**MANUAL GESTION DE SOFTWARE**

**1.- CONTENIDO**

1. VISIÓN GENERAL

1.1. Introducción.

1.1.1. Preguntas frecuentes sobre la ingeniería de software

1.1.2. Responsabilidad profesional y ética

1.2. Sistemas socio-técnicos.

1.2.1. Propiedades emergentes de los sistemas

1.2.2. Ingeniería de sistemas

1.2.3. Organizaciones, personas y sistemas informáticos

1.2.4. Sistemas heredados

1.3. Sistemas críticos.

1.3.1. Un sistema de seguridad critico sencillo

1.3.2. Confiabilidad de un sistema

1.3.3. Disponibilidad y fiabilidad

1.3.4. Seguridad

1.3.5. Protección

1.4. Procesos del software

1.4.1. Modelos del proceso de software

1.4.2. Iteración de procesos

1.4.3. Actividades del proceso

1.4.4. El Proceso Unificado de Rational

1.4.5. Ingeniería del Software Asistida por computadora

1.5. Gestión de proyectos

1.5.1. Actividades de gestión

1.5.2. Planificación del proyecto

1.5.3. Calendarización del proyecto

1.5.4. Gestión de riesgos

1.6. La industria del Software.

2. INGENIERÍA REQUERIMIENTOS

2.1. Requerimientos del software

2.1.1. Requerimientos funcionales y no funcionales

2.1.2. Requerimientos del usuario

2.1.3. Requerimientos del sistema

2.1.4. Especificación de la interfaz

2.1.5. El documento de requerimientos del software

2.2. Procesos de la ingeniería de requerimientos.

2.2.1. Estudios de viabilidad

2.2.2. Obtención y análisis de requerimientos

2.2.3. Validación de requerimientos

2.2.4. Gestión de requerimientos

2.3. Modelos del sistema.

2.3.1. Modelos de contexto

2.3.2. Modelos de comportamiento

2.3.3. Modelos de datos

2.3.4. Modelos de Objetos

2.3.5. Métodos estructurados

2.4. Especificación de sistemas críticos.

2.4.1. Especificación dirigida por riesgos

2.4.2. Especificación de la seguridad

2.4.3. Especificación de la protección

2.4.4. Especificación de la fiabilidad del software

2.5. Especificación formal.

2.5.1. Especificación formal en el proceso del software

2.5.2. Especificación de interfaces de subsistemas

2.5.3. Especificación del comportamiento

3. DISEÑO

3.1. Diseño arquitectónico.

3.1.1. Decisiones de diseño arquitectónico

3.1.2. Organización del sistema

3.1.3. Estilos de descomposición modular

3.1.4. Estilos de control

3.1.5. Arquitecturas de referencia

3.2. Arquitecturas de sistemas distribuidos.

3.2.1. Arquitecturas multiprocesador

3.2.2. Arquitecturas cliente servidor

3.2.3. Arquitecturas de objetos distribuidos

3.2.4. Computación distribuida interorganizacional

3.3. Arquitecturas de aplicaciones.

3.3.1. Sistemas de procesamiento de datos

3.3.2. Sistemas de procesamiento de transacciones

3.3.3. Sistemas de procesamiento de eventos

3.3.4. Sistemas de procesamiento de lenguajes

3.4. Diseño orientados a objetos.

3.4.1. Objetos y clases

3.4.2. Un proceso de diseño orientado a objetos

3.4.3. Evolución del diseño

3.5. Diseño de software en tiempo real.

3.5.1. Diseño del sistema de tiempo real

3.5.2. Sistemas operativos de tiempo real

3.5.3. Sistemas de monitorización y control

3.5.4. Sistemas de adquisición de datos

3.6. Diseño de interfaces de usuario.

3.6.1. Asuntos de diseño

3.6.2. El proceso de diseño de la interfaz de usuario

3.6.3. Análisis del usuario

3.6.4. Prototipado de la interfaz de usuario

3.6.5. Evaluación de la interfaz

4. DESARROLLO

4.1. Desarrollo.

4.1.1. Métodos ágiles

4.1.2. Programación extrema

4.1.3. Desarrollo rápido de aplicaciones

4.1.4. Prototipado del software

4.2. Reutilización del software.

4.2.1. El campo de la reutilización

4.2.2. Patrones de diseño

4.2.3. Reutilización basada en generadores

4.2.4. Marcos de trabajo de aplicaciones

4.2.5. Reutilización de sistemas de aplicaciones

4.3. Ingeniería del software basada en componentes.

4.3.1. Componentes y modelos de componentes

4.3.2. El proceso CBSE

4.3.3. Composición de componentes

4.4. Desarrollo de sistemas críticos.

4.4.1. Procesos confiables

4.4.2. Programación confiable

4.4.3. Tolerancia a defectos

4.4.4. Arquitecturas tolerantes a defectos

4.5. Evolución del software.

4.5.1. Dinámica de evolución de los programas

4.5.2. Mantenimiento del software

4.5.3. Procesos de evolución

4.5.4. Evolución de sistemas heredados

5. VERIFICACIÓN Y VALIDACIÓN

5.1. Verificación y validación.

5.1.1. Planificación de la verificación y validación

5.1.2. Inspecciones de software

5.1.3. Análisis estático automatizado

5.1.4. Verificación y métodos formales

5.2. Pruebas del software.

5.2.1. Pruebas del sistema

5.2.2. Pruebas de componentes

5.2.3. Diseño de casos de prueba

5.2.4. Automatización de las pruebas

5.3. Validación de sistemas críticos.

5.3.1. Validación de la fiabilidad

5.3.2. Garantía de la seguridad

5.3.3. Valoración de la protección

5.3.4. Argumentos de confiabilidad y seguridad

6. GESTIÓN DE PERSONAL

6.1. Gestión de personal.

6.1.1. Selección de personal

6.1.2. Motivación

6.1.3. Gestión de grupos

6.1.4. El modelo de madures de la capacidad del personal

6.2. Estimación de costes del software.

6.2.1. Productividad

6.2.2. Técnicas de estimación

6.2.3. Modelado algorítmico de costes

6.2.3.1. El modelo de COCOMO

6.2.3.2. El modelo Putnam

6.2.3.3. Modelos algorítmicos de costes en la planificación

6.2.4. Duración y personal del proyecto

6.3. Gestión de la calidad.

6.3.1. Calidad de proceso y producto

6.3.2. Garantía de la calidad y estándares

6.3.2.1. ISO 9000

6.3.2.2. Estándares de documentación

6.3.3. Planificación de la calidad

6.3.4. Control de la calidad

6.3.5. Medición y métricas del software

6.3.5.1. El proceso de medición

6.3.5.2. Métricas del producto

6.3.5.3. Análisis de las mediciones

6.4. Mejora de procesos.

6.4.1. Calidad de producto y proceso

6.4.2. Clasificación de los procesos

6.4.3. Medición del proceso

6.4.4. Análisis y modelado de procesos

6.4.5. Cambio en los procesos

6.4.6. El marco de trabajo para la mejora de procesos CMMI

6.5. Gestión de configuraciones.

6.5.1. Planificación de la gestión de configuraciones

6.5.2. Gestión del cambio

6.5.3. Gestión de versiones y entregas

6.5.4. Construcción del sistema

6.5.5. Herramientas CASE para gestión de configuraciones

**2.- BIBLIOGRAFÍA**

Fundamentals of Software Engineering - RajibMall - Editorial Prentice Hall of India – 2º Edición 2004.

• Ingeniería del Software - Roger S. Pressman - Editorial Mcgraw-hill - Sexta edición 2000

Proyecto de Ingeniería de Software. Consultado el 2018 de https://www.fing.edu.uy/inco/cursos/ingsoft/pis/memoria/index.htm

**MANUAL GESTION DE SOFTWARE**

**1. VISIÓN GENERAL**

**1.1. Introducción.**

El software de computadora es el producto que construyen los programadores profesionales y al que después le dan mantenimiento durante un largo tiempo. Incluye programas que se ejecutan en una computadora de cualquier tamaño y arquitectura, contenido que se presenta a medida de que se ejecutan los programas de cómputo e información descriptiva tanto en una copia dura como en formatos virtuales que engloban virtualmente a cualesquiera medios electrónicos. La ingeniería de software está formada por un proceso, un conjunto de métodos (prácticas) y un arreglo de herramientas que permite a los profesionales elaborar software de cómputo de alta calidad.

**1.1.1. Preguntas frecuentes sobre la ingeniería de software**

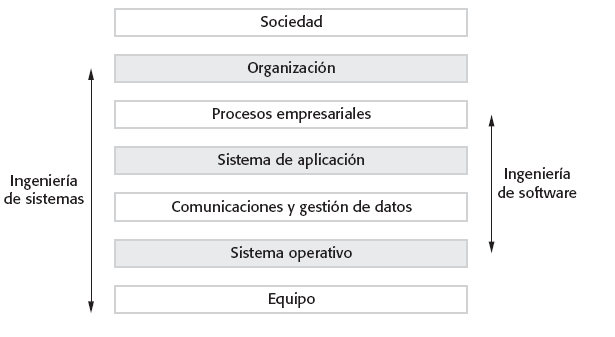
* **¿Quién lo hace?** Los ingenieros de software elaboran y dan mantenimiento al software, y virtualmente cada persona lo emplea en el mundo industrializado, ya sea en forma directa o indirecta.
* **¿Por qué es importante?** El software es importante porque afecta a casi todos los aspectos de nuestras vidas y ha invadido nuestro comercio, cultura y actividades cotidianas.
* **¿Cuáles son los pasos?** El software de computadora se construye del mismo modo que cualquier producto exitoso, con la aplicación de un proceso ágil y adaptable para obtener un resultado de mucha calidad, que satisfaga las necesidades de las personas que usarán el producto. En estos pasos se aplica el enfoque de la ingeniería de software.
* **¿Cuál es el producto final?** Desde el punto de vista de un ingeniero de software, el producto final es el conjunto de programas, contenido (datos) y otros productos terminados que constituyen el software de computadora.
* **¿Cómo me aseguro de que lo hice bien?** Lea el resto de este libro, seleccione aquellas ideas que sean aplicables al software que usted hace y aplíquelas a su trabajo.

**1.1.2. Responsabilidad profesional y ética**

La ingeniería del software se lleva a cabo dentro de un marco legal y social que limita la libertad de los ingenieros.

