**CAPITULO II**

**2 - MARCO TEORICO**

La agencia de turismo (JIWAKY) desempeñan un papel crucial en la industria turística al servir como intermediarias entre los proveedores de servicios (aerolíneas, hoteles, tours, etc.) y los viajeros. Ofrecen una variedad de servicios, como la gestión de reservas, itinerarios de viaje y asesoramiento a los clientes.

Este marco teórico proporciona un contexto sólido para investigar y desarrollar soluciones que aborden los desafíos en la operación de la agencia de turismo (JIWAKY) y promuevan la eficiencia operativa y la calidad del servicio.

**2-1** **SISTEMA DE INFORMACION**

Un sistema de información colaborativo respalda las operaciones empresariales, destacando la tecnología, pero también incluyendo procedimientos manuales. Según Rodríguez Rodríguez y Daureo Campillo, aborda la recopilación, procesamiento, almacenamiento y distribución de información para un flujo eficaz en la organización.

**2-1.1** **ELEMENTOS DE UN SISTEMA DE INFORMACIÓN**

Un sistema de información realiza tres actividades esenciales: entrada, procesamiento y salida. La entrada recopila datos, el procesamiento los transforma y la salida transmite la información procesada. Además, se requiere una función de retroalimentación, donde la información procesada se devuelve para evaluar y corregir la etapa de entrada según sea necesario (Laudon & Laudon, 2012).

* Entrada: Captura datos desde fuentes internas o externas, como transacciones, informes o formularios, marcando el inicio del flujo de información en el sistema.
* Proceso: Transforma, organiza y analiza los datos recopilados, realizando cálculos y operaciones para generar información valiosa y significativa para la organización.
* Salida: Presenta la información procesada de manera comprensible a través de informes, gráficos u otros formatos, facilitando la toma de decisiones y la ejecución de tareas. La salida comunica los resultados del procesamiento a usuarios finales.

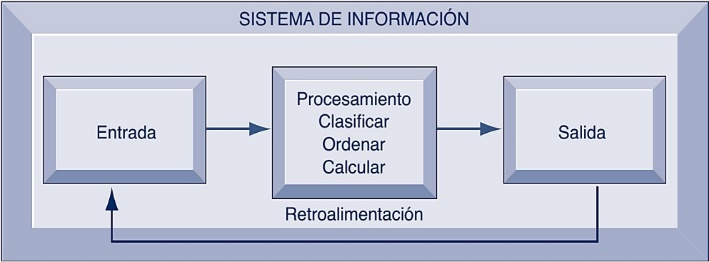


FIGURA 1. ACTIVIDADES DE UN SISTEMA DE INFORMACIÓN  
FUENTE: (Laudon & Laudon, 2012)

**2-1.2 TIPOS DE SISTEMAS DE INFORMACION**

Los sistemas de información se adaptan a las necesidades específicas de las empresas, abordando diversas funciones. Desde los sistemas de procesamiento de transacciones para operaciones básicas hasta los sistemas de apoyo a la toma de decisiones en grupo y colaborativos para enfrentar decisiones complejas, estos sistemas cubren una amplia gama de niveles operativos, de gestión y estratégicos en una organización (Kendall & Kendall, 2005).

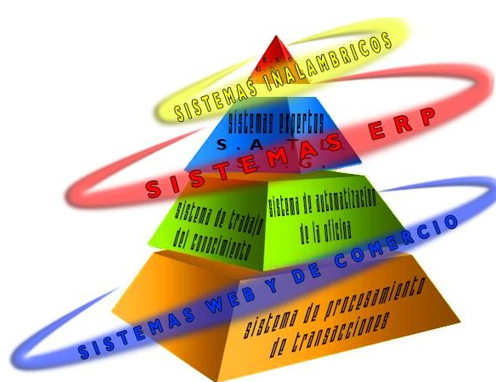


FIGURA 2. TIPOS DE SISTEMAS DE INFORMACION   
FUENTE. (Kendall & Kendall, 2005)

**2-1.2.1** **SISTEMA DE PROCESAMIENTO DE TRANSACCIONES**

Los sistemas de procesamiento de transacciones a nivel empresarial se refieren a aquellos sistemas diseñados para gestionar y registrar las actividades comerciales y financieras de manera rutinaria y cotidiana. Estas transacciones pueden abarcar una amplia gama de operaciones, como el procesamiento de nóminas, la facturación a clientes, el control de inventario y la gestión de depósitos bancarios. La naturaleza específica de estas transacciones puede variar significativamente en función del tipo de empresa y su sector de actividad.

**2-1.2.2** **SISTEMA DE APOYO A LA TOMA DE DECISIONES**

Un Sistema de Apoyo a la Toma de Decisiones, en un sentido general, es un sistema informático diseñado para facilitar y mejorar el proceso de toma de decisiones. En un contexto más específico, se trata de un sistema de información interactivo, flexible y adaptable, especialmente creado para respaldar la resolución de problemas de gestión que no siguen una estructura predefinida, con el objetivo de mejorar la calidad de las decisiones tomadas. Estos sistemas se basan en el uso de datos, proporcionan una interfaz de usuario amigable y permiten a los usuarios tomar decisiones basadas en un análisis integral de la situación.

**2-2 METODOLOGIA SCRUM**

**2-2.1 VISION GENERAL DE SCRUM**

Scrum es un marco de trabajo ágil que aborda desafíos complejos y adaptables, permitiendo la entrega eficiente de productos de alto valor. Es ligero, fácil de entender para todos los involucrados, pero requiere tiempo y experiencia para dominarlo. Utilizado ampliamente desde la década de 1990, no es un proceso específico, sino un marco flexible para aplicar diversas técnicas y procesos en el desarrollo de productos.

*“En un nivel más específico, según "SCRUM PRIMER" de Deemer, Benefield, Larman y Vodde (2012), Scrum se define como un enfoque en el cual equipos multifuncionales colaboran en la creación iterativa e incremental de productos o proyectos. El trabajo se estructura en ciclos llamados Sprints o iteraciones, con una duración típica de dos semanas. Cada Sprint se enfoca en un objetivo claro y tangible acordado por el equipo, y durante su transcurso no se pueden agregar elementos nuevos. Los cambios se adaptan en el siguiente Sprint.”*

Al final de cada Sprint, el equipo realiza una revisión con las partes interesadas y demuestra lo que han logrado. El feedback recibido se incorpora en el siguiente Sprint. Scrum pone un fuerte énfasis en entregar un producto completamente funcional al final de cada iteración, lo que significa que cada incremento es una versión "terminada" del producto.

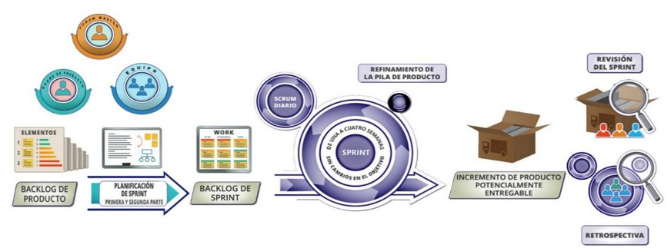


FIGURA 3. VISION GENERAL DE SCRUM   
FUENTE. (Deemer, Benefield, Larman, & Vodde, 2012)

**2-2.2** **EL EQUIPO SCRUM**

Un equipo Scrum, según Schwaber y Sutherland (2013), consta de tres roles principales: Dueño de Producto, Equipo de Desarrollo y Scrum Master. Estos equipos son autoorganizados y multifuncionales, lo que significa que tienen autonomía para tomar decisiones internas y poseen todas las habilidades necesarias para realizar sus tareas sin depender de fuentes externas. Esta estructura busca maximizar la flexibilidad, creatividad y productividad en el entorno de trabajo.

