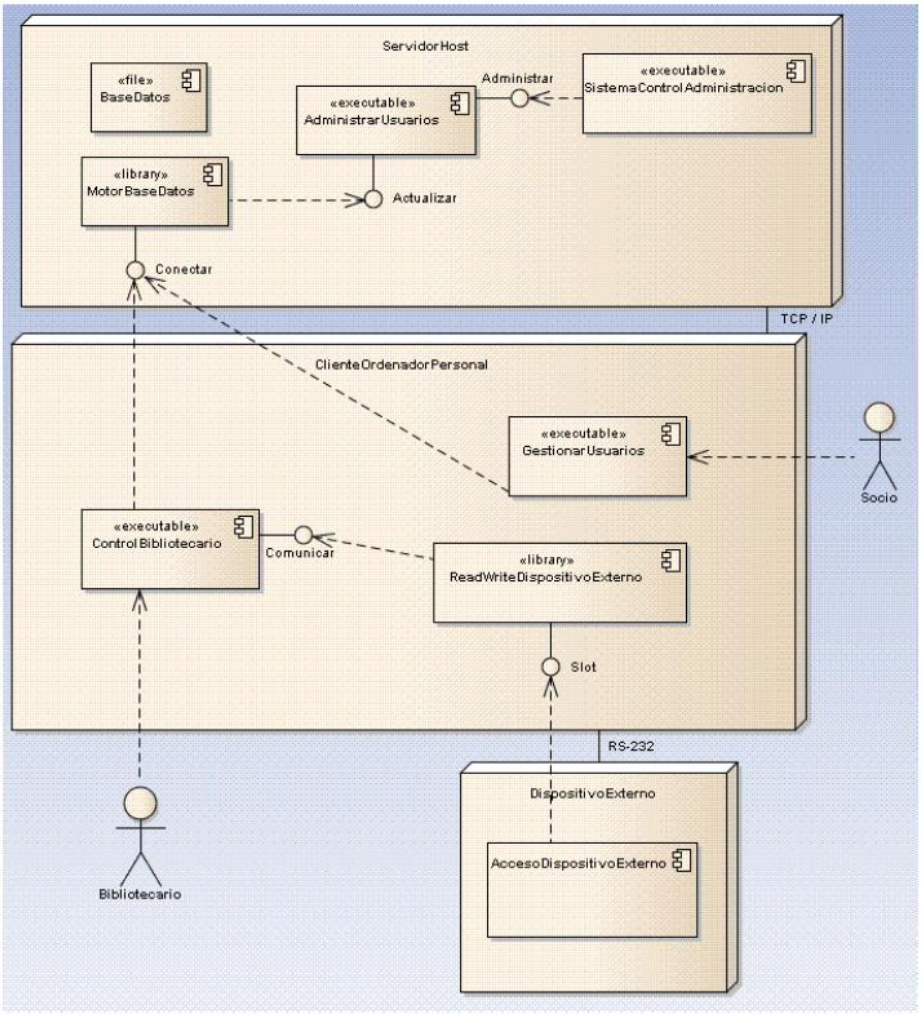
El siguiente diagrama muestra un diagrama de despliegue para una red, mostrando los protocolos de red como estereotipos y también mostrando multiplicidades en los extremos de la asociación.



Nodo como contenedor

Un nodo puede contener otros elementos, como componentes o artefactos. El siguiente diagrama muestra un diagrama de despliegue para una parte del sistema embebido y muestra un artefacto ejecutable como contenido por el nodo madre (motherboard).





Ejemplo diagrama de despliegue

2.7 Diseño de la Base de Datos del Software

Elmasri/Navathe: El proceso de Diseño de Bases de Datos es el proceso de diseñar la estructura lógica y física de una o más bases de datos para satisfacer las necesidades de información de los usuarios en una organización, para un conjunto definido de aplicaciones.

Los objetivos del diseño de BD:

satisfacer requisitos de contenido de información de usuarios y aplicaciones

proporcionar una estructuración de los datos natural y fácil de entender

soportar los requisitos de procesamiento y objetivos de rendimiento como tiempo de respuesta, tiempo de procesamiento, espacio de almacenamiento.

conseguir un esquema flexible de BD, es decir tal que sea posible modificarlo (como consecuencia de cambios en los requisitos del sistema) fácilmente una vez implementada la BD

Fases principales en el Diseño de BD

Obtención y análisis de requisitos (S.I.)

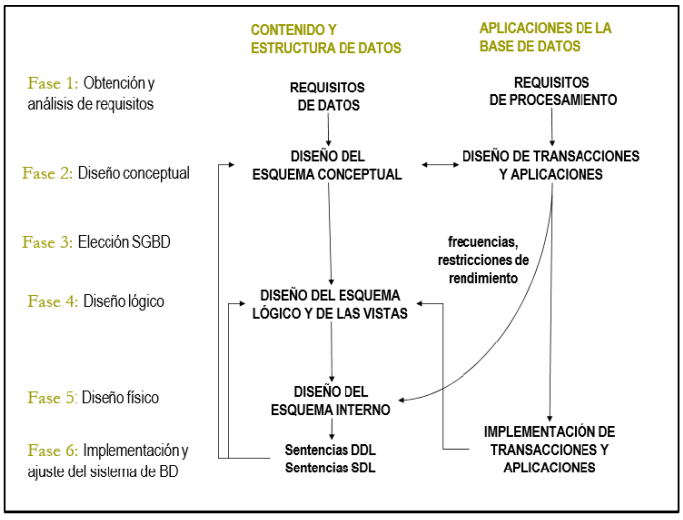
Diseño conceptual

Elección de un SGBD

Diseño lógico

Diseño físico

Implementación y ajuste del sistema de BD (S.I.)



Análisis Funcional:

- Especificación de los requerimientos de las aplicaciones

- Esquema funcional: descripción de alto nivel de actividades y flujos de información intercambiados por esas actividades.
- Bases de datos: simples depósitos de información,
- Análisis funcional integra al modelado conceptual.

Diseño:

Especificación de las aplicaciones:

Descripción a alto nivel de abstracción del comportamiento de los programas de aplicación (en particular describen como las aplicaciones acceden a la BD).

Especificaciones detalladas de programas de aplicación y eventualmente el código de los programas.

Estas fases integran los diseños lógico y físico (para estas etapas de diseño habrá que tener en cuenta los requisitos de las funciones (procesos). Ambas aproximaciones al diseño de SI deben de considerarse como complementarias y deben desarrollarse en paralelo.

Los esquemas funcional (resultado del Análisis Funcional) y conceptual (resultado del Modelado Conceptual) deben de ser mutuamente consistentes (sin conflictos) y completos:

Todos los datos requeridos por las funciones están representados en el esquema conceptual y las funciones incluyen todas las operaciones requeridas por la base de datos.

El diseño físico de la base de datos depende de las aplicaciones que van a utilizar los ficheros de la base de datos (forma y frecuencia de acceso a los datos, restricciones de rendimiento, etc.).

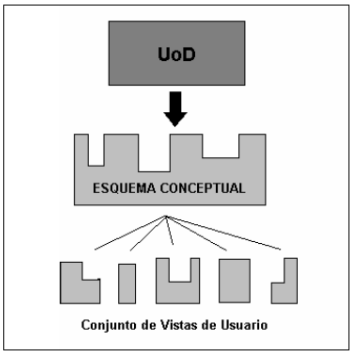
En el diseño de los programas de aplicación se hace referencia a los elementos que tiene el esquema lógico de la base de datos.