**1.2. Sistemas socio-técnicos.**

Los sistemas socio técnicos son tan complejos que es prácticamente imposible entenderlos como un todo. En vez de ello, deben verse como capas:

****

**1.2.1. Propiedades emergentes de los sistemas**

En muchos sistemas complejos se generan propiedades emergentes, que son el producto del conjunto de las relaciones entre las partes. Estas propiedades están basadas en conductas simples de éstas y que, como decía Aristóteles, las propiedades del todo generado es mayor que la suma de las propiedades individuales de dichos elementos que conforman el sistema.

**1.2.2. Ingeniería de sistemas**

La ingeniería de sistemas es un modo de enfoque interdisciplinario que permite estudiar y comprender la realidad, con el propósito de implementar u optimizar sistemas complejos.

**1.2.3. Organizaciones[[1]](#footnote-2), personas y sistemas informáticos**

El funcionamiento organizacional debe ser estudiado en relación con las transacciones continuas con el medio ambiente que lo envuelve. Esa relación conlleva los conceptos de sistemas, subsistemas y supersistemas: los sistemas sociales, como sistemas abiertos, dependen de otros sistemas sociales; su caracterización como subsistemas, sistemas o supersistemas es relativa a su grado de autonomía en la ejecución de sus funciones y a los intereses particulares del investigador. Desde el punto de vista de la sociedad, la organización es un subsistema de uno o más sistemas mayores y su vinculación o integración con ellos afecta su modo de operación y su nivel de actividad.

**1.2.4. Sistemas heredados**

Un sistema heredado (o sistema legacy) es un sistema informático (equipos informáticos o aplicaciones) que ha quedado anticuado pero que sigue siendo utilizado por el usuario (generalmente, una organización o empresa) y no se quiere o no se puede reemplazar o actualizar de forma sencilla.

**1.3. Sistemas críticos.**

Son aquellos sistemas en los que un fallo puede ocasionar consecuencias graves en el entorno en el que está trabajando, tanto humanas como materiales (desfibriladores cardíacos, sistemas de control de aviónica, sistemas de control de las plantas de energía nuclear, sistemas antifrenado de los automóviles, etc).

**1.3.1. Un sistema de seguridad critico sencillo**

Son aquellos en los que un fallo puede ocasionar daños para la vida humana o un gran impacto para el medio ambiente. (Ej. El sistema de control de una planta nuclear).

**1.3.2. Confiabilidad de un sistema**

Es una de las características más importantes que debe poseer este tipo de sistemas, ya que sin ello puede producirse una pérdida de confianza por parte del cliente, los costes económicos y de información de las pérdidas pueden ser enormes. Para lograr una confiabilidad perfecta se deben cumplir estas dimensiones.

**1.3.3. Disponibilidad y fiabilidad**

El sistema esté activo y en funcionamiento y que sea capaz de proporcionar servicios útiles en todo momento.

**1.3.4. Seguridad**

Evite consecuencias catastróficas sobre su entorno.

**1.3.5. Protección**

El sistema es capaz de resistir ataques premeditados o accidentales, la protección debe cumplir: integridad, control de accesos y autenticación.

**1.4. Procesos del software**

Un proceso de desarrollo de software tiene como propósito la producción eficaz y eficiente de un producto software que reúna los requisitos del cliente. Dicho proceso, en términos globales se muestra en la siguiente figura:

****

**1.4.1. Modelos del proceso de software**

Cada proyecto de software requiere de una forma de particular de abordar el problema. Las propuestas comerciales y académicas actuales promueven procesos iterativos, donde en cada iteración puede utilizarse uno u otro modelo de proceso, considerando un conjunto de criterios, como el: cascada, evolutivo, incremental y espiral.

**1.4.2. Iteración de procesos**

Iteración significa el acto de repetir un proceso con la intención de alcanzar una meta deseada, objetivo o resultado.

**1.4.3. Actividades del proceso**

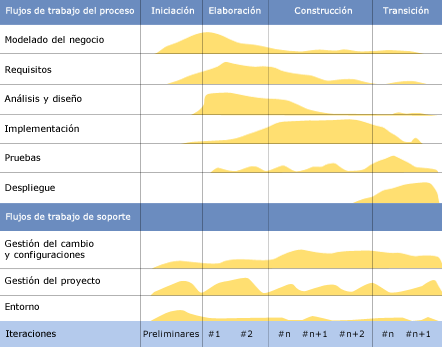
Las actividades del procesos constituyen el conjunto de pasos ha seguir para desarrollar el producto, como: capturar los requerimientos, analizar, diseñar, implementar, probar.

**1.4.4. El Proceso Unificado de Rational**

El Proceso Racional Unificado o RUP (por sus siglas en inglés de Rational Unified Process) es un proceso de desarrollo de software desarrollado por la empresa Rational Software, actualmente propiedad de IBM.1​ Junto con el Lenguaje Unificado de Modelado (UML), constituye la metodología estándar más utilizada para el análisis, diseño, implementación y documentación de sistemas orientados a objetos.

El ciclo de vida RUP es una implementación del desarrollo en espiral. Fue creado ensamblando los elementos en secuencias semi-ordenadas. El ciclo de vida organiza las tareas en fases e iteraciones.

RUP divide el proceso en cuatro fases, dentro de las cuales se realizan pocas pero grandes y formales iteraciones en número variable según el proyecto. En la Figura muestra cómo varía el esfuerzo asociado a las disciplinas según la fase en la que se encuentre el proyecto RUP.

****

**1.4.5. Ingeniería del Software Asistida por computadora**

La ingeniería de software asistida por ordenador (o CASE) es un conjunto de herramientas de programación que utilizan una interfaz común para diseñar, desarrollar y depurar software.

**1.5. Gestión de proyectos**

Gestión de Proyectos se refiera a todas las actividades que se realizan para cumplir con un fin principal definido, en un tiempo establecido utilizando recursos tanto humanos como materiales y para el cual se debe tener presupuestados los costos en que se incurrirán.

**1.5.1. Actividades de gestión**

El objetivo principal de la Gestión de Proyectos es administrar, planificar, coordinar, seguimiento y control de todas las actividades y los recursos asignados para la ejecución del proyecto de una forma que se pueda cumplir con el alcance en el tiempo establecido y con los costos presupuestados.

**1.5.2. Planificación del proyecto**

Esta etapa define el alcance de lo que se quiere hacer esta planificación debe ser conciso y expresar de forma precisa

La planificación se refiere a la identificación de actividades, hitos y entregable del proyecto, incluso posibilidades de mitigación de riesgos.

Las tres restricciones principales o posibilidades de alteración en los proyectos son el alcance, tiempo y costos. Es así que aquí definimos de una forma clara el alcance que queremos conseguir (objetivos), en que tiempo lo haremos (cronograma) y el coste que tendrá lograrlo (presupuesto).

Aquí se establece como el equipo de trabajo deberá satisfacer las restricciones de prestaciones, la planificación temporal y el coste. Una planificación bien detallada disminuye la aparición de sorpresas.

Posteriormente se hace la la redacción de la propuesta específica objeto, objetivos, alcance, calidad y estima riesgos del proyecto, y describe cómo él se llevaría a cabo. Incluye también estimaciones de costo y tiempo, y efectúa la integración de todo lo anterior con lo que sigue, y justifica -evaluando credenciales y circunstancias- por qué el contrato del proyecto se debe dar a una organización o equipo en particular, y bajo qué condiciones.

**1.5.3. Calendarización del proyecto**

Es el conjunto de actividades que se ejecutaran durante el proyecto esta deberá tener de manera visible hitos y actividades de control.

**1.5.4. Gestión de riesgos[[2]](#footnote-3)**

Se debe prever mitigar riesgos para evitar el fracaso del proyecto implementando monitoreo y seguimiento continuo de todas las actividades planificadas que se están ejecutando.

**1.6. La industria del Software.**

El sector de software y servicios informáticos (SSI) se engloba dentro de lo que se conoce como industrias de las “tecnologías de la información” (TI), que, de acuerdo con la OECD (1997) .

El output de la industria de software puede ser clasificado como producto o como servicio. Mientras los ingresos derivados del desarrollo de productos provienen principalmente de la venta de licencias para su uso, los ingresos generados por los servicios provienen de actividades diversas como el diseño y desarrollo de soluciones a medida, la implementación y adaptación de productos de terceros, servicios de consultoría, capacitación, instalación y mantenimiento de software, etc.

**2. INGENIERÍA[[3]](#footnote-4) REQUERIMIENTOS**

**2.1. Requerimientos del software**

En aplicaciones de software y hardware, los requerimientos de software son las características que debe tener el software instalado en una computadora para poder soportar y/o ejecutar una aplicación o un dispositivo específicos. Contrasta con los requerimientos de hardware.

**2.1.1. Requerimientos funcionales y no funcionales**

Los requerimientos funcionales son declaraciones de los servicios que **debe proporcionar** el sistema, de la manera en que éste debe reaccionar a entradas particulares. O también pueden declarar explícitamente lo que el sistema **no debe hacer**.

Los requerimientos no funcionales son **restricciones** de los servicios o **funciones** ofrecidos por el sistema. Incluyen restricciones de tiempo, sobre el proceso de desarrollo y estándares. Dentro de estos requerimientos encontramos todo lo referente a **fiabilidad**, el **tiempo de respuesta** y la **capacidad de almacenamiento**.

**2.1.2. Requerimientos del usuario**

Son declaraciones, en lenguaje natural y en diagramas, de los servicios que se espera que el sistema proporcione y de las restricciones bajo las cuales debe funcionar.

**2.1.3. Requerimientos del sistema**

Estos requerimientos establecen con detalle las funciones, servicios y restricciones operativas del sistema. El documento de requerimientos del sistema deberá ser preciso, y definir exactamente lo que se va a hacer.

**2.1.4. Especificación de la interfaz**

Documento que describe los diseños de interfaz del Software a Desarrollar.

