GESTIÓN DE LA CONFIGURACIÓN DEL SOFTWARE (GCS)

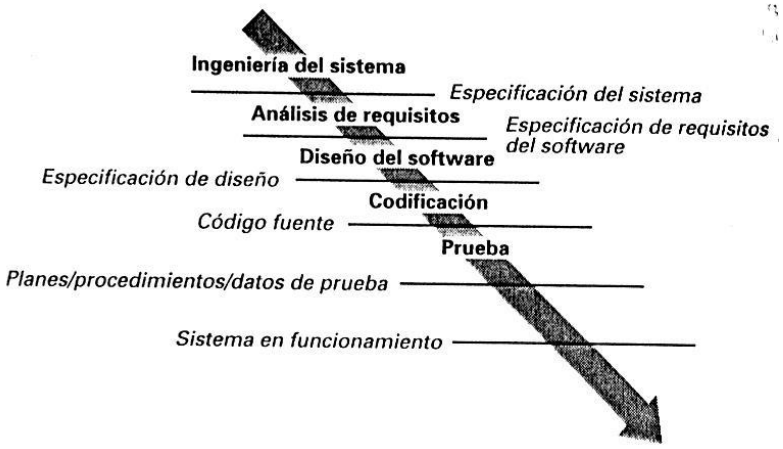
Es el proceso de identificar y definir los elementos en el sistema, controlando el cambio de estos elementos a lo largo de su ciclo de vida, registrando y reportando el estado de los elementos y las solicitudes de cambio, y verificando que los elementos estén completos y que sean los correctos.
Es una actividad de autoprotección que se aplica durante el proceso del software. El resultado del proceso de Software se puede dividir en: Programas (códigos y ejecutables). Documentos. Datos.

- Elementos de la GCS:

1. Especificación del sistema
2. Plan del proyecto software
3. a) Especificación de requerimientos del software
   b) Prototipo ejecutable o en papel
4. Manual de usuario preliminar
5. Especificación de diseño:
   a) Diseño preliminar
   b) Diseño detallado
6. Listados del código fuente
7. a) Planificación y procedimiento de prueba
   b) Casos de prueba y resultados registrados
8. Manuales de operación y de instalación
9. Programas ejecutables
   a) Módulos, código ejecutable
   b) Módulos enlazados
10. Descripción de la base de datos
    a) Esquema, modelos
    b)Datos iniciales
11. Manual de usuario
12. Documentos de mantenimiento
    a) Informes de problemas del software
    b) Peticiones de mantenimiento
    c) Órdenes de cambios de ingeniería
13. Estándares y procedimientos de ingeniería del software

El cambio se puede producir en cualquier momento, las actividades de la GCS sirven para: Identificar el cambio. Controlar el cambio. Garantizar que el cambio se implemente adecuadamente. Informar del cambio a todos aquellos que puedan estar afectados.

- Línea Base: es un concepto de GCS que nos ayuda a controlar los cambios.
  Definición de la IEEE: Una especificación o producto que se ha revisado formalmente y sobre el que se ha llegado a un acuerdo, y que de ahí en adelante sirve como base para un desarrollo posterior y que puede cambiarse solamente a través de procedimientos formales de control de cambio.



- Proceso de la GCS:
  - 1. Identificaciones de los objetos: Nombre: cadena de caracteres sin ambigüedad. Descripción: lista de elementos de datos que identifican: Tipo de ECS (documento, código fuente, datos), Identificador del proyecto, información de la versión y/o cambio.
  - 2. Control de versiones: Permite al usuario especificar configuraciones alternativas del sistema mediante la selección de versiones adecuadas (por ejemplo, asociando atributos que la identifican). Combinación de procedimientos y herramientas para gestionar las versiones de los ECS.
    - Repositorio (Se almacenan los archivos actualizados e históricos de cambio del proyecto). Versión (Determina un conjunto de archivos).
    - Master (Conjunto de archivos principales del proyecto).

- Abrir rama – branch (Bifurcación del máster para trabajar sobre dos ramas de forma independiente).
- Desplegar – check-out (Copia de trabajo local desde el repositorio).
- Publicar - Commit (Una copia de los cambios hechos a una copia local es escrita o integrada sobre repositorio).
- Conflicto (Problema entre las versiones de un mismo documento).
- Cambio – diff (Representa una modificación específica).
- Integración – Merge (Fusión entre dos ramas del proyecto).
- Actualización – sync o update (Integra los cambios que han sido hechos en el repositorio y las copias locales).