Modelado Conceptual

Propósito: describir el contenido de información de la BD, (tipos de datos, relaciones y restricciones), no las estructuras de almacenamiento que se puedan requerir para su gestión

Esquema conceptual: descripción de alto nivel de la estructura de la D; independiente del SGBD particular usado para la implementación de la BD.

Modelo Conceptual: lenguaje usado para la descripción del esquema conceptual.



La aplicación de un diseño centralizado o de integración de vistas dependerá de la complejidad y tamaño de la BD.

Normalmente, el enfoque centralizado podrá utilizarse para BD pequeñas y poco complejas

Diseño Físico

Independiente del sistema: dependiente del modelo de datos usado, no tanto del SGBD específico.

Esquema Lógico: descripción de la estructura del BD en términos del modelo de datos de implementación.

Esquema conceptual y esquemas externos (vistas) en la arquitectura a tres niveles.

Modelo Lógico: lenguaje usado para especificar el esquema lógico.

Relacional, Red, Jerárquico, Orientado a Objeto.

Objetivo: conseguir una instrumentación eficiente del esquema lógico.

Las fases de diseño físico y lógico están relacionadas (retroalimentación, adaptar diseño lógico a requisitos de físico).

Dependerá del SGBD específico utilizado en implementación.

Esquema físico: Descripción de la implementación del BD: estructuras de almacenamiento y métodos de acceso.

Esquema interno en arquitectura a tres niveles.

Tanto el esquema lógico como el físico se expresarán usando el DDL y SDL del SGBD destino.

Posteriormente se podrá pasar a la construcción de la BD, pruebas, carga y explotación.

Características (deseables) de una metodología de diseño.

Claridad y comprensibilidad

Distintas clases de personas (usuarios, técnicos de sistemas, analistas, etc.) participan en el proceso de diseño; una sencillez tal que permita que sea explicada a distintos tipos de usuarios.

Soportar la evolución de los sistemas.

Produciendo en sus distintas etapas esquemas evolutivos, cambio y/o adaptación de esquemas la metodología debe proporcionar la base para una buena documentación del sistema.

Facilitar la portabilidad

IEEE: “la facilidad con la que un producto de programación puede ser transferido de un sistema informático a otro o de un entorno a otro", esquemas portables, etapas de diseño independientes

Fase de Diseño Lógico eStándar (DLS), entre el modelado conceptual y el diseño lógico específico / físico en el SGBDR concreto que se va a utilizar.

Flexibilidad:

Independencia de la dimensión de los proyectos tanto en proyectos grandes como pequeños.

si bien las líneas metodológicas serán las mismas otras técnicas (como, por ejemplo, la de integración de vistas) simplificación de algunas de las etapas

Rigurosidad

Se pretende imprimir un carácter riguroso a los principios metodológicos propuestos.

Siempre que sea posible (como en el caso de la normalización) nos apoyaremos en sólidos fundamentos teóricos.

Un excesivo formalismo puede provocar el rechazo de determinado tipo de usuarios.

Adoptar estándares

Aplicación de todos aquellos que para la ingeniería del software en general y para las bases de datos en particular, recomiendan distintas organizaciones internacionales (como ISO, ACM, etc).

Automatización

Aplicando herramientas tipo CASE que soporten todas las fases propuestas para el diseño del BD.

En nuestro caso, al utilizar modelos, lenguajes y herramientas muy extendidos (como el ME/R, diagramas de dependencias funcionales, SQL, etc.), la metodología se puede instrumentar con facilidad en los productos CASE existentes

Versatilidad respecto a tipos de aplicaciones

No orientada a un tipo de aplicaciones concreto, sino que puede utilizarse en aplicaciones diversas

2.8 Diseño del Interfaces del software

El diseño de la interfaz de usuario es la fase del diseño que trata de crear un medio de comunicación entre el hombre y la máquina. Su correcto desarrollo es de vital importancia para la aceptación del sistema final por las personas que lo van a utilizar

La interfaz de usuario es la percepción que tiene estos del sistema de software.

Las tres "reglas de oro" de Theo Mantel

1. Dar el control al usuario:

En muchas ocasiones el diseñador introduce restricciones en la interfaz para simplificar su construcción que pueden dar lugar a sistemas frustrantes de utilizar.

Definir los modos de interacción de manera que no obligue al usuario a realizar acciones innecesarias y no deseadas. Por ejemplo forzar a que el usuario siempre corrija la ortografía de un mensaje de correo que se va a enviar.

Tener en consideración una interacción flexible, por ejemplo una aplicación debe poder manejarse solamente con el teclado, solamente con el ratón o bien con una combinación de ambos.

Permitir que la interacción que está realizando el usuario en un momento determinado pueda interrumpirse y deshacerse.

Aligerar la interacción a medida que avanza el nivel de conocimientos del usuario, utilizando por ejemplo atajos de teclado.

Ocultar al usuario los entresijos técnicos de la aplicación.

Permitir interaccionar directamente con todos los objetos que aparezcan en la pantalla (si hay un botón, que se pueda pulsar).

2. Reducir la carga de memoria del usuario:

Cuantas más cosas tenga que recordar el usuario, más propenso a errores será su interacción con el sistema. Puede desarrollarse en:

Reducir la carga de memoria a corto plazo, evitando que el usuario tenga que recordar datos entre una interacción y otra.

Establecer valores por defecto útiles.

Definir las deficiencias para que sean intuitivas: Cuando sea necesario implementar una acción basada en un mnemónico, conviene hacerlo de la manera más intuitiva posible. Aunque parezca una contradicción, con un ejemplo se visualiza mejor esta regla: Utilizar CTRL-C para copiar es una acción nada intuitiva que se basa en un mnemónico, sin embargo es mucho más intuitiva (Control-Copiar) que por ejemplo CTRL-V para pegar.

El formato visual de la interfaz debe basarse en una metáfora del mundo real.

La información debe de desglosarse de manera progresiva.

3. Construcción de una interfaz consistente:

La información deberá ser presentada y adquirida por la interfaz de forma consistente, esto es:

La información estará organizada en base a un criterio estándar en toda la aplicación.

Los mecanismos de entrada se limitarán a un conjunto definido y se utilizarán coherentemente a lo largo de toda la aplicación.

Los mecanismos para ir pasando de tarea en tarea estarán definidos e implementados correctamente.

MODELOS DE DESCRIPCIÓN DE LA INTERFAZ

Modelo de diseño:

Modelo del sistema completo que crea el ingeniero del software, que incluye diseño arquitectónico, de dotar, de procedimientos y de interfaces. En este modelo se determinan las entradas y salidas que debe recibir o enviar el sistema a sus usuarios

Modelo de usuario:

Representa el perfil de los usuarios del sistema que tienen los analistas e ingenieros del software. Para construir una interfaz eficaz, es necesario comenzar por conocer la mayor cantidad posible de características de los usuarios finales. Los usuarios pueden además catalogarse en varias categorías, y es conveniente que la interfaz de usuario se adapte a todas las categorías aunque dependiendo del estudio del perfil se podrá incidir más en una u otra:

Principiantes:

No tienen conocimientos ni a nivel sintáctico ni a nivel semántico de la utilización de la aplicación, es decir, no saben ni como se hacen las acciones ni porqué se hacen las acciones.