**2.1.5. El documento de requerimientos del software**

A continuación se describe el contenido de este documento:

documento de requerimientos

**Fecha:** <Fecha en que se hace el Documento>

**Versión:** <Número de Versión del Documento>

**Responsables:** <Nombre de las Personas que realizan este documento>

Índice

[1 Sistema a Construir: 1](#_Toc195677905)

[2 Usuarios: 1](#_Toc195677906)

[3 Requerimientos Funcionales: 1](#_Toc195677907)

[3.1 <Funcionalidad 1>: 1](#_Toc195677908)

[4 Requerimientos No Funcionales: 1](#_Toc195677909)

[4.1 <Requerimiento no Funcional 1>: 1](#_Toc195677910)

[5 Restricciones: 1](#_Toc195677911)

[5.1 <Restricción 1> 1](#_Toc195677912)

[6 Interfaces: 1](#_Toc195677913)

[6.1 Interfaces de Usuario: 2](#_Toc195677914)

[6.2 Interfaces con Hardware: 2](#_Toc195677915)

[6.3 Interfaces con Software: 2](#_Toc195677916)

[6.4 Interfaces de Comunicación: 2](#_Toc195677917)

# Sistema a Construir:

<Descripción del Sistema a construir, de sus principales funcionalidades y de los posibles usos del producto>

# Usuarios:

<Caracterización de los principales usuarios del Sistema a Construir>

# Requerimientos Funcionales:

<Se detallan todos los Requerimientos Funcionales del Sistema>

## <Funcionalidad 1>:

<Explicación de la Funcionalidad 1 del Sistema>

# Requerimientos No Funcionales:

<Se detallan los Requerimientos no Funcionales del Sistema, éstos pueden tratarse de requerimientos necesarios para el uso del sistema, de performance, de mantenimiento, de soporte, etc >

## <Requerimiento no Funcional 1>:

<Explicación del Requerimiento no Funcional 1 del Sistema>

# Restricciones:

<Detalle de las Restricciones impuestas sobre el proyecto pero que no son restricciones sobre el producto, ejemplos son proceso de software a seguir, fechas de entrega, hitos de entrega fijos, etc >

## <Restricción 1>

<Explicación de la Restricción >

# Interfaces:

<Se detallan las interfaces que debe proveer o utilizar la aplicación>

## Interfaces de Usuario:

<Describe las interfaces de usuario que deben ser implementadas>

## Interfaces con Hardware:

<Incluye cualquier interfaz con el Hardware que debe ser provista por el software, si es que son necesarias>

## Interfaces con Software:

<Describe las interfaces del sistema con cualquier otro sistema de software, pueden ser componentes reusados, componentes del mercado o cualquier otro tipo con el cual el sistema debe interactuar>

## Interfaces de Comunicación:

<Describe cualquier interfaz de comunicación con otro sistema o dispositivo, tales como redes, dispositivos remotos, etc>

**2.2. Procesos de la ingeniería de requerimientos.**

Es el proceso de recopilar, analizar y verificar las necesidades del cliente para un sistema de software.

**2.2.1. Estudios de viabilidad**

Los resultados del estudio de viabilidad deberían ser un informe que recomiende si merece o no la pena seguir con la ingeniería de requerimientos y el proceso de desarrollo del sistema.

**2.2.2. Obtención y análisis de requerimientos**

1. Descubrimiento de los requerimientos: es el proceso de interactuar con los stakeholders[[4]](#footnote-5) para recopilar sus requerimientos, los de su dominio y la documentación. Se realiza mediante entrevistas, cuestionarios, prototipos, ..etc.

2. Clasificación y organización de los requerimientos: toma la recopilación no estructurada de los requerimientos, grupos relacionados de requerimientos y los organiza en grupos coherentes. (se refiere a la identificación de requerimientos coincidentes de diferentes stakeholders[[5]](#footnote-6) y a la agrupación de requerimientos relacionados)

3. Ordenamiento por prioridades y negociación de requerimientos: ordenar por prioridades, y encontrar y resolver los requerimientos en conflicto a través de la negociación

4. Documentación de requerimientos: se documenta requerimientos y se entra a la siguiente vuelta del espiral. Se pueden producir documentos formales o informales.

**2.2.3. Validación de requerimientos**

Rara vez se encuentran todos los problemas en los requerimientos durante este proceso. Es inevitable que haya cambios adicionales de requerimientos para corregir las omisiones y las malas interpretaciones después de que el documento de requerimientos haya sido aprobado.

Se llevan a cabo verificaciones sobre el documento de requerimientos, estas verificaciones comprenden: consistencia (el documento de requerimientos no se contradice), validez (se requiere funciones adicionales), completitud (incluir funciones de restricción propuestas por el usuario), realismo (deben verificarse que los requerimientos pueden completarse), verificabilidad (debe evitarse discusiones con el cliente para esto debe ser verificable el requerimiento).

**2.2.4. Gestión de requerimientos**

Gestión de requerimientos establece los deseos y las necesidades de los interesados y entonces los revisa estos para crear una línea base de requerimientos para usar en desarrollo de soluciones y la gestión de beneficios. Sus objetivos son:

* Asegurar que los interesados relevantes tienen la oportunidad de expresar sus deseos y necesidades;
* Reconciliar los requerimientos múltiples de los interesados para crear un solo conjunto de objetivos viables; y
* Lograr una línea base de requerimientos en consenso con los interesados.

**2.3. Modelos del sistema.**

El modelo de proceso de desarrollo de software es quizás la pieza más importante de este engranaje conocido como ingeniería de software. Existen varios modelos para el proceso de desarrollo software. Los modelos están conformados por etapas que son generales a todos los enfoques. Las diferencias están básicamente en los tiempos en los cuales se realizan dichas etapas, la simultaneidad, la prioridad que se le da a cada etapa, entre otros elementos. Otras características se pueden observar en detalle en la definición de cada uno de los modelos.

**2.3.1. Modelos de contexto**

EL modelo fundamental del sistema o modelo de contexto, representa al elemento de software completo como una sola burbuja con datos de entrada y de salida representados por flechas de entrada y de salida, respectivamente.

**2.3.2. Modelos de comportamiento**

Los modelos de comportamiento se utilizan para describir el comportamiento del sistema en su totalidad. Entre los modelos de comportamiento existentes se distinguen dos de estos: modelos de flujo de datos, que modelan el procesamiento de los datos en el sistema, y modelos de máquinas de estado, que modelan como el sistema reacciona a los eventos. Estos modelos pueden usarse de forma separada o conjuntamente, dependiendo del tipo de sistema que se esté desarrollando.

**2.3.3. Modelos de datos**

Un modelo de base de datos muestra la estructura lógica de la base, incluidas las relaciones y limitaciones que determinan cómo se almacenan los datos y cómo se accede a ellos.

**2.3.4. Modelos de Objetos**

Un modelo de objetos muestra la estructura lógica de las clases y sus dependencias, incluidas las relaciones y limitaciones que determinan cómo se instancian .

**2.3.5. Métodos estructurados**

El modelo estructurado se basa en el modelo básico de entrada proceso y salida Ej. Diagrama Entidad Relación y Diagramas de Flujo.

**2.4. Especificación de sistemas críticos.**

Se debe especificar cuáles son los sistemas que deben tener plan de continuidad para evitar contingencias.

**2.4.1. Especificación dirigida por riesgos**

Se debe especificar el análisis de la probabilidad e impacto para medir el nivel de riesgo absoluto y residual después de aplicar el control que mitigue su riesgo.

**2.4.2. Especificación de la seguridad**

Se debe documentar como evitar vulnerabilidades en la fiabilidad, confiabilidad, disponibilidad, control de accesos y autenticidad.

**2.4.3. Especificación de la protección**

Se debe especificar los tipos de protección para evitar que se materialicen la amenazas sobre las vulnerabilidades mediante controles de seguridad.

**2.4.4. Especificación de la fiabilidad del software**

Se debe documentar las necesidades y condiciones sobre las capacidades de: disponibilidad, tolerancia a fallos, tiempo de recuperación y madurez en condiciones normales para satisfaces las operaciones que se requiere en condiciones normales.

**2.5. Especificación formal**

Una especificación formal usa notación matemática para describir de manera precisa las propiedades que un sistema de información debe tener, sin preocuparse por la forma de obtener dichas propiedades. Describe lo que el sistema debe hacer sin decir cómo se va a hacer.

**2.5.1. Especificación formal en el proceso del software**

Esta abstracción hace que las especificaciones formales sean útiles en el proceso de desarrollar un sistema, porque permiten responder preguntas acerca de lo que el sistema hace con confianza, sin la necesidad de tratar con una gran cantidad de información no relevante que se encuentra en el código de programa del sistema en un lenguaje de programación cualquiera, o especular sobre el significado de frases en un impreciso Pseudocódigo.

**2.5.2. Especificación de interfaces de subsistemas**

Las especificaciones de la interfaz de cada sub sistema a desarrollar deben estar diseñadas antes de implementarlas con sus requerimientos específicos.

**2.5.3. Especificación del comportamiento**

El comportamiento en función al tiempo de ejecución facilitan el manejo de comportamientos complejos.

**3. DISEÑO**

El diseño de software, es una de las etapas que deben componer el ciclo de vida del software, casi de una forma obligatoria, aunque algunas metodologías no le den la importancia que requiere. Se ha descrito al diseño como un proceso de etapas múltiples en el que, a partir de los requerimientos de información, se sintetizan las representaciones de los datos y la estructura del programa, las características de la interfaz y los detalles del procedimiento. El diseño es una actividad que tiene que ver con la toma de decisiones importantes, con frecuencia de naturaleza estructural. Comparte con la programación el objetivo de abstraer una representación de la información y de las secuencias de procesamiento (Pressman, 2010).

Básicamente, después de haber analizado a mano y papel los requisitos que se tienen para nuestro sistema a desarrollar, es entonces cuando entra en juego el diseño de software. Su objetivo será armar el cascarón bajo el cuál se estará implementando el código o realizando la programación. Pues no puedes empezar a programar en el aire sin saber hacia donde va tu software.

Para definir el diseño de software con una sola palabra, posiblemente Calidad sea la indicada. Pues si realmente deseas tener un software que supere básicamente cualquier error y que esté hecho a la perfección como el cliente le pide, el diseño de software es fundamental. Pues en el, estaremos analizando cada una de las especificaciones solicitadas por el cliente, además estaremos seccionando el software, viendo sus funciones, como se mostrará en pantalla y muchas cosas más que conlleva el diseño de software, por si pensabas que era una etapa sencilla y que no tendría complejidad alguna.

**3.1. Diseño arquitectónico.**

El concepto de arquitectura de software se refiere a la estructuración del sistema que, idealmente, se crea en etapas tempranas del desarrollo. Esta estructuración representa un diseño de alto nivel del sistema que tiene dos propósitos primarios: satisfacer los atributos de calidad (desempeño, seguridad, modificabilidad), y servir como guía en el desarrollo.

La arquitectura del software de un programa o sistema de cómputo es la estructura o estructuras del sistema, lo que comprende a los componentes del software, sus propiedades externas visibles y las relaciones entre ellos. Los diagramas de despliegue de UML permiten representar la arquitectura del Sistema.

