



**GOBIERNO DE
MÉXICO**

EDUCACIÓN
SECRETARÍA DE EDUCACIÓN PÚBLICA



**DEPARTAMENTO ADSCRIPCIÓN
SISTEMAS Y COMPUTACIÓN**

**Opción elegida
B.1 Elaboración de Apuntes / B.1.2 Elaboración de apuntes de
asignatura:
“Ingeniería de Software”**

**INFORME FINAL DEL
PROYECTO DE SEMESTRE SABÁTICO**

Nombre del Docente: M. en TE ROSA ELVIRA MORENO RAMÍREZ

**No dictamen:
SS-1-088/2025**

**REVISOR:
M.C. Julieta Ruíz Jiménez
Imelda Vertti Guzmán
Mauro Sánchez Sánchez**

Fecha: 19 JULIO 2025

Introducción

El presente documento contiene los apuntes de la materia Ingeniería de Software, los cuales son la base para la creación de software de alta calidad, aportando al perfil del Ingeniero en Sistemas Computacionales las habilidades para abstraer información del usuario final para elaborar el análisis de requerimientos del software a desarrollar.

Se pretende que el estudiante adquiera las habilidades necesarias que complementan su perfil profesional tales como:

- Coordinar y participar en equipos multidisciplinarios desarrollando habilidades interpersonales, así como la capacidad de comunicación oral y escrita. para la aplicación de soluciones innovadoras en diferentes contextos,
- Diseñar, desarrollar e implementar aplicaciones computacionales para solucionar problemas de diversos contextos, tomando decisiones para ir integrando diferentes tecnologías, plataformas o dispositivos y apoyar así la productividad y competitividad de las organizaciones cumpliendo con estándares de calidad.
- Implementar interfaces para automatización de sistemas de hardware del software asociado.
- Desarrollar y administrar software, demostrando su capacidad de aplicar los conocimientos en la práctica.

El objetivo de haber seleccionado la opción de apuntes para esta materia constituye un esfuerzo encaminado a cubrir una de las principales necesidades del docente en el aula para la correcta impartición de una materia fundamental del perfil del Ingeniero en Sistemas Computacionales.

Dicha necesidad se centra en la recopilación de información seleccionada a partir de las diferentes fuentes especializadas y actualizadas para cubrir el programa de la materia.

La presentación de este trabajo se hace en formato que contiene la recopilación de material consultado integrando propuesta de ejercicios para el reforzamiento de los contenidos temáticos.

Estos conceptos teóricos no se encuentran condensados en una sola fuente bibliográfica sino al producto de investigar y consultar una serie de libros y fuentes diversas a fin de cumplir con los objetivos establecidos dentro del programa de la asignatura, de ahí la importancia que adquiere este trabajo como coadyuvante a esta labor.

Anexo entrego este trabajo como reporte de las actividades desarrolladas dentro del periodo de semestre sabático en los formatos digitales.

ATENTAMENTE

M. en T.E ROSA ELVIRA MORENO RAMÍREZ

JULIO 2025

Contenido

TEMA 1. ANÁLISIS	6
Introducción.....	7
1.1 Revisión de especificación de requisitos	12
1.1.1 Norma IEEE830-1998.....	15
1.1.2 Trazabilidad de requisitos	17
1.2 Descripción de procesos actuales	20
1.3 Diagramas UML	22
1.3.1 Diagramas Estructurales.....	22
1.3.2 Diagramas Comportamentales	27
1.4 Estudio de Factibilidad	32
1.5 Análisis Costo-Beneficio.....	34
ACTIVIDADES TEMA 1.....	38
TEMA 2. DISEÑO	43
Introducción.....	44
2.1 Diseño de procesos propuestos	47
2.1.1 Herramientas CASE para diseño	48
2.2 Diseño arquitectónico.....	51
2.3 Diseño de datos	55
2.4 Diseño de interfaz de usuario.....	58
ACTIVIDADES TEMA 2.....	65
TEMA 3. DESARROLLO.....	67
Introducción.....	68
3.1 Lenguajes para el desarrollo del software	70
3.2 Manejadores de bases de datos.....	75
3.3 Construcción del sistema	77
ACTIVIDADES TEMA 3.....	85
TEMA 4. PRUEBAS E IMPLANTACIÓN.....	86
Introducción.....	87
4.1 Diseño de caso de prueba.....	89
4.2 Pruebas de componentes.....	94
4.3 Pruebas del sistema.....	98
4.4 Documentación de resultados de las pruebas.	103
4.5 Entrega del sistema y Capacitación a usuarios	107
4.6 Entrega de documentación técnica y de usuario del sistema.	109
ACTIVIDADES TEMA 4.....	114

Fuentes Bibliográficas.....	117
Fuentes Electrónicas	119

Contenido de figuras

Figura 1. 1 Ciclo de Vida de Desarrollo de Software	8
Figura 1. 2 Actividades para seguimiento de revisión de requisitos.....	14
Figura 1. 3 Nomenclatura de una Clase.....	23
Figura 1. 4 Diagrama de Clases.....	24
Figura 1. 5 Diagrama de Objetos	24
Figura 1. 6 Diagrama de componentes Sistema de Gestión de Biblioteca	25
Figura 1. 7 Diagrama de Despliegue del Sistema de Gestión Hospitalaria.....	27
Figura 1. 8 Representación de Diagrama de Caso de Uso	27
Figura 1. 9 Diagrama de Caso de Uso	28
Figura 1. 10 Diagrama de Secuencia	28
Figura 1. 11 Diagrama de Secuencia	29
Figura 1. 12 Representación del Diagrama de Actividades	29
Figura 1. 13 Diagrama de Actividad	30
Figura 1. 14 Diagrama de estado.....	31
Figura 2. 1 Traducción del Modelo de Análisis al Modelo de Diseño (Pressman, 2007)	44
Figura 2. 2 Arquitectura en capas	51
Figura 2. 3 Arquitectura de flujo de datos.....	52
Figura 2. 4 Arquitectura orientada a servicios (SOA).....	52
Figura 2. 5 Arquitectura orientada a microservicios.....	53
Figura 2. 6 Arquitectura cliente-servidor.....	53
Figura 2. 7 Diseño arquitectónico de un sistema de suscripciones	54
Figura 2. 8 Ejemplo del Diagrama Entidad-Relación	55
Figura 2. 9 Interfaz sistema Mecánico Automotriz	59
Figura 3. 1 Metodología de Proceso Iterativo-Incremental (Somerville, 2005).....	68
Figura 3. 2 Creación de Tablas para sistema de Ventas.....	76
Figura 4. 1 Estrategia de pruebas (Pressman, R, 2010).....	88
Figura 4. 2 Actividades para diseño de caso de prueba	89
Figura 4. 3 Pasos de la prueba de Software (Pressman, 2010)	94

Contenido de tablas

Tabla 1. 1 Matriz de trazabilidad de Requisitos	19
Tabla 1. 2 Análisis Costo-Beneficio para desarrollo de un Sistema de Biblioteca	36
Tabla 3. 1 Lenguajes de programación	72
Tabla 3. 2 Lista de DBMS con principales características	75
Tabla 4. 1 Ejemplo de un Caso de Prueba	90
Tabla 4.2 Ejemplo de Formato de Prueba unitaria de un sistema de inventarios.....	92
Tabla 4.3 Ejemplo de formato de Pruebas de Componentes de Software	96
Tabla 4.4 Ejemplo de Formato de Prueba de Integración.....	97

Tabla 4.5 Ejemplo del formato para Prueba de Seguridad	99
Tabla 4.6 Ejemplo de formato de prueba de esfuerzo (estrés)	102

TEMA 1. ANÁLISIS

Introducción

Para comprender el concepto de Ingeniería de Software, se recomienda a los analistas de sistemas entender los conceptos básicos que abordan esta disciplina, comenzando con la identificación de las necesidades de información de los clientes y la calidad del producto a entregar, generalmente sistemas de información.

Partiendo de concepto de sistema “es el conjunto de elementos que interactúan para lograr un objetivo”, un Sistema de Información es el conjunto de elementos, hardware (el uso efectivo del equipo), software (programas), datos, procedimientos y usuarios (personas) que interactúan para alcanzar los resultados específicos que apoyan los objetivos del negocio o compañía.

Ejemplos de sistemas pueden ser: el inventario de una papelería o el sistema de nómina.

Para la creación de un sistema de información con calidad, se sigue una serie de pasos llamado el Ciclo de Vida de Desarrollo de Software (CVDS), comenzando con la identificación de la necesidad de información hasta que el sistema es sustituido ya sea de uno anterior con procedimientos manuales o un sistema obsoleto.

Su estudio se basa en una metodología que se divide en las siguientes fases:

- 1- Investigación Preliminar (Incluye la Planeación)
- 2- Análisis
- 3- Diseño
- 4- Codificación
- 5- Pruebas
- 6- Implementación
- 7- Mantenimiento

Algunos autores manejan la fase 8 como documentación, pero es imprescindible la documentación desde la primera fase.

La figura 1.1 muestra la secuencia del CVDS.

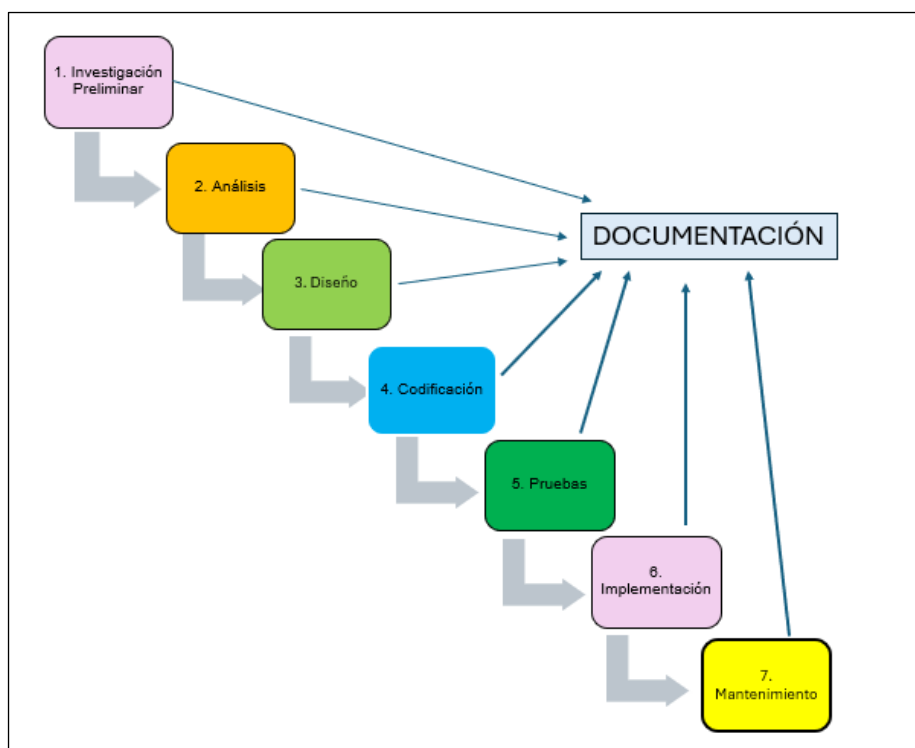


Figura 1. 1 Ciclo de Vida de Desarrollo de Software

Todo proyecto debe ser documentado desde su inicio y con este tipo de metodología se estará asegurando la calidad el software.

Una vez implementado el sistema agregaría el proceso de mantenimiento, no como una fase, sino como las actividades necesarias para dar continuidad a la funcionalidad del sistema hasta que este llegue a ser obsoleto que es cuando se dice que terminó su ciclo de vida.

Fase 1: Investigación Preliminar

El propósito de esta fase es identificar claramente la naturaleza y el alcance del problema.

Comienza con un pedido escrito llamado “system request”, donde el cliente plantea una problemática que se presenta debido a la ausencia de información sistematizada y pueden ser cambios mayores (un nuevo sistema) o cambios menores como pueden ser:

Mejorar el servicio – Proveer nuevas herramientas al usuario para que realice su trabajo.

Mejorar el desempeño del sistema – Son solicitudes que se hacen cuando un sistema no cumple con los requisitos de desempeño. Por ejemplo, un sistema muy lento o que no es compatible con un equipo nuevo.

Más información – Puede que la información actual no sea suficiente para satisfacer las necesidades cambiantes de la compañía.

Controles más fuertes – El sistema debe tener controles efectivos que aseguren que los datos son correctos y seguros. Incluye el uso de “passwords”, niveles de accesos, etc.

Reducir costos – Puede que el sistema actual sea costoso para operar o mantener, ya sea por problemas técnicos, cambios en la tecnología, demanda del negocio, etc.

Las solicitudes al sistema pueden ser motivados por factores internos o factores externos de la organización. Algunos factores internos son:

- los usuarios

- la alta gerencia
- los sistemas existentes
- el mismo departamento de sistemas de información

Los factores externos que causan solicitudes al sistema son:

- cambios en la economía
- cambios en leyes del gobierno
- la tecnología
- los clientes
- la competencia

La determinación de si el proyecto es viable a desarrollar depende de la naturaleza y alcance del problema, el analista de sistemas debe comenzar con la elaboración del plan de actividades de los procesos a llevar a cabo asignando la jerarquía para la atención de dichos procesos a desarrollar.

Esta es una etapa de investigación que no se basa solamente en la recolección de datos, ni que la definición precisa de la problemática a atender, sino la atención de los siguientes objetivos:

1º. Entender la naturaleza del problema: Es el objetivo más importante, por lo que es necesario interactuar con el usuario del sistema para identificar claramente el requisito de información y plantear la solución a las áreas de oportunidad detectadas. Es conveniente eliminar la palabra problema en el lenguaje del analista de sistemas para evitar una impresión negativa al usuario.

2º. Definir el alcance, restricciones o limitaciones del sistema: Esta parte es la clave para guiar al cliente a que defina hasta dónde se debe llegar. Debe determinar los procesos afectados por la falta de información apoyando al usuario para que describa las restricciones que controlará el sistema y las limitaciones que puede contener, solicitándole documentación que defina los procesos de la empresa, por el ejemplo un manual de procedimientos, es decir, algún documento que involucre el manejo de equipo, programas, tiempo, leyes, costos, entre otros.

3º. Identificar beneficios que se obtendrían si el sistema propuesto es implementado. Estos pueden ser catalogados como beneficios tangibles (que se pueden expresar en términos de dinero) o beneficios intangibles (información disponible para tomar decisiones, satisfacción de los empleados, mejorar la imagen de la compañía y otros aspectos que no se miden en término de dinero).

4º. Se debe presentar un estimado del tiempo y el costo que tomará realizar cada uno de las siguientes fases del desarrollo del sistema y del costo que la compañía debe incurrir para completar el sistema. Es recomendable incluir los costos de desarrollo separando los que incurren solo una vez y los costos continuos que se pagan periódicamente.

5º. Presentar un informe a la gerencia describiendo el problema y detallando si se recomienda continuar con la fase de análisis del sistema. Es importante mencionar que se debe realizar a la par el documento de requisitos previos del sistema e incluir la evaluación del “system request”, que identifica el sistema de información y los cambios deseados, estimado de tiempo, costo-beneficios y las recomendaciones.

-Evaluación de los “system request”

Se realizan de diferentes formas dependiendo de la organización. Si el cambio que se pide es mínimo, tal vez sólo necesita la evaluación del analista de sistemas o del gerente del departamento de sistemas de información. Sin embargo, si el pedido requiere muchos cambios al sistema actual o un sistema totalmente nuevo, probablemente sea un comité el que evalúe el “system request”.

Una de las primeras preguntas que se debe hacer sobre el “system request” es su viabilidad.

Pasos para realizar la investigación preliminar:

1. Obtener la autorización de la gerencia (si el proyecto es en empresa)

2. Identificar la información necesaria para el proyecto para cumplir con los cinco objetivos de la investigación (ya mencionados).
3. Realizar las acciones que sean necesarias para conseguir la información, como por ejemplo:
 - a. Analizar el organigrama para conocer la estructura de los departamentos y las personas claves para el sistema.
 - b. Realizar entrevistas a los usuarios, éste es el método principal de obtener información.
 - c. Revisar la documentación actual, validando con los usuarios si la documentación es correcta y completa.
 - d. Observar la operación actual para identificar fuentes de Input y Output.
 - e. Realizar encuestas, método usado cuando se necesita información de muchas personas.
4. Analizar la información obtenida, identificando alternativas con sus costos y beneficios y recomendando la acción que se debe tomar.
5. Presentar los resultados y recomendaciones a la gerencia. Este documento se llama Informe de Investigación Preliminar. La investigación preliminar también es conocida como Estudio de Viabilidad.

Existen tres tipos de viabilidad (feasibility): viabilidad operacional, viabilidad técnica y viabilidad económica.

- **Viabilidad operacional:** Un sistema es operacionalmente viable si será utilizado efectivamente después de su desarrollo. Si los usuarios tienen dificultad con el nuevo sistema, éste no producirá los beneficios esperados.

Es necesario preguntarse:

- la gerencia y los usuarios apoyan el proyecto?
- el nuevo sistema requiere capacitación para usuarios y la compañía está preparada para ofrecerlo?
- el nuevo sistema traerá como consecuencia una reducción del personal?
- los usuarios estarán involucrados en planificar el nuevo sistema desde sus comienzos?
- el nuevo sistema requiere algún cambio en la manera en que se realizan las tareas?
- los clientes se verán afectados?
- el itinerario para el nuevo sistema es adecuado?
- se debe considerar alguna situación legal o ética?

Viabilidad Técnica: Un sistema es técnicamente viable si la organización tiene los recursos para desarrollarlo o comprarlo, instalarlo y operarlo. Se debe incluir lo siguiente:

- si la organización tiene el equipo necesario para el sistema o si puede adquirirlo sin dificultad
- si el equipo propuesto tiene la suficiente capacidad para las necesidades futura
- si la plataforma de hardware y software son confiables, integrables con otros sistemas de la compañía y de los clientes
- la combinación del equipo y los programas ofrecen el rendimiento adecuado y esperado

Viabilidad Económica: Un sistema es económicamente viable si los beneficios proyectados superan los costos estimados en el desarrollo o la compra, instalación y operación del sistema. Los costos pueden ser "one-time" (que ocurren una sola vez) o continuos. Para determinar esta viabilidad, es necesario estimar adecuadamente el costo para satisfacer las necesidades en:

- personal, incluyendo los usuarios y el personal de sistemas
- equipos y programas, incluyendo tanto los programas desarrollados en la compañía como los comprados
- capacitación formal e informal
- licencias

- costos por consultoría y otros.

Fase 2: Análisis

Una vez obtenido el informe de Investigación Preliminar autorizado por la gerencia, se continuará con la fase de análisis del sistema, actividad que realiza el analista con la recopilación de requisitos, centrada totalmente en las especificaciones de operación del software a desarrollar, basado en la creación del modelo de negocio, para que el usuario pueda revisar el flujo de información representada en un diagrama de bloques para la validación del proceso.

Fase 3: Diseño. Esta etapa está íntimamente relacionada con el paradigma que se esté usando (ya sea de manera estructurada u orientada a objetos), en función de este, es el tipo de diagramas que se implementan para plantear la solución del sistema. El caso de sistemas estructurados se puede modelar con diagramas DFD (Diagramas de Flujo de Datos), DP (Diagrama de Procesos), o algoritmos. En el caso de la orientación a objetos se consideran principalmente UML (Unified Modeling Lenguaje), y DFO (Diagrama de Flujo de Objetos).

Fase 4: Codificación. Es la etapa donde los diseños son interpretados por los programadores, y codificados en un lenguaje de programación que cubra con los requisitos de operación del sistema, como: rendimiento, precio, viabilidad, soporte, portabilidad, etc.

Fase 5: Pruebas. Aunque existen diversos métodos para aplicar las pruebas del software, se consideran principalmente dos tipos de pruebas: caja blanca y caja negra; la primera de ellas es para garantizar que cada una de las líneas de código, han sido probadas y con ello minimizar el problema de errores (lógica interna). Las pruebas de caja negra buscan los problemas de interpretación o presentación en pantalla (lógica externa) de la aplicación.

Fase 6: Implementación. Esta fase comprende el proceso de verificar e instalar nuevo equipo, capacitar a los usuarios, instalar la aplicación y construir todos los archivos de datos necesarios para utilizarla, para entregar un sistema de información completo y documentado, que haya sido revisado y aprobado por la gerencia y usuarios.

Los preparativos finales incluyen la conversión de datos, adiestramientos y la transición del sistema viejo al nuevo. En esta fase se debe realizar una evaluación del sistema luego de implantado para verificar costo-beneficio.

Fase 7: Mantenimiento. El software indudablemente sufrirá cambios una vez entregado al cliente, debido a nuevas aplicaciones, problemas en la aplicación, cambios en las políticas de la empresa o gubernamentales y por actualizaciones del mismo producto en el que fue desarrollado.

1.1 Revisión de especificación de requisitos

El propósito de la fase Análisis, es conocer exactamente cómo trabaja el sistema actual, sea manual o computadorizado, lo que se necesita para apoyar las necesidades del negocio, determinar y documentar qué debe hacer el sistema y recomendar las posibles soluciones. Es una etapa fundamental en el desarrollo de sistemas de información que ayuda a garantizar que el producto final sea el adecuado, cumpla con las expectativas del cliente y sea factible de desarrollar, evitando costosos errores en fases posteriores del proyecto.

Un nuevo sistema requiere planificación, construcción y prueba. Los programas y módulos deben ser diseñados, codificados, probados y documentados. Cuando se planifica el sistema, muchas veces se usa un estilo de arriba-hacia-abajo (top-down), que procede de un diseño general a una estructura detallada siguiendo unos pasos lógicos.

En el estilo top-down, el analista de sistemas define los objetivos generales, y luego los descompone en subsistemas y módulos en un proceso llamado “partitioning”. Este estilo también se conoce como diseño modular.

Se utilizan diferentes herramientas para explicar cómo funciona el sistema actual. Primero se hace una determinación de requisitos, donde se define todas las funciones que realiza el sistema actual y se determinan las mejoras necesarias.

Luego se desarrolla un plan específico para resolver los problemas del sistema actual; este proceso se llama Análisis de requisitos. El producto de esta fase es el Documento de requisitos del sistema, donde se describe los requisitos de la gerencia y los usuarios, los diferentes planes, sus costos y la recomendación.

Los pasos a seguir en el análisis del sistema son:

1. Determinar los requisitos del sistema
2. Analizar los requisitos
3. Llegar a conclusiones y presentar recomendaciones a la gerencia.

La ingeniería de software es una disciplina que se ha implementado para mejorar la calidad del desarrollo de sistemas de información. Incluye todas las características necesarias para la creación del software, desde la etapa inicial que es el análisis de los requerimientos del cliente, modelando y diseñando estos requerimientos a través de técnicas y herramientas aplicadas para interpretar y

comprender todos los procesos que se requieren automatizar, el desarrollo del software hasta el mantenimiento, que es la etapa en la que se detectan requerimientos posteriores a los ya establecidos y especificados en el desarrollo de las primeras versiones.

Compromete al desarrollador a realizar un sistema de información basado en los requerimientos reales del cliente.

El análisis de requerimientos es la actividad principal en el desarrollo de sistemas del cual se parte a través de diferentes técnicas de abstracción de la información, como la entrevista, entre otras.

La entrevista con el cliente se puede considerar la principal fuente de obtención de información, de la cual se obtiene el objetivo principal y el alcance del sistema para realizar el estudio de factibilidad técnica y determinar si es posible el desarrollo del sistema.

Para que el desarrollo de software se lleve a cabo de manera eficiente, es necesario recopilar toda la información que se requiere y llevar a cabo en forma continua la **revisión de especificación de requisitos**, actividad indispensable para asegurar que los requisitos definidos para el desarrollo del sistema de información sean precisos, completos y viables, porque es en la fase de análisis donde los requisitos se revisan, validan y verifican para detectar posibles errores, inconsistencias, omisiones o ambigüedades antes de comenzar con el desarrollo con el fin de garantizar que sean claros, completos, coherentes y correctos.

Sommerville destaca que “la ingeniería de software es una disciplina que los ingenieros hacen que las cosas funcionen. Aplican teorías, métodos y herramientas donde sea conveniente utilizarlas, de forma selectiva siempre tratando de atender los problemas aun cuando no existan teorías ni métodos aplicables para resolverlos, comprende también actividades para resolverlos, como la gestión de proyectos y el desarrollo de herramientas, métodos y teorías de apoyo a la producción del software” (Sommerville, 2005).

Propósito de la revisión de especificación de requisitos

1. **Validación de requisitos:** Se verifica que los requisitos sean correctos y estén alineados con las necesidades del cliente o usuarios. Según Sommerville (2011), este proceso ayuda a garantizar que el sistema desarrollado realmente resuelva el problema o satisfaga la necesidad para la cual fue solicitada.
2. **Identificación de ambigüedades y errores:** Durante la revisión, se deben identificar posibles ambigüedades en el lenguaje de los requisitos que puedan generar interpretaciones diferentes. Según Pressman (2014), las revisiones permiten detectar problemas en las etapas iniciales del proyecto, lo que reduce los riesgos de fallas en fases posteriores del desarrollo.

3. **Verificación de consistencia y viabilidad:** Es fundamental que los requisitos sean consistentes entre sí y que puedan ser implementados dentro de las restricciones de tiempo, recursos y presupuesto. El análisis de viabilidad técnica asegura que los requisitos sean alcanzables con la tecnología disponible (Wiegers & Beatty, 2013).

Actividades en una revisión de especificación de requisitos.

Cuando ya se tiene definido el documento de especificación de requisitos (SRS), es importante dar continuidad y seguimiento mediante actividades de revisión de documentos, reuniones con interesados, análisis de riesgos y en algunas ocasiones, implementar prototipos o simuladores con la información entregada por el cliente y probar si los requisitos cumplen con las expectativas de los usuarios y las capacidades del sistema (Sommerville, 2011).

La figura 1.2 muestra las actividades sugeridas para llevar a cabo la continuidad de las revisiones de especificación de requisitos:

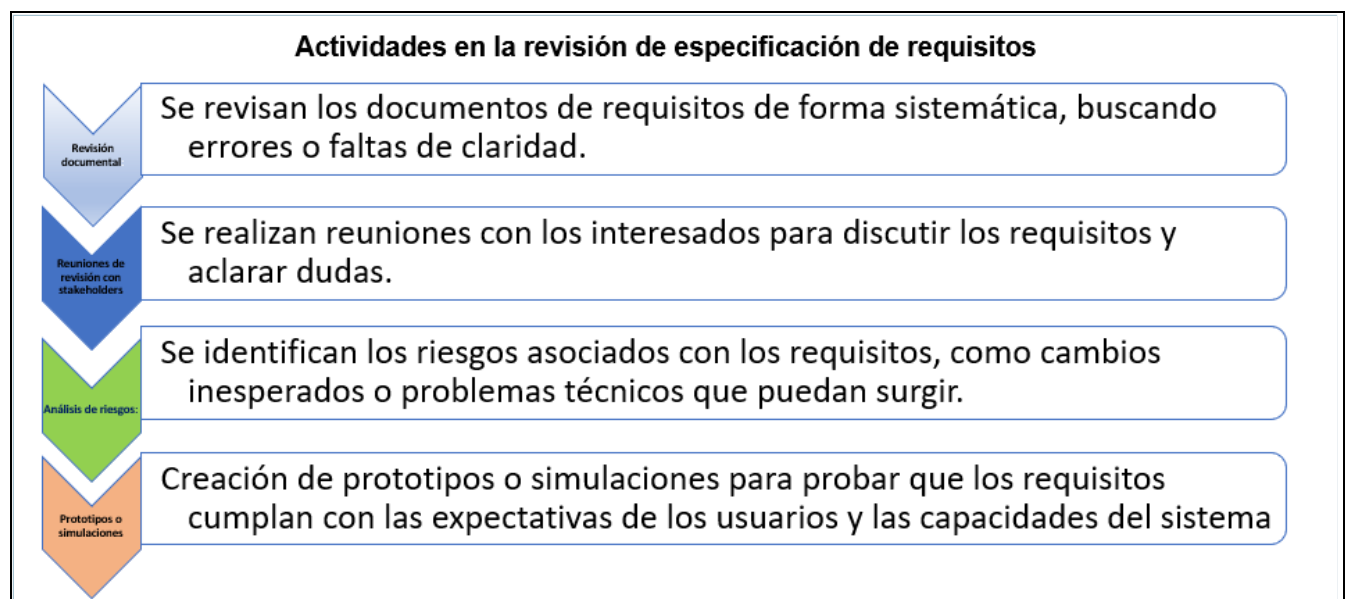


Figura 1. 2 Actividades para seguimiento de revisión de requisitos

Beneficios de la revisión de especificación de requisitos

- **Mejora de la calidad del software:** Al detectar errores o malentendidos en los requisitos, se evita que estos se conviertan en problemas durante el desarrollo, lo que mejora la calidad del sistema final (Pressman, 2014).
- **Reducción de costos y tiempo:** Corregir errores en las primeras etapas del desarrollo es más económico que hacerlo en etapas posteriores, cuando el código ya está desarrollado y los cambios son más costosos (Wiegers & Beatty, 2013).

- **Alineación con las expectativas del cliente:** Una revisión exhaustiva asegura que los requisitos coincidan con las expectativas del cliente, lo que mejora la satisfacción y reduce el riesgo de modificaciones de última hora.

El principal objetivo de la ingeniería de software es el desarrollo de software con calidad, por lo que es importante mantener estándares en la documentación de la especificación de requisitos, con lo que se apoya en diferentes estándares de calidad que ya han sido implementados en diferentes empresas.

Uno de ellos es el estándar del **IEEE830** que define cómo especificar los requerimientos de software, desarrollado por el Instituto de Ingenieros Eléctricos y Electrónicos (Institute of Electrical and Electronics Engineers, por sus siglas en inglés) es la sociedad profesional técnica más grande del mundo, dedicada a promover la innovación tecnológica y la excelencia en beneficio de la humanidad.