**2-2.2.1 DUEÑO DE PRODUCTO (PRODUCT OWNER)**

El propietario del producto (product owner) es quien toma las decisiones del cliente. Su responsabilidad es el valor del producto. Para simplificar la comunicación y toma de decisiones es necesario que este rol recaiga en una única persona. Si el cliente es una organización grande, o con varios departamentos, puede adoptar la forma de comunicación interna que consideren oportuna, pero en el equipo de desarrollo sólo se integra una persona en representación del cliente, y ésta debe tener el conocimiento suficiente del producto y las atribuciones, necesarias para tomar las decisiones que le corresponden. (Palacio, 2015)

**2-2.2.2 MAESTRO SCRUM (SCRUM MASTER)**

El Scrum Master desempeña un papel esencial en la implementación exitosa de Scrum, actuando como educador, mentor y líder del equipo. Su responsabilidad principal es guiar y apoyar al Equipo Scrum, eliminando obstáculos y creando un entorno propicio. Además, actúa como intermediario entre el equipo y las partes externas, facilitando la colaboración y optimizando la comunicación para maximizar el valor del equipo Scrum.

**2-2.2.3 EL EQUIPO DE DESARROLLO (DEVELOPMENT TEAM)**

El Equipo de Desarrollo en Scrum es un grupo autónomo y responsable de crear un Incremento de producto "Terminado" al final de cada Sprint. Con flexibilidad y autoridad para organizar su trabajo, este equipo colabora de manera sinérgica para lograr eficiencia y alta calidad en la entrega. Su autonomía les permite tomar decisiones independientes, promoviendo la eficacia en la ejecución de sus tareas.

**2-2.3 PROCESOS DE SCRUM**

El desarrollo se lleva a cabo de manera iterativa e incremental, organizado en ciclos definidos como "Sprints," que tienen una duración preestablecida, generalmente de 2 a 4 semanas. Al concluir cada Sprint, se obtiene una versión del software con nuevas funcionalidades listas para su uso. Durante cada nuevo Sprint, el equipo ajusta y mejora la funcionalidad existente, al tiempo que agrega nuevas características, priorizando aquellas que aportan un mayor valor de negocio.

El proceso de desarrollo se rige por los siguientes componentes clave:

* **Pila de Producto:** Lista de requisitos priorizados y descritos en lenguaje no técnico, ajustada regularmente.
* **Planificación del Sprint:** Reunión donde se seleccionan y organizan historias del Backlog para el Sprint.
* **Sprint:** Iteración fija donde el equipo convierte historias comprometidas en una versión funcional del software.
* **Pila del Sprint:** Lista de tareas necesarias para completar las historias del Sprint.
* **Reunión de Sprint Diario:** Encuentro breve para sincronizar actividades y discutir avances y obstáculos.
* **Demostración y Retrospectiva:** Reunión al final del Sprint para presentar historias completadas y reflexionar sobre mejoras.

**2-3** **LENGUAJE UNIFICADO DE MODELADO (UML)**

El Lenguaje Unificado de Modelado (UML) es un lenguaje visual para especificar, visualizar y documentar sistemas de software. Propuesto por Booch, Rambaugh y Jacobson, conceptualiza sistemas como colecciones de objetos colaborativos. Con cuatro objetivos clave, busca representar sistemas completos, conectar conceptos modelados con componentes ejecutables, abordar la complejidad y ser utilizable por humanos y máquinas. La creación de un modelo conceptual implica bloques básicos, reglas de interacción y mecanismos comunes, proporcionando una base sólida para comunicar la estructura y el comportamiento de sistemas complejos.

Los autores tenían cuatro objetivos principales en mente al desarrollar el UML:

* Representar sistemas en su totalidad, superando la mera representación de un solo programa.
* Establecer una conexión explícita entre los conceptos modelados y los componentes ejecutables del sistema.
* Tener en cuenta la complejidad inherente a sistemas críticos y de gran escala.
* Crear un lenguaje de modelado que resulte utilizable tanto para seres humanos como para máquinas.

**2-3.1 MODELOS**

Un modelo es una representación, generalmente en un medio distinto al objeto o sistema que se desea representar. Esta representación busca capturar los aspectos esenciales y significativos del objeto o sistema desde un punto de vista específico, a menudo simplificando o omitiendo detalles menos relevantes.

* Modelo de Casos de Uso: Facilita la comunicación entre desarrolladores y clientes al definir actores, casos de uso y sus relaciones, estableciendo los requisitos funcionales del sistema.
* Modelo de Análisis: Organiza en paquetes el análisis más alto, abstrayendo subsistemas y capas de diseño para gestionar eficazmente un sistema complejo.
* Modelo de Diseño: Se centra en la realización física de los casos de uso, considerando requisitos funcionales y no funcionales, y sirve como base para la implementación.
* Modelo de Despliegue: Describe la distribución física del sistema entre nodos de cómputo, influyendo en el diseño y siendo crucial para la implementación.
* Modelo de Implementación: Detalla cómo los elementos del modelo de diseño se convierten en componentes de implementación, organizándolos según estructuras y modularización.

Modelo de Prueba: Enfocado en las pruebas de componentes ejecutables, asegura el correcto funcionamiento del sistema, abordando aspectos específicos que requieran pruebas particulares.

Cada uno de estos modelos cumple un papel crucial en el ciclo de desarrollo de software, abordando aspectos específicos y contribuyendo a la creación exitosa del sistema en su conjunto.

**2-3.2 DIAGRAMAS**

Según la Real Academia de la Lengua Española, un diagrama es un dibujo geométrico utilizado para ilustrar una proposición, resolver un problema o representar gráficamente la ley de variación de un fenómeno. A través de él, se presentan visualmente las relaciones entre las diversas partes de un conjunto o sistema.

**2-3.2.1 DIAGRAMA DE CASOS DE USOS**

Un diagrama de casos de uso es una representación gráfica que muestra un conjunto de casos de uso, actores y sus relaciones en un sistema. Estos diagramas capturan el comportamiento del sistema desde la perspectiva de los usuarios que interactúan con él. Esencialmente, un diagrama de casos de uso ilustra las interacciones entre el sistema, sus actores externos y los usuarios. Proporciona una visión gráfica de quiénes utilizan el sistema y cómo esperan interactuar con él.