- 3. Control de cambios: A lo largo del proyecto los cambios son inevitables y el control es vital para el desarrollo del mismo. Combina los procedimientos humanos y las herramientas adecuadas para proporcionar un mecanismo para el control del cambio.
  La autoridad de control de cambios (ACC) evalúa:
  ¿Cómo impactará el cambio en el hardware?
  ¿Cómo impactará el cambio en el rendimiento?
  ¿Cómo alterará el cambio la percepción del cliente sobre el producto?
  ¿Cómo afectará el cambio a la calidad y a la fiabilidad?
- 4. Auditoría de la configuración: La identificación y el control de versiones y el control de cambio, ayudan al equipo de desarrollo de software a mantener un orden, pero sólo se garantiza hasta que se ha generado la orden de cambio. Cómo aseguramos que el cambio se ha realizado correctamente: Revisiones técnicas formales y auditorías de configuración.
- 5. Generación de informes de estado de la configuración: Responde ¿Qué pasó? ¿Quién lo hizo? ¿Cuándo pasó? ¿Qué más se vio afectado? La generación de informes de estado de la configuración desempeña un papel vital en el éxito del proyecto.

DISEÑO DE SOFTWARE

- Representación significativa de ingeniería de algo que se va a construir.
- Es el proceso creativo de transformación del problema en una solución.
- Es el núcleo técnico de la ingeniería de software.
  - Una vez que se analizan y especifican los requisitos del software, el diseño es la primera actividad técnica a realizar.
- Es independiente del modelo de proceso que se utilice.
- El diseño se centra en cuatro áreas importantes:
  - Datos, Arquitecturas, Interfaces y Componentes.

- Diseño de datos: transforma el modelo del dominio, obtenido del análisis, en estructura de datos, objetos de datos, relaciones, etc.
- Diseño arquitectónico: define la relación entre los elementos estructurales más importantes del software, los estilos arquitectónicos, patrones de diseño, etc., para lograr los requisitos del sistema. La información para realizar el diseño puede derivarse de la especificación, del modelo de análisis y de la interacción de los subsistemas definidos.
- Diseño a nivel componentes: Transforma los elementos estructurales de la arquitectura de software en una descripción procedimental de los componentes del software. La información obtenida de los modelos basados en clases, modelos de flujos, de comportamiento (DTE) sirven como base.
- Diseño de interface: Describe la forma de comunicación dentro del mismo sistema, con otros sistemas, y con las personas. Una interface implica flujo de información (datos o control) y comportamiento.

- El diseño es la etapa en la que se fomentará la calidad.
- Proporciona las representaciones del software susceptibles de evaluar respecto de la calidad.

- Sin diseño se corre el riesgo de construir un sistema inestable, el cual fallará cuando se realicen cambios pequeños, será difícil de probar.

- Características para la evaluación de un diseño:
  El diseño deberá implementar todos los requisitos explícitos del modelo de análisis, y deberá ajustarse a todos los requisitos implícitos que desea el cliente.
  Deberá ser una guía legible y comprensible para aquellos que generan código y para aquellos que comprueban y consecuentemente, dan soporte al software.
  Deberá proporcionar una imagen completa del software, enfrentándose a los dominios de comportamiento funcionales y de datos desde una perspectiva de implementación.

- Criterios técnicos para un buen diseño:
  1. Deberá presentar una estructura arquitectónica que: Se haya creado mediante patrones de diseño reconocibles. Que esté formado por componentes que exhiban características de buen diseño. Se implemente en forma evolutiva.
  2. Deberá ser modular, el software deberá dividirse lógicamente en elementos que realicen funciones y sub-funciones específicas.
  3. Deberá contener distintas representaciones de datos, arquitectura, interfaces y componentes (módulos).
  4. Deberá conducir a estructuras de datos adecuadas para los objetos que se van a implementar y que procedan de patrones de datos reconocibles.
  5. Deberá conducir a componentes que presenten características funcionales independientes.
  6. Deberá conducir a interfaces que reduzcan la complejidad de las conexiones entre los módulos y con el entorno externo.
  7. Deberá derivarse mediante un método repetitivo y controlado por la información obtenida durante el análisis de los requisitos del software.
  8. Debe representarse por medio de una notación que comunique de manera eficaz su significado.

DISEÑO

- El diseño es tanto un proceso como un modelo.
- El proceso de diseño es una secuencia de pasos que hacen posible que el diseñador describa todos los aspectos del software que se va a construir.