**3.1.1. Decisiones de diseño arquitectónico**

Nos permite asimismo cubrir el hueco existente entre los requisitos y los productos de software, de manera que podemos establecer una traza completa tanto hacia delante como hacia atrás.

**3.1.2. Organización del sistema**

Encontrar una forma concreta de organizarse (sistema) que permitiera lograr lo siguiente:

* Que todo el mundo fuera desahogado y trabajara sin prisas
* Que no se acumularan las tareas de manera infinita
* Que todos los colaboradores finalizaran sus tareas y entregaran sus proyectos a tiempo
* Que nadie se llevara trabajo para hacer durante el fin de semana
* Que nadie se peleara porque ningún compañero se atrasó en sus tareas
* Que toda la empresa supiera rápidamente dónde puede encontrar todos los archivos compartidos

Es utópico[[6]](#footnote-7) porque tu manera de organizarte tiene que ver en cómo eres tú, por lo tanto cada persona tendrá su propio sistema.

**3.1.3. Estilos de descomposición modular**

La descomposición modular es un método de diseño proporciona un mecanismo sistemático para descomponer el problema en subproblemas, reducirá la complejidad de todo el problema consiguiendo de esta manera una solución modular efectiva.

El diseño modular propone dividir el sistema en partes diferenciadas y definir sus interfaces. Sus ventajas: Claridad, reducción de costos y reutilización.

Los pasos a seguir son:

* Identificar los módulos
* Describir cada módulo
* Describir las relaciones entre módulos.

Una descomposición modular debe poseer ciertas cualidades mínimas para que se pueda considerar suficiente validad.

* Independencia funcional
* Acoplamiento[[7]](#footnote-8)
* Cohesión[[8]](#footnote-9)
* Comprensibilidad[[9]](#footnote-10)
* Adaptabilidad

**3.1.4. Estilos de control**

Los modelos para estructurar un sistema están relacionados con la forma en que un sistema se descompone en subsistemas.

Para trabajar como un sistema, los subsistemas deben ser controlados para que sus servicios se entreguen en el lugar correcto, en el momento preciso.

El diseñador debe organizar los subsistemas, de acuerdo con algún modelo de control que complemente el modelo de estructura usado.

Los modelos de control a nivel arquitectónico están relacionados con el flujo de control entre subsistemas.

**3.1.5. Arquitecturas de referencia**

La arquitectura de referencia es una solución de infraestructura prediseñada que utiliza configuraciones comprobadas de recursos informáticos, almacenamiento, redes y virtualización. La tecnología está integrada, y estandarizada, para garantizar un mejor rendimiento, una mayor confiabilidad y una implementación simplificada. Hoy en día, muchas organizaciones están implementando la infraestructura convergente y la infraestructura hiperconvergente en vez de la arquitectura de referencia a fin de mejorar la simplicidad y ofrecer resultados de negocios más rápidos.

**3.2. Arquitecturas de sistemas distribuidos.**

Un sistema distribuido se define como una colección de computadores autónomos conectados por una red, y con el software distribuido adecuado para que el sistema sea visto por los usuarios como una única entidad capaz de proporcionar facilidades de computación.



Los sistemas distribuidos se implementan en diversas plataformas hardware, desde unas pocas estaciones de trabajo conectadas por una red de área local, hasta Internet, una colección de redes de área local y de área extensa interconectados, que en lazan millones de ordenadores.

Las aplicaciones de los sistemas distribuidos varían desde la provisión de capacidad de cómputo a grupos de usuarios, hasta sistemas bancarios, comunicaciones multimedia y abarcan prácticamente todas las aplicaciones comerciales y técnicas de los ordenadores. Los requisitos de dichas aplicaciones incluyen un alto nivel de fiabilidad, seguridad contra interferencias externas y privacidad de la información que el sistema mantiene.

Un sistema distribuido es un sistema de información en el cual las funciones se reparten por áreas de trabajo diferentes que trabajan de forma coordinada para asumir los objetivos que la organización asigna a ese sistema de información.

Elementos de un sistema Distribuido:

En él se integran:

La plataforma de proceso. Una vez diseñado el sistema, es el elemento encargado de proporcionar los recursos físicos y el software de base para ejecutarlo. Está formado por los Mainframe, PC’s, PDA’s, teléfonos, etc… Los elementos de la conectividad. Son los encargados se proporcionar el transporte para comunicar e integrar los elementos de la plataforma de proceso. Son básicamente las redes y las comunicaciones. El almacenamiento de datos, formado por los datos en si y los gestores donde se localizan. Los elementos de software donde se incluyen las aplicaciones, los servicios que ayudan a crearlas y las interfaces que ayudan a usarlas.

En este componente se integran las arquitecturas posibles para crearlas: centralizada, Batch, transaccional, cliente / servidor basado en sistema operativo, cliente / servidor basada en Internet y aplicaciones Web Internet. A lo largo de la exposición pondremos especial cuidado en presentar las características y posibilidades las tres últimas. Sistemas de seguridad. Finalmente, debe realizarse la gestión del sistema como un conjunto integrado y coordinado a través de los recursos de dirección y administración. La gestión del sistema debe permitir la coexistencia de varios centros de gestión diferentes. Parte fundamental del sistema de gestión es el cuadro de mandos. Hay dos cuadros de mandos diferentes: El cuadro de mandos de seguimiento de los objetivos de negocio pensado para proporcionar información automática a los gestores de como la realidad se mueve respecto a las previsiones de los objetivos de negocio en “tiempo real”. El cuadro de mandos de explotación desde donde se centraliza y coordina toda la administración, supervisión y explotación del sistema.

Características:

* Compartición de Recursos
* Apertura (opennesss)
* Concurrencia
* Escalabilidad
* Tolerancia a Fallos
* Transparencia

**3.2.1. Arquitecturas multiprocesador**

Este sistema consiste de varios procesos que pueden ejecutarse sobre procesadores diferentes (aunque no es necesario), es muy común en sistemas grandes de tiempo real, recolectan información, toman decisiones, con la afirmación, y envían señales a los actuadores que modifican el entorno del sistema.

El uso de múltiples procesadores mejora el rendimiento y adaptabilidad del sistema. La distribución de los procesos de los procesadores se puede predeterminar o puede estar bajo el control de un despachador que decide cuales procesos ubicar en cada procesador. Los sistemas de múltiples procesos no son necesariamente sistemas distribuidos. Si se dispone de más de un procesador, entonces se puede implementar la distribución, pero los diseñadores del sistema no siempre consideran forzosamente cuestiones de distribución mediante el proceso de diseño.

El enfoque de diseño para este tipo de sistema es el mismo para los de Tiempo Real.

En otras palabras, un sistema multiproceso o multitarea es aquel que permite ejecutar varios procesos de forma concurrente o paralela, la razón es porque actualmente la mayoría de las CPU’s sólo pueden ejecutar un proceso cada vez. La única forma de que se ejecuten de forma simultánea varios procesos es tener varias CPU’s (ya sea en una máquina o en varias, en un sistema distribuido.

Para el desarrollo de estos procesos se ocupan modelos de programación concurrente y paralela:

Los objetivos de la programación paralela:

* Reducir el tiempo de cómputo.
* Reducir la complejidad del algoritmo,
* Aprovechar al máximo la capacidad de las computadoras multiproceso.

Objetivos de la programación concurrente:

* Compartir dinámicamente el tiempo de procesador de una maquina entre varios trabajos y aplicaciones activas.

Existen diferentes tipos de programación:

* Multihilo: El cual permite a una aplicación realizar varias tareas concurrentemente. Los distintos hilos que se ejecutan comparten una serie se recursos.
* Pase de mensaje: MPI (“Message Passing Interface”) es un estándar que define la sintaxis y la semántica de las funciones usada en programas que exploten la existencia de múltiples procesadores.

Ventajas:

* La ventaja de un sistema multiproceso reside en la operación llamada cambio de contexto. Esta operación consiste en quitar a un proceso de la CPU, ejecutar otro proceso y volver a colocar el primero sin que se entere de nada.
* Los hilos que se ejecutan comparten ciertos recursos como el espacio del mensaje, la cual permite simplificar el diseño de una aplicación que debe llevar a cabo distintas funciones simultáneamente.
* Es económica
* Las computadoras paralelas son inherentes escalables permitiendo actualizarlas para adecuarse a la necesidad.
* El uso de componentes comúnmente disponibles, en grandes cantidades, permite ofrecer mayor rendimiento.

Desventajas:

* Puede ser limitante física, existen factores que limitan la velocidad máxima de un procesador independiente del factor económico.
* Las barreras físicas infranqueables tales como la velocidad de la luz, efectos al reducir el tamaño.
* Problemas causados por fenómenos eléctricos a pequeñas escalas restringen la capacidad máxima del sistema multiprocesador.

****

**3.2.2. Arquitecturas cliente servidor**

Esta arquitectura consiste básicamente en un cliente que realiza peticiones a otro programa (el servidor) que le da respuesta.

Aunque esta idea se puede aplicar a programas que se ejecutan sobre una sola computadora es más ventajosa en un sistema operativo multiusuario distribuido a través de una red de computadoras. La interacción cliente-servidor es el soporte de la mayor parte de la comunicación por redes. Ayuda a comprender las bases sobre las que están construidos los algoritmos distribuidos.

Existen varias configuraciones propuestas de modelos cliente/servidor:

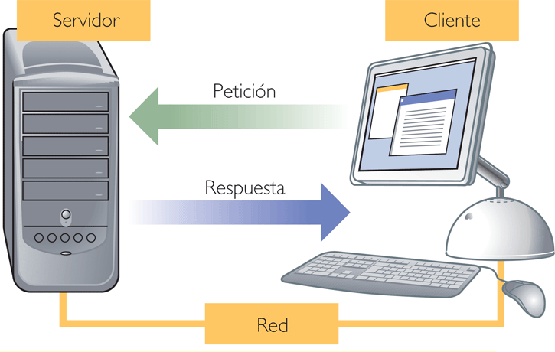
* El cliente procesando solamente el despliegue de información.
* El cliente procesando el despliegue y participando en parte del proceso.
* El cliente manejando el despliegue, todo el proceso, y accesando los datos del servidor.

Las aplicaciones tradicionales se caracterizan por ser muy intensivas en cuanto a la manipulación de los datos, normalmente consisten de cuatro componentes principales: la base de datos, la lógica de la transacción, la lógica de la aplicación y la interfaz de usuario. En las aplicaciones tradicionales –no cliente/servidor- todo este proceso se realiza en un solo procesador central, lo cual impedía al usuario el poder cargar sus propios datos y procesarlos posteriormente a su conveniencia en su propia máquina.