FIGURA 4. COMPONENTES DEL DIAGRAMA DE CASOS DE USO  
FUENTE. (Booch, Rambaugh, & Jacobson, 2004)

En detalle, los elementos de un diagrama de casos de uso incluyen:

* Casos de Uso: Representan requisitos funcionales y describen interacciones entre usuarios y el sistema, definiendo la interfaz externa y comportamiento funcional.
* Actores: Representan roles de usuarios, dispositivos u otros sistemas en las interacciones. Se visualizan como rectángulos con el estereotipo "actor" y se relacionan mediante asociación, generalización y uso.
* Relación de Dependencia: Indica que un cambio en un elemento puede afectar a otro, representado por una flecha discontinua orientada hacia el elemento dependiente.
* Relación de Generalización: Representa la relación entre un elemento general y un caso específico, indicando que los objetos hijos pueden usarse en cualquier lugar donde aparezca el elemento padre. Se representa con una flecha continua.
* Relación de Asociación: Denota la interacción entre un actor y un caso de uso, representada visualmente por una línea sólida. Se diferencian con flechas para indicar qué actor inicia el caso de uso.

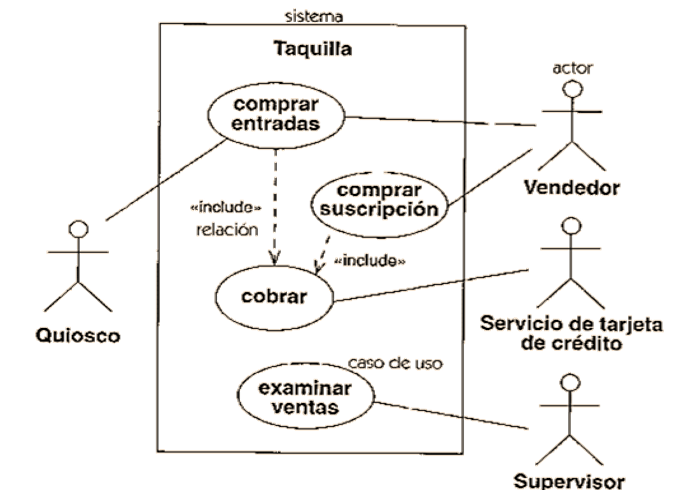
****

FIGURA 5. DIAGRAMA DE CASOS DE USO   
FUENTE. (Booch, Rambaugh, & Jacobson, 2004)

**2-3.2.2 DIAGRAMAS DE CLASES**

El diagrama de clases es una herramienta que se utiliza para representar la estructura estática de un sistema. Es importante destacar que se puede presentar a diferentes niveles de detalle, no siendo siempre necesario mostrarlo en su máxima complejidad. Este tipo de diagrama incluye la estructura de clases, que comprende sus atributos y métodos, y las relaciones que existen entre ellas, con detalles como cardinalidad, roles, herencia y pertenencia. La representación de una clase puede variar dependiendo de la profundidad de información requerida.

La simbología en un diagrama de clases se detalla de la siguiente manera:

* Clases: Representan conjuntos de objetos con estructuras similares, incluyendo comportamiento y relaciones. Se visualizan como un rectángulo sólido con secciones para nombre, atributos y operaciones.
* Interfaces: Describen el comportamiento visible de una clase, componente o paquete. Se representan como un rectángulo etiquetado como "interface," dividiéndose en compartimentos con las operaciones soportadas.
* Relación de Dependencia: Se establece entre clases cuando un cambio en un elemento podría requerir cambios en otro elemento dependiente.
* Relación de Asociación: Representa relaciones estructurales entre clases y objetos, indicando una conexión sin especificar la duración de vida relevante.
* Relación de Generalización: Describe una relación jerárquica entre clases, donde una se considera superclase y la otra subclase. Puede haber varias subclases de una superclase.

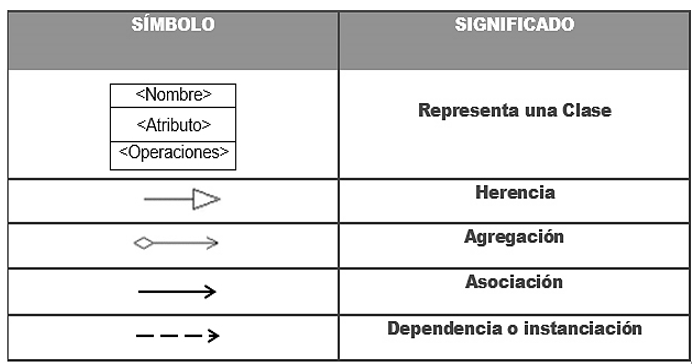


FIGURA 6. SIMBOLOGÍA DEL DIAGRAMA DE CLASES   
FUENTE. Elaboración Propia

**2-3.2.3 DIAGRAMAS DE SECUENCIA**

Según Booch, Rambaugh y Jacobson (2004), un diagrama de secuencia representa una interacción de manera gráfica y bidimensional. La dimensión vertical corresponde al eje del tiempo, donde el tiempo transcurre a través de la página. La dimensión horizontal muestra los roles que representan objetos individuales en la colaboración. Cada uno de estos roles se representa mediante una columna vertical que contiene un símbolo de inicio y una línea vertical, a menudo llamada "lifeline."

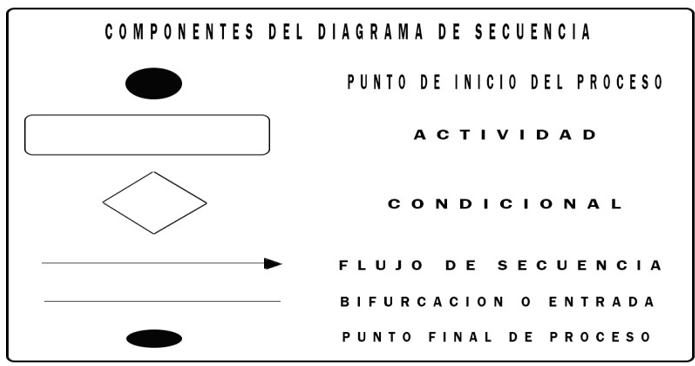


FIGURA 7. COMPONENTES DEL DIAGRAMA DE SECUENCIA   
FUENTE. ELABORACIÓN PROPIA

A continuación, se detallan los componentes clave de un diagrama de secuencia:

* Participantes: Representan los roles u objetos que interactúan en la secuencia. Cada participante se muestra como una línea vertical que refleja su actividad en el tiempo.
* Mensajes: Son las interacciones o comunicaciones entre los participantes. Estos mensajes se muestran mediante flechas que indican la dirección de la comunicación y su contenido.
* Activaciones: Indican el período durante el cual un participante está activo o involucrado en la interacción. Se representan mediante cajas verticales adyacentes a la línea del participante.
* Fragmentos de Interacción: Estos fragmentos ayudan a mostrar la lógica condicional o bucles en la secuencia de interacción. Pueden incluir condiciones como "if" o "else."
* Notas y Comentarios: Se utilizan para proporcionar información adicional o aclaraciones en el diagrama.

**2-3.2.4 DIAGRAMA DE ESTADOS**

Los diagramas de estados se emplean para crear una representación visual de las máquinas de estados finitos. Una máquina de estados finitos es un modelo de comportamiento que detalla las diferentes secuencias de estados que un objeto puede experimentar a lo largo de su vida en respuesta a eventos específicos, así como las acciones o respuestas asociadas a dichos eventos. Este enfoque permite una comprensión visual y estructurada de cómo un objeto o sistema se comporta en diferentes situaciones, lo que es fundamental para el diseño y la descripción de sistemas que implican transiciones de estado y eventos.