Usuarios esporádicos y con conocimientos:

Poseen un razonable conocimiento semánticos (es decir, saben en principio que acciones debe permitir el sistema) pero pocos conocimientos sintácticos (no saben como se hacen las cosas), además de una baja retención de la información necesaria para manejar la interfaz.

Usuarios frecuentes y con conocimientos:

Poseen conocimientos elevados tanto a nivel semántico como sintáctico y es posible que busquen como sacar el máximo partido a la interfaz, utilizando por ejemplo modos abreviados de interacción.

Percepción del usuario

Consiste en la imagen mental que se crea el usuario de cómo es y como funciona el sistema a través de lo que le transmite la interfaz. Esta percepción también está muy influida por el tipo de usuario que se trate.

MODELOS DE DESCRIPCIÓN DE LA INTERFAZ

Un buen diseño de la interfaz es aquel en el que la Imagen del sistema y la percepción del usuario coinciden adecuadamente, lo que hace que los usuarios se sientan a gusto con el software y con su funcionamiento.

Para conseguir esto, el diseñador de la interfaz debe unir la información procedente del modelo de diseño con el modelo de los usuarios para crear una imagen del sistema que refleje de manera veraz y entendible para el perfil del usuario definido la información tanto a nivel sintáctico como semántico.

El objetivo es definir un conjunto de objetos y acciones sobre estos objetos que posibiliten al usuario llevar a cabo todas las tareas que se consideren necesarias para que el sistema cumpla con los objetivos de usabilidad obtenidos en el análisis de requisitos

Comienza por la identificación de los requisitos desde el punto de vista del usuario, que sirven para crear e identificar los escenarios de usuario que posteriormente se utilizan como base para definir el conjunto de objetos y de acciones de la interfaz.

A partir de este conjunto de objetos se creará el formato de la pantalla y se especificarán los elementos principales de éste. Por último se generará un prototipo que será validado por los usuarios finales.

ETAPAS DEL PROCESO

Análisis y modelado de las tareas:

Un sistema interactivo basado en computadora se utiliza para reemplazar una actividad manual o semi-manual. Es necesario entender las tareas que se realizan de manera manual como primer paso, para posteriormente hacerlas corresponder con un conjunto de tareas similares

que se implementan en la interfaz de usuario. Este primer paso de diseño busca estas tareas, o escenarios de usuario, que se utilizaran como punto de partida para el siguiente paso de diseño.

Identificación de los objetos y acciones a realizar:

En esta fase se analizan los escenarios de usuario anteriormente identificados buscando objetos que el usuario va a utilizar, así como las acciones que se puedan realizar sobre estos objetos para llevar a cabo la tarea.

Formato de pantalla:

A cada objeto identificado se le dota de un diseño gráfico (un icono, un botón, un menú…) asociando las acciones a eventos que el usuario produzca sobre este objeto. Los objetos se distribuyen por la pantalla siguiendo una lógica (como un estándar o bien de manera coherente con el resto del sistema). En esta fase además de define el texto explicativo que aparecerá en la interfaz, el tamaño, ubicación y título de las ventanas, texto descriptivo de los menús y en general todos los elementos visuales y sus características que aparezcan en la interfaz.

Implementación de un prototipo:

Una vez realizado un modelo del diseño de la interfaz este debe ser implementado como prototipo y evaluado por los usuarios que validarán el diseño o sugerirán mejoras y críticas que sirvan para refinar este.

Problemas de diseño

Tiempo de respuesta:

Está caracterizado por dos valores, su duración y su variabilidad, la primera asociada con la media de los tiempos de respuesta y la segunda con su varianza. Si la duración es muy larga, el usuario puede frustrarse y si es demasiado corta puede ocasionar que el usuario se precipite y cometa errores.

La variabilidad se refiere a la desviación del tiempo de espera promedio, una variabilidad baja permite al usuario mantener un ritmo de trabajo constante. Es conveniente incluso aumentar la duración del tiempo de respuesta si se consigue con eso disminuir su variabilidad.

Servicios de ayuda al usuario:

Los tipos de ayuda más frecuentes son ayudas integradas y ayudas complementarios. Las primeras se añaden al software desde el principio de su construcción y son sensibles al contexto en el que se encuentra el usuario, lo que le permite buscar rápidamente ayuda sobre las acciones que está

llevando a cabo en este momento. Una función de ayuda complementaria se añade a la interfaz tras su construcción y se parece más a un manual de usuario en línea y es posible que el usuario tenga que buscar entre largos índices antes de encontrar lo que busca. Siempre es preferible el primer tipo de ayuda al segundo.

Información de errores:

Los mensajes de error deben cumplir tres características:

Describir el error ocurrido de manera que el usuario lo entienda.

Proporcionar consejos útiles para recuperarse del error

Indicar las consecuencias o las posibles consecuencias del error.

Etiquetado de órdenes.

En caso de que se proporcione la posibilidad de interaccionar con teclado y ratón, hay que plantearse la forma de las órdenes de teclado (teclas de función, combinación de teclas, comandos,…), que haya órdenes para todos los elementos de los menús y la posibilidad de que el sistema recuerde las últimas órdenes enviadas.

Principios de composición en el diseño

La composición se define como una distribución o disposición de todos los elementos que incluiremos en un diseño o composición de un menú o ventana, de una forma perfecta y equilibrada. En un diseño, lo primero que se debe elegir son todos los elementos que aparecerán en él, luego debemos distribuirlos para colocarlos con el espacio disponible.

Los elementos pueden ser tanto imágenes, como espacios en blanco, etc. Es muy importante tener en cuenta de que forma situaremos estos elementos en nuestra composición, para que tengan un equilibrio formal y un peso igualado.

El color en las interfaces gráficas

La tipografía y el tratamiento del color son dos elementos a los que hay que prestar especial importancia a la hora de establecer una buena interfaz, poniendo especial cuidado en el diseño de las formas y la coherencia interna entre ellas.

Disposición de la ventana

La disponibilidad de una ventana en una interfaz gráfica se le domina disposición de la ventana.

El número y la disposición de las ventanas gráficas activas y los parámetros asociados a ellas son parte de la configuración de ventanas.

Alineación de la ventana

Una de las formas de que se dispone para organizar los elementos del gráfico consiste en alinear la vista de una de las ventanas de presentación con la vista de otra ventana.

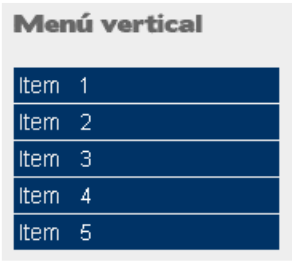
2.9 Diseño del menú del Software

El menú de una página web es la principal herramienta de navegación que le podemos facilitar al visitante para que encuentre lo que busca. Es imprescindible para que las personas encuentren las demás páginas de la web, además del índice.

Los menús de navegación son básicamente listas de enlaces a las diferentes páginas o secciones de la web. De su estructuración dependerá en gran medida que los visitantes encuentren lo que buscan, por lo que es conveniente pensarse dos veces cómo hacerlo antes de implementarlo en la web.

Podemos encontrar un menú horizontal:

O un menú vertical;



Como mencionamos antes, para sitios web con un número de páginas pequeño puede ser bueno enlazar a todas ellas desde cada página para que el usuario tenga en todo momento la información disponible en la web a su alcance.