Cliente/servidor divide la aplicación, en esta división normalmente se deja la lógica de la transacción y la base de datos en un lugar y la lógica de la aplicación y la interfaz de usuario distribuidas en otro lugar, de esta forma el usuario puede tener múltiples fuentes de datos y permite al modelo descentralizar el ambiente.

Partes que componen el sistema:

* **Cliente:** Programa ejecutable que participa activamente en el establecimiento de las conexiones. Envía una petición al servidor y se queda esperando por una respuesta. Su tiempo de vida es finito una vez que son servidas sus solicitudes, termina el trabajo.
* **Servidor:** Es un programa que ofrece un servicio que se puede obtener en una red. Acepta la petición desde la red, realiza el servicio y devuelve el resultado al solicitante. Al ser posible implantarlo como aplicaciones de programas, puede ejecutarse en cualquier sistema donde exista TCP/IP y junto con otros programas de aplicación. El servidor comienza su ejecución antes de comenzar la interacción con el cliente. Su tiempo de vida o de interacción es “interminable”.



**Ventajas del esquema Cliente-Servidor**

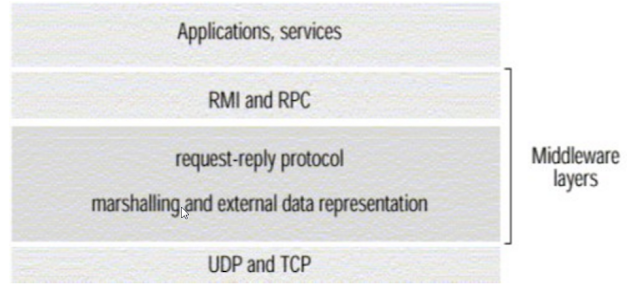
* Facilita la integración entre sistemas diferentes y comparte información, permitiendo por ejemplo que las máquinas ya existentes puedan ser utilizadas pero utilizando interfaces más amigables el usuario. De esta manera, se puede integrar PCs con sistemas medianos y grandes, sin necesidad de que todos tengan que utilizar el mismo sistema operativo.
* Al favorecer el uso de interfaces gráficas interactivas, los sistemas construidos bajo este esquema tienen una mayor y más intuitiva con el usuario. En el uso de interfaces gráficas para el usuario, presenta la ventaja, con respecto a uno centralizado, de que no siempre es necesario transmitir información gráfica por la red pues esta puede residir en el cliente, lo cual permite aprovechar mejor el ancho de banda de la red.
* La estructura inherentemente modular facilita además la integración de nuevas tecnologías y el crecimiento de la infraestructura computacional, favoreciendo así la escalabilidad de las soluciones.
* Contribuye además a proporcionar a los diferentes departamentos de una organización, soluciones locales, pero permitiendo la integración de la información.

**Desventajas**

* El mantenimiento de los sistemas es más difícil pues implica la interacción de diferentes partes de hardware y de software, distribuidas por distintos proveedores, lo cual dificulta el diagnóstico de fallas.
* Cuenta con muy escasas herramientas para la administración y ajuste del desempeño de los sistemas.
* Es importante que los clientes y los servidores utilicen el mismo mecanismo (por ejemplo sockets o RPC), lo cual implica que se deben tener mecanismos generales que existan en diferentes plataformas.
* Hay que tener estrategias para el manejo de errores y para mantener la consistencia de los datos.
* El desempeño (performance), problemas de este estilo pueden presentarse por congestión en la red, dificultad de tráfico de datos, etc.

**3.2.3. Arquitecturas de objetos distribuidos**

Los objetos se distribuyen a través de varias computadoras en una red y se comunican a través de middleware que proporciona un conjunto de servicios que permiten la comunicación entre objetos y para que estos puedan ser añadidos o eliminados del sistema. Veamos algunas características del modelo cliente servidor y modelo de objetos distribuidos, además de sus ventajas.



Algunas ventajas

* Es una arquitectura muy abierta que permite añadir nuevos recursos fácilmente. (Implementación de estándares de comunicación entre objetos que permite escribir objetos en lenguajes de programación distintos).
* Es posible reconfigurar el sistema de forma dinámica mediante la migración de objetos a través de la red.
* Los objetos que proporcionan servicios pueden ejecutarse sobre cualquier nodo de la red. No será necesario decidir con antelación donde ser situá la lógica de la aplicación.
* Mayor flexibilidad y escalabilidad debido a que se pueden crear diferentes instancias del sistema proporcionando los mismo servicios por objetos diferentes. (según la carga del sistema)

Arquitecturas:

* RMI: Es un sistema distribuido orientado a objetos provee los mecanismos necesarios para un manejo transparente de la comunicación, de modo que el programador se ocupe de la lógica de la aplicación. Esta idea de abstraer la parte de comunicación fue introducida por primera vez con RPC (Remote Procedure Call). RPC no sigue el paradigma de orientación a objetos por lo que el diseño de sistemas distribuidos orientados a objetos debe hacerse desde los cimientos.
* OMG: El OMG alcanza sus objetivos promoviendo la adopción de especificaciones de interfaz y de protocolo, que permiten la interoperabilidad y portabilidad de las aplicaciones orientadas a objetos distribuidos. En este consorcio no se producen guías de cómo implementar o producir software, sólo especificaciones.
* OMA: En el modelo de objeto de OMA, un objeto es una entidad encapsulada con una identidad inmutable y distinguible, cuyos servicios pueden ser accedidos a través de interfaces bien definidas.
* CORBA: La arquitectura CORBA está orientada a objetos. Los objetos CORBA presentan muchas características de otros sistemas orientados a objetos, incluyendo la herencia de interfaces y el polimorfismo. Lo que hace a CORBA más interesante es que proporciona estas capacidades, incluso cuando es utilizado en lenguajes no orientados a objeto como C o COBOL, aunque CORBA trabaja particularmente bien con los lenguajes orientados a objeto como C++ y Java.

**3.2.4. Computación distribuida interorganizacional**

La Computación distribuida proporciona mejores condiciones para aplicar estándares locales y procesos operacionales.

Esta ha sido implementada principalmente a nivel organizacional. Una organización tiene varios servidores y distribuye la carga entre ellos. Actualmente están disponibles modelos mas recientes de computación distribuida que permiten computación distribuida interorganizacional en lugar de interorganizacional.

CARACTERISTICAS

* Escalabilidad
* Anonimato
* Rendimiento
* Seguridad
* Propiedad Compartida

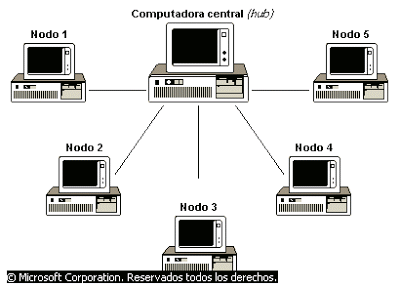
VENTAJAS:

* Compartición de Recursos
* Apertura con Protocolos Estándares
* Tolerancia a Defectos

DESVENTAJAS:

* Complejidad mas difícil de entender y probar
* La Seguridad a las red de computadoras tenían ruidos
* Manejabilidad era impredecible y podría verse afectados por varios factores

Un sistemas de computación distribuida requiere de un software que gestione las partes llamadas MIDDLEWARE se ubica entre los componentes del sistema distribuido.



**3.3. Arquitecturas de aplicaciones.**

*"Si cree que la buena arquitectura es cara, pruebe la mala".*

*- Brian Foote y Joseph Yoder*

Cuando aumenta la complejidad de las aplicaciones, una manera de administrarla consiste en dividir la aplicación según sus responsabilidades o intereses. Esto sigue el principio de separación de intereses y puede ayudar a mantener organizado un código base que crece para que los desarrolladores puedan encontrar fácilmente donde se implementa una funcionalidad determinada. Pero la arquitectura en capas ofrece una serie de ventajas que van más allá de la simple organización del código.

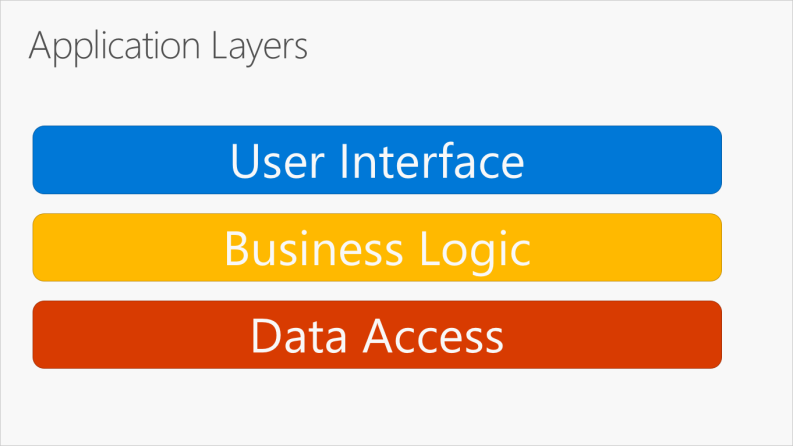
Al organizar el código en capas[[10]](#footnote-11), la funcionalidad común de bajo nivel se puede reutilizar en toda la aplicación.Esta reutilización es beneficiosa ya que significa escribir menos código y puede permitir que la aplicación se estandarice en una sola implementación, siguiendo el principio [Una vez y solo una (DRY)](https://en.wikipedia.org/wiki/Don%27t_repeat_yourself).

Con una arquitectura en capas, las aplicaciones pueden aplicar restricciones sobre qué capas se pueden comunicar con otras capas. Esto ayuda a lograr la encapsulación. Cuando se cambia o reemplaza una capa, solo deberían verse afectadas aquellas capas que funcionan con ella. Mediante la limitación de qué capas dependen de otras, se puede mitigar el impacto de los cambios para que un único cambio no afecte a toda la aplicación.

Las capas (y la encapsulación) facilitan considerablemente el reemplazo de funcionalidad dentro de la aplicación. Por ejemplo, es posible que una aplicación use inicialmente su propia base de datos de SQL Server para la persistencia, pero más adelante podría optar por usar una estrategia de persistencia basada en la nube, o situada detrás de una API web. Si la aplicación ha encapsulado correctamente su implementación de persistencia dentro de una capa lógica, esa capa específica de SQL Server se podría sustituir por una nueva que implementara la misma interfaz pública.