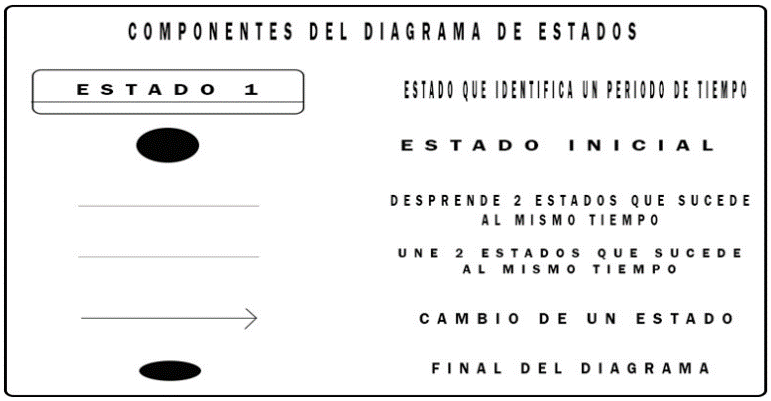
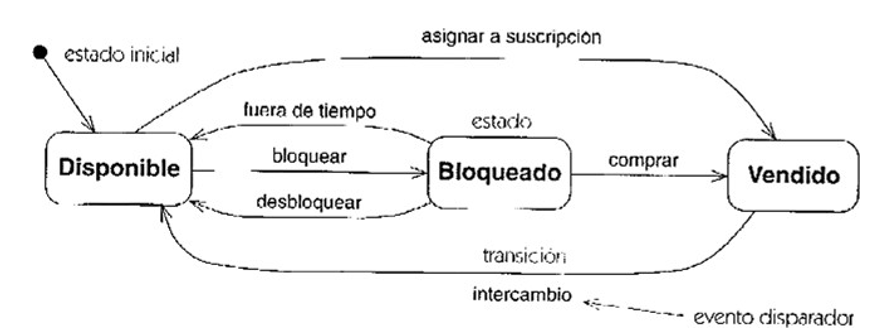


FIGURA 8. COMPONENTES DEL DIAGRAMA DE ESTADOS   
FUENTE. ELABORACION PROPIA

****

*FIGURA 9. DIAGRAMA DE ESTADOS   
FUENTE. (Booch, Rambaugh, & Jacobson, 2004)*

**2-3.2.5 DIAGRAMA DE ACTIVIDADES**

De acuerdo con la definición de Fowler (1999), un diagrama de actividades es una herramienta que se utiliza para representar un procedimiento lógico, un proceso de negocio o un flujo de trabajo. La interpretación de estos diagramas varía según su contexto en el proyecto. En las fases conceptuales, las actividades pueden considerarse como tareas que deben ser realizadas por una computadora o un ser humano.

**2-4** **TECNICA DE MODELADO DE OBJETOS (OMT)**

OMT (Object Modeling Technique) fue desarrollada por James Rumbaugh y Michael Blaha en 1991, mientras James lideraba un equipo de investigación en los laboratorios General Electric (Medina, 2005). La esencia del análisis y diseño orientado a objetos radica en la identificación y organización de conceptos dentro del dominio de la aplicación, independientemente de su posterior implementación en un lenguaje de programación orientado a objetos (Rumbaugh & Blaha, 1991).

Para construir sistemas complejos, los desarrolladores deben abstraer distintas perspectivas del sistema, crear modelos utilizando notaciones precisas, asegurarse de que estos modelos cumplan con los requisitos del sistema y, de manera gradual, agregar detalles para transformar los modelos en una implementación concreta (Rumbaugh & Blaha, 1991).

OMT divide el ciclo de vida del software en cuatro fases consecutivas:

* Análisis de Objetos: En esta etapa, se enfoca en comprender y modelar el problema dentro del dominio de la aplicación.
* Diseño del Sistema: Aquí se determina la arquitectura del sistema en términos de subsistemas.
* Diseño de Objetos: Esta fase refina y optimiza el análisis de objetos para prepararlo para la implementación.
* Implementación: En esta etapa, se procede a la codificación y pruebas de lo diseñado previamente.

OMT, como técnica, proporciona un enfoque estructurado y sistemático para el análisis y diseño de sistemas basados en objetos, lo que contribuye a la construcción eficiente de sistemas de software robustos y de alta calidad.

**2-4.1 ANALISIS DE OBJETOS**

El proceso de desarrollo de software sigue un enfoque estructurado que comienza con la descripción del problema y la obtención de requisitos claros. Se procede a la creación de modelos, incluyendo el Modelo de Objetos para la estructura estática, el Modelo Dinámico que aborda el comportamiento temporal, y el Modelo Funcional que describe las funciones del sistema. Cada modelo se verifica exhaustivamente, iterando para lograr refinamientos. Este enfoque garantiza una cobertura completa y metódica de todos los aspectos clave del sistema, contribuyendo a un desarrollo de software sólido.

**2-4.1.1 MODELO DE OBJETOS**

Los pasos para construir el modelo de objetos son los siguientes:

* Identificación de Objetos y/o Clases: En esta etapa se identifican los objetos y clases que formarán parte del sistema, considerando sus roles y responsabilidades.
* Creación de un Diccionario de Datos: Se elabora un diccionario que contiene información detallada sobre los objetos, clases y conceptos del sistema, incluyendo sus atributos y relaciones.
* Identificación de Atributos y Conexiones para la Creación del Diagrama de Clases u Objetos: Aquí se definen los atributos de los objetos y se establecen las conexiones entre ellos para representar las relaciones entre clases. Con esta información, se puede crear el diagrama de clases u objetos que muestra la estructura estática del sistema.
* Realización de Iteraciones para el Refinamiento del Modelo: Es importante realizar iteraciones en todo el proceso para garantizar que el modelo sea preciso y completo. Esto implica revisar y ajustar el modelo según sea necesario.

**2-4.1.2 MODELO DINAMICO**

Los pasos para construir el modelo dinámico son los siguientes:

* Preparación de Escenarios de Secuencias Típicas de Iteración: En esta etapa inicial, se preparan escenarios que representan las secuencias típicas de interacción entre los objetos dentro del sistema. Estos escenarios ayudan a comprender cómo los objetos interactúan en situaciones específicas.
* Identificación de Sucesos que Actúan entre Objetos: Se identifican los eventos o sucesos que ocurren entre los objetos en cada escenario. Estos sucesos representan las interacciones y acciones que los objetos realizan en respuesta a eventos.
* Construcción de un Diagrama de Estado para Cada Objeto: Para cada objeto que tiene un comportamiento dinámico, se crea un diagrama de estado que representa sus posibles estados y las transiciones entre ellos. Estos diagramas de estado muestran cómo un objeto responde a eventos y cambia de estado en función de su comportamiento.
* Construcción del Diagrama Global de Flujo de Sucesos (Diagrama de Secuencia): Finalmente, se construye un diagrama de secuencia que representa la secuencia global de sucesos en el sistema. Este diagrama muestra cómo los objetos interactúan a lo largo del tiempo y cómo se comunican entre sí en función de los sucesos identificados en los escenarios.

En conjunto, estos pasos permiten modelar de manera efectiva el comportamiento dinámico del sistema, lo que incluye cómo los objetos interactúan, responden a eventos y evolucionan a lo largo del tiempo. La construcción de diagramas de estado y diagramas de secuencia es fundamental para representar estos aspectos del sistema de manera clara y comprensible.