Otra herramienta muy utilizada para la especificación de requisitos es la matriz de trazabilidad, en la que se detallan las características que debe cumplir el sistema de manera práctica para un adecuado seguimiento al cumplimiento de lo que solicita el cliente.

A continuación, se describen cada una de estas herramientas para la documentación de requerimientos de información del cliente.

1.1.1 Norma IEEE830-1998

La norma **IEEE 830-1998** es un estándar importante en la ingeniería de software que proporciona directrices para la especificación de requisitos de software y asegura que los requisitos sean completados y comprendidos correctamente ya que establece un formato común para documentar estos requisitos de manera clara, coherente y comprensible, lo que facilita la comunicación entre los stakeholders del proyecto: clientes, desarrolladores, entre otros (Pressman, 2014).

El **IEEE 830-1998** define los requisitos que debe incluir un **Documento de Especificación de Requisitos de Software (SRS)** . Este documento tiene como objetivo describir qué debe hacer el sistema, sin entrar en detalles sobre cómo se implementará.

Una especificación bien escrita y completa reduce el riesgo de malentendidos y cambios costosos durante las fases posteriores del proyecto (Wiegiers & Beatty, 2013).

El propósito es proporcionar una base sólida para la ingeniería de software, abordando tanto los **requisitos funcionales**. como los **no funcionales**.

La norma **IEEE 830-1998** proporciona un conjunto de directrices esenciales para la creación de especificaciones de requisitos en proyectos de ingeniería de software, asegurando que los requisitos

sean documentados de manera clara y comprensible. Su uso es fundamental para garantizar el éxito de un proyecto de software, ya que establece una base sólida para la planificación, el diseño y la implementación del sistema.

A continuación, se mencionan los elementos clave del Documento de Especificación de Requisitos de Software (SRS) según **IEEE 830** (este formato se encuentra en el apartado de Anexos de este documento):

1. Introducción :

- Descripción general del sistema que se va a desarrollar.
- Propósito del software, su ámbito y los requisitos de alto nivel.
- Definición de las interfaces de usuario y de hardware, si aplica.

2. Descripción general :

- Requisitos funcionales y no funcionales.
- Restricciones del sistema (por ejemplo, en cuanto a hardware, interfaces, etc.).
- Sucesos o condiciones que podrían afectar el rendimiento del sistema.

3. Requisitos específicos :

- Descripción detallada de las funcionalidades y comportamientos del sistema.
- Descripción de entradas, salidas, procesos y reglas de negocio.
- Especificaciones técnicas y de rendimiento.

4. Requisitos no funcionales :

- Requisitos relacionados con la usabilidad, rendimiento, seguridad, confiabilidad, etc.
- Restricciones de diseño o implementación.

5. Apéndices :

- Detalles adicionales que puedan ser útiles, como diccionarios de datos o diagramas.

1.1.2 Trazabilidad de requisitos

En la mayor parte del desarrollo de sistemas de información, se les exige a los programadores de manera legal que muestren no solo que los requisitos hayan sido implementados de manera correcta sino además que no exista algún código no identificable, es decir, sin registros o trazas.

La trazabilidad de requisitos es "el proceso de documentar las relaciones entre los requisitos, el diseño, la implementación y las pruebas de un sistema, lo que permite seguir el camino de los requisitos a través de todas las fases del ciclo de vida del software" (Wiegers, 2003).

El Instituto de Ingeniería de Software (SEI, por sus siglas en inglés) ha definido estándares que requieren que exista en el proceso una trazabilidad básica. (SEI, 2001).

Por otra parte, la certificación de cualquier proceso, a partir de las normas ISO 9000, exige la existencia de evidencias documentales sobre el cumplimiento de cada etapa del desarrollo, de las revisiones del cumplimiento de los requisitos establecidos por el cliente y demás partes interesadas, y de la validación y verificación del diseño y desarrollo.

La gestión de la configuración, dentro de la norma ISO define la necesidad de proporcionar identificación y trazabilidad, el estado de cumplimiento de los requisitos físicos y funcionales del producto, y acceso a información exacta en todas las fases del ciclo de vida. (ONN, 2009).

Por la necesidad de mantener un proceso formal para la adecuada documentación de requisitos para el desarrollo del software, surge la ingeniería de requisitos (IR) como un proceso tradicional del desarrollo de software que tiene como objetivos identificar, analizar, documentar y validar requisitos del sistema que se va a desarrollar.

Es frecuente encontrar en la literatura que la IR y en particular los modelos de trazabilidad son incompatibles con las prácticas de desarrollo ágil; debido en lo fundamental porque la IR hace un marcado énfasis en la documentación para compartir e intercambiar conocimiento mientras que los métodos ágiles se enfocan en la colaboración cara a cara entre clientes y desarrolladores para alcanzar objetivos similares (Fowler, 2005).

En ingeniería de software, **la trazabilidad de requisitos** se refiere a la capacidad de seguir (o rastrear) los requisitos a lo largo de todo el ciclo de vida del proyecto, desde su definición hasta su implementación y verificación. Es un proceso que asegura que los requisitos iniciales del sistema sean correctamente comprendidos, se implementen adecuadamente y facilita la gestión de cambios, la validación del sistema y la comunicación entre las partes interesadas.

A través de la trazabilidad, se puede verificar que todos los requisitos definidos se hayan considerado durante el diseño, desarrollo, pruebas y mantenimiento del software.

La trazabilidad de requisitos ayuda a:

1. **Garantizar que se cumplan los requisitos:** Permite verificar que cada requisito del sistema esté cubierto por una funcionalidad del software y que haya sido implementado correctamente.
1. **Facilitar la gestión de cambios:** Si un requisito cambia durante el proyecto, la trazabilidad permite identificar qué otros elementos (como casos de prueba, diseño, código, etc.) se ven afectados, facilitando el proceso de actualización y asegurando que el cambio no cause problemas en otras áreas.
2. **Mejorar la comunicación:** La trazabilidad ayuda a que todas las partes interesadas (clientes, desarrolladores, testers) tengan claro cómo se están abordando los requisitos y cómo se están implementando en el sistema.
3. **Verificación de la cobertura y validación:** Asegura que todos los requisitos tengan una correspondencia directa con elementos tangibles en el sistema, como el código, las pruebas o los documentos. Permite verificar que el sistema final cumple con todos los requisitos definidos inicialmente, lo que es clave en procesos de validación y verificación (Pressman, 2014).

Existen herramientas y técnicas específicas para gestionar la trazabilidad de requisitos, como matrices de trazabilidad, que vinculan los requisitos con otros artefactos del proyecto (diseño, pruebas, etc.).

Según Sommerville (2011), la trazabilidad es crucial porque "permite asegurar que todos los requisitos del sistema se han implementado correctamente y que el sistema cumple con las expectativas de los usuarios y otras partes interesadas."

La trazabilidad de requisitos también facilita la gestión de calidad en el desarrollo y el mantenimiento de sistemas de software, garantizando que se mantenga la coherencia entre los requisitos, el diseño y la implementación.

Aunque no existe un formato definido como un estándar para seguir la trazabilidad de requisitos, se sugiere que cada equipo de desarrollo de software proponga e implemente formatos personalizados dependiendo del nivel de complejidad del software.

Algunos elementos por considerar para definir estos formatos son los siguientes:

ID: Identificador que se asigna a cada uno de los requisitos y que corresponderá con el número de requisito definido en la Lista de Requisitos del Proyecto. Generalmente corresponde a un código o número asignado a cada uno de los requisitos del proyecto.

Nombre del requisito: Es la descripción que lo hace único

Tipo: Se refiere a la clasificación que se quiere establecer dentro del proyecto. Tanto éste como cualquier otro atributo del requisito que se quiera incluir en la matriz de seguimiento, ayudarán a una mejor gestión y trazabilidad de su evolución, y asegurar el resultado final del proyecto.

Prioridad: Es el valor que se asigna con base en la urgencia o necesidad primordial del cliente. Es útil para valorar las decisiones a tomar durante la evolución del proyecto. No todos los requisitos tienen la misma importancia. Hay primarios, secundarios y terciarios en función del valor que supongan para el proceso en su conjunto. La matriz debe dejar claro este dato.

Interesado o destinatario: Es la persona que ha originado el requerimiento o requisito (stakeholder).

Estado (del requisito): Se refiere al estado (status) en que se encuentra el requisito establecido, y que generalmente será “activo” (o similar), pero es posible que pueda cambiar, por ejemplo, si se renunciara a cierta funcionalidad o si algún cambio hiciera que el requisito en cuestión pasara a “cancelado”, aunque ello quizás generara la aparición de otros requisitos.

La tabla 1.1 Matriz de trazabilidad de Requisitos muestra el ejemplo de la matriz de trazabilidad de requisitos para el desarrollo del sistema de control de actividades del Departamento de Orientación Educativa de una institución de educación superior.

Tabla 1. 1 Matriz de trazabilidad de Requisitos

Matriz de Trazabilidad de Requisitos								
					Fecha	22/02/2024		
Nombre del Proyecto					Código Proyecto	20176		
ID	REQUISITO	TIPO	PRIORIDAD (1-5)	OBJETIVO	DOCTOS. ENTRADA	ENTREGABLES	ESTADO (entregabl)	VALIDACION (persona/fecha)
1	Proceso de Admisión	Complejo	5	Seleccionar al alumnado mas prometedor en su carrera profesional	Ficha Escuela Ficha ceneval	Folio Aceptación Lista aprobados ceneval	Inactivo	Ana Laura 18/01/24
2	Proceso de Atención al alumno aceptado	Medio	4	Atención aspirantes aceptados de proceso de admisión.	Lista aprobados	No. Control Asignado Agenda eventos Carrera	Inactivo	Ana Laura 18/02/24
3	Seguimiento y Control Actividades Orientacion Educativa	Complejo	4	Apoyar en la formación integral ofreciendo conferencias de superación personal	Lista de detección de necesidades por departamento	Agenda de cursos de actualización profesional y formación docente	Inactivo	Ana Laura 10/01/24
4	Proceso de Cambio de Carrera	Simple	3	Dar apoyo al alumno que desea cambiar e ingresar a otro programa educativo	Solicitud de cambio de carrera	Folio de cambio de carrera	Inactivo	Bertha 15/02/24

En resumen, la trazabilidad de requisitos es fundamental para mantener la calidad, el control y la organización en proyectos de software complejos.

1.2 Descripción de procesos actuales

En ingeniería de software, la **descripción de procesos actuales** se refiere a la documentación y análisis detallado de los procesos, flujos de trabajo y actividades que se están llevando a cabo en un sistema o proyecto en su estado actual. Este proceso es clave para entender cómo se están realizando las tareas en la actualidad, sin considerar aún los posibles cambios o mejoras, y proporciona una base para la evaluación, mejora y optimización de dichos procesos.

Definición y Objetivos

La **descripción de procesos actuales** tiene como objetivo proporcionar una visión clara y precisa de cómo se ejecutan los procesos dentro de una organización, equipo o sistema. Esto ayuda a identificar ineficiencias, cuellos de botella, redundancias y otras áreas susceptibles de mejora. También proporciona un punto de referencia para comparar el estado actual con los futuros procesos optimizados.

Según Sommerville (2011), en su libro *Software Engineering*, "la descripción de procesos actuales implica comprender cómo se gestionan las actividades de desarrollo de software, cómo se toman las decisiones, y cómo las herramientas y las personas interactúan dentro del proceso."

Importancia

1. **Entender el estado actual:** Permite tener una visión clara de cómo se están ejecutando las tareas y procesos, lo cual es esencial para la toma de decisiones y para entender las prácticas vigentes.
2. **Identificación de áreas de mejora:** Como sostiene Pressman (2014) en *Software Engineering: A Practitioner's Approach*, "la descripción de los procesos actuales es crucial para detectar ineficiencias, redundancias y cuellos de botella que podrían afectar la productividad y calidad en el desarrollo del software."
3. **Base para la mejora continua:** La documentación del proceso actual sirve como un punto de partida para mejorar los procesos, optimizar el flujo de trabajo, e incorporar nuevas metodologías o herramientas (Sommerville, 2011).
4. **Facilitar la transición hacia nuevos procesos:** Al tener bien documentados los procesos actuales, se facilita la transición hacia metodologías más eficientes, como Agile o DevOps, al proporcionar un punto de referencia claro de lo que se está cambiando.

Componentes de la Descripción de Procesos Actuales

La descripción de los procesos actuales generalmente incluye:

- **Flujos de trabajo y diagramas de procesos:** Representaciones visuales de las actividades que forman un proceso y sus interacciones.
- **Documentación de tareas y responsabilidades:** Descripción de las actividades, pasos y los roles responsables de cada tarea.
- **Herramientas y tecnologías utilizadas:** Listado de las herramientas y tecnologías que se emplean para llevar a cabo los procesos, como plataformas de desarrollo, software de gestión de proyectos, etc.

- **Entradas y salidas del proceso:** Definición de los datos, recursos o información que entran en cada proceso y los resultados generados.

En resumen, la **descripción de procesos actuales** en ingeniería de software es un paso fundamental para obtener una visión clara del estado operativo de los procesos, detectar áreas de mejora y establecer una base sólida para la implementación de nuevas metodologías o herramientas que optimicen la eficiencia y calidad del software.

1.3 Diagramas UML

En ingeniería de software, los **diagramas UML** (Unified Modeling Language, por sus siglas en inglés) son herramientas fundamentales para representar visualmente el diseño, el comportamiento dinámico de los componentes y la estructura estática de un sistema de software, comunicando de manera efectiva y facilitando el entendimiento, la toma de decisiones y la colaboración entre los miembros del equipo de desarrollo, arquitectos y otras partes interesadas a entender el sistema desde diferentes puntos de vista.

Según Booch, Rumbaugh y Jacobson (1999) en su libro *The Unified Modeling Language User Guide*, "UML es un lenguaje unificado que integra varios diagramas que representan tanto la estructura estática como el comportamiento dinámico de un sistema de software."

En tanto que C. Fontela (2012) en su libro "UML modelado de software para Profesionales", menciona que en UML no deberían sobrecargarse los diagramas con detalles innecesarios ya que se utilizará para la comunicación, el énfasis se centrará en facilitarla.

Tipos de Diagramas UML

UML incluye dos grandes categorías de diagramas: **diagramas estructurales** y **diagramas comportamentales** que proporcionan una representación visual clara que facilita la comunicación entre los miembros del equipo, mejora el diseño del sistema y asegura que el sistema final cumpla con los requisitos especificados.

1.3.1 Diagramas Estructurales

Los diagramas estructurales muestran la **estructura estática** de un sistema y sus componentes, y está integrado por el conjunto de los siguientes diagramas que se describen a continuación:

- diagramas de clases
- diagramas de objetos
- diagramas de componentes
- diagramas de despliegue

Diagrama de clases: Representa las clases del sistema y sus relaciones. Describen el conjunto de objetos con roles equivalentes en un sistema. Es fundamental para modelar la estructura del software. Según Fowler (2003), en *UML Distilled*, "los diagramas de clases son probablemente los más

importantes de los diagramas estructurales en UML, ya que representan las entidades que componen el sistema y sus relaciones."

Una clase está compuesta por: nombre, atributos, métodos y responsabilidad tal como se muestra en la Figura 1.3.

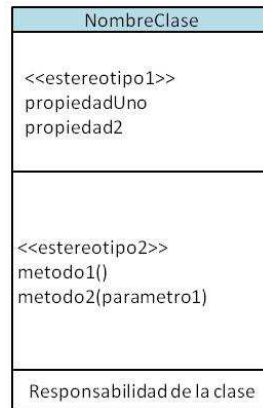
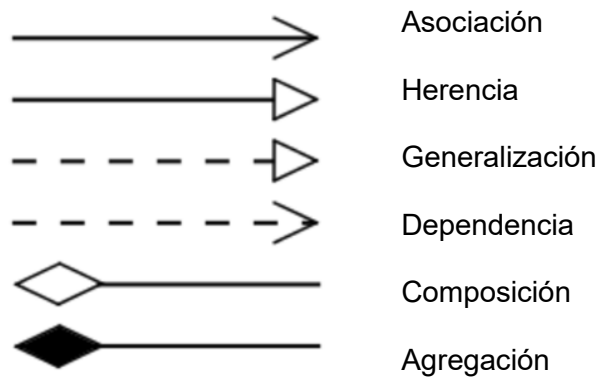


Figura 1. 3 Nomenclatura de una Clase

La nomenclatura de las relaciones entre clases se define de la siguiente manera:



Ejemplo:

La Figura 1.4 muestra un sencillo diagrama de Clases para un sistema de control de Biblioteca. Con las siguientes Clases: **Libro**, **Usuario** y **Prestamo** donde especifican las siguientes **Relaciones**:

- Un **Usuario** puede tener muchos **Prestamos** (relación uno a muchos).
- Un **Prestamo** está asociado a un **Libro** y a un **Usuario**.
- La clase **Libro** puede estar en muchos **Prestamos** (si se presta varias veces a diferentes usuarios).

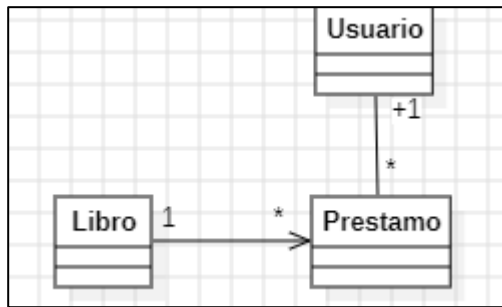


Figura 1. 4 Diagrama de Clases

Diagrama de objetos: Representa las instancias de las clases en un momento específico, resaltando las asociaciones y la multiplicidad (cardinalidad) que guarda cada objeto.

La Figura 1.5 muestra el ejemplo de diagrama de objetos, que indica la necesidad de instanciar cinco objetos de la clase Jugador para ser asociados a una sola instancia de la clase Equipo.

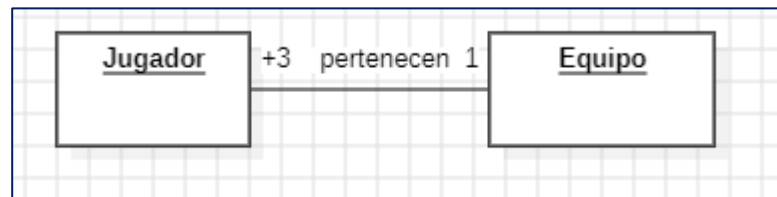


Figura 1. 5 Diagrama de Objetos

Diagrama de componentes: Muestra la organización y dependencias entre los componentes de software. Esto significa, es una representación visual que muestra cómo los diferentes componentes de un sistema se relacionan y se organizan. Es muy útil para entender la estructura modular de un sistema y cómo interactúan sus partes.

Para crear un diagrama de componentes se sugiere:

- 1º. Identificar los componentes principales del sistema, tales como módulos principales del menú de opciones, APIs, etc.
- 2º. Dibujar rectángulos con el nombre de los componentes en la parte superior.
- 3º. Unir los componentes mediante líneas que representan las relaciones o dependencias y añadir detalles adicionales si es necesario.

Ejemplo de Diagrama de Componentes para un Sistema de Gestión de Biblioteca

La Figura 1.6 muestra el diagrama de componentes de este ejemplo.

Componentes principales:

- Interfaz de Usuario
- Módulo de Gestión de Libros
- Módulo de Gestión de Usuarios
- Módulo de Préstamos

Descripción:

- ✓ La **Interfaz de Usuario** interactúa con los usuarios y se comunica con los módulos de gestión.

- ✓ El **Módulo de Gestión de Libros** maneja la información de los libros y se conecta con la base de datos.
- ✓ El **Módulo de Gestión de Usuarios** administra la información de los usuarios y también se conecta con la base de datos.
- ✓ El **Módulo de Préstamos** coordina los préstamos de libros, interactuando con los módulos de gestión de libros y usuarios, y también con la base de datos.

Representación visual (descripción):

- Cada componente se dibuja como un rectángulo con su nombre.
- Las dependencias se muestran con líneas de dependencia o asociación.
- La interfaz de usuario tiene una línea de comunicación con los módulos de gestión.

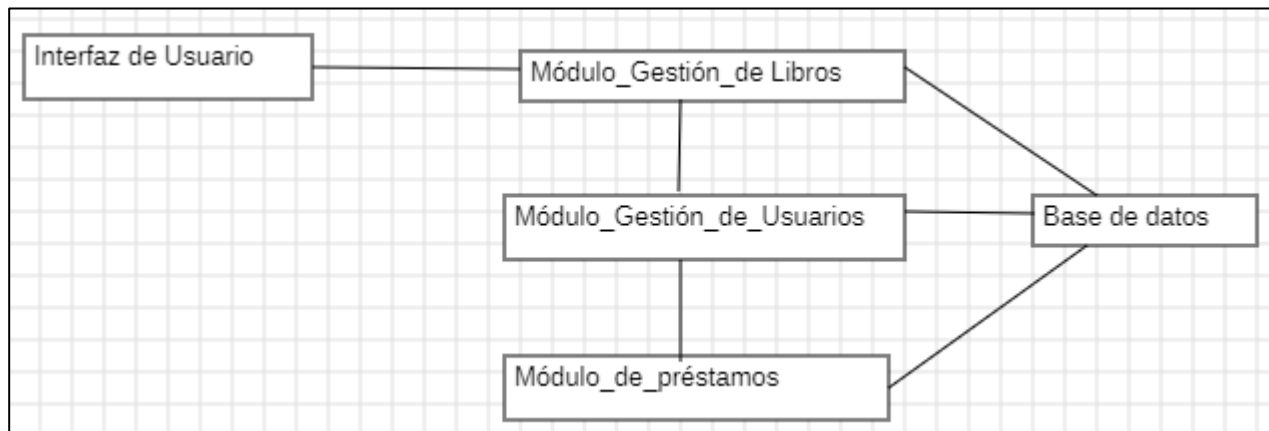


Figura 1. 6 Diagrama de componentes Sistema de Gestión de Biblioteca

Diagrama de despliegue: Representa la configuración física del sistema, como servidores y dispositivos que alojan los componentes.

Los diagramas de despliegue en UML son una herramienta útil para visualizar la arquitectura física de un sistema y comprender cómo se despliega el software en el hardware porque muestran la distribución física de los componentes de software y hardware en una arquitectura de sistema. Se utilizan para representar cómo los componentes clave se asignan a nodos donde se instalan (hardware) y las conexiones que permiten la comunicación entre ellos.

Elementos clave en un diagrama de despliegue:

- **Nodos:** Representan los componentes de hardware (servidores, computadoras, dispositivos, etc.) donde se despliega el software.
- **Componentes:** Representan los artefactos de software (archivos ejecutables, scripts, etc.) que se instalan en los nodos.
- **Asociaciones:** Representan las conexiones de comunicación entre los nodos (por ejemplo, redes, interfaces de programación de aplicaciones).

- **Artefactos:** Son productos de software que se despliegan en los nodos, como archivos ejecutables o bases de datos.
- **Estereotipos:** Se utilizan para identificar tipos específicos de nodos o componentes (por ejemplo, <<server>> para un servidor).

Para elaborar un diagrama de despliegue se recomienda seguir estos pasos:

Paso 1: Identificar el propósito del diagrama de despliegue, identificando los nodos y dispositivos dentro del sistema que se visualizará en el diagrama.

Paso 2: Conocer las relaciones entre nodos y dispositivos para añadir las asociaciones de comunicación al diagrama.

Paso 3: Identificar los componentes y objetos activos para agregarse al diagrama.

Paso 4: Añadir las dependencias entre los componentes y los objetos según sea necesario.

La figura 1.7 muestra el ejemplo del diagrama de despliegue para el Sistema de gestión Hospitalaria que muestra lo siguiente:

- Un servidor de aplicación (un nodo) que contiene el código del sistema.
- Un servidor de base de datos (otro nodo) donde se almacena la información de usuarios y pacientes.
- Computadoras usuarios (nodos) que interactúan con el servidor de aplicación.
- Conexiones de red que permiten la comunicación entre estos nodos.
- Impresora

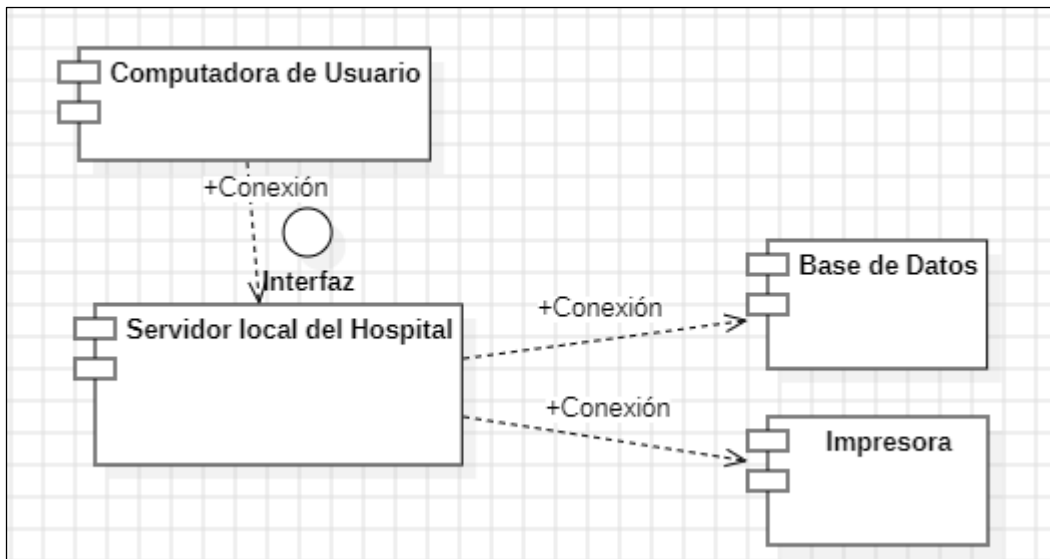


Figura 1. 7 Diagrama de Despliegue del Sistema de Gestión Hospitalaria

1.3.2 Diagramas Comportamentales

Los diagramas comportamentales modelan el **comportamiento dinámico** del sistema. Principalmente son: diagramas de caso de uso, diagramas de secuencia, diagramas de actividad y diagramas de estado. Se describen a continuación.

Diagrama de casos de uso: Representa los actores y sus interacciones con el sistema. Es crucial para la recopilación de requisitos. Se utiliza para describir las funcionalidades del sistema desde la perspectiva del usuario. La figura 1.8 muestra la representación de un caso de uso.

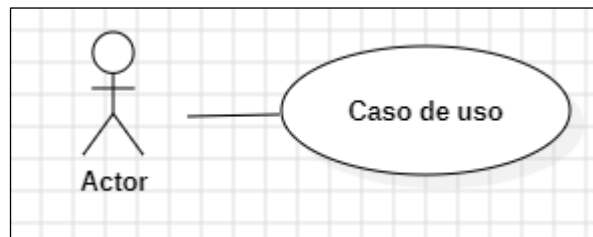


Figura 1. 8 Representación de Diagrama de Caso de Uso

- a) Según Jacobson et al. (1992), en su obra *Object-Oriented Software Engineering: A Use Case Driven Approach*, “los diagramas de casos de uso permiten a los analistas y diseñadores capturar y visualizar los requisitos funcionales desde el punto de vista del usuario.”

La figura 1.9 muestra el ejemplo del diagrama de Caso de Uso Préstamo de Libro, donde se refleja el proceso del requisito del cliente.

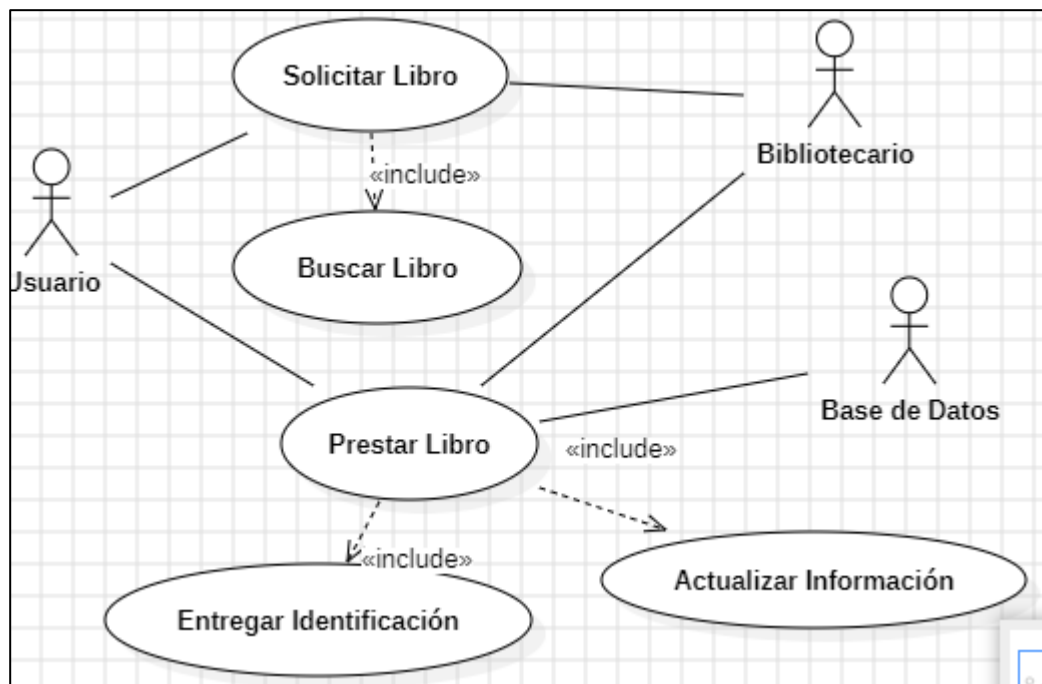


Figura 1. 9 Diagrama de Caso de Uso

Los **actores** son los usuarios que interactúan con el sistema. En este caso, Los actores son: el **Usuario**, **Bibliotecario** y la **Base de Datos**.