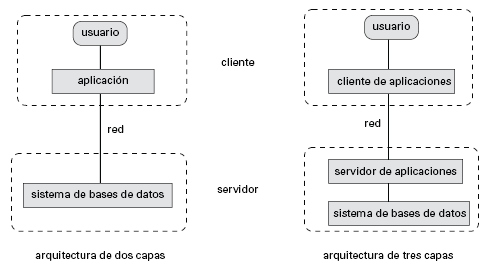
Además de la posibilidad de intercambiar las implementaciones en respuesta a cambios futuros en los requisitos, las capas de aplicación también facilitan el intercambio de implementaciones con fines de prueba.En lugar de tener que escribir pruebas que funcionan en la capa de datos reales o de la interfaz de usuario de la aplicación, estas capas se pueden reemplazar en tiempo de prueba con implementaciones falsas que proporcionen respuestas conocidas a las solicitudes. Esto normalmente hace que las pruebas sean mucho más fáciles de escribir y mucho más rápidas de ejecutar en comparación con la ejecución de pruebas sobre la infraestructura real de la aplicación.

Las capas lógicas son una técnica común para mejorar la organización del código en las aplicaciones de software empresarial, y hay varias formas de organizar el código en capas.



Estas capas se suelen abreviar como UI (interfaz de usuario), BLL (capa de lógica de negocios) y DAL (capa de acceso a datos). Con esta arquitectura, los usuarios realizan solicitudes a través de la capa de interfaz de usuario, que interactúa con la capa BLL. BLL, a su vez, puede llamar a DAL para las solicitudes de acceso de datos. La capa de interfaz de usuario no debe realizar solicitudes directamente a DAL, ni debe interactuar con la persistencia de forma directa a través de otros medios. Del mismo modo, BLL solo debe interactuar con la persistencia a través de DAL. De este modo, cada capa tiene su propia responsabilidad conocida.

Una desventaja de este enfoque de distribución en capas tradicional es que las dependencias de tiempo de compilación se ejecutan desde la parte superior a la inferior. Es decir, la capa de interfaz de usuario depende de BLL, que depende de DAL. Esto significa que BLL, que normalmente contiene la lógica más importante de la aplicación, depende de los detalles de implementación del acceso a datos (y a menudo de la existencia de una base de datos). Probar la lógica de negocios en este tipo de arquitectura suele ser difícil, y requiere una base de datos de prueba. Se puede usar el principio de inversión de dependencias para resolver este problema, como se verá en la sección siguiente.



**3.3.1. Sistemas de procesamiento de datos**

El sistema de procesamiento de datos de Hadoop es un sistema que permite el procesamiento distribuido de big data a través de clústeres de servidores usando modelos de programación sencillos. Está diseñado para escalar de servidores individuales a miles de máquinas, cada una ofreciendo computación y almacenamiento local.

**3.3.2. Sistemas de procesamiento de transacciones**

El procesamiento de datos es, en general, "la acumulación y manipulación de elementos de datos para producir información significativa."

El procesamiento de datos trata de un subconjunto del procesamiento de la información, el cambio (procesamiento) de la información de cualquier manera detectable por un observador.

El procesamiento de datos es distinto del procesamiento de textos, pues este último manipula textos nada más en lugar de los datos.

El procesamiento de datos puede involucrar diversas funciones, entre ellas :

* Validación - Asegurar que los datos suministrados son "limpio, correcto y útil".
* Clasificación - "Ordena elementos de cierta secuencia y / o en diferentes conjuntos".
* Recapitulación - Reducir los detalles de los datos a sus principales puntos.
* Agregación - Combinación de múltiples piezas de datos.
* Análisis - La "colección, organización, análisis, interpretación y presentación de datos".
* Información - Lista detallada o resumen de los datos de información computarizada.

**3.3.3. Sistemas de procesamiento de eventos**

El Procesamiento de Eventos (EP) es un área emergente impulsada, en primer lugar, por la mayor necesidad que tienen las empresas para poder responder rápidamente a este gran volumen de negocios y eventos de TI. EP reconoce la necesidad de brindar soporte para el ciclo de toma de decisiones mediante el procesamiento de eventos significativos para las empresas de una forma más efectiva. Por lo tanto, EP es una parte cada vez más importante de las estrategias de las empresas en relación con las Arquitecturas Orientadas a Servicios (SOAs).

En primer lugar, este artículo discute la necesidad de realizar el procesamiento de eventos, los diferentes tipos de procesamientos de eventos y el valor que tiene la adopción del procesamiento de eventos para los negocios. Luego de esto, el presente artículo detalla un modelo conceptual de procesamiento de eventos que se puede usar para materializar una Arquitectura Impulsada por Eventos (EDA). Los conceptos se elaboran mediante la discusión de tres escenarios de negocios que implementan el procesamiento de eventos para lograr tomar mejores decisiones.

Los elementos son:

* Un evento (una cosa relevante que ocurre dentro o fuera del negocio) puede provocar la invocación de un servicio, la iniciación de un proceso de negocios y / o la publicación o la sindicación de más información. El Procesamiento de Eventos se ocupa de la tarea de procesar uno o varios eventos con el objetivo de identificar aquellos eventos que sean más importantes dentro de la nube de eventos. Desde la perspectiva del valor de negocios, el procesamiento de eventos es la capacidad de detectar y responder a eventos que indican situaciones que impactan sobre los negocios y ocurren en toda la empresa. Por ejemplo, un mensaje de evento podría indicar que se agregó un cliente nuevo, que se vendió un producto, que se recibió un envío, que se abrió una puerta de seguridad, que se informó la ubicación actual de un activo por medio de un GPS, etc.
* Un productor (o una fuente) de eventos produce eventos. Por ejemplo, un productor de eventos puede ser una aplicación, un almacenamiento de datos, un servicio, un proceso de negocios, un transmisor, un sensor o una herramienta de colaboración (como una aplicación de mensajería instantánea o correo electrónico). Al recibir estos eventos del productor, los eventos pueden provocar resultados de manera directa o se los puede evaluar según los patrones de procesamiento de eventos. Los patrones de procesamiento de eventos se definen de acuerdo con las necesidades de las partes interesadas, y no de acuerdo con las necesidades de los productores de eventos. Los resultados del procesamiento de eventos incluyen, pero no se limitan a, la invocación de un servicio, la iniciación de un proceso de negocios, la publicación de un evento en un hub de subscripción, la notificación directa a humanos o sistemas, la generación de un evento nuevo y / o la captura de los propósitos históricos del evento. Se suele hacer referencia a los eventos y su interacción con los servicios de negocios de la siguiente manera: sistema de información impulsada por eventos o arquitectura impulsada por eventos.

El procesamiento de Eventos de Negocios extiende y mejora el procesamiento de eventos haciendo que esté disponible para los usuarios de negocios de forma consumible. Procesamiento de Eventos de Negocios extiende las capacidades y las herramientas de los usuarios de negocios, lo que permite la aplicación de tecnologías para construir sistemas de información impulsados por eventos que contribuyen con el negocio. La capacidad de percibir cuando un evento ocurrió (o no ocurrió), lo que indica una situación de negocios procesable, hace que el negocio pueda responder de manera rápida a las oportunidades y a las amenazas.

**3.3.4. Sistemas de procesamiento de lenguajes**

El procesamiento de lenguajes naturales —abreviado PLN, o NLP del idioma inglés Natural Language Processing— es un campo de las ciencias de la computación, inteligencia artificial y lingüística que estudia las interacciones entre las computadoras y el lenguaje humano. El PLN se ocupa de la formulación e investigación de mecanismos eficaces computacionalmente para la comunicación entre personas y máquinas por medio de lenguajes naturales. El PLN no trata de la comunicación por medio de lenguajes naturales de una forma abstracta, sino de diseñar mecanismos para comunicarse que sean eficaces computacionalmente —que se puedan realizar por medio de programas que ejecuten o simulen la comunicación—. Los modelos aplicados se enfocan no solo a la comprensión del lenguaje de por sí, sino a aspectos generales cognitivos humanos y a la organización de la memoria. El lenguaje natural sirve solo de medio para estudiar estos fenómenos. Hasta la década de 1980, la mayoría de los sistemas de PLN se basaban en un complejo conjunto de reglas diseñadas a mano. A partir de finales de 1980, sin embargo, hubo una revolución en PLN con la introducción de algoritmos de aprendizaje automático para el procesamiento del lenguaje.

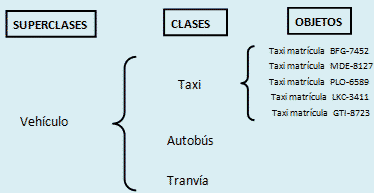
**3.4. Diseño orientados a objetos.**

Consiste en representar un modelo de datos que pueda ser fácilmente implantable con algún lenguaje de programación orientado a objetos.

**3.4.1. Objetos y clases**

Objeto: entidad existente en la memoria del ordenador que tiene unas propiedades (atributos o datos sobre sí mismo almacenados por el objeto) y unas operaciones disponibles específicas (métodos).

Clase: abstracción que define un tipo de objeto especificando qué propiedades (atributos) y operaciones disponibles va a tener.



**3.4.2. Un proceso de diseño orientado a objetos**

El diseño de software es un proceso iterativo por medio del cual se traducen los requerimientos en un “plano” para construir el software. Al principio, el plano ilustra una visión holística del software. Es decir, el diseño se representa en un nivel alto de abstracción, en el que se rastrea directamente el objetivo específico del sistema y los requerimientos más detallados de datos, funcionamiento y comportamiento. A medida que tienen lugar las iteraciones del diseño, las mejoras posteriores conducen a niveles menores de abstracción.

Lineamientos y atributos de la calidad del software.

El diseño es importante porque permite que un equipo de software evalúe la calidad de éste antes de que se implemente, momento en el que es fácil y barato corregir errores, omisiones o inconsistencias.