**2-4.1.3 MODELO FUNCIONAL**

* Identificación de Entradas y Salidas: Clarificación de datos que ingresan y salen del sistema para comprender la interacción con el entorno.
* Construcción de Diagramas de Flujo de Datos: Creación de representaciones gráficas que muestran las dependencias funcionales y transformaciones de datos dentro del sistema.
* Descripción de Funciones: Detalle de operaciones y procesos, incluyendo cálculos y transformaciones, en respuesta a los datos de entrada.
* Identificación de Restricciones: Documentación de limitaciones como tiempo, recursos o regulaciones que afectan las funciones del sistema.

Estos pasos son esenciales para comprender y representar eficazmente cómo el sistema realiza sus funciones y procesa los datos, ofreciendo una visión clara de su lógica interna y logro de objetivos.

**2-5 ARQUITECTURA DEL SOFTWARE**

La Arquitectura del Software, como lo define Clements en su trabajo en 1996, es un concepto fundamental en el diseño y desarrollo de sistemas informáticos. Se puede desglosar de la siguiente manera:

* Representación de Alto Nivel: Ofrece una visión simplificada de la arquitectura del software, enfocándose en componentes clave y relaciones para lograr los objetivos del sistema.
* Componentes Fundamentales: Identifica los elementos esenciales del sistema, como módulos de software o componentes de hardware, actuando como bloques de construcción.
* Comportamiento desde la Perspectiva del Sistema: Se centra en cómo interactúan y se comportan en conjunto los componentes para alcanzar los objetivos del sistema.
* Interacciones y Coordinación entre Componentes: Describe cómo los componentes se comunican y cooperan, definiendo interfaces, protocolos y flujos de datos para una colaboración efectiva.
* Cumplir la Misión del Sistema: La arquitectura del software se crea con el propósito de asegurar que el sistema cumpla su función o misión, ya sea procesamiento de datos, gestión de bases de datos u otros objetivos específicos.

**2-5.1 PATRON MODELO VISTA CONTROLADOR**

Patrón de arquitectura que separa datos y lógica de la interfaz de usuario. Se divide en Modelo (gestión de datos), Vista (representación de la interfaz) y Controlador (gestión de eventos). Proporciona una organización estructurada y modular para facilitar mantenimiento y escalabilidad.

**2-5.1.1 EL MODELO**

Cumple diversas responsabilidades, como acceder a la capa de almacenamiento de datos, definir reglas de negocio y notificar a las vistas sobre cambios en los datos. Estas funciones permiten independencia del sistema de almacenamiento y facilitan la actualización de datos por agentes externos.

**2-5.1.2 LA VISTA**

Responsables de recibir datos del controlador o modelo y presentarlos al usuario. Mantienen un registro del controlador asociado y ofrecen el servicio de "Actualización" para refrescar la vista en casos de cambios en datos para modelos pasivos. Se ejemplifica su uso en la navegación web.

**2-5.1.3 CONTROLADOR**

Responsable de recibir eventos de entrada del usuario, como clics de ratón o cambios en campos de texto. Contiene reglas de gestión de eventos, determinando acciones a ejecutar en respuesta a eventos específicos, como actualizar vistas o realizar solicitudes al modelo.

**2-5.2 ARQUITECTURA DE 3 CAPAS**

El Patrón de arquitectura por capas es una técnica ampliamente utilizada por arquitectos de software para estructurar sistemas de software complejos. Este enfoque conceptualiza un sistema como una serie de capas o niveles, similar a las capas de un pastel, donde cada capa se basa en la que está debajo de ella.

Los beneficios de trabajar con un enfoque por capas son:

* Permite comprender y trabajar en cada capa de manera aislada, sin necesidad de considerar las otras capas en ese momento.
* Las capas se pueden sustituir con implementaciones alternativas que ofrecen los mismos servicios básicos, lo que facilita la flexibilidad y adaptabilidad del sistema.
* Minimiza las dependencias entre las capas, lo que hace que el sistema sea más modular y fácil de mantener.
* Promueve la estandarización de servicios dentro de cada capa, lo que mejora la coherencia y la consistencia en todo el sistema.
* Una vez que se ha construido una capa, puede ser utilizada por múltiples servicios de nivel superior, lo que fomenta la reutilización de componentes.

**2-5.2.1 CAPA DE PRESENTACIÓN**

La interfaz de usuario se refiere a la forma en que los usuarios interactúan con el software. Puede ser tan básica como un menú de línea de comandos o tan sofisticada como una aplicación basada en formularios. Su función principal es mostrar información al usuario, interpretar los comandos que este introduce y llevar a cabo algunas validaciones sencillas de los datos ingresados.

**2-5.2.2 CAPA DE REGLAS DE NEGOCIO (EMPRESARIAL)**

También conocida como Lógica de Dominio, esta capa contiene la funcionalidad que implementa la aplicación. Engloba cálculos basados en la información proporcionada por el usuario y los datos almacenados, así como las validaciones necesarias. Controla la ejecución de la capa de acceso a datos y servicios externos. La lógica de la capa de negocios puede diseñarse para su uso directo por parte de los componentes de presentación o encapsularse como un servicio que se llama a través de una interfaz de servicios, coordinando así la comunicación con los clientes del servicio o invocando cualquier flujo o componente de negocio.

**2-5.2.3 CAPA DE DATOS**

Esta capa contiene la lógica de comunicación con otros sistemas encargados de llevar a cabo tareas dentro de la aplicación. Estos sistemas pueden ser monitores transaccionales, otras aplicaciones, sistemas de mensajería, entre otros. En el contexto de aplicaciones empresariales, esta capa suele estar representada por una base de datos, que se encarga del almacenamiento persistente de la información. Su función principal es abstraer por completo a las capas superiores, como la capa de negocio, del dialecto utilizado para comunicarse con los repositorios de datos, como PL/SQL, Transact-SQL, u otros.

**2-6 SEGURIDAD**

La Norma ISO 27001 es un estándar internacional de seguridad de la información que proporciona un marco para establecer, implementar, mantener y mejorar un sistema de gestión de seguridad de la información (SGSI). Algunos conceptos clave de la norma ISO 27001 que podrían estar aplicando incluyen:

* Política de Seguridad de la Información: Establecer una política que defina los objetivos de seguridad de la información y el compromiso de la alta dirección con la seguridad.
* Análisis de Riesgos: Identificar y evaluar los riesgos de seguridad de la información para determinar las medidas de seguridad adecuadas.
* Tratamiento de Riesgos: Implementar controles y medidas para mitigar los riesgos identificados.
* Documentación: Crear y mantener documentos que describan el SGSI, incluyendo la política, los procedimientos y los registros.
* Ciclo de Mejora Continua: Seguir un enfoque de mejora continua, incluyendo la revisión periódica de la eficacia de los controles y la política de seguridad de la información.
* Auditoría Interna: Realizar auditorías internas regulares para garantizar el cumplimiento de los controles y procedimientos de seguridad.
* Certificación ISO 27001: Buscar la certificación ISO 27001, que demuestra el compromiso de la organización con la seguridad de la información.
* Gestión de Incidentes de Seguridad: Implementar un proceso para la detección, gestión y respuesta a incidentes de seguridad de la información.