Por ejemplo, si es una página web informativa sobre tu empresa y no tiene otra finalidad, puede que cuente con 10 ó 15 páginas en total, las cuales pueden perfectamente ser enlazadas desde cada página individual.

Sin embargo, cuando una web contiene mucha información este tipo de menús tiene poco sentido, porque entonces tendríamos menús con cientos de enlaces, algo excesivamente largo para ser usable.

Lo normal en estos casos es enlazar desde la página principal a las secciones más importantes, y desde cada una de ellas a sus contenidos concretos.

Consejos para mejorar nuestro menú de navegación

Si el usuario no cuenta con los medios adecuados para encontrar rápidamente lo que está buscando, o existen elementos que lo pueden distraer fácilmente de su objetivo, se puede crear una sensación de frustración y su experiencia se ve claramente afectada.

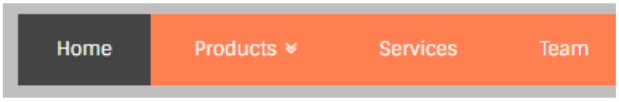
Es por esto, que debemos analizar los elementos claves para lograr una buena navegación, y el más importante de estos siempre será nuestro menú.

El menú de navegación es un aspecto central muy relevante para la usabilidad del sitio, si se le da una estructura y uso adecuado ésta se ve beneficiada en gran medida, permitiendo guiar al usuario a través de su visita.

Simplicidad

Siempre trata de mantener las cosas simples, si tienes un menú sencillo será más fácil para un visitante primerizo aprender a utilizarlo, y de esa manera navegará con mucha más confianza.

Si mantenemos la sencillez lograremos obtener una navegación eficaz. Sin importar cuántas categorías tenga el sitio, ni la cantidad de páginas que alberga, nunca debes llevar al extremo la exigencia hacia el usuario, por lo que mantener alejada la complejidad del diseño de tu sitio se debe de convertir en una de tus tareas principales



Para poder lograr la simplicidad en nuestro menú, primer debemos tener bien definida la estructura que tendrá la información, y a partir de ahí podemos definir el diseño que más nos conviene.

Es muy importante que los elementos de contenido que estén relacionados, se agrupen y se definan en categorías para que de esa manera la navegación sea más intuitiva y tenga más sentido para los usuarios.

No sobrecargues tu menú, mantén sólo los elementos que aumentan la usabilidad del sitio y mejoran la experiencia del visitante.

Buena descripción de los ítems del menú

Una vez que tenemos definida la estructura en que estableceremos la información de nuestro menú, es importante tomarnos un tiempo para definir qué palabras claves vamos a utilizar para fijar los elementos que enlazarán a la información.

Es en esta parte que la descripción juega un papel muy relevante para lograr un funcionamiento eficiente.

De nada sirve tener toda la información del sitio en un orden adecuado y con la estructura correcta, si los elementos de navegación que llevan a ella no son llamados de la manera en que se espera.

Si al momento de ver un enlace, no nos explica realmente sobre el contenido al que lleva, los datos siguen siendo difíciles de encontrar para el usuario.

Trata de utilizar palabras que sean familiares para el usuario, si es necesario hacer uso de frases o palabras extrañas trata de dar una explicación a través de algún otro medio, no dejes de ser creativo, pero no malinterpretes este concepto, selecciona una redacción que sea simple y que vaya al grano.

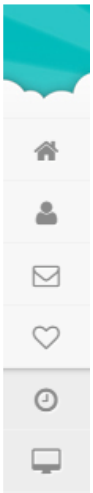
Posicionamiento en la parte superior

La posición de nuestro menú es un factor muy importante, ya que las personas que ingresan a un sitio tienden a seguir un patrón de observación, centrando su atención en ciertas partes de la página a la vez.

Generalmente este patrón forma la figura de una F, esto nos dice que su atención está puesta generalmente en la parte superior del sitio.

Si es necesario desplazarse a una parte media de la página, puede que la atención del usuario se pierda y su vista se centre en otros elementos, causando una distracción involuntaria, lo que puede llevar a la frustración que mencionábamos anteriormente. Es por eso que es importante mantener el menú en la parte superior de la ventana.

Visualmente atractivo



Está comprobado que nuestros cerebros procesan, a cierta medida, mejor la información visual que la textual.

Cuando ingresamos a un sitio de manera apresurada, tendemos a hacer un análisis rápido del contenido, por lo que podemos omitir un cierto nivel del texto que se muestra, pero las imágenes son mucho más difíciles de omitir.

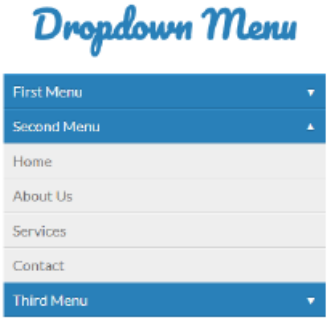
La idea de hacer un menú visualmente atractivo, es ayudar a los visitantes a interpretar el enlace mediante imágenes de fácil relación.

Elementos como iconos y patrones de colores, hacen que el usuario recuerde la posición de alguna categoría o sección de su interés.

Los colores también son importantes para resaltar los elementos que actualmente se encuentra activos, o que son llamados con algún evento, de esta manera el usuario podrá percatar si realmente esta realizando alguna acción y recordará la parte del menú a la que accedió.

¿Es conveniente que mi página web tenga menús desplegables?

Los menús desplegables son muy agradables a la vista, pero los menús no tienen en sí mismos una función decorativa, sino que se trata de elementos que colaboran en la usabilidad de la página y por consiguiente, mejoran la experiencia de los usuarios en nuestra página web.



Para determinar si una página web requiere de un menú desplegable o es mejor utilizar menús sencillos, debemos preguntarnos cuál es la utilidad que se agregará y evaluar beneficios e inconvenientes.

No cabe la menor duda que cuando tenemos una página web con contenidos que se encuentran divididos en muchas categorías y secciones, es necesario que los mismos sean presentados a los usuarios en forma ordenada, de modo que pueda acceder rápido a cualquier parte de la página.

Para estos casos, la utilización de los menús desplegables, más que una opción se trata de una necesidad, ya que presentar los títulos de secciones y categorías de otra forma podría ocupar gran cantidad de espacio y quedar parcialmente fuera de la vista del usuario haciéndose necesario desplazarse por la página para ver todo, lo que es un grave inconveniente.

Pero teniendo en cuenta esta último, debemos saber que los menús desplegables es información oculta, y que obliga al usuario a realizar algunos movimientos para que se produzca el despliegue del menú.

Esto hace que la utilización de los menús desplegables deba ser racionalizada y llevada al mínimo posible, dado que es preferible que la información de navegación se encuentre visible.

Para los sitios pequeños o con contenidos que se encuentran poco subdivididos, lo más aconsejable es que la información de navegación sea colocada en menús simples

(horizontales o verticales), tanto en la parte superior de la página como en las columnas laterales.

Malas prácticas en el uso de menús desplegables

Si la página web requiere de la utilización de estos recursos, tengamos en cuenta que no debe realizarse de cualquier forma, y que en todos los casos, lo que debe primar es la usabilidad. Cuando no se tiene en cuenta esto, se producen los fracasos que son frecuentes en la web, donde los menús son una molestia para el usuario o son prácticamente inútiles.

Uno de los errores más frecuentes es la utilización de scripts pesados para conseguir efectos diversos, sin tener en cuenta que los mismos afectan la carga de la página, como los menús hechos en flash con carga gráfica importante.