- Principios del diseño:
  1. En el proceso de diseño se deben tener en cuenta enfoques alternativos.
  2. Deberá poderse rastrear hasta el modelo de análisis.
  3. No deberá inventar nada que ya esté inventado.
  4. Deberá minimizar la distancia intelectual entre el software y el problema.
  5. Deberá presentar uniformidad e integración.
  6. Deberá estructurarse para admitir cambios.
  7. Deberá estructurarse para degradarse poco a poco, incluso cuando se enfrenta con datos, sucesos o condiciones de operaciones aberrantes.
  8. El diseño no es escribir código y escribir código no es diseñar.
  9. Deberá evaluarse en función de la calidad mientras se va creando, no después de terminado.
  10. Deberá revisarse para minimizar los errores conceptuales.

- Conceptos: Los conceptos de diseño de software fundamentales proporcionan el marco de trabajo necesario para conseguir que lo haga correctamente. Cada concepto proporciona al diseñador una base para aplicar los métodos de diseño. Los conceptos van a ayudar al diseñador a responder esas preguntas.
  - Abstracción: La noción de abstracción permite concentrarse en un problema a un nivel de generalización sin tener en cuenta los detalles irrelevantes de bajo nivel.
    Nivel de abstracción: A medida que profundizamos en la solución del problema se reduce el nivel de abstracción. Desde los requerimientos (abstractos) hasta llegar al código fuente.
    Tipos de abstracción:

        - Procedimental: Secuencia “nombrada” de instrucciones que tienen una funcionalidad específica. Ej.: Módulos (procedimientos, funciones, unidades, etc.).
        - De datos: Colección “nombrada” de datos que definen un objeto real Ej.: un registro que representa una persona con sus datos, el objeto persona en POO.
    - Arquitectura: Es la estructura general del software y las formas en que la estructura proporciona una integridad conceptual para un sistema.
    - Patrones: “Un patrón es una semilla de conocimiento, la cual tiene un nombre y transporta la esencia de una solución probada a un problema recurrente dentro de cierto contexto”. Dicho de otro modo, describe una estructura de diseño que resuelve un problema de diseño particular dentro de un contexto específico. La finalidad de cada patrón de diseño es proporcionar una descripción que le permita al diseñador determinar si es aplicable al trabajo actual, si se puede reutilizar, si puede servir como guía para desarrollar un patrón similar pero diferente en cuanto a la funcionalidad o estructura.
    - Modularidad: El software se divide en componentes nombrados y abordados por separado, llamados frecuentemente módulos, que se integran para satisfacer los requisitos del problema. Códigos Monolíticos (un único módulo) y Modularización excesiva (a nivel de instrucciones)
    - Ocultamiento de información: La información que está dentro un módulo es inaccesible a otros que no la necesiten. Se consigue una modularidad efectiva definiendo un conjunto de módulos independientes que se comunican entre sí intercambiando sólo la información necesaria para su funcionalidad.
    - Independencia funcional: Modularidad + Abstracción + Ocultamiento de Información. Es deseable que cada módulo trate una subfunción de requisitos y tenga una interfaz sencilla para que sea más fácil de desarrollar, mantener, probar y reusar. Se mide mediante la cohesión y el acoplamiento entre los módulos. Se busca una alta cohesión y bajo acoplamiento
        - Cohesión (Coherente): Se define como la medida de fuerza o relación funcional existente entre las sentencias o grupos de sentencias de un mismo módulo. Un módulo es altamente cohesivo cuando lleva a cabo solo una tarea dentro del procedimiento y requiere poca interacción con el resto de los procedimientos. Un módulo es poco cohesivo cuando realiza tareas muy diferentes o sin relación entre ellas
        - Tipos de Cohesión:
            - Funcional → Cuando las sentencias o grupos de sentencias de un mismo módulo están relacionadas en el desarrollo de una única función (la cohesión más alta).
            - Coincidental (Casual) → Cuando las sentencias llevan a cabo un conjunto de tareas que no están relacionadas o tienen poca relación (la cohesión más baja).
            - Lógica → Cuando las sentencias se relacionan lógicamente.
            - Temporal → Cuando las sentencias se deben ejecutar en el mismo intervalo de tiempo.
            - Procedimental → Cuando la sentencia tiene que ejecutarse en un orden especifico.
            - Comunicacional → Cuando los elementos de procesamiento se centran en los datos de entrada y salida.
        - Acoplamiento: Es la medida de interconexión entre los módulos. Punto donde se realiza la entrada o referencia y los datos que pasan a través de la interfaz. Una conectividad sencilla entre módulos da como resultado una conectividad más fácil.
        - Niveles de Acoplamiento:
            - Bajo: Acoplamiento de datos. Acoplamiento de marca.
            - Moderado: Acoplamiento de control.
            - Alto: Acoplamiento común. Acoplamiento externo. Acoplamiento de contenido.
    - Refinamiento: Se refina de manera sucesiva los niveles de detalle procedimentales. El refinamiento es un proceso de elaboración. Se comienza con una descripción de información de alto nivel de abstracción, sobre una funcionalidad puntual, sin conocer las características del funcionamiento, se va trabajando sobre la funcionalidad original proporcionando en cada iteración un mayor nivel de detalle hasta obtener todos los detalles necesarios para conocer su funcionamiento. La abstracción y