**2-6.1 PROPIEDADES DE UN SISTEMA DE INFORMACION SEGURO**

**2-6.1.1 INTEGRIDAD**

Este principio de seguridad informática garantiza la autenticidad y precisión de la información en cualquier momento que se solicite, asegurando que los datos no han sido alterados ni destruidos de manera no autorizada. Para prevenir y detectar posibles fallos en la integridad de los datos, el sistema debe estar equipado con mecanismos que permitan identificar y abordar los errores que puedan surgir.

**2-6.1.2 CONFIDENCIALIDAD**

La OCDE (Organización para la Cooperación y el Desarrollo Económico) define la confidencialidad, en sus Directrices para la Seguridad de los Sistemas de Información, como el hecho de que los datos o información estén disponibles únicamente para las personas, entidades o mecanismos autorizados, en los momentos autorizados y de una manera autorizada. Para garantizar la confidencialidad y prevenir posibles fallos, es fundamental diseñar un control de acceso al sistema que determine quién puede acceder, a qué parte del sistema, en qué momento y para qué tipo de operaciones.

**2-6.1.3 DISPONIBILIDAD**

La disponibilidad de la información garantiza que esta esté accesible para los usuarios autorizados cuando la necesiten, lo que implica que se pueda acceder al sistema de información en un periodo de tiempo considerado aceptable. La disponibilidad está estrechamente relacionada con la fiabilidad técnica de los componentes del sistema de información. Para asegurar la disponibilidad de la información, se deben implementar medidas de protección de los datos, así como establecer sistemas de copias de seguridad y mecanismos para la restauración de datos que puedan haber sido dañados o destruidos, ya sea de manera accidental o intencionada.

**2-6.2 MÉTODOS**

**2-6.2.1 CORTAFUEGOS (FIREWALL)**

Un cortafuegos o firewall es la parte de un sistema informático que se utiliza para impedir un acceso a través de la red no autorizado, pero permitiendo sin problemas todos los accesos autorizados. Los cortafuegos pueden ser software o hardware. (Gárcia-Cervigón Hurtado & Alegre Ramos, 2011)

**2-6.2.2 AUTENTICACIÓN Y AUTORIZACIÓN**

La autenticación y la autorización están estrechamente relacionadas debido a que la autenticación es el establecimiento de confirmación para dar paso a la autorización de acceso a las zonas restringidas, realizar acciones permitidas con su correspondiente nivel de privilegio, controlar el acceso a recursos protegidos, para prevenir ataques de escalada de privilegios.

**2-6.2.3 CIFRADO DE DATOS**

El cifrado de datos es uno de los métodos de protección más fiables en seguridad informática. Este proceso implica la transformación de los datos de tal manera que resulten incomprensibles para cualquier persona no autorizada a acceder a ellos. El cifrado puede llevarse a cabo mediante componentes lógicos o físicos, y es esencial para garantizar la confidencialidad de la información. Sin embargo, es importante tener en cuenta que el cifrado de datos puede ser intensivo en recursos computacionales, lo que significa que puede requerir un considerable poder de procesamiento y tiempo, dependiendo de la complejidad del cifrado empleado.

**2-6.2.4 SESIÓN**

Una sesión en una aplicación web implica interacciones entre un cliente y un servidor. En el contexto RESTful, las sesiones se gestionan sin estado mediante tokens de acceso, como JWT. Estos tokens, emitidos tras la autenticación, transportan información de estado en cada solicitud, eliminando la necesidad de mantener una sesión continua en el servidor y permitiendo la escalabilidad de la aplicación.

**2-7** **MODELO DE ESTIMACION DE COSTOS COCOMO**

El Modelo de Estimación de Costos COCOMO (Constructive Cost Model) es un método ampliamente utilizado para estimar los costos, el tiempo y los recursos necesarios en el desarrollo de proyectos de software. Este modelo fue desarrollado por Barry W. Boehm en la década de 1980 y ha evolucionado en varias versiones a lo largo del tiempo. COCOMO se basa en una serie de fórmulas y parámetros que ayudan a los equipos de desarrollo de software a calcular los recursos requeridos y las estimaciones de costos con base en las características y el alcance del proyecto.

El modelo COCOMO se compone de tres niveles o modos de estimación, cada uno adaptado a diferentes niveles de detalle en la planificación del proyecto:

* COCOMO Básico: Este nivel de estimación se utiliza en las primeras etapas del proyecto, cuando se tienen pocos detalles específicos sobre el sistema. Se utiliza para realizar estimaciones de alto nivel y se basa principalmente en el tamaño estimado del software (en líneas de código o puntos de función).
* COCOMO Intermedio: En este nivel, se consideran factores adicionales, como la complejidad del proyecto, la experiencia del equipo y otras características específicas del proyecto. Estas estimaciones son más detalladas que las del COCOMO Básico y se utilizan cuando se cuenta con más información sobre el proyecto.
* COCOMO Detallado: Este es el nivel más detallado de estimación y se utiliza en etapas avanzadas del proyecto, cuando se cuenta con información detallada sobre los requerimientos, la arquitectura y otros aspectos del sistema. Permite realizar estimaciones muy precisas de costos y recursos.

**2-8 METRICAS DE CALIDAD DEL SOFTWARE**

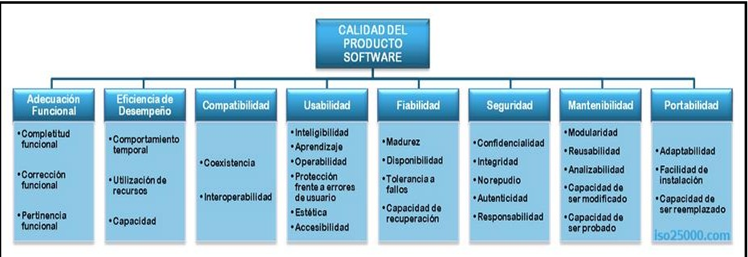
La búsqueda de software de alta calidad es esencial en la ingeniería de software, donde la calidad se define por la capacidad de cumplir de manera confiable y eficaz con los requisitos del usuario. Este objetivo se vuelve crucial con la creciente integración de aplicaciones informáticas en nuestras vidas. La calidad del software abarca aspectos como funcionalidad, confiabilidad, rendimiento, seguridad, usabilidad y mantenibilidad, adaptándose a las necesidades específicas de cada proyecto y cliente. En este contexto, el Modelo de Estimación de Costos COCOMO emerge como una herramienta valiosa para gerentes y equipos de desarrollo, proporcionando una metodología sólida para la estimación precisa de recursos y costos a lo largo del ciclo de vida del desarrollo de software.

**2-8.1** **EL ESTANDAR ISO/IEC 25010**

La ISO (International Organization for Standardization) y la IEC (International Electrotechnical Commission) constituyen un sistema especializado de estandarización a nivel mundial. En el ámbito de las tecnologías de la información, ISO e IEC han colaborado en la creación de un comité técnico conjunto conocido como ISO/IEC JTC, cuya responsabilidad principal es desarrollar estándares internacionales (Pérez Medina & Sánchez, 2012).

El modelo de calidad del producto definido por la ISO/IEC 25010 está compuesto por las ocho características de calidad que se presentan en la siguiente figura:

Este modelo se ha convertido en un marco fundamental para evaluar y garantizar la calidad de productos y sistemas de software, brindando una base sólida para la estandarización y la mejora continua en el desarrollo de software y tecnologías relacionadas.