Los **casos de uso** representan las acciones que un actor puede realizar, como **Solicitar Libro** y **Prestar Libro**

- b) **Diagrama de secuencia:** Muestra cómo los objetos interactúan entre sí a lo largo del tiempo, siendo muy útil para modelar el comportamiento del sistema.

La figura 1.10 muestra la representación del diagrama de secuencia.

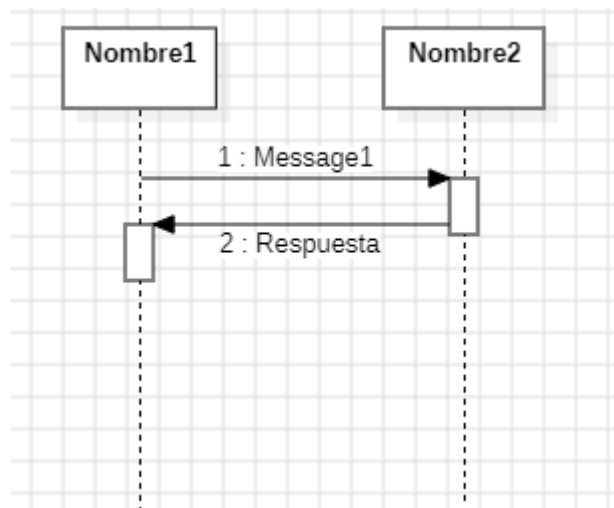


Figura 1. 10 Diagrama de Secuencia

La figura 1.11 muestra el diagrama de secuencia de préstamo de libro

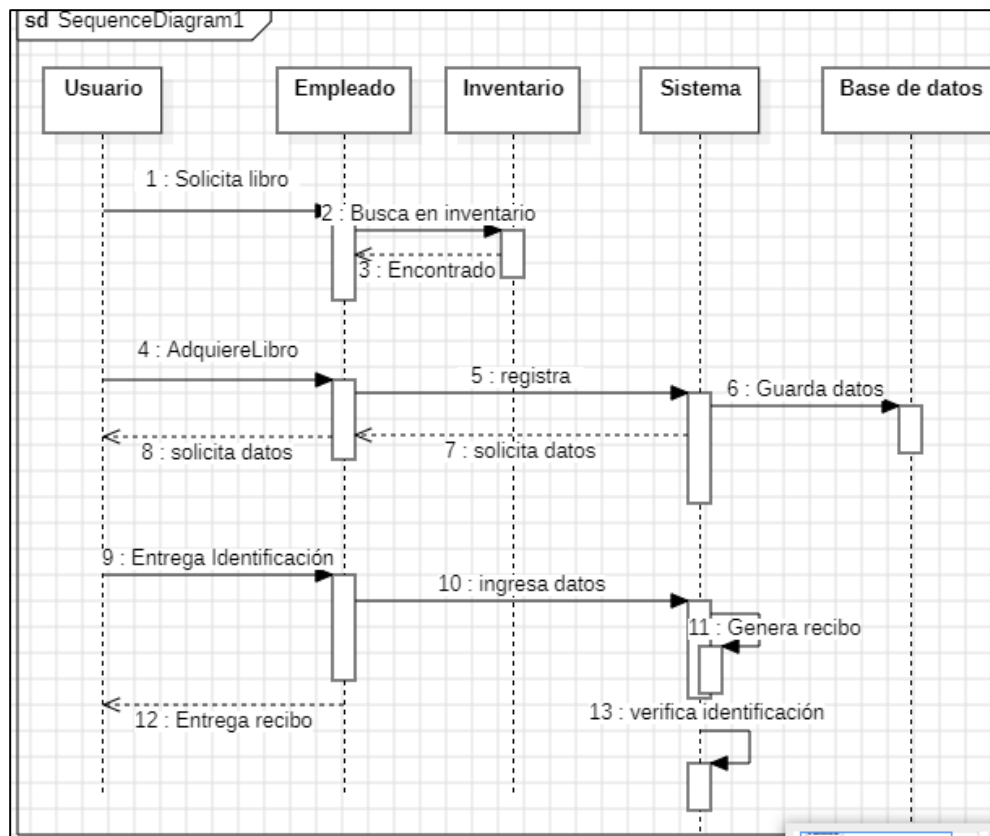


Figura 1. 11 Diagrama de Secuencia

- c) **Diagrama de actividad:** Visualizan el flujo de control y los objetos que interactúan dentro de un proceso o sistema. Se utilizan para modelar los aspectos dinámicos de un sistema, mostrando las acciones, decisiones y eventos que ocurren en él. Representa el flujo de trabajo o los procesos de un sistema, útil para describir la lógica de negocio.

La figura 1.12 muestra la representación del diagrama de actividades.

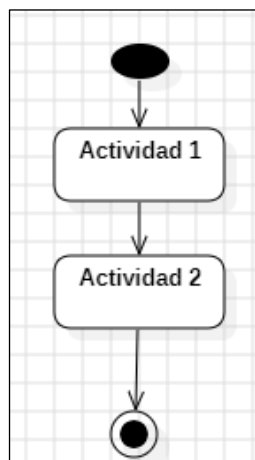


Figura 1. 12 Representación del Diagrama de Actividades

La figura 1.13 muestra el ejemplo de las actividades que se llevan a cabo para generar un documento.

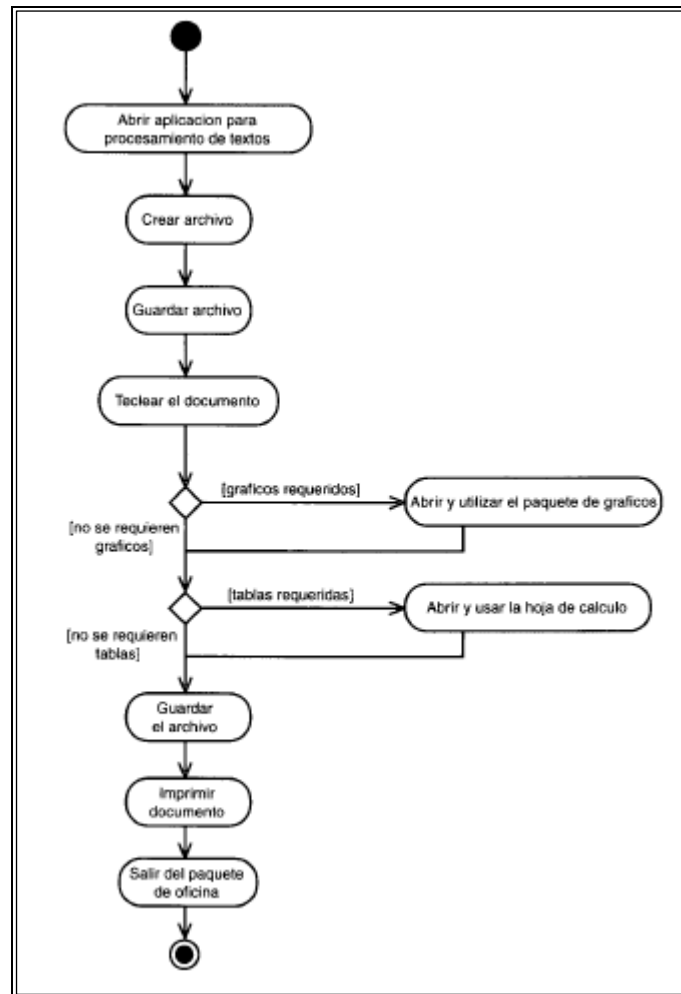


Figura 1. 13 Diagrama de Actividad

- d) **Diagrama de estado:** Muestra los estados por los que puede pasar un objeto y las transiciones entre esos estados, mostrando los puntos inicial y final de una secuencia de cambios de estado. Muestra las condiciones de un solo objeto. Estos diagramas se utilizan para optimizar cualquier proceso de desarrollo donde sea útil visualizar los estados del objeto y las condiciones para que se produzca la transición de un estado a otro.
- Suelen emplearse, por ejemplo, en el **diseño de sistemas embebidos** (en inglés, *embedded systems*), donde las señales automatizadas y los procesos en segundo plano deben estar perfectamente coordinados. En este caso, el diagrama de estado ayuda a los desarrolladores a **visualizar** todas las funciones de control y regulación más importantes en un solo esquema.

La figura 1.15 muestra el diagrama de estado para activar o desactivar la alarma de una radio-despertador, **el estado cambia** de “alarma activada” a “alarma desactivada”.



Figura 1. 14 Diagrama de estado

Importancia de UML en la Ingeniería de Software

El uso de UML facilita la **comunicación entre los miembros del equipo de desarrollo** y otras partes interesadas (como los clientes), ya que proporciona una representación clara y comprensible del sistema. Además, ayuda a **documentar** y **organizar** el proceso de desarrollo, lo que facilita el mantenimiento y la evolución del software.

Booch et al. (1999) explican que “UML proporciona un lenguaje común que permite a los desarrolladores y arquitectos compartir una visión unificada del sistema, reduciendo el riesgo de malentendidos y errores durante el desarrollo.”

Para facilitar el diseño y desarrollo de software con la creación de diagramas UML, se encuentran herramientas de software gratuitas, llamadas Herramientas CASE que sirven de apoyo para realizar análisis de sistemas y facilitar el desarrollo de software. La elección de la herramienta dependerá de las necesidades del proyecto, la experiencia del equipo y los recursos disponibles.

1.4 Estudio de Factibilidad

En ingeniería de software, el **Estudio de Factibilidad** es un proceso que se lleva a cabo dentro de la etapa del análisis preliminar que se realiza antes de iniciar un proyecto de desarrollo de software. Su objetivo principal es evaluar la viabilidad del proyecto desde diferentes perspectivas, asegurando que sea posible llevarlo a cabo, considerando los recursos, el tiempo, los costos, y lo más importante, ayuda a reducir los riesgos del proyecto, garantizando que no se invierta tiempo ni dinero en un proyecto que no pueda completarse de manera efectiva o que no ofrezca los beneficios esperados. Además, es una base importante para la **planificación del proyecto**, ya que permite a los stakeholders tomar decisiones sobre si proceder o no con el proyecto con los recursos que se cuenta y el enfoque de desarrollo que se aplicará. Pressman, R. S. (2014). *Software Engineering: A Practitioner's Approach* (8th ed.). McGraw-Hill menciona que “el estudio de factibilidad es la clave para identificar posibles obstáculos antes de que el proyecto entre en fases de desarrollo intensivo, evitando así inversiones innecesarias en proyectos inviables.”

Definición y Objetivos del Estudio de Factibilidad

El proceso del **Estudio de Factibilidad** en ingeniería de software es una fase crítica que proporciona una evaluación profunda sobre si un proyecto de software debe proceder, basándose en factores técnicos, económicos, operativos y legales. Este análisis inicial permite a los interesados tomar decisiones informadas, reducir riesgos y establecer una base sólida para la planificación del proyecto.

Se realiza en varias etapas que incluyen: **recopilación de requisitos iniciales, análisis de factibilidad técnica, económica, operativa y legal**, con el fin de determinar si el proyecto es viable y qué tan viable es en cada una de esas áreas:

- **Recopilación de Requisitos Iniciales:** El primer paso es obtener una comprensión básica de los objetivos y requisitos del sistema. Esto incluye reuniones con los stakeholders para recopilar la información relevante.
- **Análisis de Factibilidad Técnica:** Evalúa si los recursos tecnológicos disponibles son suficientes para cumplir con los requisitos del proyecto (hardware y software). Esto incluye el análisis de la infraestructura, plataformas, herramientas y conocimientos del equipo con habilidades necesarias para desarrollar el sistema. Pressman R. S. (2014) destaca la importancia de evaluar la tecnología y los recursos técnicos en esta fase.
- **Análisis de Factibilidad Económica:** Examina si el costo estimado de desarrollo y operación del sistema es justificable en relación con los beneficios que se esperan obtener. Implica análisis de costos, tiempo requerido, retorno de inversión (ROI) y presupuestos. También se realiza una

comparación de costos entre diferentes alternativas de solución. Boehm (1981) explica cómo realizar análisis económicos para justificar proyectos de software.

- **Análisis de Factibilidad Operativa:** Determina si el sistema propuesto podrá ser operado y mantenido de manera efectiva dentro del entorno organizacional existente. Se considera si los usuarios pueden adaptar fácilmente el sistema y si cumple con las necesidades operativas. Esto incluye la capacitación de usuarios, la integración con los sistemas existentes y la infraestructura de soporte. Kendall, K. E., & Kendall, J. E. (2014). *Systems Analysis and Design*. Pearson. Los autores resaltan la importancia de evaluar la aceptación y compatibilidad operativa.
- **Análisis de Factibilidad Legal:** Evalúa si el proyecto cumple con las leyes locales, regulaciones de seguridad y cumplimiento de normativas relevantes, como la protección de datos y las licencias de software. Sommerville (2011) señala que la evaluación legal es esencial para evitar problemas futuros.

Informe de Factibilidad: El resultado del estudio es un informe detallado que describe la viabilidad del proyecto en todas las áreas analizadas, y proporciona recomendaciones sobre si proceder con el desarrollo del sistema o no.

Se incluye un **Análisis de factibilidad de tiempo** para evaluar si el proyecto puede completarse en el plazo establecido. Pressman, R. S. (2014) menciona que la planificación temporal es clave para la viabilidad del proyecto.

1.5 Análisis Costo-Beneficio

En **ingeniería de software**, el **Análisis Costo-Beneficio** es una técnica utilizada para evaluar si los beneficios de un proyecto superan los costos asociados. Este análisis se realiza durante las fases iniciales del proyecto y ayuda a tomar decisiones informadas sobre si un proyecto de software es financieramente viable. Implica una evaluación detallada de los **costos** (desarrollo, operación, mantenimiento, etc.) frente a los **beneficios** (mejora en la eficiencia, incremento en los ingresos, reducción de costos operativos, etc.), y su objetivo es garantizar que la inversión en el proyecto genere un retorno positivo.

Definición y Propósito del Análisis Costo-Beneficio

El **Análisis Costo-Beneficio** tiene como propósito proporcionar una justificación económica para el proyecto de software, evaluando si los recursos invertidos generarán un valor suficiente para la organización. Este análisis se centra en cuantificar los **costos** directos e indirectos del proyecto y compararlos con los **beneficios tangibles e intangibles** que se espera obtener.

Según Sommerville (2011), “el análisis costo-beneficio es una herramienta fundamental para la toma de decisiones en ingeniería de software, ya que proporciona una base cuantitativa para evaluar si un proyecto de software es financieramente viable.”

Tipos de Costos y Beneficios

1. Costos

- **Costos iniciales:** Incluyen los gastos asociados al desarrollo del software, como salarios de los desarrolladores, adquisición de herramientas, equipos y otros recursos.
- **Costos operativos:** Son los costos relacionados con la ejecución y mantenimiento del sistema una vez implementado, como la capacitación, el soporte técnico y las actualizaciones.
- **Costos indirectos:** Se refieren a los costos no directamente relacionados con el software, como el tiempo dedicado por personal no técnico en el proceso de adopción.

2. Beneficios

- **Beneficios tangibles:** Son aquellos que se pueden medir en términos financieros, como la reducción de costos operativos, mejoras en la productividad y aumento de ingresos.
- **Beneficios intangibles:** Son más difíciles de medir, pero igualmente importantes, como la mejora en la satisfacción del cliente, la mayor capacidad de adaptación al mercado o el fortalecimiento de la reputación de la empresa.

Proceso de Análisis Costo-Beneficio

El proceso de **Análisis Costo-Beneficio** en ingeniería de software generalmente sigue los siguientes pasos:

1. **Identificación de Costos y Beneficios:** El primer paso es identificar todos los posibles costos y beneficios asociados con el proyecto. Esto incluye la recopilación de datos sobre los costos de desarrollo, implementación y mantenimiento, así como los beneficios proyectados del sistema.

2. **Cuantificación de Costos y Beneficios:** En esta etapa, se intenta cuantificar tanto los costos como los beneficios en términos monetarios, cuando sea posible. Algunos beneficios, como la mejora en la satisfacción del cliente o la mejora de la moral del equipo, pueden ser más difíciles de medir, pero deben ser considerados.
3. **Evaluación de la Rentabilidad:** Una vez que los costos y beneficios han sido cuantificados, se pueden calcular indicadores de rentabilidad como el **Valor Actual Neto (VAN)**, la **Tasa Interna de Retorno (TIR)** o el **Periodo de Recuperación**. Estos indicadores ayudan a determinar si el proyecto es financieramente viable.
4. **Toma de Decisiones:** Si los beneficios superan los costos, el proyecto puede considerarse viable desde una perspectiva económica, y se puede proceder con su desarrollo. Si los costos superan los beneficios, el proyecto puede ser desestimado o ajustado.

Importancia del Análisis Costo-Beneficio

El análisis costo-beneficio es crucial porque permite a los stakeholders comprender **la relación entre el costo de un proyecto y los beneficios esperados**, lo que ayuda a tomar decisiones informadas sobre la viabilidad económica del proyecto. Además, contribuye a **priorizar proyectos** dentro de una cartera de desarrollo de software, asignando recursos a aquellos proyectos con el mayor retorno de inversión.

Pressman (2014) argumenta que “el análisis costo-beneficio permite a las organizaciones tomar decisiones estratégicas sobre la implementación de proyectos de software, ayudando a maximizar los beneficios y minimizar los costos.”

Ejemplo:

El objetivo de este estudio es evaluar la viabilidad técnica y operativa de implementar un sistema de gestión para una biblioteca.

La tabla 1.2 muestra el análisis costo-beneficio para el desarrollo de un sistema para una biblioteca.

La primera columna, nombra el concepto y en la segunda columna describe el concepto.

La tercera columna es capturada el costo estimado en dólares.

La cuarta columna incluye el costo estimado en Moneda Nacional a un tipo de cambio de 20.00 MN

La quinta columna muestra la fuente bibliográfica que justifica la elaboración de éste análisis.

Los beneficios anuales es la suma de las columnas costo estimado en dólares y en moneda nacional.

El tiempo de recuperación de la inversión es el resultado de dividir los costos de la inversión inicial entre el costo de los beneficios anuales.

Tabla 1. 2 Análisis Costo-Beneficio para desarrollo de un Sistema de Biblioteca

1. Costos				
Concepto	Descripción	Costo estimado \$USD	Costo estimado \$MN (20 MN/USD)	Fuente / Justificación
Desarrollo del Sistema	Contratación de programador y analista (2 personas, 6 meses)	30,000	600,000	(Pressman, 2014) indica que el costo promedio de desarrollo de software puede variar, pero para un sistema básico, \$30,000 USD es razonable.
Hardware e Infraestructura	Servidores, computadoras, redes	10,000	200,000	(Kendall & Kendall, 2014) sugieren que la inversión en infraestructura para una biblioteca mediana puede rondar este monto.
Capacitación del personal	Entrenamiento para 10 empleados, 2 días	2,000	40,000	(Boehm, 1981) recomienda presupuestar capacitación para asegurar uso eficiente.
Mantenimiento y actualizaciones	10% del costo de desarrollo anual	3,000	60,000	(Sommerville, 2016) señala que el mantenimiento suele representar aproximadamente del 10 al 20% del costo inicial anual.
Implementación y migración	Transferencia de datos y puesta en marcha	5,000	100,000	Incluye tiempo y recursos para migrar datos existentes y pruebas.
Total costos iniciales:		50,000	1,000,000	
2. Beneficios Anuales				
Reducción en tiempo de gestión	Automatización reduce tareas manuales en 50%, ahorrando 1,000 horas anuales	20,000	400,000	Si el personal gana en promedio \$20/hora, y se ahorra 1,000 horas, el valor es \$20,000. (Kendall & Kendall, 2014)
Mejor servicio al usuario	Incremento en satisfacción y uso, estimado en 15% más préstamos	3,000	60,000	Basado en aumento de préstamos y multas, estimado en un valor monetario.
Reducción de costos operativos	Menor uso de papel y archivos físicos, ahorro de \$2,000 USD anuales	2,000	40,000	(Boehm, 1981) indica que la digitalización puede reducir costos en papel y almacenamiento.
Control y seguimiento mejorados	Datos precisos para decisiones, valorado en \$1,500 USD	1,500	30,000	Mejoras en gestión que permiten optimizar recursos.
Acceso remoto y en línea	Facilita gestión desde diferentes ubicaciones, valor estimado en \$1,500 USD	1,500	30,000	Valor intangible traducido en eficiencia y flexibilidad.
Total beneficios anuales:		28,000	560,000	
3. Análisis Financiero \$USD				
Inversión inicial	50,000			
Beneficios anuales	28,000			
Tiempo de Recuperación de la Inversión	1.8 Años			

- El tiempo de recuperación es la Inversión Inicial/ Beneficios anuales

Conclusión

El **Análisis Costo-Beneficio** es una herramienta fundamental en la **gestión de proyectos de software**, ya que ayuda a los stakeholders a tomar decisiones informadas sobre la viabilidad económica del proyecto. Este análisis permite comparar los costos y beneficios de manera cuantitativa y cualitativa, lo que ayuda a priorizar proyectos y asignar recursos de manera eficiente. Además, contribuye a la reducción de riesgos, asegurando que los proyectos que se desarrollen sean financieramente sostenibles y rentables a largo plazo.

Sommerville destaca la importancia del análisis costo-beneficio para evaluar la viabilidad económica de un proyecto de software, ayudando a determinar si la inversión inicial justifica los beneficios esperados a largo plazo.

ACTIVIDADES TEMA 1

El complemento educacional basado en **ESTUDIO DE CASOS** en ingeniería de software consiste en presentar a los estudiantes situaciones reales o simuladas relacionadas con proyectos de desarrollo de software, para que puedan analizar, tomar decisiones y aplicar sus conocimientos en un contexto práctico.

1. CASO DE ESTUDIO:

Elaborar el estudio de Factibilidad Técnica y Operativa para la Implementación de un Sistema de Pedidos en Línea

Introducción

El objetivo de este estudio es evaluar la viabilidad técnica y operativa para implementar un sistema de pedidos en línea para una empresa de alimentos, con el fin de justificar la inversión mediante un análisis costo-beneficio.

- Factibilidad Técnica

Evaluación de la infraestructura tecnológica y compatibilidad del sistema.

- **Infraestructura existente:** La empresa cuenta con una red Wi-Fi estable y servidores en la nube que soportan aplicaciones web.
- **Tecnología requerida:** La solución propuesta es una plataforma web compatible con dispositivos móviles y de escritorio, utilizando tecnologías modernas como HTML5, CSS3, y frameworks como React.js.
- **Seguridad:** Se implementarán protocolos SSL y medidas de protección de datos conforme a las normativas vigentes (ISO/IEC 27001, 2013).
- **Recursos humanos:** El personal de TI tiene experiencia en desarrollo y mantenimiento de sistemas web, lo que facilita la implementación y soporte técnico.

- Factibilidad Operativa

Evaluación de la aceptación del usuario, procesos internos y capacitación.

- **Capacitación:** Se planifica un programa de entrenamiento para empleados en el uso del sistema, con sesiones prácticas y manuales.
- **Procesos internos:** La digitalización de pedidos reducirá errores manuales y acelerará el proceso de atención al cliente.
- **Aceptación del usuario:** Se realizarán pruebas piloto con retroalimentación continua para garantizar la aceptación y facilidad de uso.
- **Impacto en operaciones:** Estimar la reducción en tiempos de procesamiento de pedidos y una disminución en errores de entrada de datos.

- Análisis Costo-Beneficio

Estimación de costos y beneficios asociados.

- **Determinar Costos con base en estos rubros:**
 - Desarrollo e implementación
 - Capacitación
 - Mantenimiento anual
- **Beneficios:**
 - Reducción en errores y retrabajo

2. EJERCICIOS

Elaborar el diagrama de despliegue para una Aplicación web:

Un diagrama de despliegue de una aplicación web para mostrar:

- Un servidor web (un nodo) que contiene el código de la aplicación.
- Una base de datos (otro nodo) donde se almacena la información de la aplicación.
- Nodos de cliente (por ejemplo, computadoras o dispositivos móviles) que interactúan con el servidor web a través de una conexión de red.
- Asociaciones entre los nodos, que representan las conexiones de red (por ejemplo, HTTP, TCP/IP) que permiten la comunicación entre ellos.

2. Sistema de procesamiento de tarjetas de crédito:

Un diagrama de despliegue para este sistema podría mostrar:

- Una máquina expendedora o POS (un nodo) donde el cliente introduce la tarjeta.
- Un servidor de procesamiento de transacciones (otro nodo) que realiza la verificación de la transacción.
- Un servidor de base de datos (otro nodo) que almacena la información de las tarjetas y transacciones.
- Conexiones de red (TCP/IP, RMI, JDBC) que permiten la comunicación entre estos nodos.

3. Desarrollo de Proyecto

1. Para el siguiente caso, elaborar el diagrama de Modelo de negocio y diagramas de Caso de Uso para identificar los requisitos funcionales y requisitos no funcionales y documentar en el formato de la IEEE830.

“Cinto Seguro” es una pequeña empresa del ramo automotriz que fabrica cinturones de seguridad, pero aún no tiene sistematizado el sistema de pedidos.

Los pedidos de los clientes llegan a la compañía y son recibidos por Any, quien trabaja en la compañía desde hace varios años. Una vez recibidos, al día siguiente, Any se los envía a Nancy, quien verifica que los datos del pedido sean correctos y que el cliente exista en el archivo maestro de clientes de la compañía. Si algún dato es incorrecto (Nombre o dirección), el pedido es rechazado. Si el cliente no existe en el archivo maestro, se da de alta creando un nuevo registro para ese cliente.

Nancy también verifica que los artículos solicitados en el pedido existan en el catálogo de productos terminados. Si alguno no existe, se elimina del pedido y se atiende el resto.

Como todas las ventas son a crédito, Nancy llena una solicitud de aprobación de crédito la cual manda a Nicolás, quien se encuentra en el departamento de créditos. Nicolás accede al maestro de clientes y compara el saldo. Dependiendo de la comparación, llena una respuesta indicando si el crédito fue aprobado o no y se lo envía a Nancy.

Si el crédito fue rechazado, Nancy le pide a Any que se lo notifique al cliente por escrito. En caso contrario, se lo envía a Juanita para que inicie la atención del pedido.

Juanita revisa un listado con existencias de producto terminado y las compara contra lo solicitado en el pedido. Anota lo que se puede surtir totalmente y lo que se puede surtir parcialmente.

Con estas cantidades anotadas, envía el pedido a Goyo, quien se encuentra en el departamento de Facturación elaborando la factura que se enviará al cliente. Juanita también prepara una solicitud de salida de almacén y se la envía a Pedro quien controla las salidas de almacén. En caso de haber faltantes de algún producto, Juanita prepara una solicitud de producción y lo envía a Armando, quien se encuentra en producción.

Goyo elabora la factura en base al pedido que le envía Juanita en original y dos copias. La original y una copia van a un archivo de facturas pendientes de pago.

Si al pedido le queda una cantidad pendiente de surtir, se envía a un archivo de pedidos pendientes, el cual es accedido por Juanita para reiniciar su atención. En caso contrario, Goyo archiva el pedido.

Con la otra copia de la factura, Goyo prepara una orden de salida de la planta y la envía junto con la copia de la factura al área de embarques (Rito) para que el vigilante del acceso a la compañía pueda autorizar la salida del camión que lleva los productos del cliente. El vigilante se queda con la orden de salida y la copia de la factura va para el cliente junto con el cargamento.

Con la solicitud de salida de almacén que le envía Juanita, Pedro prepara una orden de salida del almacén de producto terminado y la envía a Rito, el cual extrae los artículos del almacén y los empaca.

Una vez empacados, los artículos son embarcados en un camión por varios cargadores. Cuando el camión tiene completa la carga, se le entrega al chofer la orden de salida de la planta y la copia de la factura para que pueda iniciar el flete.

Con la solicitud de salida del almacén, Pedro prepara un movimiento de salida del almacén y lo envía al centro de cómputo para procesarse en el sistema de inventarios.

Armando recibe las solicitudes de producción que le manda Juanita y con ella prepara las órdenes de producción, las cuales se envían a Luis, Supervisor de Producción, con lo cual se producen los artículos solicitados.

Luis envía los artículos al almacén y prepara un movimiento de entrada al almacén, el cual envía al centro de cómputo para procesarse en el sistema de inventarios. Con los movimientos de entrada y salida y accediendo al maestro de productos terminados, el sistema computacional actualizará las existencias de los productos generando un reporte con entradas, salidas y existencias.

Una copia de este reporte se la envía a Juanita y el original va para las gerencias de Ventas y Producción.

Cada semana, Juanita elabora el reporte de pedidos pendientes y se lo envía al Gerente de Ventas.

Los clientes envían los pagos y los recibe Any quien elabora un movimiento contable y lo envía a Contabilidad junto con el comprobante de transferencia bancaria o efectivo. Una copia del movimiento lo manda a Pepe, el cual acredita el pago de la factura correspondiente y actualiza el pago del cliente. Si la factura es totalmente pagada, se la envía al cliente de lo contrario se regresa al archivo de facturas pendientes.

Cada semana Pepe prepara un reporte de análisis de pedidos entregados y pagados, el cual envía al Gerente de Ventas.

El gerente de Ventas envía la información al departamento de seguimiento de pedidos para que informe el estatus del pedido del cliente.

Al final del mes, se realiza el reporte de Producción y Ventas que se envía al director de la empresa para determinar la rentabilidad del negocio y proceder a la acertada toma de decisiones con el equipo directivo.

TEMA 2. DISEÑO

Introducción

En ingeniería de software, el diseño es una etapa fundamental del proceso de desarrollo de software que se centra en cómo se estructurará y funcionará el sistema.