el refinamiento son conceptos complementarios. La abstracción permite especificar procedimientos y datos sin considerar detalles de grado menor. El refinamiento ayuda a revelar los detalles de grado menor mientras se realiza el diseño.

- Refabricación o rediseño (Refactoring): Técnica de reorganización (sugerida por las metodologías ágiles) que simplifica el diseño de un componente sin cambiar su función o comportamiento. Cuando se refabrica el diseño existente, se examina en busca de redundancias, elementos inútiles, algoritmos innecesarios, estructuras de datos inapropiadas, etc. Ej: Una primera iteración del diseño podría producir un componente con poca cohesión. El diseñador puede decidir que el componente debe refabricarse en componentes distintos para elevar la cohesión.

DISEÑO ARQUITECTONICO

- Define la relación entre los elementos estructurales, para lograr los requisitos del sistema. Es el proceso de identificar los subsistemas dentro del sistema y establecer el marco de control y comunicación entre ellos. Los grandes sistemas se dividen en subsistemas que proporcionan algún conjunto de servicios relacionados.
- La arquitectura afecta directamente a los requerimientos no funcionales:
  - Los más CRÍTICOS:
    - Rendimiento: deben agrupar las operaciones críticas en un grupo reducido de sub-sistemas (componentes de grano grueso, baja comunicación).
    - Protección: la arquitectura deberá diseñarse para que las operaciones relacionadas con la protección se localicen en un único sub-sistema (o grupo pequeño), para reducir los costos y problemas de validación de la protección.
    - Seguridad: se debe utilizar una arquitectura en capas, protegiendo los recursos más críticos en las capas más internas.
    - Disponibilidad: la arquitectura se deberá diseñar con componentes redundantes para que sea posible el reemplazo sin detener el sistema, arquitectura muy tolerante a fallos.
    - Mantenibilidad: la arquitectura del sistema debe diseñarse con componentes autocontenidos de grano fino que puedan modificarse con facilidad.

1- Organización del sistema: representa la estrategia básica usada para estructurar el sistema. Los subsistemas de un sistema deben intercambiar información de forma efectiva (Todos los datos compartidos, se almacenan en una base de datos central. Cada subsistema mantiene su información y los intercambia entre los subsistemas).

- Estilos organizacionales (PATRONES arquitectónicos):
  - Repositorios: La mayoría de los sistemas que usan grandes cantidades de datos se organizan alrededor de una base de datos compartida (repositorio). Los datos son generados por un subsistema y utilizados por otros subsistemas. Ejemplo: Sistemas de gestión, Sistemas CAD, Herramientas Case, etc.

    Ventajas: Forma eficiente de compartir grandes cantidades de datos, no hay necesidad de transmitir datos de un subsistema a otro. Los subsistemas que producen datos no deben saber cómo se utilizan. Las actividades de backup, protección, control de acceso están centralizadas. El modelo compartido es visible a través del esquema del repositorio. Las nuevas herramientas se integran de forma directa, ya que son compatibles con el modelo de datos.

    Desventajas: Los subsistemas deben estar acordes a los modelos de datos del repositorio. Esto en algunos casos puede afectar el rendimiento. La evolución puede ser difícil a medida que se genera un gran volumen de información de acuerdo con el modelo de datos establecido. La migración de estos modelos puede ser muy difícil, en algunos casos imposible. Diferentes subsistemas pueden tener distintos requerimientos de protección o políticas de seguridad y el modelo de repositorio impone las mismas para todos. Es difícil distribuir el repositorio en varias máquinas, existen repositorios centralizados lógicamente, pero pueden ocasionar problemas de redundancia e inconsistencias.