**2-8.1.1 ADECUACIÓN FUNCIONAL**

La característica de "Funcionalidad" representa la capacidad del producto de software para ofrecer funciones que satisfacen las necesidades explícitas e implícitas, cuando el producto se utiliza en las condiciones especificadas. Esta característica se desglosa en las siguientes subcaracterísticas:

* Completitud Funcional: Esta subcaracterística evalúa en qué medida el conjunto de funcionalidades del producto abarca todas las tareas y objetivos especificados por el usuario.
* Corrección Funcional: La corrección funcional se refiere a la capacidad del producto o sistema para proporcionar resultados correctos con el nivel de precisión requerido.
* Pertinencia Funcional: Evalúa la capacidad del producto de software para ofrecer un conjunto adecuado de funciones que se ajusten a las tareas y objetivos específicos del usuario.

**2-8.1.2 EFICIENCIA DE DESEMPEÑO**

La característica de "Eficiencia del Desempeño" se refiere al rendimiento del software en relación a la cantidad de recursos utilizados en condiciones específicas. Esta característica se divide en las siguientes subcaracterísticas:

* Comportamiento Temporal: Evalúa los tiempos de respuesta y procesamiento, así como las tasas de transferencia de datos de un sistema cuando realiza sus funciones en condiciones predefinidas en comparación con un conjunto de pruebas de referencia (benchmark).
* Utilización de Recursos: Mide las cantidades y tipos de recursos que el software utiliza al llevar a cabo sus funciones en condiciones determinadas.
* Capacidad: Se refiere al grado en que los límites máximos de los parámetros de un producto o sistema de software cumplen con los requisitos especificados.

**2-8.1.3 COMPATIBILIDAD**

La característica de "Compatibilidad" se refiere a la capacidad de dos o más sistemas o componentes para intercambiar información y realizar sus funciones requeridas cuando comparten el mismo entorno hardware o software. Esta característica se divide en las siguientes subcaracterísticas:

* Coexistencia: Evalúa la capacidad del producto para operar en conjunto con otro software independiente en un entorno compartido, compartiendo recursos comunes sin afectar negativamente su funcionamiento.
* Interoperabilidad: Mide la capacidad de dos o más sistemas o componentes para intercambiar información y utilizar la información intercambiada de manera efectiva.

**2-8.1.4 USABILIDAD**

Capacidad del producto software para ser entendido, aprendido, usado y resultar atractivo para el usuario, cuando se usa bajo determinadas condiciones. Esta característica se subdivide a su vez en las siguientes subcaracterísticas:

* Capacidad para reconocer su adecuación: capacidad del producto que permite al usuario entender si el software es adecuado para sus necesidades.
* Capacidad de aprendizaje: capacidad del producto que permite al usuario aprender su aplicación.
* Capacidad para ser usado: capacidad del producto que permite al usuario operarlo y controlarlo con facilidad.
* Protección contra errores de usuario: capacidad del sistema para proteger a los usuarios de hacer errores.
* Estética de la interfaz de usuario: capacidad de la interfaz de usuario de agradar y satisfacer la interacción con el usuario.
* Accesibilidad: capacidad del producto que permite que sea utilizado por usuarios con determinadas características y discapacidades.

**2-8.1.5 FIABILIDAD**

Capacidad de un sistema o componente para desempeñar las funciones especificadas, cuando se usa bajo unas condiciones y periodo de tiempo determinados. Esta característica se subdivide a su vez en las siguientes subcaracterísticas:

* Madurez: capacidad del sistema para satisfacer las necesidades de fiabilidad en condiciones normales.
* Disponibilidad: capacidad del sistema o componente de estar operativo y accesible para su uso cuando se requiere.
* Tolerancia a fallos: capacidad del sistema o componente para operar según lo previsto en presencia de fallos hardware o software.
* Capacidad de recuperación: capacidad del producto software para recuperar los datos directamente afectados y reestablecer el estado deseado del sistema en caso de interrupción o fallo. (iso 25000, 2015)

**2-8.1.6 SEGURIDAD**

La característica de "Seguridad" se refiere a la capacidad del software para proteger la información y los datos de manera que personas o sistemas no autorizados no puedan acceder a ellos o modificarlos. Esta característica se desglosa en las siguientes subcaracterísticas:

* Confidencialidad: Capacidad del sistema para proteger los datos e información contra el acceso no autorizado, ya sea de manera accidental o intencionada.
* Integridad: Capacidad del sistema o componente para prevenir accesos o modificaciones no autorizadas en los datos o programas de computadora.
* No repudio: Capacidad del sistema para demostrar las acciones o eventos que han tenido lugar, de modo que dichas acciones o eventos no puedan ser negadas posteriormente.
* Responsabilidad: Capacidad del sistema para rastrear de manera inequívoca las acciones de una entidad.
* Autenticidad: Capacidad del sistema para demostrar la identidad de un sujeto o un recurso.

**2-8.1.7 MANTENIBILIDAD**

La característica de "Mantenibilidad" se refiere a la capacidad del producto software para ser modificado de manera efectiva y eficiente, ya sea para abordar necesidades evolutivas, correcciones de errores o mejoras. Esta característica se desglosa en las siguientes subcaracterísticas:

* Modularidad: Capacidad de un sistema o programa de computadora, compuesto por componentes discretos, que permite que un cambio en un componente tenga un impacto mínimo en los demás.
* Reusabilidad: Capacidad de un activo que permite que sea utilizado en más de un sistema de software o en la construcción de otros activos.
* Analizabilidad: Facilidad con la que se puede evaluar el impacto de un cambio específico en el software en el resto del sistema, diagnosticar deficiencias o causas de fallos en el software, o identificar las partes que requieren modificaciones.
* Capacidad para ser modificado: Capacidad del producto que permite que sea modificado de manera efectiva y eficiente sin introducir defectos o degradar el rendimiento.
* Capacidad para ser probado: Facilidad con la que se pueden establecer criterios de prueba para un sistema o componente y llevar a cabo pruebas para determinar si se cumplen dichos criterios.

**2-8.1.8 PORTABILIDAD**

La característica de "Portabilidad" se refiere a la capacidad del producto o componente de software para ser transferido de manera efectiva y eficiente entre diferentes entornos de hardware, software, operacionales o de uso. Esta característica se desglosa en las siguientes subcaracterísticas:

* Adaptabilidad: Capacidad del producto que le permite ser adaptado de manera efectiva y eficiente a diferentes entornos específicos de hardware, software, operaciones o uso.
* Capacidad para ser instalado: Facilidad con la que el producto se puede instalar y/o desinstalar con éxito en un entorno determinado.
* Capacidad para ser reemplazado: Capacidad del producto para ser utilizado en lugar de otro producto de software específico con el mismo propósito y en el mismo entorno.

**2-9 HERRAMIENTAS DE DESARROLLO WEB**

**2-9.1 SISTEMA GESTOR DE BASE DE DATOS MYSQL**

Sistema de gestión de bases de datos (DBMS) líder y ampliamente utilizado con arquitectura cliente-servidor. Compatible con SQL, ofrece transacciones, escalabilidad, soporte multiplataforma y diversos motores de almacenamiento. Utilizado en aplicaciones web, CMS, sistemas empresariales y más, se integra con varios lenguajes y herramientas de desarrollo. Dispone de ediciones como la gratuita Community Edition y la Enterprise para entornos empresariales, respaldadas por una activa comunidad y soporte en línea.