El menú debe siempre ser una mejora en la usabilidad de la página, por lo que aumentar el tiempo de carga resulta contraproducente.

Otro de los errores comunes es la utilización de menús con tiempos de retardo para su despliegue. Los retardos excesivos pueden hacer que el usuario se impaciente al pasar de un ítem a otro y tener que esperar que se desplieguen.

Todo ítem que se despliega cuando se hace clic, debe estar señalizado de forma que el usuario pueda advertir rápidamente de qué se trata.

Se debe prestar especial atención a la utilización de transparencias en los menús desplegables, ya que puede verse en algunas páginas, que el texto de fondo interfiere con la lectura de los ítems de menú, o que la imagen de fondo de la página hace que se pierda la visibilidad de los ítems.

Las transparencias en los menús desplegables son un efecto muy agradable, pero no en todos los casos es posible utilizarlas.

También es importante tener en cuenta que no debe haber alteraciones durante el desplazamiento del puntero por el menú, para que cuando el usuario realice este movimiento el menú no se cierre y se deba repetir la acción; este error, que parece elemental, es bastante frecuente.

Consejos para la instalación de menús desplegables

Siempre se debe tratar de que haya continuidad visual entre el menú que se despliega y el ítem desde el que parte, de forma tal que el usuario siempre tenga visible el punto de partida y le resulte sencillo desplazarse a otro ítem.

Cuando se realice o instale un menú en el que se debe hacer clic para que se despliegue, es conveniente señalizar esto mediante la colocación de una flecha, de forma tal que el usuario tenga presente que algo va a suceder mediante una acción suya.

Hay menús desplegables que tienen un tiempo de retardo para cerrarse. Siempre debe tenerse en cuenta que esto se hace principalmente para que el menú no se cierre por una acción involuntaria del usuario, pero el tiempo de retardo no debe ser tal que cuando el usuario deba esperar en forma innecesaria para que el efecto termine.

Y no debe perderse de vista que los menús, sean desplegables o no, no son simples elementos decorativos, sino que son ayudas que se ofrecen a los usuarios para que la navegación sea más sencilla.

Ante la duda, siempre es preferible optar por la opción más simple. Basta con hacer una recorrida por las páginas web más visitadas, para ver que los efectos no son un elemento principal, y en muchas, ni siquiera existen.

Técnicas para realizar menús adaptables

Diseño responsive

Manejar un diseño adaptable (responsive design) te permitirá proporcionar una mejor experiencia para el usuario, previendo los problemas relacionados a los diferentes tipos de resoluciones y tamaños de pantalla, que se presentan por la variedad de dispositivos que se utilizan en la actualidad para desplegar páginas web.

Debes tener en mente, que los usuarios que acceden a tu sitio utilizando dispositivos con resoluciones de pantalla pequeñas, también necesitan navegar a través de él, por lo que es necesario darles una clase de soporte.

El diseño adaptable (responsive design), se ha convertido en una consideración casi obligatoria al momento de elaborar la estructura de un sitio web.

Actualmente son muchos los usuarios y clientes potenciales que acceden a través de algún dispositivo móvil, a alguna de las páginas que hemos desarrollado para presentar un proyecto o producto, como tal debemos de proporcionar el soporte adecuado a estos dispositivos para no tener una fuga de inversión.

Vale destacar que sólo haremos uso de CSS para llevar a cabo estos ejemplos, pero nos respaldaremos con una pequeña línea de código JavaScript.

Uso de menus que ocupen toda la horizontal

Esta es la técnica más sencilla para crear un menú responsive, ya que simplemente consiste en crear los elementos en forma de lista y darles un ancho para que abarquen totalmente la pantalla en sus distintos tamaños.

Con este código cuando la ventana se haga pequeña o la página se visualice en una pantalla de dispositivo móvil, los elementos del menú tomarán un ancho de 100% para desplegarse de manera horizontal.

Las ventajas de este método son que no te utilizamos JavaScript, no es necesario añadir código HTML extra y el CSS se mantiene sencillo.

Desplegar elementos en un select

Con este método nos encargaremos de ocultar el menú cuando la pantalla sea muy pequeña, y en vez de él mostraremos un elemento de tipo select donde desplegaremos todos elementos que componen dicho menú.

Para poder lograr esto necesitaremos agregar un nuevo elemento HTML par así crear el select. Además de esto, haremos uso de JavaScript para redireccionar a la página que se seleccione de la lista.

Para las pantallas y ventanas grandes ocultaremos el select. Mientras que para las pequeñas ocultaremos el menú y desplegaremos el select.

De igual manera mediante el pseudo elemento “:before” agregamos la palabra Menú para que el usuario sepa que debe dar clic ahí para acceder a las secciones.

Las ventajas de esta técnica es que no se necesita mucho espacio y se utilizan controles nativos, mientras que las desventajas son la necesidad de usar JavaScript, el duplicado de contenido y que el elemento select no puede ser muy estilizado.

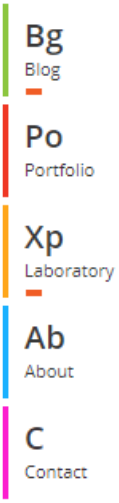
Menú fijo con scroll suavizado (sticky menu)

Otra opción muy interesante es tener el menú fijo en la parte superior de la página y hacer que al pinchar en una de las opciones de menú, vaya a la sección con una animación que haga el movimiento más suave.

Puedes ver el siguiente ejemplo y pinchar en los elementos para ver de lo que hablamos.

Pasos para hacer un menú web eficaz

La navegación web es una de los principales elementos que podemos mejorar y adecuar con las hojas de estilo, esto se ve reflejado en el constante uso de menús y barras para guiar al usuario en las páginas de internet.



Por ello, debemos de realizar un esfuerzo a la hora de diseñar un menú y no simplemente tomarlo como un paso más, debemos planearlo y codificarlo de tal manera de que el producto sea efectivo y sobre todo útil para los internautas.

La creación de menús de navegación con CSS es un tema creativo, con muchas formas de explotarse y que nos permitirá desplegar nuestras ideas para que los usuarios se sientan cómodos en nuestra página.

Existen cientos de galerías y códigos en internet para obtener menús desplegables, horizontales, verticales, transparentes, con hijos, sin hijos, etc., es cuestión de buscar y con la ayuda de Google tendremos un sinfín de menús que lucirían fantásticos en nuestros sitios.

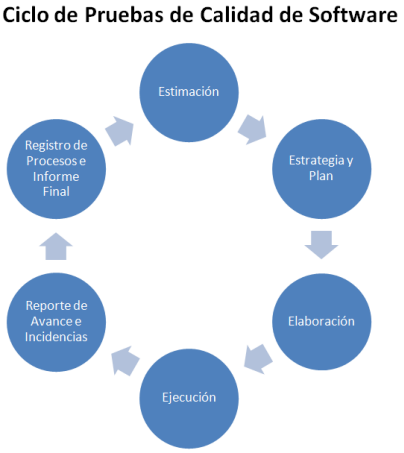
2.10 Prueba demostrativa del Software

Las pruebas de software (Software Testing) comprenden el conjunto de actividades que se realizan para identificar posibles fallos de funcionamiento, configuración o usabilidad de un programa o aplicación, por medio de pruebas sobre el comportamiento del mismo.