El diseño crea una representación o modelo del software, a diferencia del modelo de los requerimientos (que se centra en describir los datos que se necesitan, la función y el comportamiento), el modelo de diseño proporciona detalles sobre arquitectura del software, estructuras de datos, interfaces y componentes que se necesitan para implementar el sistema (Pressman, 2007).

El diseño es la etapa clave en la que el software comienza a tomar forma tangible, es el puente entre el análisis de los requisitos y el desarrollo del software. La figura 2.1 muestra esta traducción.

El diseño incluye la creación de arquitectura, que determina la estructura general del sistema, desde la elección del modelo de análisis hasta la forma en que se manejarán los datos, lo que implica cómo se organizarán los diferentes componentes y cómo interactuarán entre sí, modelos, diagramas y especificaciones que sirven como guía para los desarrolladores al construir el sistema.

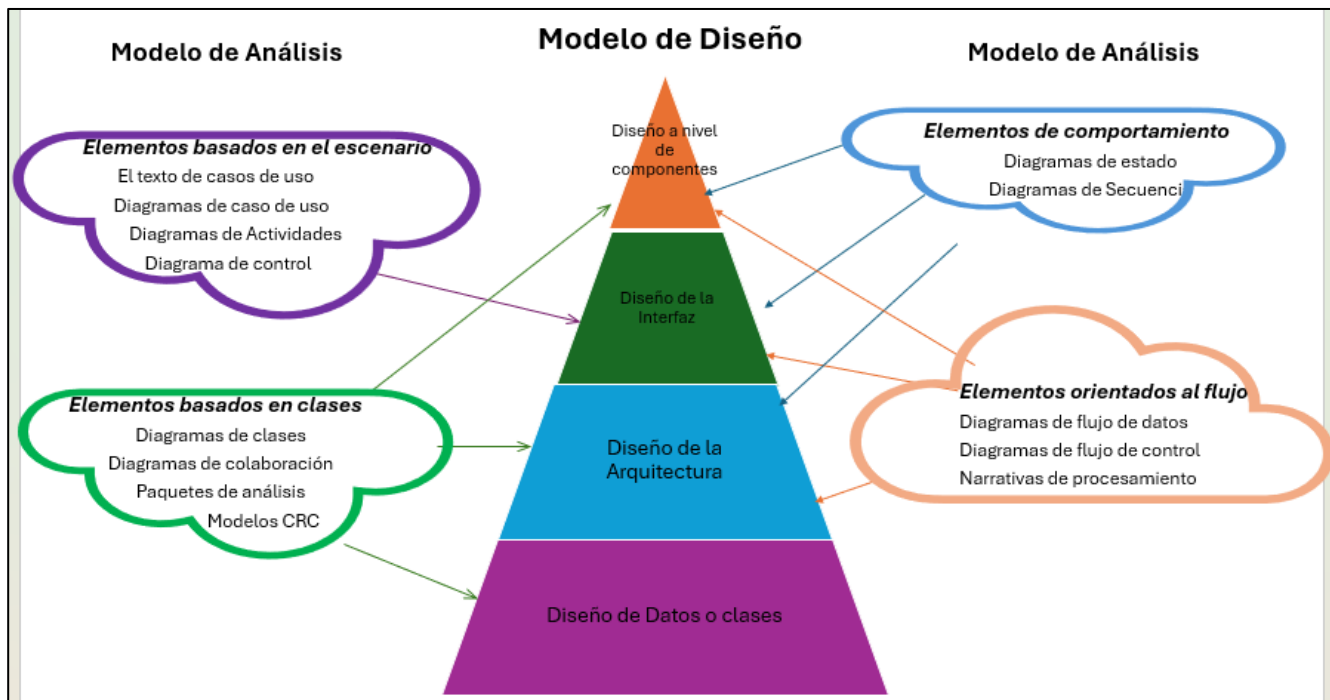


Figura 2. 1 Traducción del Modelo de Análisis al Modelo de Diseño (Pressman, 2007)

También se estructura la base de datos bien diseñada que implica definir cómo se almacenarán y gestionarán los datos, asegurando el alto rendimiento y la escalabilidad del software, incluye la creación de relaciones entre las distintas tablas y el uso de principios de normalización para evitar redundancias.

En esta fase se define la interfaz de usuario (UI) para que el diseño visual y la experiencia de usuario (UX) garanticen que el software funcione bien, que sea intuitivo y agradable de usar.

El diseño de la interfaz de usuario incluye la disposición de los elementos, los colores y los botones. También se enfoca en la navegación. El objetivo es que los usuarios interactúen con el sistema de forma eficiente y sin problemas.

Todo el proceso de diseño es fundamental para que el software cumpla con los requisitos funcionales, que sea fácil de usar, intuitivo y eficiente en su rendimiento.

Una buena planificación en esta fase reduce los riesgos. También mejora el trabajo en las etapas siguientes. Esto asegura un producto final que cumpla con las expectativas del cliente y de los usuarios.

Algunos aspectos clave del diseño en ingeniería de software son:

1. **Diseño arquitectónico:** Define la estructura general del sistema, cómo los componentes interactuarán entre sí y qué tecnologías se utilizarán.
2. **Diseño detallado:** Se centra en los detalles de cada componente, como algoritmos, estructuras de datos y cómo se implementarán.
3. **Diseño de interfaz:** Establece cómo los usuarios interactuarán con el sistema, asegurando que sea fácil de usar y accesible.

El diseño no solo busca funcionalidad, sino también optimización, mantenibilidad, escalabilidad y cumplimiento de los requisitos del cliente. Es como el plano de construcción de un edificio, esencial para crear un sistema sólido y eficiente

En la ingeniería de software, los conceptos de diseño son fundamentales para crear sistemas eficientes, mantenibles y escalables. Algunos de los principios clave incluyen:

Modularidad: Dividir el sistema en componentes independientes, desarrollarse y probarse por separado.

Abstracción: Ocultar los detalles innecesarios y exponer solo los necesarios simplificando la interacción.

Encapsulamiento: Proteger los datos internos de un módulo para evitar modificaciones accidentales.

Cohesión y acoplamiento: Maximizar la cohesión (relación fundamental de los elementos dentro de un módulo) y minimizar el acoplamiento entre módulos.

Principio de responsabilidad única: cada módulo o clase de responsabilidad dentro del sistema.

Diseño orientado a objetos: Aplicar conceptos como herencia y composición para mejorar la reutilización y mantenibilidad.

Arquitectura basada en capas: Organizar el sistema en capas de la lógica del negocio

Diseño basado en patrones: Los patrones de diseño de software son soluciones generales y probadas que los desarrolladores pueden aplicar a problemas comunes en el diseño de software.

Estos patrones son una forma de capturar el conocimiento y la experiencia colectiva de la comunidad de desarrollo de software y se utilizan para mejorar la calidad del software y la eficiencia del proceso de desarrollo.

2.1 Diseño de procesos propuestos

El diseño de procesos propuestos es una metodología o práctica común en proyectos de mejora de procesos o desarrollo de sistemas. Implica la creación y planificación de procesos nuevos o mejorados para cumplir con los objetivos específicos de una organización. Este diseño es clave para optimizar la eficiencia, reducir costos, mejorar la calidad o adaptarse a cambios en el entorno organizacional. Este enfoque incluye la identificación de necesidades, la creación de modelos de procesos y la implementación de mejoras.

Aquí están los pasos principales para diseñar procesos propuestos:

1. **Identificación de requisitos:** comprender las necesidades y expectativas de los interesados, así como las limitaciones existentes.
2. **Análisis de procesos actuales:** examinar los procesos existentes para identificar ineficiencias, cuellos de botella y áreas de mejora.
3. **Definición de objetivos:** establecer metas claras para el nuevo proceso, como tiempos de respuesta más cortos o una mayor satisfacción del cliente.
4. **Diseño del proceso:** utilizar herramientas como diagramas de flujo, mapas de procesos y modelado **BPMN** (Business Process Model And Notation) para visualizar un nuevo proceso.
5. **Evaluación de la viabilidad:** asegurarse que el proceso propuesto sea factible desde el punto de vista técnico, financiero y organizacional.
6. **Documentación:** crear documentación detallada, como manuales y guías, para facilitar la implementación y el entendimiento.
7. **Simulación y prueba:** probar el diseño del proceso en un entorno controlado para identificar posibles problemas antes de su implementación.
8. **Implementación:** introducir el nuevo proceso en la organización acompañado de capacitación para los involucrados.
9. **Seguimiento y optimización:** monitorear el desempeño del nuevo proceso y realizar ajustes según sea necesario.

El diseño de procesos propuestos requiere colaboración entre todas las partes interesadas para asegurar que el nuevo flujo sea efectivo y aceptado.

2.1.1 Herramientas CASE para diseño

Las **herramientas CASE (Computer-Aided Software Engineering)** son aplicaciones que ayudan a los ingenieros de software en el proceso de desarrollo, desde la planificación hasta la implementación y mantenimiento. Estas herramientas pueden ser utilizadas para la creación de modelos, diagramas, análisis de requisitos, pruebas, y más. Existen diversas herramientas CASE gratuitas que permiten realizar estas tareas sin necesidad de grandes inversiones.

A continuación, menciono algunas de las más destacadas, entre las que figuran las que son gratuitas, las que manejan proyectos de baja a mediana complejidad:

1. StarUML

StarUML es una herramienta de modelado UML ampliamente utilizada que permite la creación de diagramas UML como diagramas de clases, secuencia, actividad, etc. Aunque tiene una versión de pago, también dispone de una versión gratuita con funcionalidades esenciales para modelar sistemas de software.

- **Características:**
 - Soporta UML 2.x.
 - Herramientas de edición de diagramas fáciles de usar.
 - Soporte para extensiones a través de complementos.
- **Enlace:** [StarUML](#)

2. ArgoUML

ArgoUML es una herramienta de modelado UML de código abierto y gratuita. Aunque es menos moderna que otras herramientas, sigue siendo muy útil para la creación de diagramas UML y otros modelos de software.

- **Características:**
 - Compatible con diagramas UML 1.4.
 - Interfaz de usuario sencilla.
 - Generación de código a partir de diagramas UML.
- **Enlace:** [ArgoUML](#)

3. Dia

Dia es una herramienta de diagramación que permite crear diagramas de flujo, diagramas de red, diagramas UML y otros tipos de representaciones gráficas. Es simple de usar y gratuita, aunque no tiene todas las características avanzadas de herramientas más complejas.

- **Características:**
 - Soporta diagramas UML básicos, diagramas de flujo y otros tipos de diagramas.

- Interfaz de usuario intuitiva.
- Exportación de diagramas en varios formatos gráficos.

- **Enlace:** [Dia](#)

4. Visual Paradigm Community Edition

Visual Paradigm es una suite poderosa de herramientas CASE que incluye opciones para modelado UML, análisis de requisitos, y diseño de bases de datos. La versión Community Edition es gratuita, pero tiene algunas limitaciones en comparación con las versiones comerciales.

- **Características:**

- Soporta UML, SysML y BPMN.
- Integración con herramientas como Jira.
- Edición de diagramas colaborativa.

- **Enlace:** [Visual Paradigm Community Edition](#)

5. Modelio

Modelio es una plataforma de modelado de código abierto que permite crear diagramas UML, así como diagramas BPMN (Business Process Model and Notation). Es una herramienta bastante potente para los usuarios que necesitan realizar un modelado visual en el desarrollo de software.

- **Características:**

- Soporta UML 2.0 y BPMN.
- Herramientas para generar código desde los diagramas.
- Complementos disponibles para ampliar las funcionalidades.

- **Enlace:** [Modelio](#)

6. Bouml

Bouml es una herramienta CASE gratuita para modelado UML que soporta la mayoría de los diagramas estándar de UML 2.x. Es liviana y ofrece características como la generación de código para varios lenguajes de programación.

- **Características:**

- Soporta UML 2.x.
- Generación de código para varios lenguajes (C++, Java, PHP, etc.).
- Interfaz altamente personalizable.

- **Enlace:** [Bouml](#)

7. UMLet

UMLet es una herramienta ligera y fácil de usar para crear diagramas UML rápidamente. Es ideal para aquellos que buscan una solución sencilla para modelar sus sistemas sin necesidad de características complejas.

- **Características:**

- Interfaz rápida y simple.
- Fácil integración con Eclipse.
- Generación de diagramas UML de manera rápida y eficiente.

- **Enlace:** [UMLet](#)

8. Creately (versión gratuita)

Creately es una herramienta de diagramación en línea que ofrece una versión gratuita limitada que soporta la creación de diagramas UML, diagramas de flujo y otros tipos de diagramas de software. Aunque la versión gratuita tiene limitaciones, es útil para proyectos pequeños o para usuarios que necesitan crear diagramas de forma rápida.

- **Características:**

- Herramientas de diagramación en línea.
- Plantillas predefinidas para diagramas UML y otros tipos de diagramas.
- Colaboración en tiempo real.

- **Enlace:** [Creately](#)

9. Pencil Project

Pencil Project es una herramienta gratuita de prototipado y diagramación que también soporta la creación de diagramas UML. Si bien no es una herramienta UML tradicional, se puede usar para crear diagramas básicos de software.

- **Características:**

- Herramientas de prototipado de interfaces de usuario.
- Soporte básico para diagramas UML y diagramas de flujo.
- Exportación a diversos formatos gráficos y de documentación.

- **Enlace:** [Pencil Project](#)

10. Lucidchart (versión gratuita)

Lucidchart es una herramienta de diagramación basada en la web que ofrece una versión gratuita limitada. Es ampliamente utilizada para la creación de diagramas UML, diagramas de flujo y otros tipos de representaciones visuales.

- **Características:**

- Crear diagramas UML, diagramas de flujo, wireframes y más.
- Integración con herramientas de colaboración como Google Drive.
- Versiones gratuitas y de pago con diferentes limitaciones.

- **Enlace:** [Lucidchart](#)

2.2 Diseño arquitectónico

El diseño arquitectónico en ingeniería de software se refiere a la estructura y organización de los componentes de un sistema, así como a la interacción entre ellos.

Pressman (2010) menciona que el *diseño de la arquitectura* del software es el equivalente del plano de una casa. Éste ilustra la distribución general de las habitaciones, su tamaño, forma y relaciones entre ellas, así como las puertas y ventanas que permiten el movimiento entre los cuartos.

El plano da una visión general de la casa. Los elementos del diseño de la arquitectura dan la visión general del software.

El modelo arquitectónico proviene de tres fuentes: 1) información sobre el dominio de la aplicación del software que se va a elaborar, 2) los elementos específicos del modelo de requerimientos, tales como diagramas de flujo de datos o clases de análisis, sus relaciones y colaboraciones para el problema en cuestión y 3) la disponibilidad de estilos arquitectónicos

Ejemplos:

1. Arquitectura en capas: Divide el sistema en capas funcionales, como presentación, lógica de negocio y acceso a datos. Cada capa interactúa con la capa adyacente, lo que facilita el mantenimiento y la escalabilidad. La figura 2.2 muestra la estructura básica de la arquitectura en capas.

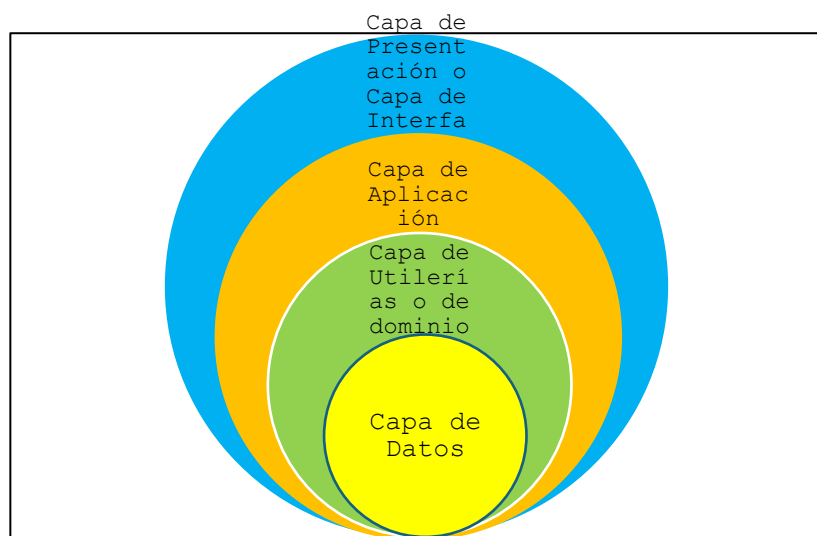


Figura 2. 2 Arquitectura en capas

2. Arquitectura de flujo de datos: Ideal para sistemas que procesan grandes volúmenes de datos. Los componentes están organizados para permitir el flujo continuo de datos entre ellos. La figura 2.3 muestra la imagen cómo se representa esta arquitectura. Se incluye el término “dashboard report” Los dashboards proporcionan una visión general rápida y procesable, facilitando el monitoreo de los objetivos, mientras que los reportes profundizan en el análisis histórico para identificar tendencias.

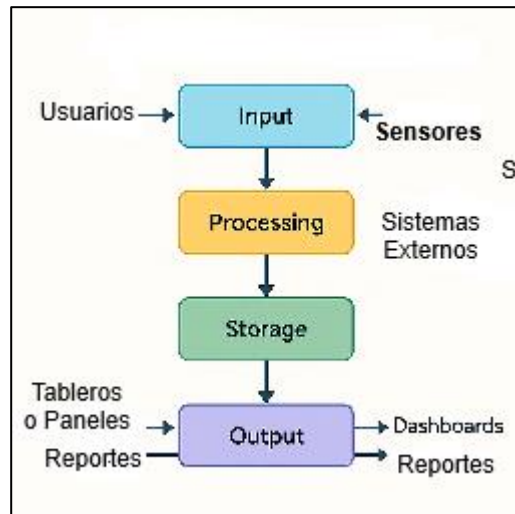


Figura 2. 3 Arquitectura de flujo de datos

3. Arquitectura orientada a servicios (SOA): Los componentes del sistema se diseñan como servicios independientes que se comunican entre sí a través de protocolos estándar. La figura 2.4 muestra esta arquitectura.

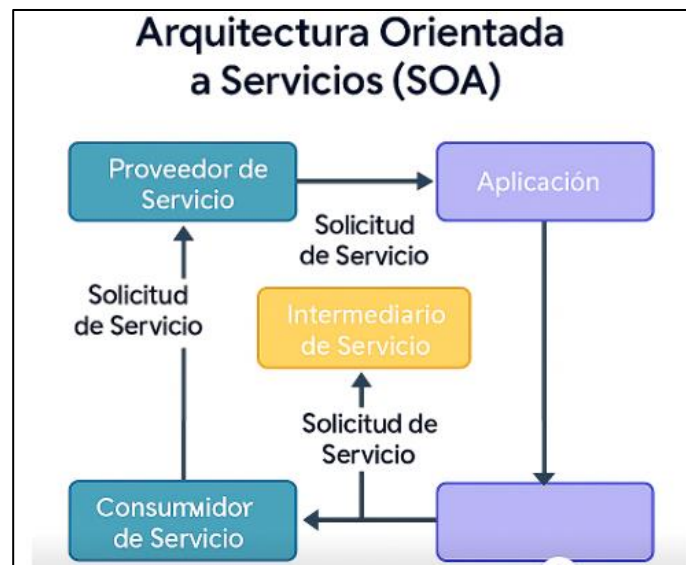


Figura 2. 4 Arquitectura orientada a servicios (SOA)

4. Arquitectura de microservicios: Similar a SOA, pero con servicios más pequeños y especializados que se pueden desplegar y escalar de manera independiente. La figura 2.5 muestra esta arquitectura.

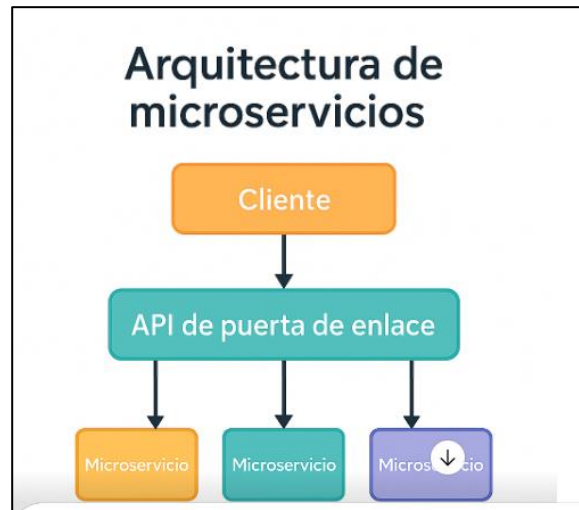


Figura 2. 5 Arquitectura orientada a microservicios

5. **Arquitectura cliente-servidor:** Un modelo clásico donde el cliente solicita servicios al servidor, que procesa y responde. La figura 3.6 muestra esta arquitectura.

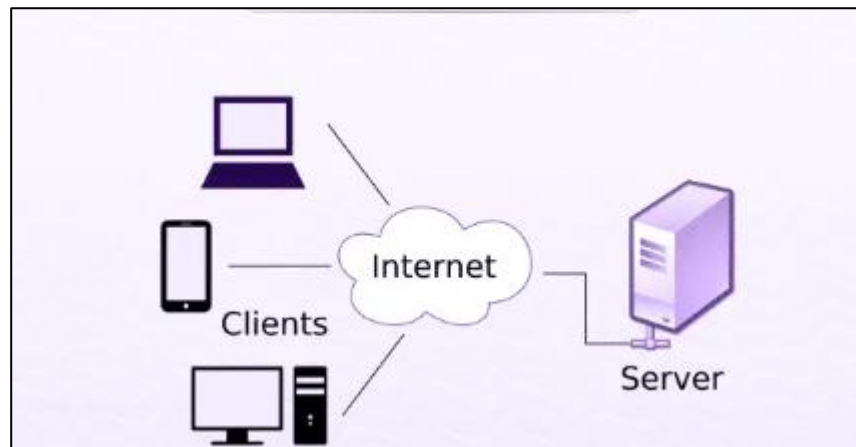


Figura 2. 6 Arquitectura cliente-servidor

La arquitectura de un sistema de información no depende solamente de un solo estilo. Sommerville (2005) sugiere que, en sistemas de software grandes, se pueden tener diferentes estilos. Pueden diseñarse diferentes partes del sistema utilizando distintos estilos arquitectónicos y en algunas ocasiones puede ser una arquitectura compuesta mediante la combinación de diferentes estilos arquitectónicos.

Garlan y Shaw (2000) Sugieren que se debe de elegir la estructura más adecuada incluyendo 3 cuestiones de diseño: se debe elegir la estructura más adecuada tal como una estructura cliente-servidor o por capas que permita satisfacer los requerimientos del sistema.

Para descomponer las unidades del sistema estructural en módulos el analista tiene que decidir la estrategia para descomponer subsistemas en sus componentes o módulos. Las aproximaciones que pueden usarse permiten implementar diferentes tipos arquitectónicos. Finalmente, en el proceso de modelado de control se toma la decisión sobre cómo se controla la ejecución de los subsistemas.

Se desarrolla un modelo general de las relaciones de control entre las partes establecidas del sistema. La evaluación de un diseño arquitectónico es difícil debido a que la verdadera prueba de una arquitectura consiste en averiguar el grado de satisfacción de sus requerimientos funcionales.

El diseño arquitectónico básicamente permite definir cómo se estructura un sistema. También se considera si existen patrones a seguir para desarrollar un software. La figura 2.7 muestra el diagrama de componentes de la arquitectura básica para un sistema de suscripciones con pagos en PayPal.

Existen tres pasos principales para definir cómo se estructura el sistema:

1. Entender los requerimientos no funcionales (¿Cómo debe operar?)

- Debe estar conectado a Internet
- Debe tener versión web
- Debe tener aplicación móvil
- Debe contener una API para la base de datos

Identificar los requerimientos funcionales ¿Qué va a hacer en operación?

- Sistema de suscripciones de pago
- Control de tiempo de visitas
- Validación con aplicación móvil.

2. Identificar los componentes clave del software

- Sistema de pagos
- Seguimiento a suscripciones
- Aplicación móvil
- Aplicación web
- API para la aplicación móvil
- Sistema de Autenticación

3. Establecer los medios de conexión entre componentes

Conexiones de los componentes (Crear diagrama de componentes de UML)

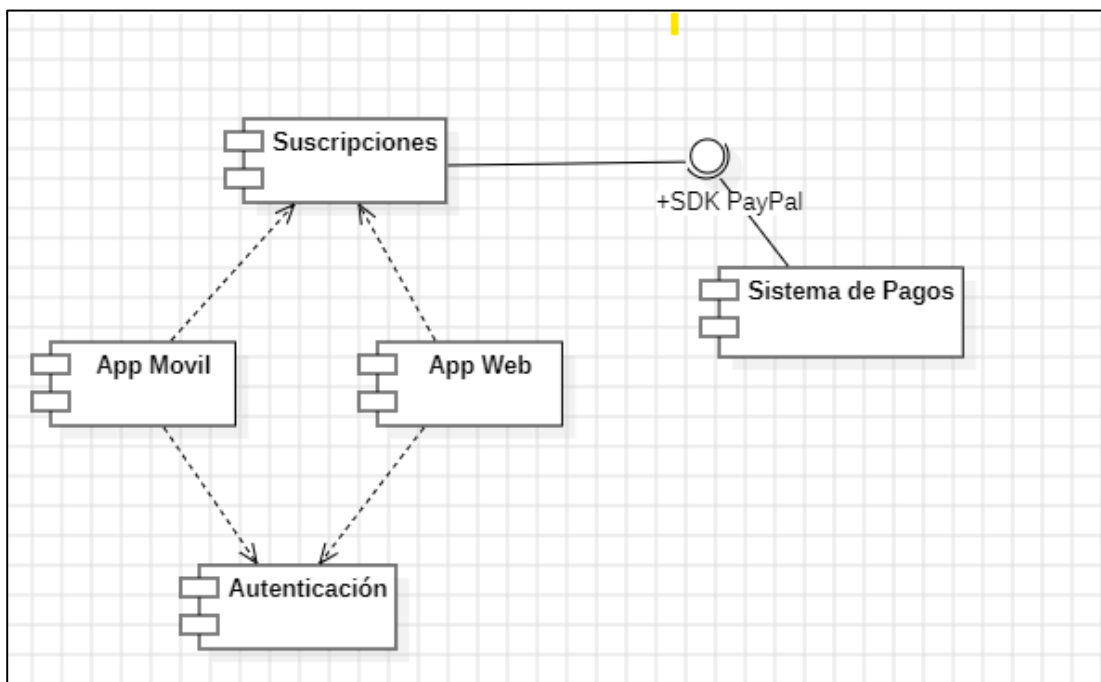


Figura 2. 7 Diseño arquitectónico de un sistema de suscripciones

2.3 Diseño de datos

El diseño de datos en ingeniería de software se refiere al proceso de definir cómo se organizan y almacenan los datos dentro de un sistema de software. Esto implica seleccionar estructuras de datos adecuadas para representar los objetos de datos y las relaciones entre ellos. El diseño de datos es fundamental para la eficiencia, la calidad y la integridad de los datos dentro de una aplicación de software. Pressman (2010) menciona que diseño de datos o clases transforma los modelos de clases en realizaciones de clases de diseño y en las estructuras de datos que se requieren para implementar el software.

El diseño de datos se realiza a través del modelado de datos, que es una representación visual de los datos y sus relaciones, y puede ser utilizado para crear o rediseñar bases de datos. Existen diferentes tipos de modelos de datos, como los modelos relacionales, los modelos orientados a objetos y los modelos dimensionales, cada uno con sus propias fortalezas y debilidades, y cada uno se adapta a diferentes necesidades de diseño.

El diseño de datos es una etapa crucial en el desarrollo de software, ya que afecta la eficiencia, la calidad y la integridad de los datos almacenados y manipulados por la aplicación. Una buena selección de estructuras de datos y una representación clara de las relaciones entre los datos pueden facilitar el desarrollo, la implementación y el mantenimiento de la aplicación.

En resumen, el diseño de datos en ingeniería de software es el proceso de definir la estructura, organización y almacenamiento de los datos dentro de un sistema, a través del modelado de datos y la selección de estructuras de datos adecuadas, para garantizar la accesibilidad, eficiencia, la calidad y la integridad de los datos. La figura 2.8 muestra un fragmento del modelo entidad-relación para un sistema de aeronaves.

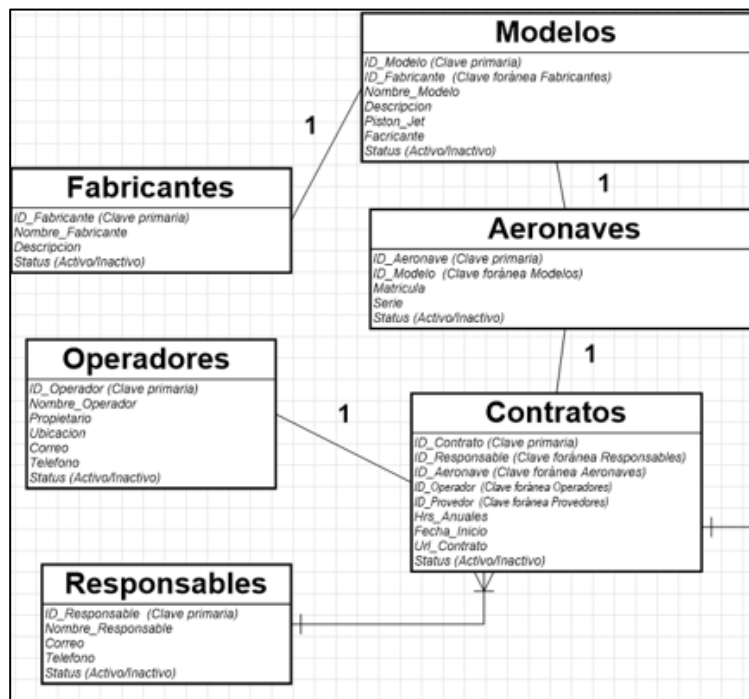


Figura 2. 8 Ejemplo del Diagrama Entidad-Relación

A continuación, se mencionan algunos aspectos clave:

1. **Modelado de datos:** Utiliza diagramas como el modelo entidad-relación (ER) para representar las relaciones entre datos. Esto ayuda a visualizar cómo se conectan los datos dentro del sistema.
2. **Normalización:** Es el proceso de organizar los datos en tablas para reducir la redundancia y mejorar la integridad de los datos.
3. **Diseño de bases de datos:** Incluye la elección de tipos de datos, índices y claves primarias para optimizar el rendimiento y la recuperación de datos.
4. **Seguridad de datos:** Implementa medidas como el cifrado y el control de acceso para proteger los datos sensibles.