- Cliente – Servidor: Es un modelo donde el sistema se organiza como un conjunto de servicios y servidores asociados, más unos clientes que utilizan los servicios.

  Componentes: Un conjunto de servidores que ofrecen servicios, otros sistemas. Un conjunto de clientes que llaman a los servicios. Una red que permite a los clientes acceder a los servicios. Caso particular cuando los servicios y el cliente corren en la misma máquina. Los clientes conocen el nombre del servidor y el servicio que brinda, pero el servidor no necesita conocer al cliente.
- Arquitectura en capas: El sistema se organiza en capas, donde cada una de ellas presenta un conjunto de servicios a sus capas adyacentes.

  Ventajas: Soporta el desarrollo incremental. Es portable y resistente a cambios. Una capa puede ser reemplazada siempre que se mantenga la interfaz, y si varía la interfaz se genera una capa para adaptarlas. Permite generar sistemas multiplataforma, ya que solamente las capas más internas son dependientes de la plataforma (se genera una capa interna para cada plataforma).

  Desventajas: Difícil de estructurar. Las capas internas proporcionas servicios que son requeridos por todos los niveles. Los servicios requeridos por el usuario pueden estar brindados por las capas internas teniendo que atravesar varias capas adyacentes. Si hay muchas capas, un servicio solicitado de la capa superior puede tener que ser interpretado varias veces en diferentes capas.

2- Descomposición modular: Una vez organizado el sistema, a los subsistemas los podemos dividir en módulos, se puede aplicar los mismos criterios que vimos en la organización, pero la descomposición modular es más pequeña y permite utilizar otros estilos alternativos.
   - Estrategias de descomposición modular:
     - Descomposición orientada a flujo de funciones: Conjunto de módulos funcionales (ingresan datos y los transforman en salida). En un Modelo orientado a flujo de funciones, los datos fluyen de una función a otra y se transforman a medida que pasan por una secuencia de funciones hasta llegar a los datos de salida. Las transformaciones se pueden ejecutar en secuencial o en paralelo.
     - Descomposición orientada a objetos: Conjunto de objetos que se comunican. Un modelo arquitectónico orientado a objetos estructura al sistema en un conjunto de objetos débilmente acoplados y con interfaces bien definidas.
   - Definiciones:
     - Subsistema: Es un sistema en sí mismo cuyo funcionamiento no depende de los servicios proporcionados por otros. Los subsistemas se componen de módulos con interfaces definidas que se utilizan para comunicarse con otro subsistema.
     - Módulo: Es un componente de un subsistema que proporciona uno o más servicios a otros módulos. A su vez utiliza servicios proporcionados por otros módulos. Por lo general no se los considera un sistema independiente.

3- Modelos de control: En un sistema, los subsistemas están controlados para que sus servicios se entreguen en el lugar correcto en el momento preciso.
   - Los modelos de control a nivel arquitectónico:
     - Control Centralizado: Un subsistema tiene la responsabilidad de iniciar y detener otro subsistema. Un subsistema se diseña como controlador y tiene la responsabilidad de gestionar la ejecución de otros subsistemas, la ejecución puede ser secuencial o en paralelo. Modelo de llamada y retorno (Modelo de subrutinas descendentes. Aplicable a modelos secuenciales) Modelo de gestor (Un gestor controla el inicio y parada coordinado con el resto de los procesos. Aplicable a modelos concurrentes)
     - Control Basado en Eventos: Cada subsistema responde a eventos externos al subsistema. Se rigen por eventos generados externamente al proceso. Eventos (Señal binaria. Un valor dentro de un rango. Una entrada de un comando. Una selección del menú). Modelos de sistemas dirigidos por eventos (Modelos de transmisión (Broadcast): es un evento que se

trasmite a todos los subsistemas, cualquier subsistema programado para manejar ese evento lo atenderá. Modelo dirigido por interrupciones: se utilizan en sistemas de tiempo real donde las interrupciones externas son detectadas por un manejador de interrupciones y se envía a algún componente para su procesamiento).