**2-9.1.1 FUNCIONAMIENTO**

Una base de datos MySQL opera mediante un servidor central que gestiona solicitudes de clientes. Los clientes, que pueden ser aplicaciones web o de escritorio, se conectan al servidor a través de conexiones de red. MySQL admite varios motores de almacenamiento, destacando InnoDB por su soporte de transacciones. Los clientes envían comandos SQL al servidor, que crea un plan de ejecución para acceder y manipular datos. La gestión de conexiones, seguridad, transacciones y administración de la base de datos son esenciales. La optimización y el mantenimiento aseguran un rendimiento eficiente.

**2-9.1.2 BASE DE DATOS RELACIONAL**

Una base de datos relacional, desde la perspectiva del usuario, puede describirse como un conjunto de tablas interconectadas que permiten el almacenamiento de información con el propósito de acceder y utilizar estos datos en el futuro. Estas bases de datos se fundamentan en el modelo de datos relacional para organizar y gestionar las tablas, el cual, a su vez, se apoya en conceptos derivados de la teoría de conjuntos para establecer relaciones entre los datos. Este enfoque estructurado y basado en tablas proporciona un medio efectivo para almacenar y recuperar información de manera lógica y coherente, siguiendo las formas normales de base de datos, como la Primera Forma Normal (1NF), la Segunda Forma Normal (2NF) y la Tercera Forma Normal (3NF). Estas formas normales son esenciales para garantizar la integridad y eficiencia de los datos, evitando la redundancia y asegurando una gestión óptima de la información.

**2-9.2 LENGUAJE DE PROGRAMACION PHP ("PREPROCESADOR DE HIPERTEXTO")**

Lenguaje de programación de código abierto, interpretado en tiempo real, integrado directamente en HTML y ejecutado en el servidor. A diferencia de lenguajes de script como JavaScript, PHP es de servidor, lo que oculta su código fuente en el navegador del cliente. Con una sintaxis similar a C, se utiliza en desarrollo web, scripts de línea de comandos y aplicaciones de interfaz gráfica, destacando por su versatilidad y amplio uso.

**2-9.3 MARCO DE DESARROLLO YIIFRAMEWORK II**

Framework PHP de código abierto, Yii2, destaca por su enfoque en seguridad, eficiencia y facilidad de uso. Con un patrón MVC, herramientas integradas de seguridad, rendimiento optimizado, soporte AJAX, generación automática de código y una comunidad activa, Yii2 es una opción robusta para el desarrollo eficiente de aplicaciones web de alta calidad.

**2-9.4 CARACTERISTICAS GENERALES DE ANGULAR**

Framework versátil para desarrollo web y móvil con arquitectura basada en componentes, TypeScript, enrutamiento SPA, inyección de dependencias, directivas personalizadas, manejo de formularios, comunicación con el servidor, pruebas unitarias y E2E, optimización de rendimiento y amplio soporte de la comunidad. Angular ofrece una estructura sólida, herramientas integradas y flexibilidad para construir aplicaciones web, móviles y de escritorio.

**2-9.5 CARACTERISTICAS GENERALES DE IONIC**

Marco de desarrollo para aplicaciones móviles híbridas, utiliza tecnologías web estándar (HTML, CSS, JavaScript) y se integra con Angular. Características destacadas: desarrollo multiplataforma, interfaz atractiva, acceso a capacidades nativas, desarrollo rápido con CLI, soporte PWA, comunidad activa, ecosistema de plugins, herramientas en tiempo real, personalización, facilidad de prueba y rendimiento optimizado. Ionic simplifica la creación de aplicaciones móviles con una apariencia moderna y funcionalidad nativa.

**2-9.6 CARACTERISTICAS GENERALES DE CAPACITOR**

Facilita la creación de aplicaciones móviles nativas con tecnologías web estándar (HTML, CSS, JavaScript). Características clave incluyen multiplataforma, acceso a API nativas, integración de plugins, personalización visual, soporte PWA, desarrollo en tiempo real, CLI, implementación sencilla, comunidad activa y código abierto. Capacitor permite a los desarrolladores web aplicar sus conocimientos para crear aplicaciones móviles, siendo una herramienta versátil y gratuita.

**2-9.7 HTML (LENGUAJE DE MARCADO DE HIPERTEXTO)**

Lenguaje estándar para crear páginas y aplicaciones web. Utiliza etiquetas angulares para estructurar contenido. Compuesto por elementos como encabezados, párrafos, imágenes y formularios. Permite enlaces, multimedia, formularios interactivos y metadatos. Recomendado junto con CSS para control avanzado de presentación. Compatible con la mayoría de navegadores. Evolucionado a HTML 5, es la base de la programación web junto con CSS y JavaScript.

**2-9.8 CSS (STYLE)**

CSS (Cascading Style Sheets - Hojas de Estilo en Cascada) es un lenguaje de diseño ampliamente utilizado en desarrollo web para controlar la apariencia y presentación visual de páginas y aplicaciones web. Su concepto central es separar la estructura (HTML) del contenido del diseño y estilo, lo que ofrece flexibilidad, consistencia y mantenibilidad en el diseño web.

**2-9.8 JAVASCRIPT:**

JavaScript es un lenguaje de programación esencial en el desarrollo web, destinado a añadir interactividad y dinamismo a las páginas web. Se trata de un lenguaje de secuencias de comandos interpretado por el navegador del usuario, lo que significa que se ejecuta en su dispositivo.

**2-9.8.1 JQUERY:**

jQuery es una biblioteca de JavaScript de código abierto que simplifica la manipulación del DOM y la interacción con JavaScript. Proporciona una serie de funciones y métodos predefinidos para realizar tareas comunes de manera más sencilla y eficiente.

**2-9.8.2 AJAX (ASYNCHRONOUS JAVASCRIPT AND XML):**

Técnica de desarrollo web con JavaScript que realiza solicitudes asincrónicas al servidor sin recargar la página. A pesar de su nombre, puede manejar diferentes formatos como JSON. Características clave: solicitudes asincrónicas, carga selectiva de partes de una página, mejora la interfaz de usuario, comunica con el servidor sin recargar la página, amplia compatibilidad con navegadores. Es esencial en aplicaciones web modernas para mejorar la experiencia del usuario.

**2-9.8.3 JQUERY UI**

Biblioteca de interfaz de usuario (UI) de código abierto basada en jQuery, simplifica la creación de interfaces interactivas en aplicaciones web. Ofrece widgets personalizables, interacción con el usuario, temas, efectos visuales y compatibilidad con navegadores. Mejora la usabilidad y experiencia del usuario, siendo valiosa en el desarrollo web.

**2-9.9 BOOTSTRAP**

Marco de diseño frontend de código abierto, mantenido por la comunidad. Facilita la creación de sitios web responsivos con sistema de rejilla, componentes CSS y JavaScript. Destaca por su fácil uso y personalización a través de temas, sin necesidad de escribir CSS o JavaScript personalizado. Solo se incorporan los archivos en el código HTML.