Igual que otras actividades de la ingeniería de software, el diseño de datos (en ocasiones denominado arquitectura de datos) crea un modelo de datos o información que se representa en un nivel de abstracción elevado (el punto de vista del usuario de los datos). Este modelo de los datos se refina después en forma progresiva hacia representaciones más específicas de la implementación que puedan ser procesadas por el sistema basado en computadora. En muchas aplicaciones de software, la arquitectura de los datos tendrá una influencia profunda en la arquitectura del software que debe procesarlo.

La estructura de los datos siempre ha sido parte importante del diseño de software. En el nivel de componentes del programa, del diseño de las estructuras de datos y de los algoritmos requeridos para manipularlos, es esencial la creación de aplicaciones de alta calidad. En el nivel de la aplicación, la traducción de un modelo de datos (obtenido como parte de la ingeniería de los requerimientos) a una base de datos es crucial para lograr los objetivos de negocios de un sistema.

En el nivel de negocios, el conjunto de información almacenada en bases de datos incompatibles y reorganizados en un “data warehouse” permite la minería de datos o descubrimiento de conocimiento que tiene un efecto en el éxito del negocio en sí. En cada caso, el diseño de los datos juega un papel importante.

Un modelo de datos bien diseñado tiene el beneficio de la reutilización, en gran medida lograda mediante la especialización, y el mantenimiento simple. El diseño del modelo de datos es tan importante como el diseño del proceso para un proyecto de software.

El modelo de datos ofrece un buen proyecto del flujo de datos completo en el ciclo de vida de los procesos de negocio. Un buen proyecto de los datos ayuda a utilizar los datos con eficiencia y perfección.

Cuando comienza a diseñar la solución de modelo de datos para el escenario de negocio dado, los pasos básicos del modelado de datos son estos:

- Identificar objetos de datos.
- Identificar atributos para cada objeto de datos.
- Identificar objetos de datos existentes que se pueden reutilizar (herencia o composición).
- Relacionar los objetos de datos.
- Planificar para lograr persistencia/integración de los objetos de datos creados por la aplicación.

El diseño del modelo de datos es muy importante y una fase clave en el desarrollo de la aplicación a fin de cumplir las necesidades del negocio. Un Lead System Architect (LSA) se centra en el modelado de datos para aprovechar los siguientes beneficios.

- La complejidad de los datos se puede reducir mediante la clasificación y el modelado de datos.
- Con el modelado de datos, se pueden identificar los datos requeridos para cumplir los requerimientos del negocio. Se pueden hacer planes para capturar los datos desde el origen y evitar cualquier pérdida de datos.
- El modelado de datos ayuda en el diseño de un plan de guardado de datos para que todos los datos obtenidos se conserven y se mantenga la integridad del sistema.
- La precisión y exactitud de los datos se pueden mantener teniendo en cuenta los diseños de integridad de datos (manteniendo una fuente de verdad única).
- En el modelado de datos se puede planear el proceso de resguardar y publicar los datos para otras partes.
- El modelado de datos ayuda a definir y describir los datos según la necesidad de las diferentes partes interesadas. Por ejemplo, un gerente puede necesitar datos para la creación de reportes, pero el desarrollador usa los mismos datos para ejecutar la lógica del negocio.
- Los requerimientos futuros se consideran en el modelado de datos, por lo que la aplicación sigue siendo sólida, flexible y escalable.

Para comprender los beneficios del modelo de datos, considere el siguiente ejemplo:

Una aplicación de solicitud de compra tiene dos procesos diferentes: gestión del pedido y gestión del producto. Dos equipos de prueba están implementando la gestión del pedido y la gestión del producto al mismo tiempo, cada uno modelando su propia definición de un producto. Existe la posibilidad de que ambos equipos de prueba entreguen dos diseños diferentes para un producto en lugar de una definición reutilizable para toda la empresa. Esta situación genera que haya diferentes formas de detectar el mismo producto y duplicidad de los datos. Por consecuencia, el mantenimiento se torna más difícil, la aplicación es menos flexible y no es escalable.

2.4 Diseño de interfaz de usuario

Es el proceso para crear la apariencia visual y la interacción de una aplicación o sitio web, centrándose en la usabilidad, accesibilidad y experiencia del usuario. En otras palabras, se trata de diseñar cómo los usuarios interactúan con un producto digital, desde la disposición de los elementos en la pantalla hasta la elección de colores y tipografía.

El diseño de la interfaz de usuario tiene el propósito principal de preparar la comunicación entre los usuarios y los dispositivos electrónicos, tales como tabletas, teléfonos celulares y computadoras, y comienza con la identificación de los requerimientos del usuario, la tarea y el ambiente.

Pressman, (2010) ha definido principios que guían la práctica, de los cuales describe que *“El diseño de la interfaz de usuario debe ajustarse a las necesidades del usuario final.”*

La interfaz de usuario es la manifestación visible del software. No importa que tan sofisticadas sean sus funciones internas, ni lo incluyentes que sean sus estructuras de datos, ni lo bien diseñada que esté su arquitectura, un mal diseño de la interfaz con frecuencia conduce a la percepción de un software “malo”.

Siguiendo un conjunto de principios de diseño de la interfaz, el diseño identifica los objetos y acciones de ésta y luego crea una plantilla de pantalla que constituye la base del prototipo de la interfaz de usuario (Pressman, 2010).

Una vez identificadas las tareas del usuario, se crean y analizan los escenarios para éste y se define un conjunto de objetos y acciones de la interfaz.

Esto forma la base para crear una plantilla de la pantalla que ilustra el diseño gráfico y la colocación de los iconos, la definición de textos descriptivos, la especificación y títulos de las ventanas, y la especificación de aspectos mayores y menores del menú.

La figura 2.9 muestra un ejemplo de diseño de interfaz.



Figura 2. 9 Interfaz sistema Mecánico Automotriz

Con el empleo de herramientas digitales, tales como Figma, permite crear el prototipo, se implementa en definitiva el modelo del diseño para evaluar la calidad del resultado.

Las herramientas más comunes para el diseño de interfaz de usuario (UI) son Figma, Adobe XD, Sketch y Adobe Photoshop. Estas herramientas facilitan la creación de wireframes, mockups, prototipos interactivos y la colaboración entre diseñadores y desarrolladores.

Herramientas de diseño de interfaz de usuario (UI):

- **Figma:**

Una plataforma de diseño web de código abierto que permite la colaboración en tiempo real y la creación de prototipos interactivos.

- **Adobe XD:**

Una herramienta de Adobe Creative Cloud que permite diseñar interfaces de usuario y prototipos para aplicaciones web y móviles.

- **Sketch:**

Una herramienta de diseño vectorial popular entre los diseñadores UI, especialmente en macOS.

- **Adobe Photoshop:**

Aunque principalmente una herramienta de diseño gráfico, puede ser utilizada para crear elementos de UI como iconos, imágenes y otros recursos visuales.

Otras herramientas útiles:

- **Balsamiq:**

Ideal para crear wireframes y bocetos de baja fidelidad rápidamente.

- **InVision:**

Una herramienta de prototipado que permite crear prototipos interactivos y colaborativos.

- **Wireframe:**

Herramientas de wireframing como Mockflow ayudan a visualizar la estructura y flujo de una interfaz.

- **Herramientas de colaboración:**

Herramientas como Slack o Trello pueden ayudar a facilitar la comunicación y colaboración entre diseñadores y desarrolladores.

Las herramientas de diseño de UI se complementan entre sí, y la elección de la herramienta dependerá del tipo de proyecto y las preferencias del diseñador.

El diseño de interfaces web es un aspecto básico a la hora de facilitar lo máximo posible la interacción entre el usuario y el sistema. Un diseño de interfaz bien estructurado facilita la usabilidad y mejora la experiencia del usuario, dando lugar a una mayor conexión emocional con el producto. Por eso es importante conocer las directrices para el diseño de interfaces de calidad.

En el ámbito informático, la interfaz de una página web se refiere a cualquier software o herramienta que permite la comunicación entre dos entidades diferentes, ya sea entre un usuario y un sistema, dos sistemas informáticos o entre diferentes partes de un sistema.

Una interfaz web, en particular, es un tipo de interfaz que permite a los usuarios interactuar y comunicarse con sistemas o aplicaciones a través de un navegador web. Estas interacciones pueden ser tan simples como hacer clic en un enlace o introducir información en un formulario, o tan complejas como realizar transacciones financieras o gestionar bases de datos.

Las interfaces web están diseñadas para ser intuitivas, permitiendo a los usuarios navegar y realizar acciones sin necesidad de tener conocimientos técnicos profundos.

Es decir, una interfaz actúa como un puente de comunicación y, en el contexto web, facilita nuestra interacción con sistemas y aplicaciones en el mundo digital. Así, cada vez que navegamos por una página web, estamos interactuando con una interfaz web diseñada para ofrecernos una experiencia de usuario óptima. Es a través de estas interfaces que la tecnología se vuelve accesible y útil para todos, independientemente del nivel de conocimiento técnico de cada usuario.

Características de una interfaz web

Una interfaz web bien diseñada posee varias características clave:

Intuitiva: Debe ser fácil de entender y usar, incluso para aquellos que no están familiarizados con la tecnología.

Responsiva: Se adapta a diferentes dispositivos y tamaños de pantalla, ofreciendo una experiencia uniforme tanto en ordenadores de escritorio como en celulares y otros dispositivos móviles.

Consistente: Mantiene una apariencia y funcionalidad uniformes en toda la web, lo que facilita que los usuarios sepan qué esperar en cada página.

Accesible: Cumple con los estándares de accesibilidad, garantizando que todos, incluidas las personas con discapacidades, puedan acceder y utilizar el sitio.

Estéticamente agradable: Posee un diseño visualmente atractivo que se alinea con la marca y el propósito del sitio.

Importancia de la interfaz en el diseño web

La interfaz no es solo un aspecto visual, es el nexo entre el usuario y la funcionalidad de la página web. Una interfaz inadecuada puede alejar a los usuarios, mientras que una bien diseñada puede facilitar la retención de usuarios y aumentar la conversión.

La interfaz determina la facilidad de navegación, la comprensión del contenido y la eficiencia en la realización de tareas específicas. Por lo tanto, invertir tiempo y recursos en una interfaz efectiva puede tener un impacto directo en el éxito y la rentabilidad de una página web.

¿Qué es el diseño de interfaces?

El diseño de interfaces se refiere a la creación de la apariencia visual y la interacción de un software o aplicación. Abarca desde la disposición de los elementos en la pantalla hasta la elección de colores y tipografías, así como la forma de navegar por ella. Los mejores ejemplos de diseño de interfaz de usuario (UI) consideran objetivos que la comunicación entre los usuarios y el sistema resulte intuitiva, accesible y lo más agradable posible a nivel estético.

Principios básicos del diseño de interfaces

Para crear una interfaz, hay ciertos principios que los diseñadores de interfaces deben tener en cuenta:

Claridad: Todo elemento en la interfaz debe tener un propósito claro y comunicar su función de manera inequívoca.

Feedback: Los usuarios deben recibir retroalimentación inmediata sobre sus acciones, ya sea un clic de un botón o la entrada de datos.

Tolerancia a errores: Las interfaces deben ser diseñadas de tal manera que minimicen los errores del usuario y permitan una recuperación fácil en caso de que ocurran.

Flexibilidad: Las interfaces deben ser flexibles para adaptarse a las diversas necesidades y habilidades de los usuarios.

Eficiencia: Con el tiempo, los usuarios deben ser capaces de realizar tareas de manera más rápida y directa a medida que se familiarizan con la interfaz.

Simplicidad: La interfaz debe ser simple y fácil de entender, lo que ayuda a evitar una complejidad innecesaria.

Visibilidad: Los elementos más importantes de la interfaz deben ser fácilmente accesibles y visibles para el usuario, reduciendo la necesidad de realizar búsquedas.

Simplicidad: La interfaz debe ser simple y fácil de entender, lo que ayuda a evitar una complejidad innecesaria.

Visibilidad: Los elementos más importantes de la interfaz deben ser fácilmente accesibles y visibles para el usuario, reduciendo la necesidad de realizar búsquedas.

Elementos del diseño de interfaces

Ahora que conocen los principios para el diseño de interfaces de usuario más básicos, les interesa conocer los elementos que no deben faltar en sus proyectos web para crear una experiencia de usuario tan agradable como coherente.

La elección de fuentes influye en la legibilidad y la percepción visual del contenido.

Los colores crean una identidad visual, pero también guían la atención del usuario y generan emociones.

Los iconos simplifican la comprensión de funcionalidades y mejoran la navegación.

Agregar botones es fundamental para facilitar la interacción, ¡por eso deben ser fácilmente identificables y predecibles!

Los formularios deben diseñarse para resultar intuitivos y fáciles de usar, disminuyendo así la posibilidad de errores.

Además, la estructura de la navegación debe ser clara para permitir al usuario moverse fácilmente por el sistema.

Proceso de diseño de interfaces

El proceso de diseño de interfaces sigue una serie de pasos que les detallamos de forma breve y concisa a continuación.

Investigación y análisis de usuarios para comprender sus necesidades y comportamientos.

Creación de wireframes, que es como se denomina a los bocetos que representan la estructura básica de la interfaz y que ayudan a planificar la disposición de los elementos.

Desarrollo de prototipos que permitan visualizar y probar la interfaz para identificar y aplicar posibles mejoras.

Realizar pruebas de usabilidad con usuarios reales para evaluar la efectividad de la interfaz y hacer ajustes necesarios.

Iteración y refinamiento, lo que significa que es necesario hacer un seguimiento del diseño y aplicación de interfaces que permita aplicar mejoras en base a los comentarios y resultados obtenidos durante las pruebas.

Herramientas populares para el diseño de interfaces

Con el auge de la digitalización, han surgido diversas herramientas que facilitan el proceso de diseño de interfaces:

Sketch: Una aplicación de diseño vectorial que ha ganado popularidad por su simplicidad y poderosas funcionalidades para diseñar interfaces.

Adobe XD: Esta herramienta de Adobe permite diseñar y prototipar interfaces de usuario para aplicaciones web y móviles.

Figma: Una plataforma basada en la nube que permite el diseño colaborativo en tiempo real, haciéndola especialmente popular entre equipos de diseño.

InVision: Ofrece soluciones para el prototipado, la colaboración y el flujo de trabajo del diseño de interfaces.

Balsamiq: Muy popular por permitir la creación rápida de bocetos o mockups de baja fidelidad, facilitando la visualización temprana de ideas.

El diseño de interfaces es esencial para crear experiencias de usuario efectivas y atractivas en el ámbito digital. Las herramientas y principios mencionados anteriormente son solo la punta del iceberg en este campo en constante evolución. Con el continuo avance tecnológico y las cambiantes expectativas de los usuarios, el diseño de interfaces seguirá siendo un aspecto crucial en el panorama digital.

Aunque el diseño de la interfaz de una página web es importante, conviene no dejar de lado aspectos tan básicos como contratar un alojamiento web de calidad, necesario para que la interfaz reaccione de forma fluida y para que la página esté siempre operativa ofreciendo además la mejor experiencia de usuario en todo momento.

Ejemplos de diseño de interfaces de usuario

El diseño de interfaces de usuario es esencial para el éxito de los proyectos digitales, y un buen ejemplo de ello es Material Design de Google, un sistema de diseño utilizado en productos de talla mundial como Google Drive y Gmail y que proporciona una experiencia eficiente y fácil de usar mediante el uso de rejillas, transiciones y efectos de profundidad.

Otro ejemplo es la Human Interface Guidelines de Apple, que define un enfoque minimalista y elegante en aplicaciones de iOS, lo que garantiza una simplicidad y usabilidad excelentes.

Ambas guías muestran cómo un buen diseño de interfaz mejora la estética, la funcionalidad y la satisfacción del usuario.

Directrices para el diseño de interfaces

El diseño de interfaces que ofrezcan buenos resultados requiere de la aplicación de algunas directrices que garanticen una experiencia de usuario positiva y accesible.

- Asegurarse de que todas las personas, independientemente de sus capacidades, puedan utilizar la interfaz sin problemas.
- Poner al usuario en el centro del proceso de diseño, garantizando que las decisiones estén basadas en sus necesidades y comportamientos.
- Crear interfaces que se adapten fluidamente a diferentes dispositivos y tamaños de pantalla, ofreciendo siempre la mejor experiencia posible en todas las plataformas.

Aplicación de interfaces en proyectos

Integrar el diseño de interfaces en el ciclo de desarrollo de software es imprescindible para el éxito de cualquier producto, y esto implica la colaboración estrecha entre diseñadores y desarrolladores desde las primeras etapas del proyecto.

Es necesario empezar con la investigación de usuarios para comprender sus necesidades, seguido de la creación de wireframes y prototipos. Durante el desarrollo, es indispensable realizar pruebas de usabilidad para iterar y refinar el diseño.

Además, es aconsejable mantener una comunicación constante y utilizar herramientas colaborativas como Figma o Sketch, que facilitan la sincronización entre el diseño y la implementación técnica.

El diseño de interfaces web es un aspecto básico a la hora de facilitar lo máximo posible la interacción entre el usuario y el sistema. Un diseño de interfaz bien estructurado facilita la usabilidad y mejora la experiencia del usuario, dando lugar a una mayor conexión emocional con el producto. Por eso es importante conocer las directrices para el diseño de interfaces de calidad.

ACTIVIDADES TEMA 2

El complemento educacional basado en **ESTUDIO DE CASOS** en ingeniería de software consiste en presentar a los estudiantes situaciones reales o simuladas relacionadas con proyectos de desarrollo de software, para que puedan analizar, tomar decisiones y aplicar sus conocimientos en un contexto práctico.

CASO DE ESTUDIO:

Presentación del caso:

El profesor entrega un documento o video que describe la situación:

"Una pequeña empresa de Metepec quiere lanzar una tienda en línea para vender productos artesanales. Necesitan un sistema que sea fácil de usar, seguro y capaz de manejar un crecimiento en el número de usuarios."

Trabajo en grupos:

Los estudiantes se dividen en equipos y analizan:

- Requisitos funcionales y no funcionales
- Tecnologías posibles (por ejemplo, frameworks, bases de datos, servicios en la nube)
- Riesgos y desafíos (seguridad, rendimiento, escalabilidad)
- Planificación del proceso de desarrollo (etapas, entregables, cronograma)

Discusión y presentación:

Cada grupo presenta su propuesta, explicando sus decisiones y justificaciones. Luego, en clase, se comparan las diferentes soluciones, discutiendo ventajas y posibles mejoras.

Retroalimentación y reflexión:

El profesor comenta sobre las propuestas, corrige conceptos y comparte experiencias reales relacionadas. Los estudiantes reflexionan sobre lo aprendido y cómo aplicar esas habilidades en proyectos reales.

EJERCICIOS:

Una empresa del sector salud necesita desarrollar un sistema de gestión de citas médicas, enfrentando desafíos en privacidad de datos y compatibilidad con dispositivos móviles. Los estudiantes deben diseñar una solución considerando normativas de protección de datos y usabilidad.

Se debe elaborar el análisis de requisitos basado en entrevistas a personal médico y administrativo de la institución. Con la información obtenida, deben diseñar diagramas correspondientes al modelo de análisis: Diagramas de Casos de Uso, Diagramas de Secuencias, Diagramas de Actividades, Diagramas de Estados y Diagramas de Clases.

A partir de esta documentación, deberán diseñar el modelo arquitectónico, modelo de datos y diseño de interfaces.

TEMA 3. DESARROLLO

Introducción

El desarrollo de software se refiere a un conjunto de actividades informáticas que se dedican al proceso de creación, diseño, implementación y soporte de software (Quiroz, 2025).

El proceso de desarrollo de software no solo consiste en codificar las instrucciones de un algoritmo, sino también abarca las siguientes actividades:

- **Análisis de requisitos:** Identificar las necesidades y requerimientos del software que se va a desarrollar.
- **Diseño:** Planificar la arquitectura, la interfaz de usuario y la funcionalidad del software.
- **Codificación:** Es la creación del código fuente utilizando lenguajes de programación específicos y un gestor de base de datos.
- **Pruebas:** Verificar la calidad y el correcto funcionamiento del software a través de diferentes tipos de pruebas.
- **Implementación:** Poner el software a disposición de los usuarios finales.
- **Mantenimiento:** Realizar actualizaciones, correcciones de errores y mejoras continuas del software.

En resumen, el desarrollo de software es el proceso de concebir, diseñar, implementar, probar y mantener software.

Según Somerville (Somerville, 2005), es importante considerar la arquitectura del software desde el principio del proceso, de no ser así, es probable que la estructura general del sistema sea inestable y se degrade conforme avance el desarrollo del sistema.

La figura 3.1 muestra el proceso de desarrollo de sistemas aplicando la Metodología de desarrollo Iterativo-Incremental, en la cual se muestra que se deben definir claramente los productos a entregar del sistema para diseñar su arquitectura y de esta manera ir desarrollando los incrementos del software según se van validando con el cliente, hasta terminar todas las validaciones de los incrementos.

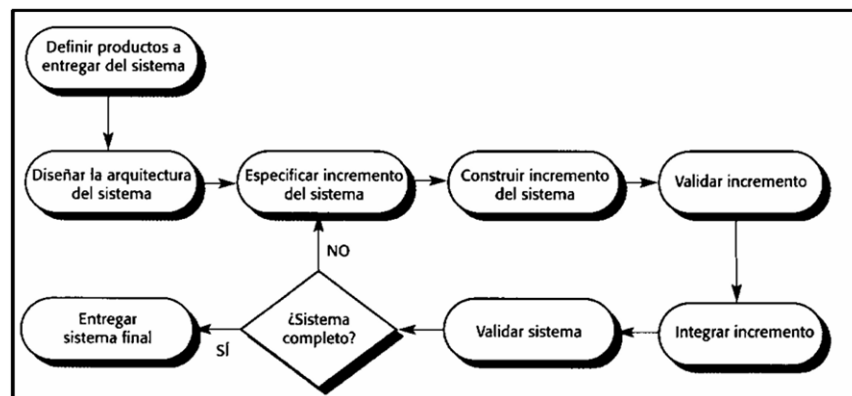


Figura 3. 1 Metodología de Proceso Iterativo-Incremental (Somerville, 2005)

Independientemente de la metodología de desarrollo de software a aplicar, es importante considerar la arquitectura desde el inicio del desarrollo del sistema, a partir de que se cuente con el **Documento de Especificación de Requisitos del Software (SRS** por sus siglas en inglés Software Requirements Specification), documento técnico que sirve como base para construir un sistema de software.

Este documento puede incluir las siguientes secciones:

1. Introducción

- Propósito del sistema: ¿Qué problema resuelve o qué necesidad cubre?
- Alcance del sistema: Breve descripción de sus funcionalidades principales.
- Audiencia prevista: A quién está dirigido este documento (desarrolladores, clientes, usuarios).

2. Descripción general

- Contexto del sistema: Cómo se relaciona con otros sistemas o procesos.
- Supuestos y dependencias: Tecnologías, herramientas o plataformas necesarias.
- Restricciones: Límites técnicos, legales, temporales o presupuestales.

3. Requisitos funcionales

Lista clara y detallada de funcionalidades específicas, por ejemplo:

- El usuario puede registrarse con correo electrónico y contraseña.
- El sistema permite generar reportes mensuales en formato PDF.

4. Requisitos no funcionales

Características relacionadas con el rendimiento, seguridad, disponibilidad, etc.:

- La plataforma debe soportar al menos 1,000 usuarios concurrentes.
- El sistema debe estar disponible el 99.9% del tiempo.

5. Requisitos de pruebas

Criterios para saber si el sistema cumple lo esperado:

- Pruebas unitarias para cada módulo.
- Pruebas de carga para el sistema completo.

6. Roles y responsabilidades

Lo que hace cada integrante del equipo de desarrollo:

- El equipo de frontend se encarga de la interfaz.
- El analista funcional valida los requisitos con el cliente.

El desarrollo de software es un proceso complejo que involucra diversas etapas y personal capacitado, con el objetivo de crear soluciones informáticas eficientes, funcionales y de alta calidad.

3.1 Lenguajes para el desarrollo del software

A partir del diseño de algoritmos, se requiere convertir el código fuente en un lenguaje propiamente diseñado para que la computadora entienda las instrucciones y son llamados lenguajes de programación.

Los lenguajes de programación se componen de un formato predefinido y sintaxis específica para cada lenguaje.

El software puede diseñarse como software a medida (con los requisitos definidos por el cliente) o software comercial. El desarrollo de software a la medida es el proceso de diseño, creación, implementación y mantenimiento de software para un conjunto específico de usuarios, funciones u organizaciones.

Por el contrario, el software comercial listo para usar (COTS, por sus siglas en inglés Commercial Off-The-Shelf) está diseñado para un amplio conjunto de requisitos, lo que permite empaquetarlo, comercializarlo y distribuirlo. Por ejemplo, el software de contabilidad que lo puede comercializar la empresa Siigo Aspe, llamada anteriormente Aspel COI en México.

También lo distribuye la empresa SYCA Consultores y Heti Computación Integral, que ofrecen servicios de venta, implementación y soporte técnico.

Los lenguajes de programación han ido evolucionando conforme a los avances tecnológicos, aunque existen lenguajes de programación para desarrollar Inteligencia Artificial, es necesario entender lo que se quiere hacer a nivel sistémico para poder diseñar y desarrollar el software que atienda los requerimientos del cliente.

Para el desarrollo de software, existen numerosos lenguajes de programación, cada uno con sus fortalezas y debilidades. Los más populares y utilizados incluyen Python, Java, JavaScript, C++, C#, y Go. Otros lenguajes como PHP, Ruby y Rust también son relevantes en diversos contextos de desarrollo. Al elegir un lenguaje, es importante considerar las necesidades del proyecto, el tipo de aplicación que se va a desarrollar, la experiencia del equipo y la disponibilidad de recursos y herramientas.

Se han escrito muchos libros sobre programación (codificación) y sobre los principios y conceptos que la guían que sobre cualquier otro tema del proceso de software que incluyen obras tempranas sobre estilo de programación, construcción de software práctico, perlas de programación, el arte de programar, temas pragmáticos de programación, entre otros.

A continuación, se listan los lenguajes de programación más populares para desarrollo de software:

- **Python** (www.python.org):

Es conocido por su sintaxis clara y legible, lo que facilita su aprendizaje y uso, especialmente en áreas como análisis de datos, inteligencia artificial, desarrollo web (con frameworks como Django y Flask) y scripting.

- **Java:** Lenguaje de propósito general, utilizado en el desarrollo empresarial, aplicaciones Android, sistemas distribuidos y sistemas backend. Es compilado. Se transforma en bytecode que corre en la Máquina Virtual de Java (JVM). Orientado a objetos puro, basado en clases. Corre en cualquier sistema con JVM (Windows, Linux, Android). Más robusto y seguro para sistemas críticos. Tipado estático y fuerte. Se debe declarar el tipo de cada variable. Ejemplo de código:

```
public class Hola { public static void main(String[] args) { System.out.println("Hola, Mundo!"); }
}
```

- **JavaScript:** Diseñado para dar interactividad a páginas web. Es interpretado. Se ejecuta directamente en el navegador o en entornos como Node.js. También se usa en backend (Corre en navegadores web y servidores con Node.js), apps móviles y más. Soporta objetos, funciones, y programación funcional. Usa prototipos en lugar de clases tradicionales. Tipado dinámico y débil. Se puede cambiar el tipo de una variable sobre la marcha. Es multiparadigma. Más flexible, pero puede ser vulnerable si no se maneja bien (por ejemplo, ataques XSS).

Ejemplo de código:

```
console.log("Hola, Mundo!");
```

(https://developer.mozilla.org/es/docs/MDN/Community/Translated_content):

Dominante en el desarrollo web para la interfaz de usuario (front-end) y también utilizado en el back-end (Node.js) para aplicaciones web completas.

- **C++ (<https://isocpp.org/get-started>):**

Un lenguaje potente y de alto rendimiento, ideal para desarrollo de sistemas, videojuegos, aplicaciones de escritorio y software que requiere eficiencia.

- **C# (<https://learn.microsoft.com/es-es/dotnet/csharp/>):**

Desarrollado por Microsoft, es un lenguaje versátil para desarrollo de aplicaciones Windows, juegos (con Unity) y aplicaciones web.

- **Go (Golang <https://go.dev>):**

Diseñado por Google, es conocido por su eficiencia y facilidad de uso en sistemas distribuidos, redes y aplicaciones web.

- **PHP (www.php.net):**

Un lenguaje popular para el desarrollo web back-end, especialmente en sistemas de gestión de contenidos (CMS) como WordPress.

- **Ruby (<https://www.ruby-lang.org/>):**

Un lenguaje dinámico y flexible, popular por su framework para desarrollo web rápido.

- **Rust (<https://www.rust-lang.org/es>):**

Un lenguaje enfocado en la seguridad y rendimiento, ideal para sistemas embebidos, sistemas operativos y aplicaciones donde la confiabilidad es crítica.

- **SQL (<https://www.oracle.com/latam/database/technologies/appdev/sql.html>):**

Es un lenguaje fundamental para la gestión de bases de datos, especialmente en el contexto de aplicaciones web y análisis de datos. SQL es un estándar definido por organizaciones como ISO e INCITS, y la documentación de referencia se encuentra principalmente en sitios relacionados con sistemas de gestión de bases de datos que lo implementan, como [Oracle](https://www.oracle.com/latam/database/technologies/appdev/sql.html), Microsoft, y MySQL.