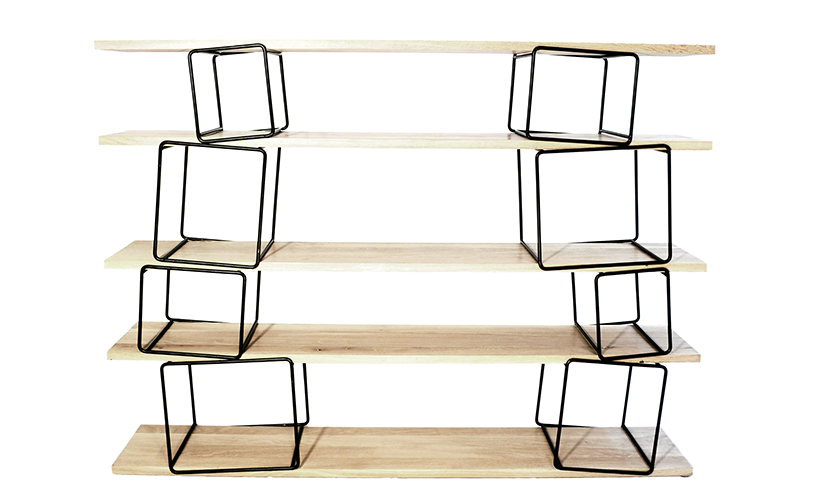
Durante el diseño, la calidad se evalúa por medio de la realización de una serie de revisiones técnicas. Una revisión técnica es una reunión celebrada por miembros del equipo de software. Por lo general, participan dos, tres o cuatro personas, en función del alcance de la información del diseño que se revisará.

Lineamientos para el diseño:

* Debe tener una arquitectura que a) se haya creado con el empleo de estilos o patrones arquitectónicos reconocibles, b) esté compuesta de componentes con buenas características de diseño y c) se implementen en forma evolutiva de modo que faciliten la implementación y las pruebas.
* Debe ser modular, es decir, el software debe estar dividido de manera lógica en elementos o subsistemas.
* Debe contener distintas representaciones de datos, arquitectura, interfaces y componentes.
* Debe conducir a estructuras de datos apropiadas para las clases que se van a implementar y que surjan de patrones reconocibles de datos.
* Debe llevar a componentes que tengan características funcionales independientes.
* Debe conducir a interfaces que reduzcan la complejidad de las conexiones entre los componentes y el ambiente externo.
* Debe obtenerse con el empleo de un método repetible motivado por la información obtenida durante el análisis de los requerimientos del software.
* Debe representarse con una notación que comunique con eficacia su significado.

**3.4.3. Evolución del diseño**

¿Crees que una mesa, una silla o un librero no pueden ser reinventados?, ¿piensas que las posibilidades que tiene un objeto cotidiano ya están agotadas? En realidad no: el Diseño Industrial es una disciplina siempre cambiante y los diseñadores constantemente están buscando nuevos acercamientos a piezas que podríamos pensar que ya están resueltas, sorprendiéndonos cada nueva temporada con sus propuestas.

****

En 1990 La crisis del software se fundamentó en el tiempo de creación de software, ya que en la creación del mismo no se obtenían los resultados deseados, además de un gran costo y poca flexibilidad.

Software es un término informático acuñado en 1968, en la primera conferencia organizada por la OTAN sobre desarrollo de software, de la cual nació formalmente la rama de la ingeniería de software. El término se adjudica a F. L. Bauer, aunque previamente había sido utilizado por Edsger Dijkstra en su obra The Humble Programmer.

Básicamente, la crisis del software se refiere a la dificultad en escribir programas libres de defectos, fácilmente comprensibles, y que sean verificables. Las causas son, entre otras, la complejidad que supone la tarea de programar, y los cambios a los que se tiene que ver sometido un programa para ser continuamente adaptado a las necesidades de los usuarios.

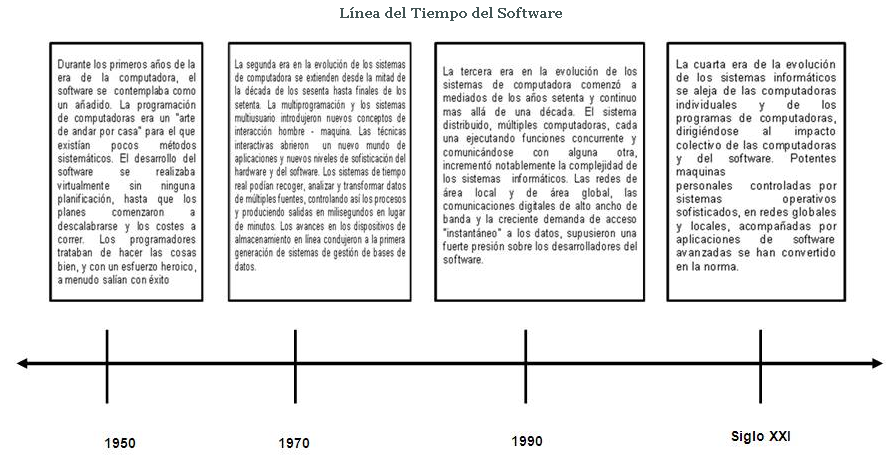
Además, no existen todavía herramientas que permitan estimar de una manera exacta, antes de comenzar el proyecto, cuál es el esfuerzo que se necesitará para desarrollar un programa. Este hecho provoca que la mayoría de las veces no sea posible estimar cuánto tiempo llevará un proyecto, ni cuánto personal será necesario. Cuando se fijan plazos normalmente no se cumplen por este hecho. Del mismo modo, en muchas ocasiones el personal asignado a un proyecto se incrementa con la esperanza de disminuir el plazo de ejecución.

Por último, las aplicaciones de hoy en día son programas muy complejos, inabordables por una sola persona. En sus comienzos se valoró como causa también la inmadurez de la ingeniería de software, aunque todavía hoy en día no es posible realizar estimaciones precisas del coste y tiempo que necesitará un proyecto de software.

Englobó a una serie de sucesos que se venían observando en los proyectos de desarrollo de software:

* Los proyectos no terminaban en plazo.
* Los proyectos no se ajustaban al presupuesto inicial.
* Baja calidad del software generado.
* Software que no cumplía las especificaciones.
* Código inmantenible que dificultaba la gestión y evolución del proyecto.

Aunque se han propuesto diversas metodologías para intentar subsanar los problemas mencionados, lo cierto es que todavía hoy no existe ningún método que haya permitido estimar de manera fiable el coste y duración de un proyecto antes de sus comienzos.



**3.5. Diseño de software en tiempo real.**

El software de tiempo real está muy acoplado con el mundo externo, esto es, el software de tiempo real debe responder al ámbito del problema en un tiempo dictado por el ámbito del problema. Debido a que el software de tiempo real debe operar bajo restricciones de rendimiento muy rigurosas, el diseño del software esta conducido frecuentemente, tanto por la arquitectura del hardware como por la del software, por las características del sistema operativo, por los requisitos de la aplicación y tanto por los extras del lenguaje de programación como prospectos de diseño.

**3.5.1. Diseño del *Sistema de Tiempo Real*[[11]](#footnote-12)**

Las computadoras se utilizan para controlar una amplia variedad de sistemas que van desde maquinas domesticas sencillas hasta plantas enteras de fabricación. Estas computadoras interactúan directamente con dispositivos de hardware. El software de dichos sistemas es software de tiempo real embebido que debe reaccionar a eventos generados por el hardware y emitir señales de control de respuesta a estos eventos.

Una forma de ver un sistema en tiempo real es como un sistema de estimo/respuesta. Dado un estímulo de entrada, el sistema debe producir la correspondiente salida. Se puede, por lo tanto, definir el comportamiento de un sistema de tiempo real haciendo una lista de estímulos recibidos por el sistema, las respuestas asociadas y el tiempo real en que dicha respuestas deben producirse.

Un sistema real tiene que responder a estímulos que ocurren en diferentes instantes de tiempo por lo tanto se tiene que organizar su arquitectura para que, tan pronto como se reciba un estímulo, el control sea trasferido al manejador adecuado.

**3.5.2. Sistemas operativos de tiempo real**

Los SO Tiempo real son aquellos en los cuales no tiene importancia el usuario, sino los procesos. Por lo general, están subutilizados sus recursos con la finalidad de prestar atención a los procesos en el momento que lo requieran. Se utilizan en entornos donde son procesados un gran número de sucesos o eventos. Algunos ejemplos de Sistemas Operativos de tiempo real son: VxWorks, Solaris, Lyns OS y Spectra.

**3.5.3. Sistemas de monitorización y control**

Los Sistemas de Monitorización y Control son infraestructuras que permiten obtener importantes ahorros ya que se configuran como elemento clave en la gestión de industrias o inmuebles (edificios, redes de oficinas comerciales, cadenas de supermercados, cadenas de distribución retail, etc.)

La medición se corresponde con la toma de datos o captura de información (variables eléctricas, parámetros de confort, consumo de agua, etc.) asociadas a distintos sistemas dependiendo del sector (Iluminación, climatización, cargas, cámaras de frío, etc.).

La ventaja fundamental asociada a la medición es la monitorización en tiempo real en modo remoto y la gestión de inmuebles e instalaciones. Además de su funcionalidad como sistema de medida interno, permite monitorizar incidencias técnicas asociadas al suministro y al funcionamiento de los equipos reforzando la gestión técnica y las operaciones de mantenimiento.

**3.5.4. Sistemas de adquisición de datos**

La adquisición de datos (DAQ) es el proceso de medir con una PC un fenómeno eléctrico o físico como voltaje, corriente, temperatura, presión o sonido. Un sistema DAQ consiste de sensores, hardware de medidas DAQ y una PC con software programable. Comparados con los sistemas de medidas tradicionales, los sistemas DAQ basados en PC aprovechan la potencia del procesamiento, la productividad, la visualización y las habilidades de conectividad de las PCs estándares en la industria proporcionando una solución de medidas más potente, flexible y rentable.

**3.6. Diseño de interfaces de usuario[[12]](#footnote-13).**

El diseño de ***interfaz de usuario*** o ingeniería de la interfaz es el resultado de definir la forma, función, usabilidad, ergonomía, imagen de marca y otros aspectos que afectan a la apariencia externa de las interfaces de usuario en sistemas de todo tipo (computadoras de uso general, sistemas de control, dispositivos de comunicación móviles, software de sistemas, software de aplicaciones, sitios web, etc). El diseño de la interfaz de usuario es una disciplina asociada al diseño industrial (aparece como tal recogido en la Clasificación de Locarno en el apartado 14-04) y se enfoca en maximizar la usabilidad y la experiencia de usuario. El objetivo final del diseño de la interfaz de usuario es hacer que la interacción entre el usuario y el sistema del que es interfaz sea tan simple y eficiente como sea posible, en términos de cumplimiento de los objetivos del usuario. Sigue por ello una filosofía de diseño centrado en el usuario.