Los sistemas informáticos, programas y aplicaciones han crecido a niveles inimaginables en complejidad e interoperabilidad, con lo cual también se han incrementado las posibilidades de defectos (bugs), a simple vista insignificantes, pero que pudieran adquirir proporciones catastróficas.

Además, factores como el uso de software de terceros desde aplicaciones móviles han añadido niveles adicionales de complejidad y por ende incrementado los posibles puntos de fallas.

Frente a esto, el reto de los profesionales de Software Testing es modernizar sus metodologías, tecnologías y herramientas que les permitan automatizar tareas, ejecutar ciclos de pruebas más rápidos y así reducir al mínimo las posibilidades de errores en el Software.



Como realizar pruebas de Software

Identificar los requisitos del software

El proceso de Software Testing tiene su punto de partida en los requerimientos funcionales y no funcionales. Aquí un modelo de matriz de trazabilidad de requisitos.

Elaborar los casos de uso o historias de usuario si estamos bajo metodología Agile

Plantilla de casos de uso.

Plantilla de historias de usuario.

Luego se elabora el Plan de pruebas de software:

Hoy en día es indispensable contar con un Plan de Pruebas de Software para especificar minuciosamente las funciones a probar, como serán ejecutadas esas pruebas, quienes serán los responsables y el cronograma para su ejecución, en toda planificación es necesario definir quién es reponsable de que y a que nivel.

El plan de pruebas de software se elabora para atender los objetivos de calidad en un desarrollo de sistemas, encargandose de definir aspectos como por ejemplo los módulos o funcionalidades sujeto de verificación, tipos de pruebas, entornos, recursos asignados, entre otros aspectos.

Pasos para elaborar un plan de pruebas de software.

1.- Analizar los requerimientos de desarrollo de software

Para elaborar un plan de pruebas de software lo primero que debes hacer es entender los requerimientos de usuario que componen la iteración o proyecto, que son el sujeto de la verificación de calidad que se va a realizar.

Deberás analizar toda la información de la ingeniería de requisitos, incluyendo la matriz de trazabilidad, especificaciones y diseño funcional, requisitos no funcionales, casos de uso, historias de usuario (si estás trabajando con metodologías ágiles), entre otra documentación.

También es muy importante realizar entrevistas con el equipo encargado de la ingeniería de requisitos para aclarar dudas y ampliar la información que sea necesaria.

Además de las preguntas específicas de cada requisito, es importante hacer preguntas del alcance general, por ejemplo:

¿Es un sistema nuevo o existente?

¿Cuáles funcionalidades existentes están siendo modificadas?

¿Cuáles son los requisitos no funcionales? Entre otras.

2.- Identificar las funcionalidades nuevas a probar

A partir de la documentación del análisis de requisitos y de las entrevistas con el equipo de ingeniería de requisito y desarrollo, debes identificar e incluir en el plan de pruebas de software la lista de las funcionalidades (Características) totalmente nuevas.

Si estás trabajando con un sistema informático nuevo, no tendrás problemas en discernir, pues todas serán nuevas.

En el caso de desarrollos de software integrados a un sistema existente es necesario revisar con los analistas de negocio y también con los arquitectos de software las funcionalidades que forman parte del desarrollo de software, en todas las capas de la arquitectura.

3.- Identificar las funcionalidades de sistemas existentes que deben probarse

Se debe identificar las funcionalidades existentes que estén siendo impactadas por el desarrollo de alguna forma, considerando todos los componentes afectados en todas las capas de la arquitectura de software.

Existen dos situaciones que se puede encontrar al identificar estas funcionalidades:

• Funcionalidades modificadas de cara al usuario: Por ejemplo si una funcionalidad está siendo modificada agregando más pantallas o cambios a su flujo de proceso, debe ser incluida en el plan de pruebas de software.

• Funcionalidades modificadas en sus componentes internos: Son funcionalidades no modificadas de cara al usuario, manteniendo la misma interfaz gráfica y flujo de procesos, sin embargo, si se modifican componentes internos que comparten con otras funcionalidades del sistema, en las capas de lógica de negocio o acceso a datos. Estas deben incluirse en el plan de pruebas de software para determinar a partir de ellas pruebas de regresión a realizar.

Quienes pueden suministrar la información serán los Analistas de negocio o Arquitectos de software, familiarizados con el sistema informático implementado en entorno de producción.

4.- Definir la estrategia de pruebas

Consiste básicamente en seleccionar cuáles son los tipos de pruebas de software que se deben realizar.

Es recomendable seguir un marco de referencia para determinar los tipos de prueba, como por ejemplo los tipos de pruebas de software definidos por el ISTQB.

Pruebas funcionales

Se determinan los conjuntos de pruebas a realizar, correspondiente con cada funcionalidad nueva o existente que se esté modificando.

Se tienen distintos tipos de pruebas funcionales, por ejemplo, las pruebas de sistema (o pruebas integradas de sistemas), que se realizan después que el equipo de desarrollo ha integrado los componentes de distintas capas.

Las pruebas funcionales

Son definidas como pruebas basadas en especificación. Son diseñadas usando técnicas de diseño de pruebas de caja negra. Entre las pruebas funcionales, también están las pruebas de aceptación, en las cuales el equipo de calidad e inclusive personas del área de negocio o cliente del proyecto verifican el funcionamiento del sistema o funcionalidad.

Pruebas no funcionales

Se define un conjunto de pruebas no funcionales para cada requisito de este tipo. Aquí se pueden incluir pruebas sobre el desempeño, tiempo de respuesta, mantenibilidad, Pruebas de seguridad de software, entre otros aspectos, según la clasificación de requisitos no funcionales que se tenga para el proyecto.

Pruebas de caja blanca

Según la estructura y arquitectura del software que se esté desarrollando.

Pruebas de regresión

Se definen sobre las funcionalidades modificadas en sus componentes internos.

Tipos de pruebas de software en metodologías ágiles

Si estás trabajando con metodologías ágiles, te recomendamos usar como base los 4 cuadrantes del Agile Testing, que define el marco de referencia con todos los tipos de pruebas que debes considerar.

5.- Definir los criterios de inicio, aceptación y suspensión de pruebas

Criterios de aceptación o rechazo:

Para definir los criterios de aceptación o rechazo, es necesario definir el nivel de tolerancia a fallos de calidad. Si la tolerancia a fallos es muy baja puede definirse como criterio de aceptación que el 100% de los casos de prueba estén sin incidencias. Lograr este margen en todos los casos de prueba principales y casos bordes será muy difícil, y podría comprometer los plazos del proyecto (incrementa los riesgos), pero asegura la calidad del producto.

Por otra parte, puede ser que la intención sea realizar un Soft Launch, o un mínimo producto viable, en ese caso se podría definir como criterio de aceptación el 100% de los casos de prueba principales (considerados clave) y 20% de casos de prueba no principales (casos borde).

Una vez logradas las condiciones, se darán por aceptadas las pruebas y el desarrollo de software.

Criterios de inicio o reanudación:

Definen las condiciones que deben cumplirse para dar inicio o reanudar las pruebas. Por ejemplo, en el caso de inicio la condición podría ser la instalación de los componentes de software en el ambiente y que los casos de pruebas de verificación de ambiente sean exitosos.

Para el caso de la reanudación las condiciones están relacionadas, se determina a partir de cuales criterios de suspensión se presentaron para detener las pruebas. Una vez que estás condiciones ya no existan (sean solventadas) se procede con la reanudación.