4- Arquitectura de los Sistemas Distribuidos: es un sistema en el que el procesamiento de información se distribuye sobre varias computadoras.
   - Tipos genéricos de sistemas distribuidos: Cliente-Servidor. Componentes distribuidos.
   - Características de los sistemas distribuidos:
     - Compartir recursos: un sistema distribuido permite compartir recursos
     - Apertura: son sistemas abiertos y se diseñan con protocolos estándar para simplificar la combinación de los recursos-
     - Concurrencia: varios procesos pueden operar al mismo tiempo sobre diferentes computadoras.
     - Escalabilidad: La capacidad puede incrementarse añadiendo nuevos recursos para cubrir nuevas demandas.
     - Tolerancia a fallos: la disponibilidad de varias computadoras y el potencial para reproducir información hace que los sistemas distribuidos sean más tolerantes a fallos de funcionamiento de hardware y software.
   - Desventajas:
     - Complejidad: Son más complejos que los centralizados, además del procesamiento hay que tener en cuenta los problemas de la comunicación y sincronización entre los equipos.
     - Seguridad: Se accede al sistema desde varias computadoras generando tráfico en la red que puede ser intervenido.
     - Manejabilidad: Las computadoras del sistema pueden ser de diferentes tipos y diferentes S.O. lo que genera más dificultades para gestionar y mantener el sistema
     - Impredecibilidad: La respuesta depende de la carga del sistema y del estado de la red, lo que hace que el tiempo de respuesta varié entre una petición y otra.
   - Arquitectura Multiprocesador: El sistema de software está formado por varios procesos que pueden o no ejecutarse en procesadores diferentes. La asignación de los procesos a los procesadores puede ser predeterminada o mediante un dispatcher. Es común en sistemas grandes de tiempo real que recolectan información, toman decisiones y envían señales para modificar el entorno.
   - Arquitectura Cliente-Servidor: Una aplicación se modela como un conjunto de servicios proporcionado por los servidores y un conjunto de clientes que usan estos servicios. Los clientes y servidores son procesos diferentes. Los servidores pueden atender varios clientes. Un servidor puede brindar varios servicios. Los clientes no se conocen entre sí.
     - Clasifican en niveles:
       - Dos Niveles: Cliente ligero (el procesamiento y gestión de datos se lleva a cabo en el servidor). Cliente pesado (el cliente implementa la lógica de la aplicación y el servidor solo gestiona los datos).
       - Multinivel: La presentación, el procesamiento y la gestión de los datos son procesos lógicamente separados y se pueden ejecutar en procesadores diferentes.
   - Arquitectura de Componentes Distribuidos: Diseña al sistema como un conjunto de componentes u objetos que brindan una interfaz de un conjunto de servicios que ellos suministran. Otros componentes u objetos solicitan estos servicios. No hay distinción tajante entre clientes y servidores. Los componentes pueden distribuirse en varias máquinas a través de la red utilizando un middleware como intermediario de peticiones.
   - Computación Distribuida inter-organizacional: Una organización tiene varios servidores y reparte su carga computacional entre ellos. Extender este concepto a varias organizaciones.
     - Arquitecturas Peer-to-Peer (P2P): Sistemas descentralizados en los que el cálculo puede llevarse a cabo en cualquier nodo de la red. Se diseñan para aprovechar la ventaja de la potencia computacional y el almacenamiento a través de una red. Pueden utilizar una arquitectura descentralizada (donde cada nodo rutea los paquetes a sus vecinos hasta

encontrar el destino) Semi-centralizada (donde un servidor ayuda a conectarse a los nodos o coordinar resultados). Ejemplos: Torrents, Skype, ICQ, SETI@Home.

- Arquitectura de sistemas orientados a servicios:
    - Servicio: Representación de un recurso computacional o de información que puede ser utilizado por otros programas. Un servicio es independiente de la aplicación que lo utiliza. Un servicio puede ser utilizado por varias organizaciones. Una aplicación puede construirse enlazando servicios. Las arquitecturas de las aplicaciones de servicios web son arquitecturas débilmente acopladas.
    - Funcionamiento: Un proveedor de servicios oferta servicios definiendo su interfaz y su funcionalidad. Para que el servicio sea externo, el proveedor publica el servicio en un "servicio de registro" con información del mismo. Un solicitante enlaza este servicio a su aplicación, es decir que el solicitante incluye el código para invocarlo y procesa el resultado del mismo.
    - Los estándares fundamentales que permiten la comunicación entre servicios:
        - SOAP (simple Object Access Protocol) Define una organización para intercambio de datos estructurados entre servicios web.
        - WSDL (Web Service Description Lenguage) Define como puede representarse las interfaces web.
        - UDDI (Universal Desscription Discovery and Integration) Estándar de búsqueda que define como puede organizarse la información de descripción de servicios.