- **HTML** (<https://lenguajehtml.com/>) y **CSS** (<https://lenguajecss.com/>): Si bien no son lenguajes de programación en sí mismos, son fundamentales para el desarrollo web, junto con JavaScript, para estructurar y dar estilo a las páginas web.
- **Kotlin** (<https://kotlinlang.org/>): Un lenguaje moderno para desarrollo Android, que se integra bien con Java.
- **Swift** (<https://www.swift.org/>): Un lenguaje desarrollado por Apple, enfocado en el desarrollo para dispositivos iOS y macOS.

La tabla 3.1 muestra un ejemplo de lenguajes de programación según el uso que se vaya a dar.

Tabla 3. 1 Lenguajes de programación

Tipo de Aplicación	Lenguaje	Uso principal
Desarrollo web	HTML / CSS	Estructura y diseño de páginas web
	JavaScript	Interactividad en el frontend y backend (Node.js)
	PHP	Backend para sitios dinámicos
	TypeScript	JavaScript mejorado con tipado estático
	Python	Backend con frameworks como Django o Flask
Aplicaciones móviles	Java	Plataforma Android
	Kotlin	Android (más moderno que Java)
	Swift	iOS
	Dart	Con Flutter, para apps Android/iOS multiplataforma
	React Native (JavaScript)	Aplicaciones híbridas
Software de escritorio	C#	Usado en aplicaciones Windows con .NET
	Java	Multiplataforma con JavaFX o Swing
	C++	Alto rendimiento para apps complejas
	Python	Aplicaciones livianas con GUI como Tkinter o PyQt
	Java	Multiplataforma con JavaFX o Swing
Sistemas embebidos o bajo nivel	C	Sistemas operativos, controladores
	C++	Aplicaciones con alto rendimiento
	Rust	Seguridad y rendimiento moderno
	Assembly	Control directo del hardware (muy especializado)
Automatización, aplicaciones en la nube	Go (Golang)	Eficiente para servicios backend y microservicios.
	Python	Usado para scripts, ciencia de datos y automatización
	JavaScript / TypeScript:	Para aplicaciones web basadas en la nube

La creación de software en Java implica una serie de pasos que pueden variar según el tipo de aplicación que se desea desarrollar (consola, GUI, web, móvil, etc.).

A continuación, se muestra el ejemplo para crear en Java una aplicación de consola con la descripción de los pasos generales.

1. Instalar Java Development Kit (JDK)

Se debe tener instalado el **JDK** (Java Development Kit) en el equipo donde se desarrollará el sistema.

- Descargar desde: <https://www.oracle.com/java/technologies/javase-downloads.html>
- Verificar la instalación: Se debe cuidar que se instale correctamente
- Bash: **Intérprete de comandos de tipo Unix/Linux** que puede usarse para ejecutar scripts relacionados con la instalación, configuración o uso del JDK.
- CopiarEditar:

`java -version`: Este comando muestra la **versión del entorno de ejecución de Java (JRE)** que está instalada en tu sistema.

Ejemplo de salida:

```
scss                                                                    Copiar  Editar

java version "17.0.9" 2024-04-16 LTS
Java(TM) SE Runtime Environment (build 17.0.9+11-LTS)
Java HotSpot(TM) 64-Bit Server VM (build 17.0.9+11-LTS, mixed mode, sharing)
```

`javac -version`: Este comando muestra la **versión del compilador de Java (javac)**, que es parte del JDK (Java Development Kit).

Ejemplo de salida:

```
nginx                                                                    Copiar  Editar

javac 17.0.9
```

2. Configurar un entorno de desarrollo

Se puede usar un IDE como **Eclipse o NetBeans**

- O simplemente un editor de texto (como VS Code o incluso Notepad++) con compilación manual por terminal.

3. Crear un nuevo proyecto

Si se usa un IDE (por ejemplo, NetBeans), seleccionar:

- **Nuevo Proyecto Java**

- Configurar la estructura: paquete, clase principal, etc.

Crear un archivo, por ejemplo: Main.java

4. Escribir el código Java

```
public class Main {  
    public static void main(String[] args) {  
        System.out.println("¡Hola, mundo!");  
    }  
}
```

5. Compilar el código

Si se está usando la terminal:

bash

CopiarEditar

javac Main.java

Esto generará un archivo Main.class.

De lo contrario seleccionar del menú del IDE la opción Compilar

6. Ejecutar la aplicación

java Main

Se despliega en pantalla:

¡Hola, mundo!

3.2 Manejadores de bases de datos

Un manejador de base de datos, también conocido como Sistema Gestor de Base de Datos (SGBD) o Database Management System (DBMS), es un software que permite crear, administrar, consultar y proteger la información almacenada en una base de datos. Actúa como una interfaz entre el usuario y la base de datos, facilitando la interacción con los datos almacenados y garantizando su integridad, seguridad y eficiencia. Su objetivo es facilitar el almacenamiento, recuperación, modificación y administración eficiente de los datos, asegurando integridad, seguridad y acceso concurrente.

Un manejador de base de datos (DBMS):

- Define cómo se almacenan los datos (estructura, tablas, campos, relaciones).
- Permite realizar operaciones como crear, leer, actualizar y eliminar datos (CRUD).
- Controla el acceso de usuarios y protege la integridad de la información. Controla quién puede acceder a la base de datos y qué operaciones puede realizar.
- Ofrece herramientas de administración: Proporciona interfaces y herramientas para facilitar la gestión de la base de datos, como la generación de informes, copias de seguridad, etc.
- Optimiza el rendimiento de consultas, transacciones y recuperación ante fallos.

Tipos de bases de datos

- Los relacionales (SQL). Organizan los datos en tablas con relaciones entre ellas
- Los no relacionales (NoSQL). Usan estructuras más flexibles como documentos, grafos o pares clave-valor.

La tabla 3.2 muestra un listado de manejadores de base de datos (DBMS) más utilizados con sus principales características.

Tabla 3. 2 Lista de DBMS con principales características

DBMS	Tipo	Características destacadas
MySQL	Relacional	Código abierto, rápido, ideal para web
PostgreSQL	Relacional	Muy robusto, extensible, ideal para proyectos complejos
SQLite	Relacional	Ligero, sin servidor, perfecto para apps móviles
SQL Server	Relacional	De Microsoft, muy usado en entornos empresariales
Oracle	Relacional	Potente, escalable, usado en grandes corporativos
MongoDB	NoSQL	Basado en documentos JSON, flexible y escalable
Redis	NoSQL	Clave-valor, ultra rápido, ideal para caché y sesiones
MariaDB	Relacional	Derivado de MySQL, con mejoras de rendimiento
Elasticsearch	NoSQL	Optimizado para búsquedas y análisis de texto
Cassandra	NoSQL	Distribuido, escalable, para grandes volúmenes de datos

Un ejemplo de la creación de tablas en Mysql para un sistema de ventas con sucursales se muestra a continuación en la figura 3.2:

```
CREATE TABLE ventas (  
  id INT AUTO_INCREMENT PRIMARY KEY,  
  sucursal_id INT,  
  empleado_id INT,  
  fecha DATETIME,  
  total DECIMAL(10,2),  
  FOREIGN KEY (sucursal_id) REFERENCES sucursales(id),  
  FOREIGN KEY (empleado_id) REFERENCES empleados(id)  
);
```

Figura 3. 2 Creación de Tablas para sistema de Ventas

3.3 Construcción del sistema

La construcción de software es el proceso de transformar el diseño de un sistema de información en código ejecutable, mediante la codificación, pruebas y depuración.

Implica la creación detallada de componentes de software y su integración en un producto funcional, asegurando que cumpla con los requisitos y especificaciones definidas en las etapas de análisis y diseño.

El proceso de construcción de software incluye un conjunto de tareas de codificación y pruebas que debe llevar el software listo para entregarse al cliente o usuario final (Pressman, 2010).

Dentro de las actividades de la ingeniería de software se involucra la calidad de la codificación ya que se pueden utilizar distintos lenguajes de programación dependiendo del tipo de sistema, la plataforma y las necesidades del proyecto.

Con base en estos antecedentes, la codificación puede ser:

- 1) la creación directa en un lenguaje de programación en código fuente (por ejemplo, Python o Java)
- 2) la generación automática de código fuente que usa una representación intermedia parecida al diseño del componente que se va a construir o
- 3) la generación automática de código ejecutable que utiliza un “lenguaje de programación de cuarta generación” (por ejemplo, Python, Visual C++).

Las pruebas dirigen su atención inicial al componente, y con frecuencia se denomina *prueba unitaria*. En el capítulo 4 se profundiza los tipos de prueba a los que se somete un software para ser liberado. A continuación, se mencionan algunos niveles de pruebas:

- 1) Pruebas *de integración* (realizadas mientras el sistema está en construcción)
- 2) Pruebas *de validación*, que evalúan si los requerimientos se han satisfecho para todo el sistema (o incremento de software) y
- 3) Pruebas *de aceptación*, que efectúa el cliente en un esfuerzo por utilizar todas las características y funciones requeridas.

Los siguientes principios y conceptos son aplicables a la codificación y prueba:

Principios de codificación. Los principios que guían el trabajo de codificación se relacionan de cerca con el estilo, lenguajes y métodos de programación.

Sin embargo, puede enunciarse cierto número de principios fundamentales:

Principios de preparación: *Antes de escribir una sola línea de código, asegúrese de:*

- Entender el problema que se trata de resolver.
- Comprender los principios y conceptos básicos del diseño.
- Elegir un lenguaje de programación que satisfaga las necesidades del software que se va a elaborar y el ambiente en el que operará.
- Seleccionar un ambiente de programación que disponga de herramientas que hagan más fácil su trabajo.

- Crear un conjunto de pruebas unitarias que se aplicarán una vez que se haya terminado el componente a codificar.

Principios de programación: *Cuando comience a escribir código, asegúrese de:*

- Restringir sus algoritmos por medio del uso de programación estructurada.
- Tomar en consideración el uso de programación por parejas.
- Seleccionar estructuras de datos que satisfagan las necesidades del diseño.
- Entender la arquitectura del software y crear interfaces que son congruentes con ella.
- Mantener la lógica condicional tan sencilla como sea posible.
- Crear lazos anidados en forma tal que se puedan probar con facilidad.
- Seleccionar nombres significativos para las variables y seguir otros estándares locales de codificación.
- Escribir código que se documente a sí mismo.
- Crear una imagen visual (por ejemplo, líneas con sangría y en blanco) que ayude a entender.

Principios de validación: *Una vez que haya terminado su primer intento de codificación, asegúrese de:*

- Realizar el recorrido del código cuando sea apropiado.
- Llevar a cabo pruebas unitarias y corregir los errores que se detecten.
- Rediseñar el código.

Tipos comunes de software

- **Aplicaciones móviles** como WhatsApp o Uber
- **Software empresarial** como sistemas ERP o CRMs
- **Sistemas web** como tiendas en línea o plataformas educativas
- **Juegos** digitales y simuladores

3. El personal para el desarrollo de software aplica en diferentes roles, entre los que se mencionan:

- **Analistas** de sistemas
- **Desarrolladores** o programadores
- **Diseñadores UX/UI**
- **Testers** o QA
- **Project managers**

Ejemplo:

Desarrollar un **sistema de cadena de tortillerías usando MySQL y PHP.**

A continuación, se muestran los pasos para el desarrollo es un sistema web que permita gestionar sucursales, ventas, empleados e inventarios.

1. Planificación y análisis del sistema: Antes de programar, se debe definir:

Módulos del sistema:

- Sucursales
- Empleados
- Productos (maíz, tortillas, insumos)
- Inventario
- Ventas
- Reportes

Roles de usuario

- Administrador general
- Gerente de sucursal
- Empleado / vendedor

Estructura del Proyecto

```
tortilleria/  
|  
├── conexion.php  
├── login.php  
├── dashboard.php  
├── registrar_venta.php  
├── inventario.php  
├── reportes.php  
├── logout.php  
├── css/  
|   └── estilos.css  
└── sql/  
    └── tortilleria.sql
```

2. Diseño de la base de datos (MySQL)

```
CREATE TABLE sucursales (  
  id INT AUTO_INCREMENT PRIMARY KEY,  
  nombre VARCHAR(100),
```

direccion TEXT

);

CREATE TABLE empleados (

id INT AUTO_INCREMENT PRIMARY KEY,

nombre VARCHAR(100),

usuario VARCHAR(50) UNIQUE,

contrasena VARCHAR(255),

rol ENUM('admin', 'gerente', 'empleado'),

sucursal_id INT,

FOREIGN KEY (sucursal_id) REFERENCES sucursales(id)

);

CREATE TABLE productos (

id INT AUTO_INCREMENT PRIMARY KEY,

nombre VARCHAR(100),

precio DECIMAL(10,2),

tipo ENUM('insumo', 'producto')

);

CREATE TABLE inventario (

id INT AUTO_INCREMENT PRIMARY KEY,

sucursal_id INT,

producto_id INT,

cantidad DECIMAL(10,2),

FOREIGN KEY (sucursal_id) REFERENCES sucursales(id),

FOREIGN KEY (producto_id) REFERENCES productos(id)

);

CREATE TABLE ventas (

id INT AUTO_INCREMENT PRIMARY KEY,

sucursal_id INT,

empleado_id INT,

fecha DATETIME,

total DECIMAL(10,2),

FOREIGN KEY (sucursal_id) REFERENCES sucursales(id),

```

    FOREIGN KEY (empleado_id) REFERENCES empleados(id)
);
CREATE TABLE detalle_ventas (
    id INT AUTO_INCREMENT PRIMARY KEY,
    venta_id INT,
    producto_id INT,
    cantidad DECIMAL(10,2),
    precio_unitario DECIMAL(10,2),
    FOREIGN KEY (venta_id) REFERENCES ventas(id),
    FOREIGN KEY (producto_id) REFERENCES productos(id)
);

```

3. Conexión a MySQL desde PHP

```

<?php
$host = "localhost";
$usuario = "root";
$contrasena = "";
$bd = "tortilleria";
$conn = new mysqli($host, $usuario, $contrasena, $bd);
if ($conn->connect_error) {
    die("Error de conexión: " . $conn->connect_error);
}
?>

```

4. Login de usuarios

Crea un archivo login.php para validar credenciales:

```

<?php
include 'conexion.php';
$usuario = $_POST['usuario'];
$contrasena = $_POST['contrasena'];
$sql = "SELECT * FROM empleados WHERE usuario = '$usuario'";
$resultado = $conn->query($sql);

```

```

if ($resultado->num_rows > 0) {
    $fila = $resultado->fetch_assoc();
    if (password_verify($contrasena, $fila['contrasena'])) {
        session_start();
        $_SESSION['usuario'] = $usuario;
        $_SESSION['rol'] = $fila['rol'];
        $_SESSION['sucursal_id'] = $fila['sucursal_id'];
        header("Location: dashboard.php");
    } else {
        echo "Contraseña incorrecta";
    }
} else {
    echo "Usuario no encontrado";
}
?>

```

5. Interfaces en PHP + HTML

dashboard.php → Menú según el rol.

gestion_ventas.php → Formulario para registrar ventas.

inventario.php → Mostrar y editar inventario.

reportes.php → Visualizar ventas por sucursal y fecha.

6. Registro de ventas (backend)

```
<?php
```

```
include 'conexion.php';
```

```
$sucursal_id = $_SESSION['sucursal_id'];
```

```
$empleado_id = $_SESSION['usuario_id'];
```

```
$fecha = date("Y-m-d H:i:s");
```

```
$total = 0;
```

```
// Ejemplo de productos recibidos del formulario
```

```
$productos = $_POST['productos']; // Array asociativo: id => cantidad
```

```
$conn->begin_transaction();
```

```
try {
```

```
    $sql = "INSERT INTO ventas (sucursal_id, empleado_id, fecha, total) VALUES (?, ?, ?, 0)";
```

```
    $stmt = $conn->prepare($sql);
```

```
    $stmt->bind_param("iis", $sucursal_id, $empleado_id, $fecha);
```

```
    $stmt->execute();
```

```
    $venta_id = $stmt->insert_id;
```

```
    foreach ($productos as $id => $cantidad) {
```

```
        $sql = "SELECT precio FROM productos WHERE id = ?";
```

```
        $stmt = $conn->prepare($sql);
```

```
        $stmt->bind_param("i", $id);
```

```
        $stmt->execute();
```

```
        $stmt->bind_result($precio);
```

```
        $stmt->fetch();
```

```
        $stmt->close();
```

```
        $subtotal = $precio * $cantidad;
```

```
        $total += $subtotal;
```

```
    $sql = "INSERT INTO detalle_ventas (venta_id, producto_id, cantidad, precio_unitario)
        VALUES (?, ?, ?, ?)";
```

```
    $stmt = $conn->prepare($sql);
```

```
    $stmt->bind_param("iidd", $venta_id, $id, $cantidad, $precio);
```

```
    $stmt->execute();
```

```
}
```

```
$sql = "UPDATE ventas SET total = ? WHERE id = ?";
```

```

$stmt = $conn->prepare($sql);
$stmt->bind_param("di", $total, $venta_id);
$stmt->execute();

$conn->commit();
echo "Venta registrada correctamente.";
} catch (Exception $e) {
    $conn->rollback();
    echo "Error en la venta: " . $e->getMessage();
}
?>

```

7. Generar reportes

```

SELECT sucursales.nombre, SUM(ventas.total) AS total_vendido
FROM ventas
JOIN sucursales ON ventas.sucursal_id = sucursales.id
WHERE fecha BETWEEN '2025-03-01' AND '2025-03-15'
GROUP BY sucursales.id;

```

ACTIVIDADES TEMA 3

Para los siguientes ejercicios, deberá realizar el análisis de requisitos para obtener el SRS y elaborar el diseño del sistema de información.

Entregar la documentación del sistema que incluye:

- El documento de especificación de requerimientos (SRS)
- Estructura del proyecto
- Módulos del sistema (Codificar los procesos identificados en el sistema)
- Roles de usuario
- Estructura de la B.D.
- Elaborar la documentación técnica de la construcción del sistema

Ejercicio 1:

El gimnasio “El Músculo” lo ha contratado para que desarrolle el sistema de información para permitir a los administradores tener un control detallado acerca de los usuarios del gimnasio. Proporcionar a los usuarios del gimnasio, herramientas que ayuden a la gestión y mejora de sus rutinas como atletas y módulos para la mejora de su experiencia antes y después de visitar el establecimiento. Dar a los usuarios del gimnasio una plataforma en la cual puedan enterarse de eventos, anuncios, promociones y actividades en general que ocurran en el establecimiento.

Ejercicio 2:

Desarrollar una aplicación móvil para mejorar la seguridad vial en el municipio de Naucalpan, Estado de México, facilitando la participación ciudadana en la denuncia de infracciones de tránsito. La aplicación permitirá a los ciudadanos reportar infracciones de manera sencilla y rápida, proporcionando a las autoridades la información necesaria para identificar y sancionar a los infractores, y contribuir así a la reducción de accidentes y a la mejora de la calidad de vida en la comunidad.

TEMA 4. PRUEBAS E IMPLANTACIÓN

Introducción

Durante el proceso de desarrollo de software, se realizan pruebas para descubrir errores que se cometieron en las etapas de diseño y construcción.

Una actividad eficiente para garantizar la calidad del software, es realizar una estrategia de prueba, esto es un documento formal que incluye un plan para probar el software completo y lo realiza el gerente de proyecto involucrando tanto a desarrolladores como a los administradores y clientes que han definido los requerimientos.

Según Pressman (Pressman, R, 2010) *“es importante la realización de pruebas, esto requiere más esfuerzo que cualquiera otra acción de ingeniería del software. Si se realiza sin orden, se desperdicia tiempo, se emplea esfuerzo innecesario y, todavía peor, es posible que algunos errores pasen desapercibidos. Por tanto, parecería razonable establecer una estrategia sistemática para probar el software”*.

Todas las pruebas que se realizan comienzan “por lo pequeño” y avanzan “hacia lo grande”, esto significa que las primeras etapas de prueba se enfocan sobre un solo componente (módulo) o un pequeño grupo de componentes relacionados y se aplican pruebas para descubrir errores en los datos y en la lógica de procesamiento que se encapsularon en los componentes.

Posteriormente todas las pruebas, deben integrarse hasta que se construya el sistema completo. En este punto, se ejecuta una serie de pruebas de orden superior para descubrir errores en la satisfacción de los requerimientos del cliente.

Los errores deben diagnosticarse y corregirse en la medida en que se van descubriendo, a este proceso se le llama depuración.

Una *Especificación de pruebas* documenta la forma en la que el equipo de software prepara la prueba al definir un plan que describe una estrategia global y un procedimiento con pasos de prueba específicos y los tipos de pruebas que se realizarán.

Para asegurarse que el plan de pruebas se hizo bien, es necesario revisar la *Especificación de pruebas* antes de realizar las pruebas, es posible valorar si están completos los casos de prueba y las tareas de la misma.

Un plan de prueba y procedimientos efectivos conducirán a la construcción ordenada del software y al descubrimiento de errores en cada etapa del proceso de construcción.

En un proyecto de desarrollo de software pueden aparecer errores en cualquiera de las etapas del ciclo de vida, algunos de ellos incluso permanecen sin ser descubiertos, de ahí la importancia de las pruebas en desarrollo de software.

Hay una gran probabilidad de que el código final tenga errores tanto de requerimientos, como de diseño o de funcionalidad. Para identificar estos problemas antes de que ocurran en un entorno crítico, es necesario realizar pruebas de software, una parte muy importante del proceso, pero también muy costosa; sin embargo, debemos tener en cuenta que el costo debido a un fallo mientras está el software en funcionamiento puede llegar a ser mucho mayor.

Las pruebas de software permiten la ejecución de un sistema con el objetivo principal de encontrar errores presentes en el mismo; así conseguimos reducirlos y mejorar la calidad del sistema, haciéndolo más robusto frente a fallos.

Las pruebas se pueden dividir en diferentes niveles, desde módulos individuales hasta las pruebas de todo el sistema en conjunto.

Objetivos de las pruebas:

- Detectar un error específico
- Descubrir errores no detectados antes
- Tener un buen caso de prueba

Pressman (Pressman, R, 2010) muestra la estrategia para probar el software también puede verse en el contexto de la espiral (Figura 4.1). La *prueba de unidad* comienza en el vértice de la espiral y se concentra en cada unidad (por ejemplo, componente, clase o un objeto de contenido de una *webapp*) del software como se implementó en el código fuente. La prueba avanza al moverse hacia afuera a lo largo de la espiral, hacia la *prueba de integración*, donde el enfoque se centra en el diseño y la construcción de la arquitectura del software. Al dar otra vuelta hacia afuera de la espiral, se encuentra la *prueba de validación*, donde los requerimientos establecidos como parte de su modelado se validan confrontándose con el software que se construyó. Finalmente, se llega a la *prueba del sistema*, donde el software y otros elementos del sistema se prueban como un todo. Para probar el software de cómputo, se avanza en espiral hacia afuera en dirección de las manecillas del reloj a lo largo de líneas que ensanchan el alcance de las pruebas con cada vuelta.

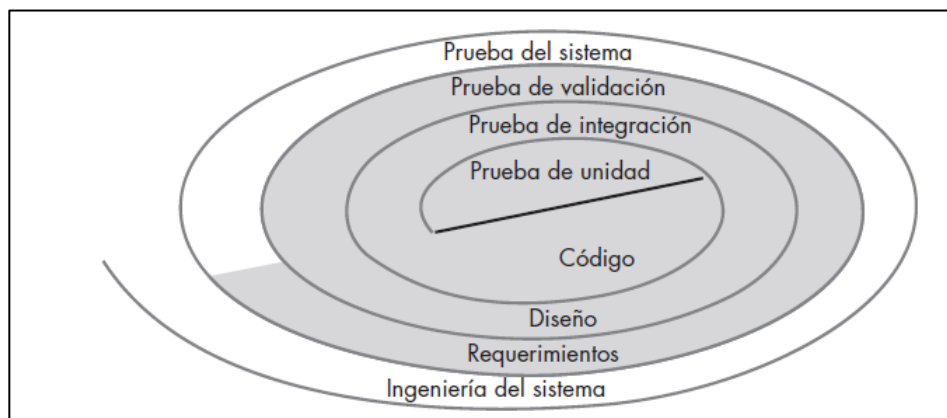


Figura 4. 1 Estrategia de pruebas (Pressman, R, 2010)

Atributos que debe tener una buena prueba:

- Intentar obtener la más alta probabilidad de encontrar un error.
- No debe ser redundante
- No debe ser demasiado sencilla ni demasiado compleja.

La clasificación de las pruebas según la forma en que se ejecutan, son manuales y automáticas.

4.1 Diseño de caso de prueba

El diseño de casos de prueba es el proceso de crear una serie de pasos, escenarios y condiciones para garantizar que un software cumpla con los requisitos y funcione correctamente. La efectividad de estas pruebas depende en gran medida de la calidad de los casos de prueba diseñados. Ayuda a garantizar la calidad y fiabilidad del software al identificar defectos y probar diferentes escenarios de uso. Al utilizar técnicas de diseño adecuadas, se pueden crear casos de prueba efectivos que cubran una amplia gama de situaciones y reduzcan el riesgo de fallos en producción.

La figura 4.2 muestra las actividades necesarias implicadas para la identificación de defectos y asegurar la calidad del software: entender el contexto esto significa conocer bien la aplicación que se está probando, definición de entradas para identificar defectos, aplicar acciones y validar los resultados esperados para cada prueba.

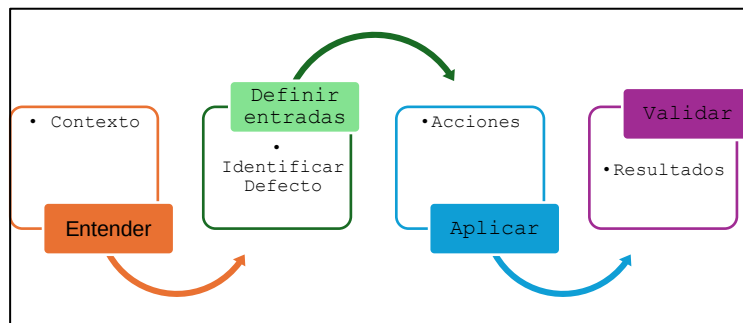


Figura 4. 2 Actividades para diseño de caso de prueba

Un diseño de caso de prueba bien elaborado ayuda a **identificar defectos**, (errores y fallos) en el software antes de que llegue a los usuarios finales, garantizando que el software cumple con los requisitos y funciona como se espera en diversas situaciones. Un caso de prueba bien diseñado mejora la cobertura de pruebas, facilita la automatización y ayuda a reducir los costos de mantenimiento del software.

En la etapa de producción, minimiza los riesgos de fallos al probar exhaustivamente el software. Es necesario prestar mucha atención a los detalles y conocer bien la aplicación que se está probando.

Los casos de prueba documentados facilitan la comprensión y ejecución por parte de otros miembros del equipo, al definir claramente los pasos y resultados esperados, y sirven como referencia para futuras pruebas.

Técnicas comunes de diseño de casos de prueba:

- **Partición de equivalencia:** Divide el dominio de entrada en clases de equivalencia, donde cada clase se espera que produzca el mismo comportamiento.
- **Análisis de valor límite:** Se centra en los valores límite de entrada, incluyendo valores justo por encima y por debajo de los límites, para probar la lógica del software.
- **Tabla de decisiones:** Se utiliza para probar combinaciones de entradas y salidas, especialmente cuando hay muchas condiciones.
- **Gráficos de transición de estados:** Útil para probar sistemas con estados definidos, como interfaces de usuario.
- **Pruebas exploratorias:** Se basa en el conocimiento y la experiencia de la persona responsable para explorar el software y encontrar posibles problemas.

Tabla 4. 1 Ejemplo de un Caso de Prueba

ID del caso de prueba	Descripción del caso de prueba	Condiciones	Gravedad	Entorno	Pasos de prueba	Datos de prueba	Resultado Esperado	Resultado Actual	Estado
1	Escenario de prueba: Verificar el inicio de sesión de Gmail Casos de prueba: Introduzca el nombre de usuario y la contraseña válidos	El usuario tiene una cuenta de Gmail válida	Critico	Phone 8 Plus + iOS 14.4	1. Ingresa el nombre de usuario correcto. 2. Ingresa la contraseña correcta. 3. Pulse el botón de inicio de sesión.	Nombre de usuario: user@itt.com Contraseña: prueba	Inicio de sesión exitoso	"Error. Inicio de sesión fallido".	Falla

Buenas prácticas para diseñar casos de prueba efectivos

- **Definir claramente los requisitos.** Los casos de prueba deben basarse en especificaciones funcionales bien documentadas. Si los requisitos no son claros, es difícil diseñar pruebas efectivas y completas.
- **Mantener la simplicidad y claridad.** Cada caso de prueba debe enfocarse en validar una sola funcionalidad o escenario. Evitar pruebas excesivamente complejas reduce errores en la ejecución y facilita su automatización.
- **Usar técnicas de diseño de pruebas.** Para optimizar la cobertura y eficiencia de las pruebas, se pueden aplicar metodologías como:
 - **Equivalence Partitioning:** Divide los datos de entrada en grupos para reducir la cantidad de pruebas sin perder efectividad.
 - **Boundary Value Analysis:** Evalúa valores en los límites de las particiones para identificar errores en validaciones de rangos.
 - **Decision Table Testing:** Ayuda a probar múltiples combinaciones de entrada y sus resultados esperados.
- **Cubrir casos positivos y negativos.** No solo es importante validar que el software funcione correctamente con datos esperados, sino también evaluar cómo maneja entradas erróneas o condiciones inesperadas.
- **Priorizar casos de prueba según riesgo y criticidad.** Dado que no siempre es posible probar todas las combinaciones posibles, los casos de prueba deben priorizarse según su impacto en el negocio y la probabilidad de fallos.
- **Automatizar pruebas repetitivas.** Casos de prueba que deben ejecutarse con frecuencia o en múltiples configuraciones pueden automatizarse con herramientas como Selenium, Cypress o TestNG, mejorando la eficiencia del proceso.