Criterios de suspensión:

Las condiciones van a depender de los acuerdos de nivel de servicio (SLAs) internos de la organización y también de los acuerdos establecidos en cada proyecto individual. Por ejemplo, si se tiene un equipo de pruebas que comparte su esfuerzo entre varios proyectos, se puede definir un criterio de suspensión exigente, un determinado porcentaje de casos fallidos que resulten en incidencias. Si la condición se cumple, se detienen las pruebas y se dedica el personal a otras actividades,

Por otra parte si se tiene un equipo de pruebas con personal dedicado, el criterio de suspensión puede ser poco exigente, por ejemplo solo ocurriendo si se bloquean por incidencia todos los casos de prueba.

6.- Identificar los entornos (ambientes) requeridos

Posteriormente se definen y documentan las características de los entornos de Hardware y Software necesarios para realizar la ejecución de las pruebas de software.

Esta información se obtiene a partir del equipo de desarrollo y de los arquitectos de software, quienes pueden suministrar los requisitos mínimos y óptimos para la operación del sistema.

Como mejor práctica, el ambiente de pruebas de software debería ser lo más similar posible al ambiente de producción, sin embargo, no siempre es posible debido a limitaciones de recursos (financieros). En estos casos debe estudiarse cuales son los requisitos que aseguran un mínimo de confiabilidad de estas pruebas respecto al entorno de producción.

Además, en esta sección del plan de pruebas, también se definen los requisitos de sistemas operativos, software y herramientas de las estaciones de trabajo de los Testers. Si el alcance del proyecto incluye pruebas de aplicaciones (Apps) para móviles, es necesario definir los emuladores y teléfonos inteligentes, con sus respectivos requisitos. También deben definirse los requisitos de harware y software para los siguientes componentes:

• Herramienta de gestión de calidad de software.

• Herramientas para automatización de pruebas.

• Herramientas de BDD, TDD y Testing de Web Services).

7.- Determinar necesidades de personal y entrenamiento

Debe completarse previamente la estimación del esfuerzo de pruebas a partir del diseño de casos de prueba.

Si aún no se cuenta con la estimación, se puede comenzar por definir los tipos de perfiles de habilidades y conocimientos en Software Testing que se necesitan.

Para ello se puede buscar la respuesta a las siguientes preguntas:

¿Qué conocimientos de procesos de negocio se necesitan?

¿Qué sistemas se están probando y quienes tienen experiencia en su funcionamiento?

¿Se necesitan conocimientos específicos en pruebas de requisitos no funcionales? Por ejemplo para pruebas de desempeño o estrés.

¿Cual herramientas de gestión de calidad de software se va a utilizar?

¿Se necesitan conocimientos en herramientas técnicas como Lenguajes de programación o herramientas de pruebas de webservices?

¿Se necesitan conocimientos en herramientas de pruebas automatizadas?

Requisitos de entrenamiento

Por ejemplo:

• Transferencia de conocimientos en el sistema y su operación con el área de negocio.

• Formación en metodologías de pruebas de software.

• Entrenamiento en tipos de pruebas especializados (desempeño, estrés, arquitectura, caja blanca).

• Formación en automatización de pruebas de software.

8.- Establecer la metodología y procedimientos de prueba

La metodología de pruebas de software dependerá de la que se esté utilizando para la gestión del proyecto.

Si se está utilizando una metodología predictiva, las pruebas de software comenzaran con la estimación del esfuerzo de pruebas, diseño y luego la ejecución de las pruebas, como te lo contamos en el artículo de Pruebas de calidad de software en 10 pasos.

Si se están utilizando metodologías ágiles de desarrollo de software, debes considerar las diferencias de las pruebas ágiles de software respecto al enfoque predictivo, por lo que la metodología debe estar alineada con el manifiesto ágil.

En nuestra serie de artículos de Agile Testing te contamos más al respecto.

Luego se seleccionar la metodología de referencia, se documentan los procedimientos para diseño y ejecución, siguiendo el orden de los pasos definidos, flujos de procesos, condiciones para tomar decisiones, y demás aspectos.

9.- Elaborar la planificación de las pruebas

La planificación de las pruebas abarca:

Matriz de responsabilidades

Puede usarse una Matriz RACI o Matriz RAM como plantilla. Esta se define con perfiles genéricos o inclusive con el equipo de trabajo si ya se conoce cuál es el que será asignado.

Las tareas del plan de pruebas deben estar alineadas con las habilidades y conocimientos de cada persona.

Cronograma

Elaborado a partir de la estimación de las actividades de Software Testing realizada por el equipo.

Para elaborar un cronograma real, es importante definir actividades críticas como por ejemplo los tiempos de instalación de versiones en los entornos de pruebas, pruebas de validación de ambientes antes de comenzar a hacer las pruebas y las iteraciones por incidencias, que es el tiempo invertido en volver a probar los casos de prueba fallidos.

Premisas

Son las condiciones que deben cumplirse para que el cronograma sea realizable, estas se determinan a partir de la documentación de entornos y de los requisitos de personal. Por ejemplo disponibilidad de ciertos entornos, disponibilidad de personal con algún conocimiento técnico especifico, la metodología que se va a utilizar, premias que deben cumplirse para poder aplicarla, entre otros.

10.- Identificar los riesgos y definir planes de respuesta

La gestión de riesgos en pruebas es muy similar a la gestión de riesgos en proyectos de la que hemos escrito en artículos como Las 5 preguntas y respuestas sobre la

identificación de riesgos, así como el artículo sobre Cómo hacer seguimiento de los riesgos del proyecto.

Para el Software Testing, los riesgos por lo general están vinculados con factores como:

• Posibles dificultades en la disponibilidad de entornos.

• Pruebas que dependen de factores externos al proyecto y la organización.

• Disponibilidad de personal con conocimientos especializados en alguna herramienta, o en la funcionalidad especifica que se está desarrollando.

• Dependencias con otros proyectos.

• Posibilidad que alguna premisa no se cumpla.

Para identificar los riesgos es necesario enumerar cada una de estas dependencias y por medio de mesas de trabajo y tormentas de ideas pensar en las posibilidades de que algo salga mal (u oportunidades para que salga bien).

Luego de la identificación, es necesario también definir planes de respuesta, los cuales deben ser específicos para cada situación particular y riesgo.

Tipos de pruebas de aplicaciones para celular

Un aspecto importante a considerar es el auge que están teniendo las aplicaciones para dispositivos móviles, lo cual abre toda una nueva gama de pruebas que debemos considerar en los planes de pruebas.

Pruebas de interrupción

La ejecución de aplicaciones móviles nativas o de la web móvil puede interrumpirse por distintos eventos desencadenados por el dispositivo, como por ejemplo una llamada entrante.

Es necesario definir casos de prueba para ver el comportamiento de la aplicación ante eventos como:

• Llamada entrante.

SMS entrante.

• Correo electrónico.

• Notificaciones de social media.

• Baja batería.

• Batería en estado crítico.

• Apagado del equipo.

• Caída de la conexión con la red, entre otros.

Pruebas de interfaz con el usuario (UI)

Buscan evaluar la interfaz con el usuario en relación con estándares para la interacción humano maquina aceptable. Entre las pruebas que podemos realizar tenemos:

• Organización de pantallas, alineación, colores, fondos y patrón de lectura.