Elementos para considerar al escribir casos de prueba

1. Identificar la característica que se va a probar. Determine qué características de su software necesitan ser probadas. Por ejemplo, si está probando la funcionalidad de búsqueda de su sitio web, identifique la función de búsqueda como un área de enfoque para la prueba. La definición clara de la característica garantiza que los casos de prueba se dirijan al aspecto correcto del software

2. Identificar los escenarios de prueba. Describa los escenarios que se pueden probar para verificar todos los aspectos de la característica. Por ejemplo, al probar la función de inicio de sesión de su sitio web, los posibles escenarios de prueba incluyen:

- Iniciar sesión con credenciales válidas o no válidas
- Intentar iniciar sesión con una cuenta bloqueada
- Uso de credenciales caducadas

Asegúrese de identificar los resultados esperados para los escenarios positivos y negativos. Por ejemplo, un inicio de sesión exitoso con credenciales válidas debería redirigir al usuario a la página de su cuenta, mientras que las credenciales no válidas deberían desencadenar un mensaje de error y denegar el acceso.

3. Identificar los datos de prueba. Especifique los datos necesarios para ejecutar y evaluar cada escenario de prueba. Por ejemplo, en un escenario de prueba de inicio de sesión con credenciales no válidas, los datos de prueba pueden incluir una combinación incorrecta de nombre de usuario y contraseña. Los datos de prueba adecuados garantizan que los escenarios sean realistas y replicables.

La tabla 4.2 muestra un ejemplo de prueba unitaria para el ingreso de valor de registro de entrada.

4. Identificar el enfoque de prueba. Después de que se ha definido la característica, los escenarios y los datos, desarrolle un enfoque estratégico para las pruebas. Considere lo siguiente:

- En el caso de las nuevas características, es posible que tenga que escribir pasos de prueba detallados para confirmar la funcionalidad específica. Por ejemplo, al probar un nuevo proceso de pago en un sitio de comercio electrónico, incluya pasos como agregar artículos al carrito, ingresar la información de envío y facturación y confirmar el pedido.
- Si sus pruebas son exploratorias o se centran en la aceptación del usuario, una carta de prueba o una declaración de misión pueden ser suficientes. Por ejemplo, durante las pruebas de aceptación del usuario de una nueva función de la aplicación móvil, su misión podría ser verificar que el proceso de inicio de sesión sea fácil de usar y satisfaga las necesidades del usuario final.

Tabla 4.2 Ejemplo de Formato de Prueba unitaria de un sistema de inventarios

ID de prueba	Descripción	Funcionalidad	Datos de entrada	Validación	Resultado esperado	Resultado obtenido	Estado	Comentarios
INV001	Validar inicio de sesión	Login	usuario: admin, contraseña: 1234	Debe ser de 8 caracteres y contener mayúsculas, minúsculas y números	Inicio exitoso	Contraseña insegura	No Aprobado	Faltan mayúsculas y minúsculas
INV002	Validar ENTRADA de datos en Kgs	Peso de producto	Peso =13	Peso > 20	Peso correcto	Peso incorrecto	No Aprobado	Dato incorrecto
INV003	Validar ENTRADA de datos en Kgs	Peso de producto	Peso =87	Peso > 20	Peso correcto	Peso correcto	Aprobado	Dato correcto

Errores comunes al diseñar casos de prueba

- **No actualizar los casos de prueba** cuando cambian los requisitos, lo que puede generar pruebas desactualizadas o irrelevantes.
- **Ignorar la reutilización de pruebas**, generando redundancia y aumentando la carga de mantenimiento.

- **Falta de cobertura en escenarios de error**, lo que puede provocar fallos en producción por no considerar comportamientos inesperados.
- **No documentar adecuadamente** los casos de prueba, dificultando la comprensión y ejecución por parte de otros testers o equipos de desarrollo.

Impacto de los casos de prueba en la calidad del software

El diseño de **casos de prueba efectivos** no solo mejora la detección de defectos, sino que también optimiza el tiempo y los recursos dedicados a la validación del software. Un enfoque estructurado en la creación de pruebas garantiza una cobertura más amplia y reduce el riesgo de fallos en producción, lo que se traduce en una mejor experiencia para el usuario final.

4.2 Pruebas de componentes

Las pruebas de componentes, también conocidas como pruebas de módulos o pruebas unitarias, son una técnica de prueba de software que se enfoca en verificar la funcionalidad, usabilidad, comportamiento y el rendimiento de cada componente o módulo de forma aislada, antes de integrarlos en un sistema más grande. Estas pruebas ayudan a detectar defectos tempranamente, facilitando su resolución y mejorando la calidad general del software.

En Ingeniería de software, las pruebas de componentes son también llamadas pruebas de unidad, se enfocan en cada componente de manera individual, lo que garantiza que funcionan adecuadamente como unidad.

Para llevar a cabo las pruebas de unidad, se utilizan las técnicas de prueba que ejercitan rutas específicas en una estructura de control de componentes para asegurar una cobertura completa y la máxima detección de errores.

Las pruebas de unidad se enfocan en la lógica de procesamiento interno y de las estructuras de datos dentro de las fronteras de un componente. Este tipo de pruebas puede realizarse en paralelo para múltiples componentes.

Pressman Al considerar el proceso de pruebas desde un punto de vista procedural, las pruebas pueden aplicarse en una serie de cuatro pasos que se implementan de manera secuencial. Éstos se muestran en la Figura 4.3 Iniciando con especificar una dirección a la prueba. Posteriormente, se puede considerar el diseño de las pruebas de unidad antes de comenzar la codificación o después de generar el código fuente. La revisión de la información del diseño proporciona una guía para establecer casos de prueba que es probable que se descubran errores en cada una de las categorías analizadas anteriormente. Cada caso de prueba debe acoplarse con un conjunto de resultados esperados.

A continuación, los componentes deben ensamblarse o integrarse para formar el paquete de software completo. La *prueba de integración* aborda los conflictos asociados con los problemas duales de verificación y construcción de programas. Durante la integración, se usan más las técnicas de diseño de casos de prueba que se enfocan en entradas y salidas, aunque también pueden usarse técnicas que ejercitan rutas de programa específicas para asegurar la cobertura de las principales rutas de control.

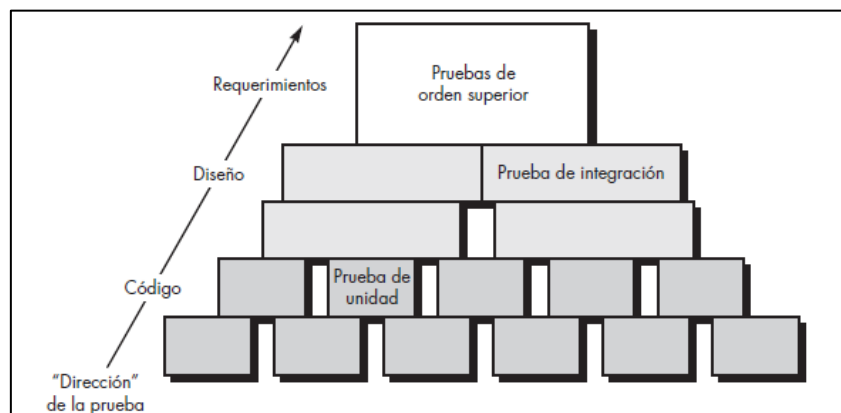


Figura 4. 3 Pasos de la prueba de Software (Pressman, 2010)

Después de integrar (construir) el software, se realiza una serie de *pruebas de orden superior*. Deben evaluarse criterios de validación (establecidos durante el análisis de requerimientos). La *prueba de validación* proporciona la garantía final de que el software cumple con todos los requerimientos informativos, funcionales, de comportamiento y de rendimiento.

El último paso de la prueba de orden superior posterior a la validación es la *prueba del sistema*, que, en conjunto con otros elementos del sistema, tales como el hardware, personal, bases de datos, entre otros, verifica que todos los elementos se mezclen de manera adecuada para lograr el funcionamiento o rendimiento global del sistema.

En resumen, los siguientes son los tipos de pruebas de componentes:

- **Pruebas unitarias:** Se enfocan en probar unidades de código individuales, como funciones o métodos.
- **Pruebas de módulos:** Se enfocan en probar módulos completos, que pueden contener varias unidades de código.
- **Pruebas de API:** Se enfocan en probar las interfaces de programación de aplicaciones, asegurando que funcionen correctamente.

Las pruebas de componentes eficaces dependen de estos principios básicos:

- **Aislamiento :** Los componentes se prueban por separado sin depender de otras partes del sistema.
- **Funcionalidad:** Se verifica que cada componente realice su función específica correctamente, según lo definido en los requisitos
- **Verificación :** Garantiza que un componente devuelva la salida correcta para una entrada determinada
- **Mocking y Stubbing :** Reemplaza dependencias externas con versiones controladas para simular escenarios específicos.
- **Depuración :** facilita el seguimiento y la solución de problemas dentro de un componente específico.
- **Caja cerrada:** En las pruebas de componentes, no se examina el código interno del componente, sino su comportamiento externo.

Las pruebas de componentes son importantes porque permiten la detección temprana de errores identificando defectos en una etapa temprana del desarrollo, lo que facilita su corrección y reduce costos. Facilita la depuración al detectar errores en componentes individuales.

Reduce la complejidad: Permite resolver problemas de forma más sencilla al identificar la causa raíz del error en un componente específico.

En resumen, las pruebas de componentes son una práctica esencial en el desarrollo de software para garantizar la calidad y fiabilidad del producto final, detectando y resolviendo errores de forma temprana y eficiente.

Beneficios de las pruebas de componentes

Las pruebas de componentes respaldan un desarrollo confiable y eficiente a través de:

- **Detección temprana de errores** : los problemas se identifican antes de que se integren los componentes
- **Calidad de código mejorada** : los controles independientes garantizan que los componentes estén bien definidos y funcionen según lo previsto, lo que genera menos defectos y un mejor diseño general.
- **Mejor mantenibilidad** : los cambios en un componente no afectan al resto del sistema.

La prueba de componentes es una técnica de prueba de software que verifica la funcionalidad, usabilidad y comportamiento de los componentes individuales de una aplicación de forma aislada para garantizar que cumplan con los requisitos especificados.

En las pruebas de componentes, cada componente se prueba de forma independiente antes de integrarlo con otros módulos. Este enfoque ayuda a identificar defectos en las primeras etapas del ciclo de desarrollo, ahorrando tiempo y reduciendo el costo de solucionar problemas posteriormente.

Los componentes se prueban en un entorno controlado y aislado mediante herramientas de depuración o entornos de prueba, a menudo con el apoyo de un IDE . Al centrarse en componentes individuales, estas pruebas garantizan su corrección y fiabilidad, sentando una base sólida para las posteriores fases de integración y pruebas del sistema. La tabla muestra un ejemplo de pruebas de componentes

Tabla 4.3 Ejemplo de formato de Pruebas de Componentes de Software

ID de Prueba	Componente	Descripción de la Prueba	Datos de Entrada	Resultado Esperado	Resultado Obtenido	Estado (Aprobado/Rechazado)	Comentarios
TC001	Login	Verificar inicio de sesión con credenciales válidas	usuario: admin, contraseña: 1234	Acceso concedido	(Aquí se registra el resultado)	Aprobado/Rechazado	(Comentarios adicionales)
TC002	Login	Verificar inicio de sesión con credenciales inválidas	usuario: admin, contraseña: 0000	Mensaje de error	(Resultado)	Aprobado/Rechazado	(Comentarios)
TC003	Registro	Verificar registro con datos válidos	nombre: Juan, email: juan@mail.com	Registro exitoso	(Resultado)	Aprobado/Rechazado	(Comentarios)

Algunos puntos clave que muestran la importancia de las pruebas de componentes:

- **Detección temprana de errores** : las pruebas de componentes ayudan a detectar errores en las primeras etapas del proceso de desarrollo, lo que hace que sea más fácil y económico solucionarlos.

- **Pruebas enfocadas:** Probar componentes de forma aislada permite a los desarrolladores concentrarse en partes específicas de la aplicación, lo que facilita la detección de problemas.
- **Verificación de funcionalidad:** Garantiza que cada parte de la aplicación funcione como se espera.
- **Soporte para integración continua:** Se pueden agregar pruebas de componentes automatizadas al flujo de trabajo del equipo de desarrollo, lo que permite realizar pruebas frecuentes y recibir comentarios rápidos.
- **Eficiencia de costos:** La detección temprana de problemas reduce el costo de corregir errores más adelante y minimiza los gastos de mantenimiento posterior al lanzamiento.
- **Calidad de software mejorada:** Las pruebas exhaustivas y periódicas de los componentes individuales garantizan que el software cumpla con estándares de alta calidad.

Tabla 4.4 Ejemplo de Formato de Prueba de Integración

ID de Prueba	Descripción de la Prueba	Módulo(s) Afectado(s)	Datos de Entrada	Resultado o Esperado	Resultado Obtenido	Estado (Aprobado/Rechazado)	Comentarios
TI-001	Verificar integración de login y perfil	Login, Perfil	Usuario válido	Acceso a perfil correcto	(Aquí se llena después de la prueba)	Pendiente / Aprobado / Rechazado	(Comentarios adicionales)
TI-002	Validar envío de formulario y confirmación	Formulario, Confirmación	Datos válidos	Mensaje de éxito	(Resultado)	Pendiente / Aprobado / Rechazado	(Comentarios)

4.3 Pruebas del sistema

Las pruebas del sistema son una fase del desarrollo de software donde se verifica que un sistema completo funcione correctamente como un todo. Se evalúa si el sistema cumple con los requisitos funcionales, de calidad y comerciales, verificando la interacción entre todos sus componentes.

“La *prueba del sistema* es una serie de diferentes pruebas cuyo propósito principal es ejercitar por completo el sistema basado en computadora. Aunque cada prueba tenga un propósito diferente, todo él funciona para verificar que los elementos del sistema se hayan integrado de manera adecuada y que se realicen las funciones asignadas” (Pressman, 2010).

En detalle:

- **Enfoque en el sistema completo:**

A diferencia de las pruebas unitarias o de integración, las pruebas del sistema se enfocan en el comportamiento del sistema en su conjunto, evaluando la funcionalidad de extremo a extremo.

- **Verificación de requisitos:**

Se asegura que el sistema cumpla con los requisitos funcionales (lo que debe hacer) y no funcionales (cómo debe hacerlo, como rendimiento y seguridad).

- **Pruebas de caja negra:**

Generalmente, las pruebas del sistema se realizan utilizando la técnica de caja negra, donde el tester no necesita conocer el código interno, sino que se enfoca en la entrada y salida del sistema.

- **Tipos de pruebas:**

Se pueden realizar pruebas funcionales (verificando las funcionalidades del sistema) y no funcionales (como pruebas de rendimiento, seguridad, usabilidad, etc.).

- **Importancia:**

Las pruebas del sistema son cruciales para identificar errores que puedan surgir de la interacción de diferentes componentes, errores que pueden no ser detectados en pruebas unitarias o de integración.

- **Ciclo de vida del desarrollo:**

Se llevan a cabo después de las pruebas de integración y antes de las pruebas de aceptación del usuario.

Ejemplos de pruebas:

Pruebas de carga, pruebas de estrés, pruebas de seguridad, pruebas de compatibilidad con diferentes navegadores o sistemas operativos.

En resumen, las pruebas del sistema son una etapa esencial para garantizar la calidad y fiabilidad de un software antes de su lanzamiento.

Pruebas de recuperación

Muchos sistemas basados en computadora deben recuperarse de fallas y reanudar el procesamiento con poco o ningún tiempo de inactividad. En algunos casos, un sistema debe ser tolerante a las fallas, es decir, las fallas del procesamiento no deben causar el cese del funcionamiento del sistema global.

En otros casos, la falla de un sistema debe corregirse dentro de un periodo de tiempo específico u ocurrirán severos daños económicos.

La *recuperación* es una prueba del sistema que fuerza al software a fallar en varias formas y que verifica que la recuperación se realice de manera adecuada. Si la recuperación es automática (realizada por el sistema en sí), se evalúa el reinicio, los mecanismos de puntos de verificación, la recuperación de datos y la reanudación para correcciones. Si la recuperación requiere intervención humana, se evalúa el tiempo medio de reparación (TMR) para determinar si está dentro de límites aceptables.

Pruebas de seguridad

Cualquier sistema basado en computadora que gestione información sensible o cause acciones que puedan dañar (o beneficiar) de manera inadecuada a individuos es un blanco de penetración inadecuada o ilegal. La penetración abarca un amplio rango de actividades: *hackers* que intentan penetrar en los sistemas por deporte, empleados resentidos que intentan penetrar por venganza, individuos deshonestos que intentan penetrar para obtener ganancia personal ilícita.

La *prueba de seguridad* intenta verificar que los mecanismos de protección que se construyen en un sistema en realidad lo protegerán de cualquier penetración impropia. Para citar a Beizar [Bei84]: “La seguridad del sistema debe, desde luego, probarse para ser invulnerable ante ataques frontales; pero también debe probarse su invulnerabilidad contra ataques laterales y traseros.”

Durante la prueba de seguridad, quien realiza la prueba juega el papel del individuo que desea penetrar al sistema. ¡Cualquier cosa vale! Quien realice la prueba puede intentar adquirir contraseñas por medios administrativos externos; puede atacar el sistema con software a la medida diseñado para romper cualquier defensa que se haya construido; puede abrumar al sistema, y por tanto negar el servicio a los demás; puede causar a propósito errores del sistema con la esperanza de penetrar durante la recuperación; puede navegar a través de datos inseguros para encontrar la llave de la entrada al sistema.

Con los suficientes tiempos y recursos, las buenas pruebas de seguridad penetran en el sistema. El papel del diseñador de sistemas es hacer que el costo de la penetración sea mayor que el valor de la información que se obtendrá. La tabla 4.5 muestra el ejemplo del formato para este tipo de pruebas.

Tabla 4.5 Ejemplo del formato para Prueba de Seguridad

ID de Prueba	Fecha	Proyecto	Tipo de Prueba	Técnica Usada	Herramienta	Vulnerabilidad Objetivo	Descripción del Escenario	Resultado	Riesgo	Evidencia
SEC001	12/08/2025	Sistema VentasX	Escaneo de vulnerabilidades	Caja negra	OWASP ZAP	Inyección SQL	Probar entradas de login con payloads SQL	Fallido	Alto	evid_sec001.png
SEC002	12/08/2025	Sistema VentasX	Prueba de penetración	Caja gris	Burp Suite	XSS reflejado	Inyección de script en campo de búsqueda	Aprobado	Nulo	evidencia_sec002.png

Pruebas de esfuerzo

Los primeros pasos de la prueba del software dieron como resultado una evaluación extensa de las funciones y el rendimiento normales del programa. Las pruebas de esfuerzo se diseñan para enfrentar los programas con situaciones anormales. En esencia, la persona que realiza las pruebas de esfuerzo pregunta: “¿cuánto podemos doblar esto antes de que se rompa?”.

La *prueba de esfuerzo* ejecuta un sistema en forma que demanda recursos en cantidad, frecuencia o volumen anormales.

Por ejemplo, pueden 1) diseñarse pruebas especiales que generen diez interrupciones por segundo, cuando una o dos es la tasa promedio,

2) aumentarse las tasas de entrada de datos en un orden de magnitud para determinar cómo responderán las funciones de entrada,

3) ejecutarse casos de prueba que requieran memoria máxima y otros recursos,

4) diseñarse casos de prueba que puedan causar *thrashing* (que es un quebranto del sistema por hiperpaginación) en un sistema operativo virtual,

5) crearse casos de prueba que puedan causar búsqueda excesiva por datos residentes en disco. En esencia, la persona que realiza la prueba intenta romper el programa.

Una variación de la prueba de esfuerzo es una técnica llamada *prueba de sensibilidad*.

En algunas situaciones (la más común ocurre en algoritmos matemáticos), un rango muy pequeño de datos contenidos dentro de las fronteras de los datos válidos para un programa puede causar procesamiento extremo, e incluso erróneo, o profunda degradación del rendimiento. La prueba de sensibilidad intenta descubrir combinaciones de datos dentro de clases de entrada válidas que puedan causar inestabilidad o procesamiento inadecuado.

La tabla 4.6 Muestra un ejemplo de una prueba de esfuerzo conocida también como prueba de estrés

Pruebas de rendimiento

Para sistemas en tiempo real y sistemas embebidos, el software que proporcione la función requerida, pero que no se adecue a los requerimientos de rendimiento, es inaceptable. La prueba de rendimiento se diseña para poner a prueba el rendimiento del software en tiempo de corrida, dentro del contexto de un sistema integrado. La prueba del rendimiento ocurre a lo largo de todos los pasos del proceso de prueba. Incluso en el nivel de unidad, puede accederse al rendimiento de un módulo individual conforme se realizan las pruebas. Sin embargo, no es sino hasta que todos los elementos del sistema están plenamente integrados cuando puede determinarse el verdadero rendimiento de un sistema. Las pruebas de rendimiento con frecuencia se aparean con las pruebas de esfuerzo y por lo general requieren instrumentación de hardware y de software, es decir, con frecuencia es necesario medir la utilización de los recursos (por ejemplo, ciclos del procesador) en forma meticulosa. La instrumentación externa puede monitorear intervalos de ejecución y eventos de registro (por ejemplo, interrupciones) conforme ocurren, y los muestreos del estado de la máquina de manera regular. Con la instrumentación de un sistema, la persona que realiza la prueba puede descubrir situaciones que conduzcan a degradación y posibles fallas del sistema.

Pruebas de despliegue

En muchos casos, el software debe ejecutarse en varias plataformas y bajo más de un entorno de sistema operativo. La *prueba de despliegue*, en ocasiones llamada *prueba de configuración*, ejercita el software en cada entorno en el que debe operar. Además, examina todos los procedimientos de instalación y el software de instalación especializado (por ejemplo, “instaladores”) que usarán los clientes, así como toda la documentación que se usará para introducir el software a los usuarios finales.

Como ejemplo, suponga que puede acceder a través de internet al software que permitiría a un cliente monitorear el sistema de seguridad desde ubicaciones remotas.

La aplicación del sistema de seguridad debe probarse usando todos los navegadores web que es probable que se encuentren.

Una prueba de despliegue más profunda puede abarcar combinaciones de navegadores web con varios sistemas operativos (por ejemplo, Linux, Mac OS, Windows). Puesto que la seguridad es un tema principal, un juego completo de pruebas de seguridad se integraría con la prueba de despliegue.

Tabla 4.6 Ejemplo de formato de prueba de esfuerzo (estrés)

ID de Prueba	Fecha	Proyecto	Objetivo de la Prueba	Tipo de Carga	Técnica Usada	Herramienta de Prueba	Usuarios Concurrentes	TPS	Tiempo de Respuesta Prom.	Recursos del Sistema	Fallas Detectadas	Punto de Ruptura	Resultado	Observaciones
ESF001	12/08/2025	InventarioWebApp	Identificar punto de falla con usuarios simultáneos	Estrés	Caja negra	Apache JMeter	1500	350	4.2 segundos	CPU 98%, RAM 95%, Red 85%	Error 503 al superar 1400 usuarios	1450 usuarios	Fallido	Sistema colapsó al llegar a 1450 usuarios
ESF002	12/08/2025	InventarioWebApp	Verificar recuperación después del estrés	Estrés	Caja negra	Apache JMeter	1000	300	2.9 segundos	CPU 88%, RAM 80%, Red 70%	Ninguna	No alcanzado	Aprobado	El sistema respondió correctamente

4.4 Documentación de resultados de las pruebas.

La documentación de resultados de pruebas es el registro detallado de las pruebas realizadas, incluyendo los resultados obtenidos, las observaciones, hallazgos, y conclusiones de las pruebas realizadas además de otros problemas identificados.

Esta documentación es crucial para el éxito de cualquier proyecto de software, garantiza la calidad del software, facilitar la toma de decisiones y mejorar el proceso de pruebas en el futuro.

La documentación de los resultados de las pruebas sirve como una referencia para futuros esfuerzos de prueba, ayuda a identificar problemas y proporciona información valiosa para el equipo de desarrollo y otras partes interesadas. El término *prueba de software* invoca imágenes de gran número de casos de prueba preparados para revisar los programas de cómputo y los datos que manipulan.

Los errores en la documentación pueden ser tan devastadores para la aceptación del programa como los errores en los datos o en el código fuente. Nada es más frustrante que seguir con exactitud una guía de usuario o un centro de ayuda en línea y obtener resultados o comportamientos que no coinciden con los predichos por la documentación. Por esta razón, las pruebas de documentación deben ser parte significativa de todo plan de prueba de software.

La prueba de documentación puede abordarse en dos fases:

- La primera fase, la revisión técnica, examina el documento en su claridad editorial.
- La segunda fase, prueba en vivo, usa la documentación en conjunto con el programa real.

Sorprendentemente, una prueba en vivo para la documentación puede abordarse usando técnicas que son análogas a muchos de los métodos de prueba de caja negra estudiados anteriormente.

La prueba basada en gráfico puede usarse para describir el uso del programa; la partición de equivalencia y el análisis del valor de frontera pueden usarse para definir varias clases de entrada e interacciones asociadas. La PBM puede usarse para garantizar que el comportamiento documentado y el comportamiento real coinciden. Entonces, el uso del programa puede rastrearse a través de la documentación.

Importancia de la documentación de resultados de pruebas:

- **Garantiza la trazabilidad:** Permite rastrear qué se probó, cómo se probó y cuáles fueron los resultados.
- **Facilita la comunicación:** Asegura que todos los miembros del equipo, incluyendo desarrolladores, testers y partes interesadas, estén al tanto de los resultados y puedan tomar decisiones informadas.
- **Identifica problemas:** Ayuda a identificar y documentar defectos, errores y problemas de rendimiento, lo que facilita su corrección.
- **Mejora la calidad:** Al documentar los resultados, se puede analizar la calidad del software y tomar medidas para mejorarla.
- **Apoya la toma de decisiones:** La documentación proporciona información valiosa para tomar decisiones sobre la liberación del software, la priorización de tareas y la mejora del proceso.

- **Sirve como referencia para futuras pruebas:** La documentación puede utilizarse como base para futuras pruebas, evitando repetir errores y mejorando la eficiencia del proceso.
- **Facilita el cumplimiento de estándares:** La documentación adecuada puede ayudar a cumplir con los estándares de calidad y regulaciones específicas de la industria.

Tipos de documentación de resultados de pruebas:

- **Informes de pruebas:**

Resumen los resultados generales de las pruebas, incluyendo el número de casos de prueba ejecutados, el número de errores encontrados y el estado general del software.

- **Registros de defectos:**

Documentan cada defecto o error encontrado, incluyendo su descripción, pasos para reproducirlo, gravedad y prioridad.

- **Informes de pruebas individuales:**

Proporcionan información detallada sobre cada caso de prueba ejecutado, incluyendo los datos de entrada, los pasos de prueba, los resultados esperados y los resultados reales.

- **Gráficos y visualizaciones:**

Pueden utilizarse para presentar los resultados de forma visual, facilitando la comprensión de la información.

Ejemplos de documentación de resultados de pruebas:

- Un informe de pruebas que indique que se ejecutaron 100 casos de prueba, 10 de los cuales fallaron, y que el software está en un estado de "prueba".
- Un registro de defectos que describa un error en la función de inicio de sesión, incluyendo los pasos para reproducirlo, el resultado esperado y el resultado real.
- Un gráfico que muestre la evolución del número de defectos a lo largo del tiempo, permitiendo identificar tendencias y áreas problemáticas.

En resumen, la documentación de resultados de pruebas es un proceso esencial para garantizar la calidad del software y facilitar la toma de decisiones informadas. Una buena documentación ayuda a identificar problemas, mejora la comunicación y permite la mejora continua del proceso de pruebas.

Importancia de la documentación de resultados de pruebas:

- **Transparencia y trazabilidad:**

Permite a todos los involucrados en el proyecto tener visibilidad sobre el proceso de prueba y los resultados obtenidos, facilitando la trazabilidad de los defectos y su resolución.

- **Identificación de problemas:**

La documentación detallada de los resultados ayuda a identificar y rastrear los defectos encontrados, permitiendo a los desarrolladores corregirlos de manera efectiva.

- **Análisis de la calidad:**

La documentación de los resultados permite analizar la calidad del software, identificar áreas de mejora y tomar decisiones informadas sobre el producto.

- Referencia para futuras pruebas:

La documentación sirve como una referencia valiosa para futuros ciclos de prueba, evitando la repetición de esfuerzos y garantizando la consistencia en el proceso de prueba.

- Comunicación efectiva:

La documentación facilita la comunicación entre los diferentes equipos involucrados en el proyecto, asegurando que todos estén al tanto del estado de las pruebas y de los problemas encontrados.

Documentar los resultados de las pruebas le ayuda a realizar un seguimiento del progreso, la calidad y el rendimiento de su software, así como a proporcionar pruebas y comentarios a las partes interesadas y a los desarrolladores.

Los resultados de las pruebas más importantes que hay que documentar y cómo se puede hacer de forma eficaz ayuda a realizar un seguimiento del progreso, la calidad y el rendimiento del software, así como a proporcionar pruebas y comentarios a las partes interesadas y a los desarrolladores.