• Posición, tamaño, datos de entradas y acciones.

• Claridad, alineación y densidad de imágenes y símbolos.

• Mensajes de error.

• Consistencia de la interfaz en toda la aplicación y entre dispositivos discimiles.

Pruebas de acciones del usuario

Hoy en día la interacción con dispositivos móviles ocurre mayormente a través de pantallas táctiles. Por lo tanto, es necesario probar la aplicación ante distintas acciones como tocar, doble tocar, arrastrar, rotar, voltear, extender los dedos, cerrar los dedos, entre otras.

Pruebas de usabilidad y accesibilidad

Este tipo de pruebas busca validar aspectos como:

• Presentación de la información en el diseño de página para móviles.

• Facilidad para completar tareas.

• Eficiencia y exactitud.

• Minimizar que el usuario tenga que recordar información.

• Tamaño de pantalla.

• Condiciones de iluminación.

• Tamaño de la interfaz táctil.

Integración de aplicaciones nativas, aplicaciones web móvil e híbridas.

Pruebas de Movilidad

Consiste en probar el desempeño de la aplicación cuando el dispositivo móvil se encuentra en movimiento, con una persona o vehículos. Esto es más importante en aplicaciones que utilizan la localización para mostrar información relevante al usuario.

Posibles pruebas a realizar:

• Geo localización (GPS, triangulación, etc.).

• Acelerómetro (movimientos, gestos).

• Magnetómetro (Brújula).

• Atravesar zonas de la red móvil (por ejemplo ir de una celda a otra).

• Señal de GPS, Wi-Fi o celular débil o intermitente.

Pruebas de conectividad

Desempeño de la aplicación al conectarse a las redes en distintos protocolos y distintas condiciones, por ejemplo:

• Wi-Fi.

• Bluetooth.

• Red analógica, 3G o 4G.

• Señales de distinta intensidad (sin señal, conexión intermitente, señal fuerte, intensidad de señal variable).

Pruebas de seguridad

Buscan validar la resistencia de la aplicación a ataques por usuarios maliciosos. Por ejemplo ataques via la red, ataques al servidor, ataques al dispositivo, análisis de flujo de datos e inyección, Jail Breaking, Testing de penetración.

Con este tipo de pruebas también se busca validar que los programadores apliquen prácticas de seguridad informática en el código de programa.

Pruebas de desempeño y estrés

Eficiencia en el consumo de batería.

• Desempeño lento.

• Cómo funciona la aplicación en modos de optimización y ahorro de energía del dispositivo.

• Pruebas con distintos niveles de batería (bajo, medio o alto).

• Utilización de procesador, memoria y espacio de almacenamiento.

• Pruebas de carga del lado de servidor (pruebas de estrés). Para lo cual pueden utilizarse herramientas como SoapUI.

• Pruebas de la red.

Pruebas de compatibilidad

Consisten en validar la compatibilidad de la aplicación con:

• Otras aplicaciones.

• Plataformas.

• Dispositivos.

• Navegadores.

• Redes de telecomunicaciones.

• Con versiones anteriores de la aplicación.

Validación de los lineamientos para envió a las App Stores

Las aplicaciones nativas Android y iPhone solo pueden distribuirse a través de sus respectivas tiendas online (Apps Stores). Las Apps Store poseen una serie de lineamientos que deben cumplir las aplicaciones.

Por ende es buena práctica que al testear aplicaciones para celular, se incluyan casos de prueba para validar el cumplimiento de estos lineamientos.

Para definir estos casos de prueba, referirse a los lineamientos de Apple para dispositivos iOS como los iPhone y las iPads, y los lineamientos de Google para teléfonos móviles y tabletas con sistema operativo Android.

B. Base de Consulta

TÍTULO	AUTOR	EDICIÓN	AÑO	IDIOMA	EDITORIAL
Ingeniería del Software: Un Enfoque Práctico	Pressman, R. S	7ª Edición	2010	Español	McGraw-Hill
El Lenguaje Unificado de Modelado. Manual de Referencia	Rumbaugh, J., Jacobson, I., Booch, G.	2ª Edición	2004	Español	Addison-Wesley
Fundamentos de Ingeniería de Software	Gómez, Cervantes y González	1ª Edición	2019	Español	Universidad Autónoma Metropolitana

C. Base práctica con ilustraciones

Modelamiento de la arquitectura del proyecto de software seleccionado con la aplicación de diagramas de casos de uso, diagramas de clase y diagrama de actividades

4. ESTRATEGIAS DE APRENDIZAJE

ESTRATEGIA DE APRENDIZAJE 1: Análisis y Planeación
Descripción: Discusión sobre las lecturas y artículos de procesos de ingeniería de software. Exposición conceptos y fundamentos de las diferentes metodologías de desarrollo, y de la arquitectura del software. Estudio de casos, análisis de soluciones informáticas a problemáticas seleccionadas. Estudio y trabajo en grupo, para el desarrollo de talleres de aplicación en actividades de diseño de software mediante diagramas. Estudio y trabajo autónomo para complementar los temas revisados en el aula referentes a levantamiento de requerimientos.
Ambiente(s) requerido: Aula amplia con buena iluminación.
Material (es) requerido: Proyector Pizarrón, marcadores Computador
Docente: Profesional con título en Ingeniería de Sistemas o afines, y experiencia en desarrollo de software

5. ACTIVIDADES

- Exposiciones, diálogos o discusiones
- Taller en grupo: Grupo 1 seis estudiantes, grupo 2 siete estudiantes, grupo 3 siete estudiantes, grupo 4 cuatro estudiantes.

- Evaluación con base estructurada
- Presentación del proyecto final

Se presenta evidencia física y digital con el fin de evidenciar en el portafolio de cada aprendiz su resultado de aprendizaje. Este será evaluable y socializable

6. EVIDENCIAS Y EVALUACIÓN

Tipo de Evidencia	**Descripción (de la evidencia)**
De conocimiento:	Evaluación de base estructurada
Desempeño:	Exposición individual: perfil del producto de software seleccionado para desarrollo. Trabajo grupal: Diagramas de: Casos de uso; Secuencia, Clases, Actividades y de Despliegue. Sistemas: Tickets-Fast para transporte, Cine Soft, BPT entrega de comida, TicketCines.
De Producto:	Documento final y diseño del proyecto de software seleccionado, aplicando una metodología de desarrollo
Criterios de Evaluación (Mínimo 5 Actividades por asignatura)	Ensayos Conversatorios Talleres en grupo Evaluaciones en linea de base estructurada Exposición del proyecto

Elaborado por: Ing. Eddy Vargas **(Docente)**	**Revisado Por:** Ing. Alexis Benavides **(Coordinador)**	**Reportado Por:** Dr. Milton Altamirano **(Vicerrector)**

INSTITUTO SUPERIOR TECNOLÓGICO

JAPÓN

Amor al conocimiento

Guía Metodológica de Desarrollo de software
Carrera de Desarrollo de Software
Ing. Eddy Vargas
2020

Coordinación editorial general:
Mgs. Milton Altamirano Pazmiño
Ing. Alexis Benavides Vinueza
Mgs. Lucia Begnini Dominguez

Diagramación:
Sebastián Gallardo Ramírez

Corrección de Estilo:
Mgs. Lucia Begnini Dominguez

Diseño:
Sebastián Gallardo Ramírez

Instituro superior tecnológico Japón
AMOR AL CONOCIMIENTO