Plan de pruebas

Un plan de pruebas es un documento que describe el alcance, los objetivos, la estrategia y el cronograma de las actividades de pruebas. Define lo que se va a probar, cómo se va a probar, quién es responsable de las pruebas y cuándo se va a probar. Este documento ayuda a garantizar que sus esfuerzos de prueba estén en línea con los requisitos, expectativas y riesgos del proyecto. Para ser eficaz, un plan de pruebas debe ser claro, conciso y coherente. Debe incluir el alcance de las pruebas, los objetivos de las pruebas, la estrategia para llevarlas a cabo, el cronograma y los hitos de las actividades de prueba, los recursos necesarios para las pruebas, los resultados producidos a partir de las pruebas y cualquier riesgo potencial asociado con las pruebas.

Plan de Pruebas para el Sistema de Gestión de Inventarios

Fecha: 27 de abril de 2024

Autor: Juan Pérez

1. Introducción

Este documento describe el plan de pruebas para validar el correcto funcionamiento del Sistema de Gestión de Inventarios, asegurando que cumple con los requisitos especificados y funciona de manera eficiente.

2. Objetivos de las Pruebas

- Verificar que las funciones principales (agregar, editar, eliminar productos) operan correctamente.
- Validar la generación de reportes de inventario.
- Detectar errores o fallos en la interfaz y en la lógica del sistema.

3. Alcance

Se probarán las funcionalidades relacionadas con la gestión de productos y reportes. No se incluirán pruebas de integración con otros sistemas externos en esta fase.

4. Estrategia de Pruebas

- Pruebas manuales para verificar funcionalidades básicas.
- Uso de casos de prueba predefinidos.
- Validación en entorno de prueba con datos simulados.

5. Criterios de Entrada y Salida

Entrada:

- Sistema instalado y configurado.
- Datos de prueba cargados.

- Accesos y permisos adecuados.

Salida:

- Funcionalidades operando sin errores críticos.
- Cumplimiento de los requisitos especificados.

6. Recursos

- Equipo: Juan Pérez, Ana Gómez.
- Herramientas: Navegador Chrome, Postman, Excel para reportes.
- Entorno: Servidor de pruebas, base de datos de prueba.

7. Cronograma

Actividad	Fecha de inicio	Fecha de finalización	Responsable
Preparación del entorno	1 mayo 2024	2 mayo 2024	Juan Pérez
Ejecución de casos de prueba	3 mayo 2024	10 mayo 2024	Ana Gómez
Revisión y reporte de errores	11 mayo 2024	13 mayo 2024	Juan Pérez

8. Casos de Prueba (Ejemplo)

Caso 1: Agregar un nuevo producto

- **Paso 1:** Ingresar al módulo de inventario.
- **Paso 2:** Seleccionar "Agregar producto".
- **Paso 3:** Completar los campos: nombre, cantidad, precio.
- **Paso 4:** Guardar.
- **Resultado esperado:** El producto aparece en la lista con los datos correctos.

Caso 2: Generar reporte de inventario

- **Paso 1:** Acceder a la sección de reportes.
- **Paso 2:** Seleccionar "Reporte de inventario".
- **Paso 3:** Elegir filtros (fecha, categoría).
- **Paso 4:** Generar reporte.
- **Resultado esperado:** Se muestra un reporte correcto y exportable.

9. Gestión de Defectos

- Los errores se reportarán en un sistema de seguimiento (Jira, por ejemplo).
- Se priorizarán según su impacto (crítico, mayor, menor).

10. Riesgos y Contingencias

- Riesgo: Datos incompletos en el entorno de prueba.
- Contingencia: Preparar datos de prueba antes de comenzar.

11. Aprobaciones

Nombre Firma Fecha

Juan Pérez

Ana Gómez

4.5 Entrega del sistema y Capacitación a usuarios

La entrega del sistema y la capacitación de usuarios son dos fases importantes en la implementación de cualquier sistema, ya sea software, hardware o un proceso.

La entrega implica poner a disposición el sistema a los usuarios, mientras que **la capacitación** asegura que los usuarios puedan utilizarlo de manera efectiva y eficiente.

Entrega del Sistema:

Esta actividad lleva implícito del proceso de implementación del software, el cual requiere una planificación cuidadosa, ejecución eficiente y una gestión adecuada (soporte continuo) para asegurar que el software cumpla con los objetivos al atender las necesidades del cliente.

Los pasos para la entrega del software incluyen entre otros:

- **Planificación:** Definir los objetivos, alcance, cronograma de actividades, recursos humanos, recursos tecnológicos y presupuesto del proyecto de implementación.
- **Instalación y configuración:** Instalar el software en los servidores y estaciones de trabajo (entorno del usuario), y configurar los parámetros (Ajuste del sistema) según las necesidades específicas del usuario o la organización.
- **Pruebas:** Realizar pruebas exhaustivas para asegurar que el software funciona correctamente, incluyendo pruebas unitarias, de integración y de aceptación por parte del usuario.
- **Migración de datos:** Transferir los datos existentes al nuevo sistema o integrar datos de sistemas anteriores.
- **Capacitación:** Entrenar a los usuarios finales sobre cómo utilizar el nuevo software de manera efectiva.
- **Documentación:** Proporcionar manuales, guías y otros materiales para el usuario.
- **Puesta en marcha:** Lanzar el software en producción y monitorear su rendimiento.
- **Soporte y mantenimiento:** Brindar soporte técnico y realizar mantenimiento continuo para garantizar la estabilidad y el buen funcionamiento del software.

La implementación de software es un requisito indispensable para el éxito de cualquier proyecto de software, ya que de ella depende la correcta puesta en marcha y el aprovechamiento de todas sus funcionalidades.

Una implementación deficiente puede generar problemas durante la operación del software, pérdida de tiempo y recursos, y lo más peligroso, afectar negativamente la productividad de la empresa.

Se recomienda llevar a cabo el proceso de implementación aplicando alguna metodología, tales como la metodología en cascada, ágil, o mixtas, cada una con sus propias ventajas y desventajas, pero la elección de la metodología adecuada dependerá de las características del proyecto y las necesidades de la organización.

Capacitación de Usuarios:

La capacitación de usuarios es fundamental para asegurar que los usuarios puedan aprovechar al máximo el sistema. Esto implica:

- Identificación de necesidades:

Determinar qué usuarios necesitan capacitación y qué habilidades deben adquirir.

- Diseño de programas de capacitación:

Crear materiales y actividades de capacitación adaptados a las necesidades identificadas.

- Ejecución de la capacitación:

Ofrecer sesiones de capacitación a los usuarios, utilizando diferentes métodos como capacitación presencial, virtual o e-learning.

- Evaluación de la capacitación:

Medir la efectividad de la capacitación y realizar ajustes si es necesario.

- Soporte continuo:

Brindar apoyo a los usuarios después de la capacitación, a través de canales como mesas de ayuda, foros o documentación actualizada.

Importancia de la capacitación:

La capacitación efectiva de los usuarios es esencial para:

- Aumentar la adopción del sistema:

Los usuarios que se sienten cómodos con el sistema son más propensos a utilizarlo.

- Reducir errores y problemas:

La capacitación adecuada ayuda a los usuarios a evitar errores comunes y a resolver problemas por sí mismos.

- Mejorar la productividad:

Los usuarios capacitados pueden utilizar el sistema de manera más eficiente y realizar sus tareas más rápidamente.

- Aumentar la satisfacción del usuario:

Los usuarios que se sienten competentes y seguros con el sistema son más propensos a estar satisfechos con él.

En resumen, tanto la entrega del sistema como la capacitación de usuarios son procesos críticos que deben planificarse y ejecutarse cuidadosamente para garantizar el éxito de cualquier proyecto de implementación de sistemas

4.6 Entrega de documentación técnica y de usuario del sistema.

La entrega de la documentación técnica y de usuario de un sistema es un proceso fundamental en el desarrollo de software, ya que facilita la comprensión, uso y mantenimiento del sistema a largo plazo.

La documentación técnica está dirigida a desarrolladores e ingenieros, proporcionándoles la información necesaria para entender el funcionamiento interno del sistema, mientras que la documentación de usuario está dirigida a los usuarios finales, ofreciendo instrucciones claras y concisas sobre cómo utilizar el sistema.

El objetivo de la **Documentación Técnica** es explicar el funcionamiento interno del sistema, incluyendo su arquitectura, diseño, código y procesos. Se debe redactar con énfasis en los desarrolladores, ingenieros, personal de mantenimiento y soporte técnico.

Esta documentación facilita el mantenimiento, actualizaciones y mejoras del sistema, así como la colaboración entre los integrantes del equipo de desarrollo de software.

La documentación técnica debe contener:

- Especificaciones de requisitos: Detallan funcionalidades y restricciones del sistema.
- Diseño del sistema: Describe la arquitectura, diagramas y modelos visuales del sistema.
- Documentación del código: Explica el código fuente y su funcionalidad.
- API y guías de integración: Explican cómo otros sistemas pueden interactuar con el sistema.
- Documentación de pruebas: Detalla los casos de prueba y resultados esperados.
- Informes de desarrollo: Registran el avance y problemas encontrados durante el desarrollo.

La Documentación de Usuario tiene como objetivo guiar al usuario final en el uso efectivo del sistema, Mejora la experiencia del usuario, reduce la curva de aprendizaje y minimiza la necesidad de soporte técnico.

Incluye funciones, características y resolución de problemas y su redacción debe ser dirigida a los usuarios finales del sistema.

La documentación de usuario debe contener:

- Manuales de usuario: Guías paso a paso sobre cómo utilizar el sistema.
- Preguntas frecuentes (FAQ): Respuestas a preguntas comunes de los usuarios.
- Tutoriales: Guías visuales o escritas para aprender a usar funciones específicas.
- Glosario: Definiciones de términos técnicos utilizados en el sistema.

En resumen, la documentación técnica y de usuario son herramientas esenciales para el éxito de cualquier sistema. La documentación técnica asegura que el sistema pueda ser mantenido, mejorado y comprendido por los equipos de desarrollo, mientras que la documentación de usuario empodera a los usuarios finales para utilizar el sistema de manera efectiva y satisfactoria.

documentación externa escrita para usuarios finales. Abarca todos los documentos escritos relacionados con el desarrollo de productos de software y se crean muchos tipos diferentes a lo largo del ciclo de vida del desarrollo de software (SDLC).

La documentación externa se refiere a materiales escritos, de audio o visuales que proporcionan información sobre productos o servicios a clientes ajenos a la empresa. Incluye manuales de usuario, guías de instalación, preguntas frecuentes y guías de solución de problemas. Esta información tiene como objetivo proporcionar a los usuarios recursos de autoayuda que les permitan utilizar el producto de forma eficaz y solucionar problemas comunes sin necesidad de ponerse en contacto con el servicio de atención al cliente.

Describe las características y la funcionalidad del producto de forma clara para que cualquiera pueda utilizarlo. El objetivo principal es explicar un aspecto particular del producto al público objetivo. Aunque se presenta en distintos formatos, la mayor parte de la documentación técnica se publica en línea y normalmente la escriben redactores técnicos, desarrolladores y gerentes de proyectos.

La documentación técnica debe ser clara, concisa y precisa y, realmente, resolver un problema para sus usuarios.

requisitos del sistema y consejos para la solución de problemas. Las guías de instalación pueden ser esenciales para productos o servicios de software complejos, ya que una instalación incorrecta puede causar muchos problemas. Las buenas guías de instalación deben ser fáciles de seguir e incluir consejos para la solución de problemas comunes.

Guías de solución de problemas:

Las guías de solución de problemas ayudan a los usuarios a resolver problemas comunes que pueden encontrar al usar el producto. Estos pueden incluir información detallada sobre cómo diagnosticar y solucionar problemas. Las guías de solución de problemas ayudan a reducir la necesidad de atención al cliente y pueden ahorrar tiempo y frustración a los usuarios. Las buenas guías de solución de problemas deben ser completas y fáciles de seguir, con instrucciones claras y capturas de pantalla cuando sea necesario.

Qué hacer y qué no hacer en la documentación técnica

Hacer:

- Manténlo simple y claro: no compliques demasiado tu documentación ni uses un lenguaje complejo.
- Tenga siempre en mente a su usuario: cada vez que escriba documentación, asegúrese de tener claro a quién va dirigida.
- Hazlo visual: si puedes representar lo que intentas transmitir con una imagen, hazlo.
- Realice un proceso de revisión exhaustivo: si puede, asegúrese de que varias personas revisen su trabajo durante la fase de escritura.

No:

- Suponga que su audiencia está familiarizada con el tema: proporcione siempre el mayor contexto posible.
- Utilice la voz pasiva; utilice siempre la voz activa: “Él presionó el botón” en lugar de “él presionó el botón”.
- Utilice acrónimos: si debe utilizar acrónimos, indique claramente el significado del acrónimo junto a ellos.

- Intenta ser gracioso: lo que puede resultar divertido para ti puede resultar insultante u ofensivo para tus lectores.

La frase “Una pluma es más poderosa que una espada” (Edward Bulwer-Lytton, 1839), enfatiza el poder de las palabras. Escribir es una herramienta poderosa para transmitir ideas impactantes y, sobre todo, influir en los lectores con sus pensamientos.

Con base en esta frase, los redactores de contenido tienen la enorme responsabilidad de usar esta herramienta con sabiduría, deben ser imparciales y producir contenido en un tono amigable para atraer a cualquiera que lo lea.

La documentación inclusiva debe tratar a diferentes tipos de personas de forma justa e igualitaria. Todo ser humano es tratado con dignidad y respeto, independientemente de su género, raza, religión, ubicación, etc. Mostrar empatía en el estilo de la documentación libera el potencial de ser justo e igualitario con todos.

La documentación inclusiva promueve la armonía en la sociedad y hace de nuestro planeta un lugar mejor para vivir. La inclusión debe ser un principio rector del marco de la documentación, y los redactores de contenido deben practicar la producción de contenido inclusivo sin sesgos.

Dado que los redactores de contenido desconocen las características de su audiencia, deben elegir el lenguaje, el tono y, sobre todo, las palabras adecuadas para expresar sus ideas. ¡La elección de palabras del redactor de contenido es fundamental! El uso de pronombres neutros fomenta sentimientos positivos hacia las mujeres y las personas LGBTQ+.

Las palabras y expresiones de pensamiento no deben discriminar ni ofender a ningún ser humano.

Cinco pasos para producir documentación inclusiva

1. Utilice términos neutrales en cuanto al género.

Los términos de género neutro ayudan a evitar favorecer un género sobre otros. Evitar el uso de pronombres específicos de género en el contenido contribuye a lograr la neutralidad de género. Por ejemplo, en lugar de usar términos como "lego", "horas-hombre" o "humanidad", elija términos como "persona lego", "horas-persona" o "humanidad".

En lugar de usar términos como "lista negra" y "lista blanca", utilice términos como "bloqueado" y "permitido". Esto elimina los sesgos culturales en la documentación. Además de la redacción del contenido, el redactor de UX debe prestar más atención a los elementos del menú, los textos de derechos de autor en los cuadros de diálogo y los textos emergentes para garantizar que se utilicen términos neutrales en cuanto al género en todo el producto.

Evite usar términos como "arquitectura esclavo-maestro"; en su lugar, use términos como "arquitectura de dos niveles". Es necesario examinar los fragmentos de código, el cuerpo de la solicitud de la API y los ejemplos de respuesta para asegurar el uso de segmentos de código neutrales en cuanto al género.

2. Elija un tono neutral y amigable

Elegir el tono adecuado es una habilidad que los redactores de contenido deben practicar y dominar. El tono, como el lenguaje mismo, funciona porque las personas coinciden en el significado de las palabras. Las normas culturales suelen dictar cómo se perciben. Se requiere conocimiento de dichas normas y expectativas, así como del contexto de la conversación, para poder transmitir las emociones deseadas. Los riesgos suelen ser mayores para corresponder con la importancia del mensaje, por lo

que se justifica una mayor atención. Escribir en un tono neutral ayuda a no inducir sentimientos positivos ni negativos sobre el contenido. Algunas recomendaciones son:

- Utilizar la elección correcta de palabras/vocabulario
- Utilice la puntuación correcta
- Utilice gerundios en lugar de palabras de acción si desea enmascarar cualquier momento relacionado
- Asegúrese de que la información fluya al ritmo adecuado en su contenido
- Revise su contenido antes de publicarlo

3. Evite usar palabras de jerga

Se deben evitar las jergas en el idioma y contexto locales. Algunas jergas locales que se usan para referirse a ciertas cosas podrían resultar ofensivas o graciosas en otro país o lugar de donde proviene tu audiencia global. Los redactores de contenido no deben usar jergas para narrar nada y que el contenido parezca gracioso e ingenioso.

Muchos países del mundo ofrecen directrices específicas para cada país sobre el lenguaje culturalmente apropiado. Se debe usar un lenguaje y unas palabras respetuosas de forma coherente al referirse al contenido relacionado con la edad. Evite usar títulos y puestos de trabajo específicos de género en cualquier contexto.

4. Utilice elementos visuales que sean de naturaleza diversa.

Si utiliza elementos visuales en su contenido con fotos y nombres de perfil, la diversidad de estos elementos refuerza sus principios de inclusión. Los elementos visuales con una mezcla de personas promueven la diversidad cultural y muestran una cultura corporativa inclusiva. Esto impulsa la imagen de marca y la fidelización de los clientes. Las imágenes de marca suelen asociarse con los valores que representan. Durante cualquier crisis social, las marcas se posicionan a favor de su inclusión y diversidad, lo que fomenta la conciencia de marca.

Durante la producción de videos de marketing y tutoriales, una buena combinación de diferentes personajes y voces promueve la equidad y la igualdad. La inclusión y la diversidad están estrechamente relacionadas con la equidad y la igualdad.

5. Utilice términos adecuados al hablar de accesibilidad y discapacidad.

La discapacidad no define a las personas. Los redactores de contenido deben evitar sesgos y prejuicios al escribir sobre personas con discapacidad o sobre accesibilidad. Deben investigar las directrices de accesibilidad sobre los términos preferidos y comprender los matices de las palabras. Deben evitarse los términos que generen prejuicios para no herir los sentimientos de las personas. No deben aparecer en el contenido términos que contengan eufemismos o sean condescendientes, ya que podrían causar daño.

Auditoría de formación y contenidos

Las buenas directrices de estilo para la documentación de productos, los términos del glosario y otras directrices que deben cumplir los redactores de contenido técnico deben cumplirse. Los redactores de contenido deben recibir capacitación periódica sobre las directrices de inclusión y diversidad. Esta capacitación puede formar parte de las prácticas internas de RH. de la empresa para incorporar a nuevos redactores de contenido y capacitar a los redactores actuales, respectivamente.

La auditoría de contenido debe realizarse anualmente para garantizar que el contenido existente cumpla con los principios de inclusión y diversidad. La auditoría identificará cualquier deficiencia en la documentación y ofrecerá recomendaciones para mejorar la calidad del contenido. Estos informes de auditoría pueden hacerse públicos para generar confianza en los clientes por su compromiso con los principios de inclusión y diversidad.

Conclusión

Redactar documentación inclusiva fomenta la coherencia social y contribuye a difundir la paz entre el público global. El lenguaje inclusivo debe formar parte de las habilidades de un redactor de contenido técnico. Los redactores de contenido técnico deben recibir capacitación periódica sobre prácticas de documentación inclusiva. Esto permite crear contenido atractivo para un público global y acorde con los valores de la empresa.

ACTIVIDADES TEMA 4

Ejercicios:

Seleccione un componente de software que haya diseñado e implementado recientemente. Diseñe un conjunto de casos de prueba que garantice que todos los enunciados se ejecutan, usando prueba de ruta o trayectoria básica.

Elaborar un formato de plan de prueba para un **Sistema de Administración de Sucursales**. Este tipo de sistema puede incluir funcionalidades como gestión de sucursales, empleados, horarios, inventario, reportes. A continuación, se muestra un ejemplo de una plantilla básica.

1. Objetivo del Plan de Pruebas

El objetivo de este plan de pruebas es verificar que el Sistema de Administración de Sucursales funcione correctamente conforme a los requisitos funcionales y no funcionales especificados. Las pruebas validarán funcionalidades clave como el registro y gestión de sucursales, administración de empleados, generación de reportes y control de inventarios.

2. Alcance

Incluye pruebas funcionales, de integración, de regresión, de interfaz de usuario, rendimiento y seguridad. El entorno principal es una aplicación web, aunque puede contemplar una app móvil si aplica.

3. Características a Probar

Módulo	Funcionalidad
Gestión de Sucursales	Crear, editar, eliminar y listar sucursales
Gestión de Empleados	Asignar empleados a sucursales, CRUD de empleados
Inventario	Control de productos por sucursal, stock, movimientos
Reportes	Generar reportes por sucursal, por fechas, por productos
Seguridad y Accesos	Roles y permisos (Administrador, Gerente, Empleado)
Interfaz de Usuario	Navegación, validación de formularios, errores visibles al usuario

4. Criterios de Aceptación

- Todas las funcionalidades deben cumplir con los requisitos definidos.
- No debe haber errores críticos (bloqueantes) ni mayores en el sistema.
- El sistema debe responder en menos de 3 segundos en operaciones críticas.

5. Tipos de Pruebas

Tipo de Prueba	Descripción
Pruebas Funcionales	Validación de cada funcionalidad especificada

Tipo de Prueba	Descripción
Pruebas de Integración	Verificación de interacciones entre módulos (ej. Sucursales ↔ Inventario)
Pruebas de UI	Pruebas de la interfaz, experiencia del usuario, validaciones
Pruebas de Regresión	Asegura que nuevas funciones no afecten las existentes
Pruebas de Rendimiento	Verifica tiempos de respuesta y carga del sistema
Pruebas de Seguridad	Control de accesos, encriptación de datos, sesiones

6. Casos de Prueba Ejemplo

6.1 Caso de Prueba 001 – Crear Sucursal

- **Objetivo:** Validar que se puede crear una sucursal correctamente.
- **Entradas:** Nombre: "Sucursal Norte", Dirección: "Av. Central 123", Estado: "Activa"
- **Pasos:**
 1. Iniciar sesión como administrador.
 2. Ir a módulo "Sucursales".
 3. Hacer clic en "Agregar nueva".
 4. Completar los campos requeridos.
 5. Guardar.
- **Resultado Esperado:** La nueva sucursal aparece en la lista con los datos correctos.

Caso de Prueba 002 – Acceso no autorizado

- **Objetivo:** Verificar que un usuario con rol "Empleado" no puede eliminar una sucursal.
- **Entradas:** Usuario con rol "Empleado", intentar eliminar sucursal.
- **Resultado Esperado:** Acceso denegado con mensaje de error.

7. Criterios de Salida

- Todos los casos de prueba ejecutados con un 95% de éxito.
- No deben existir errores críticos ni mayores pendientes por corregir.

8. Recursos

- **Herramientas:** JIRA (gestión), Selenium (automatización), Postman (API), JMeter (rendimiento)
- **Equipo:** 1 QA Lead, 2 Testers funcionales, 1 Automatizador QA

9. Cronograma Tentativo

Actividad	Fecha Inicio	Fecha Fin
Diseño de pruebas	20/07/2025	22/07/2025
Ejecución pruebas manuales	23/07/2025	30/07/2025
Automatización pruebas	23/07/2025	05/08/2025
Informe de resultados	06/08/2025	07/08/2025

Fuentes Bibliográficas

- Bloch, J. (2018). *Effective Java* (3rd ed.). Addison-Wesley Professional.
- Boehm, B. W. (1981). *Software Engineering Economics*. Prentice Hall. Bohl, M. y M. Rynn, *Tools for Structured Design: An Introduction to Programming Logic*, 5a. ed., Prentice Hall, 2000
- Booch, G., Rumbaugh, J., & Jacobson, I. (1999). *The Unified Modeling Language User Guide*. Addison-Wesley.
- Coronel, C., & Morris, S. (2016). *Database Systems: Design, Implementation, & Management*. Cengage Learning.
- Duckett, J. (2014). *JavaScript and JQuery: Interactive Front-End Web Development*. Wiley.
- Fernández, C. (2019). *Adopción tecnológica y resistencia al cambio*. *Journal of Business Innovation*, 12(4), 112-119.
- Flanagan, D. (2020). *JavaScript: The Definitive Guide* (7th ed.). O'Reilly Media.
- Fowler, M. (2003). *UML Distilled: A Brief Guide to the Standard Object Modeling Language* (3rd ed.). Addison-Wesley.
- García, P., & López, M. (2020). *Infraestructura tecnológica en pequeñas y medianas empresas*. Editorial Tecnológica.
- Gómez, D., & Ruiz, A. (2022). *Impacto de sistemas digitales en la eficiencia operativa*. *Revista de Administración*, 20(1), 34-42.
- IEEE. (1998). *Estándar IEEE 830-1998: Práctica recomendada por el IEEE para especificaciones de requisitos de software*. Instituto de Ingenieros Eléctricos y Electrónicos.
- ISO/IEC 27001. (2013). *Tecnología de la información — Técnicas de seguridad — Sistemas de gestión de la seguridad de la información*.
- Jacobson, I., Booch, G., & Rumbaugh, J. (1992). *Object-Oriented Software Engineering: A Use Case Driven Approach*. Addison-Wesley.
- Kernighan, B. y P. Plauger, *The Elements of Programming Style*, 2a. ed., McGraw-Hill, 1978.
- Knuth, D., *The Art of Computer Programming*, tres volúmenes, Addison-Wesley, 1998.
- Lemay, L. (2015). *Learn Java in One Day and Learn it Well*. Kindle Edition.

- López, M., & Pérez, J. (2020). *Optimización de procesos mediante digitalización*. Editorial Innovación.
- Martínez, A. (2019). *Frameworks modernos para desarrollo web*. Revista de Tecnología, 15(2), 45-52.
- Pressman, RS (2014). *Ingeniería de software: un enfoque práctico* (8.^a ed.). Nueva York: McGraw-Hill.
- Sánchez, R., & Torres, L. (2021). *Capacitación en sistemas tecnológicos: mejores prácticas*. Revista de Gestión Empresarial, 8(3), 67-75.
- Sanghera, P. (2018). *Learning Java: A Hands-On Approach*. Mercury Learning & Information.
- Schmuller, J. (2000). *Aprendiendo UML en 24 horas* (1^a. ed.) Pearson Educación, México, 2000.
- Silberschatz, A., Korth, H. F., & Sudarshan, S. (2019). *Database System Concepts*. McGraw-Hill.
- Sommerville, I. (2011). *Ingeniería de software* (9.^a ed.). Boston: Addison-Wesley.
- Tilkov, S., & Vinoski, S. (2010). Node.js: Using JavaScript to build high-performance network programs. *IEEE Internet Computing*, 14(6), 80–83.
- Wieruch, R. (2018). *The Road to React: Your journey to master plain yet pragmatic React.js*. Independently published.
- Wiegers, KE, y Beatty, J. (2013). *Requisitos de software* (3.^a ed.). Redmond: Microsoft Press.
- Wiegers, K. (2003). *Software Requirements* (2nd ed.). Microsoft Press.

Fuentes Electrónicas

- [Diccionario de la Real Academia Española https://dle.rae.es/ingenier%C3%ADa](https://dle.rae.es/ingenier%C3%ADa)
- Eclipse Foundation. (s.f.). *Eclipse IDE for Java Developers*.
<https://www.eclipse.org/ide/>
- Eich, B. (2020). *A Brief History of JavaScript*. Mozilla Foundation.
https://developer.mozilla.org/en-US/docs/Web/JavaScript/A_re-introduction_to_JavaScript
- <https://www.ibm.com> › Inicio › Temas
- <https://inteleguia.com/es/blog/post/gestor-de-base-de-datos-qu%C3%A9-es-funcionalidades-y-ejemplos>
- <https://creately.com/blog/es/diagramas/tutorial-de-diagrama-de-despliegue/>
- <https://academy.pega.com/es/topic/data-model-design/v1>
- Northware (11 de abril 2025). Técnicas efectivas para la toma de requerimientos. Obtenido de
<https://www.northware.mx/blog/tecnicas-efectivas-para-la-toma-de-requerimientos/>
- Oracle. (s.f.). *Getting Started with Java*. Oracle.
<https://docs.oracle.com/en/java/javase/17/getting-started/index.html>
- Oracle Corporation. (2024). *MySQL Reference Manual*. <https://dev.mysql.com/doc/>
- Oracle. (n.d.). *The Java™ Tutorials*. <https://docs.oracle.com/javase/tutorial/>
- Oracle. (s.f.). *JDK: The Java Development Kit*. Oracle.
<https://www.oracle.com/java/technologies/javase-downloads.html>
- Oracle. (2023). *The Java™ Tutorials*. Oracle.
<https://docs.oracle.com/javase/tutorial/>
- JetBrains. (s.f.). *IntelliJ IDEA: The Java IDE for Professional Developers*.
<https://www.jetbrains.com/idea/>
- PostgreSQL Global Development Group. (2024). *PostgreSQL Documentation*.
<https://www.postgresql.org/docs/>
- Quiroz, C, (2025, 7, junio) ¿Qué es el desarrollo de software?. IBM.com.
<https://www.ibm.com/think/topics/software-development>
- <https://www.yunbitsoftware.com/blog/2016/09/02/importancia-de-las-pruebas-en-desarrollo-de-software/>
- https://www.google.com/search?q=dise%C3%B1o+de+casos+de+prueba&rlz=1C1CHBF_esMX852MX852&oq=dise%C3%B1o+de+casos+de+prueba&gs_lcrp=EqZjaHJvbWUyCQgAEEUYORiABDIHCAEQABiABDIICAiQABgWGB4yCAgDEAAYFhgeMggIBBAAGBYYYHjIICAUQABg

[WGB4yCAgGEAAYFhgeMgoIBxAAGKIEGikF0gEINTgxMGowajeoAgCwAgA&sourceid=chrome&ie=UTF-8](https://www.google.com/search?q=WGB4yCAgGEAAYFhgeMgoIBxAAGKIEGikF0gEINTgxMGowajeoAgCwAgA&sourceid=chrome&ie=UTF-8)

- <https://gawerk.es/blog/como-redactar-casos-de-prueba/>