**3.6.1. Asuntos de diseño**

"Diseñar es mucho más que construir, ordenar o incluso editar, es agregar valor y significado, iluminar, simplificar, aclarar, modificar, dignificar, dramatizar, persuadir, incluso divertirse" Paul Rand

Algunos asuntos de diseño son:

* La claridad es el trabajo primero y más importante de cualquier interfaz.
* Interfaces existen para permitir la interacción entre los seres humanos y nuestro mundo.
* Los seres humanos se encuentran más cómodos de sí mismos y de su entorno cuando se sienten bajo control. El software con un diseño que no está bien definido resta esa comodidad forzando a los usuario a interacciones no planeadas, rutas confusas y resultados sorprendentes.
* No existe la mejor interfaz en absoluto, cuando cada uno de nosotros somos capaces de manipular directamente los objetos físicos en nuestro entorno. Como esto no siempre es posible, y los objetos son cada vez más complejos, creamos interfaces para ayudarnos a interactuar con ellos. Es fácil caer en el error de agregar más capas de lo necesario a una interfaz, crear botones excesivamente forjados, cromados, gráficos, opciones, preferencias, ventanas, archivos adjuntos y otros elementos para que acabemos manipulando elementos de interfaz de usuario en lugar de lo que es importante. En su lugar, esforzarse por diseñar una interfaz con la menor huella posible, del objetivo original de la manipulación, reconociendo los gestos humanos tanto como sea posible. Idealmente, el diseño de la interfaz debe ser tan leve que el usuario debe tener la sensación de una manipulación directa con el objeto, sin intermediarios.
* Cada pantalla que diseñemos debe soportar una sola acción de valor real para la persona que la usa. Esto hará que sea más fácil de aprender, más fácil de usar y más fácil de agregar o construir cuando sea necesario.
* Las pantallas con una sola acción primaria pueden tener múltiples acciones secundarias pero deben mantenerse secundarias.
* Muy pocas interacciones están destinadas a ser la última, así que hay que diseñar cuidadosamente el próximo paso para cada interacción que una persona tiene que realizar con su interfaz.
* Los usuarios estamos más cómodos con las cosas que se comportan de la manera que esperamos. Otras personas, animales, objetos, software. Cuando alguien o algo se comporta de manera consistente con nuestras expectativas, sentimos que tenemos una buena relación con ella.
* Siguiendo el principio anterior, los elementos de pantalla no deben aparecer coherentes entre sí a menos que se comporten consistentemente entre sí. Los elementos que se comportan de la misma deben ser iguales.
* Una jerarquía visual fuerte se logra cuando hay un orden de visualización claro a los elementos visuales en una pantalla. Es decir, cuando los usuarios visualizan los mismos elementos en el mismo orden cada vez. Las jerarquías visuales débiles no dan poca pistas sobre dónde descansar la mirada y terminan por percibirse como desorden y confusión.
* Como dice John Maeda en su libro “Las leyes de la Simplicidad” una organización inteligente de los elementos de pantalla puede hacer que los muchos aparezcan como los pocos.Esto ayudará a los usuarios a entender nuestra interfaz más fácil y más rápidamente, ya que se ha ilustrado las relaciones inherentes de contenido en su diseño.
* El color de las cosas físicas cambia a medida que la luz cambia. A plena luz del día vemos un árbol muy diferente del que se esboza contra una puesta de sol. Al igual que en el mundo físico, donde el color es muy subjetivo, el color no debe determinar mucho en una interfaz. Puede ayudar, ser utilizado para destacar, ser utilizado para guiar la atención, pero no debe ser el único diferenciador de las cosas. Para largas horas de lectura o de pantalla extendida, se debe utilizar colores de fondo claros o neutros, ahorrando tonos más brillantes. Por supuesto, hay un tiempo para colores de fondo vibrantes, así, sólo hay que asegurarse de que es apropiado para su audiencia.
* Mostrar sólo lo que es necesario en cada pantalla.
* El estado cero: La primera experiencia con una interfaz es crucial, pero a menudo es pasada por alto por los diseñadores.
* Una curiosa propiedad de un gran diseño es que por lo general pasa desapercibido "El gran diseño es invisible" por las personas que lo utilizan.

**3.6.2. El proceso de diseño de la interfaz de usuario**

El proceso de diseño de interfaz de usuario basado en el flujograma de Usability Professionals Association, “Designing the User Experience”, comprende las siguientes fases:

* **Fase de Análisis**
  + Reuniones con responsables para establecer una visión clara del site a diseñar
  + Inclusión de tareas relativas a usabilidad en el plan del proyecto
  + Reunir un equipo multidisplinar para asegurar un conocimiento global
  + Establecer objetivos de usabilidad
  + Organizar estudios de campo
  + Búsqueda de productos competitivos
  + Crear perfiles de usuario
  + Desarrollar un análisis de tareas
  + Describir y documentar los escenarios de usuario
  + Describir y documentar los requerimientos de operativa de usuario
* **Fase de Diseño**
  + Brainstormings: diseño de conceptos y metáforas
  + Desarrollo del flujo de pantallas y el modelo de navegación
  + Realizar revisiones de conceptos de diseño
  + Diseño con papel y lápiz
  + Elaborar prototipos "low-fidelity"
  + Organizar tests de usabilidad sobre los prototipos "low-fidelity"
  + Elaborar prototipos detallados "high-fidelity"
  + Organizar tests de usabilidad sobre los prototipos "high-fidelity"
  + Documentación de estándares y directrices
  + Elaboración de una especificación de diseño
* **Fase de Implementación**
  + Realización de evaluaciones heurísticas en curso
  + Trabajar al lado de los responsables finales de la entrega según se va implementando el diseño
  + Organizar tests de usabilidad inmediatamente a las entregas
* **Fase de Desarrollo**
  + Realizar encuestas para obtener feedback de los usuarios
  + Organizar estudios de campo para obtener información de cómo se está usando
  + Comprobar objetivos mediante tests de usabilidad
* **Entregables**
  + Con frecuencia, los estudios para la usabilidad de proyectos, tanto en en análisis y desarrollo como en producción, suelen culminar con la elaboración y presentación de un documento final de "agujeros de usabilidad detectados" y medidas paliativas propuestas.
  + Pudiendo bastar para situaciones de apremio, en proyectos de poco calado o para resolver análisis de usabilidad de cliente interno, sin embargo es absolutamente recomendable seguir una metodología formal de Productos de Reporte o **Entregables**, para cada fase del proyecto por varias razones:
    - Porque trabajar con una Metodología establece un procedimiento formal de trabajo que garantiza orden evitando improvisaciones e indefiniciones, y avala nuestro trabajo ante la organización.
    - Porque se recoge, para cada fase del proyecto, la información "ad hoc" y conclusiones para la misma, documentándola de forma apropiada.
    - Porque se establecen claramente hitos con sus consiguientes documentos asociados, lo que ayuda a estructurar el proyecto, establecer compromisos y plazos, y garantizar, indirectamente, el éxito del proyecto entero.
    - Porque permite documentar los pasos seguidos a lo largo de todo el proyecto, permitiendo repetir el proceso en el futuro de la misma forma o mejorando situaciones problemáticas.
  + Aunque el conjunto de Entregables del proyecto dependerá de como se estructure éste y de la Metodología de Análisis y Diseño de la Usabilidad, damos a continuación un ejemplo de lo que podrían ser los Entregables de una metodología bien diseñada:
    - *Presentación de Lanzamiento del Proyecto.*
    - *Organización Funcional del Proyecto (identificación de responsables, dependencias y flujos de decisiones en el marco del Proyecto)... [Fase de Análisis].*
    - *Plan de Obtención de Información y Datos... (Técnicas de Sondeo)... [Fase de Análisis].*
    - *Especificación Funcional... [Fase de Análisis].*
    - *Documento de Compromiso para el Diseño Gráfico... (Técnicas de Sondeo) [Fase de Análisis].*
    - *Estructura del Interfaz de Usuario... (Técnicas de Sondeo) [Fase de Análisis].*
    - *Identificación de Estándares... (Técnicas de Sondeo) [Fase de Análisis].*
    - *Establecimiento de Estándares Específicos... (Técnicas de Sondeo) [Fase de Análisis].*
    - *Modularización del Proyecto... (Técnicas de Sondeo) [Fase de Análisis].*
    - *Requerimientos de los Módulos del Proyecto... (Técnicas de Sondeo) [Fase de Análisis].*
    - *Flujo de Pantallas, Funciones y Campos... (Técnicas de Sondeo) [Fases de Análisis / Diseño].*
    - *Documentos de Detalle de Datos Obtenidos... (Técnicas de Sondeo) [Fases de Análisis / Diseño].*
    - *Diseño y Elaboración de Prototipos... (Técnicas Auxiliares) [Fase de Diseño].*
    - *Iteraciones de Diseño Gráfico... [Fases de Diseño / Implementación / Desarrollo].*
    - *Actas de Reuniones de Especificación de Desarrollo... [Fases de Diseño / Implementación / Desarrollo].*
    - *Informe del Test de Simulación... (Técnicas de Testing de Usabilidad) [Fases de Diseño / Implementación / Desarrollo].*
    - *Informe de la Revisión de Experto... (Técnicas de Inspección de Usabilidad) [Fases de Diseño / Implementación / Desarrollo].*
    - *Informes del Test de Usabilidad... (Técnicas de Testing de Usabilidad) [Fases de Diseño / Implementación / Desarrollo].*
    - *Documento de Seguimiento de Modificaciones del Diseño... [Fases de Diseño / Implementación / Desarrollo].*
    - *Informe del Test Final de Usabilidad... (Técnicas de Testing) [Fase de Desarrollo].*
    - *Plan de Revisiones y Mejoras... (Técnicas de Testing).*
    - *Evauación Post-Producción... (Técnicas de Testing).*
    - *Evaluación de la Localización... (Técnicas de Testing).*
    - *Especificaciones de Traducción del Interfaz... (Técnicas de Sondeo).*

**3.6.3. Análisis del usuario**

El análisis de usuario es una técnica que permite conocer al usuario, identificar la razón fundamental del tipo de tarea del usuario: que hace y cuáles son sus procedimientos. Conocer cómo piensa, con el fin de desarrollar un sistema que trabaja según ese esquema (y no según el esquema mental del equipo de desarrollo).

**3.6.4. Prototipado de la interfaz de usuario**

Los prototipos de interfaz de usuario se pueden utilizar para explorar un diseño de interfaz de usuario alcanzable y adecuado que cumpla los requisitos, ayudando a reducir las distancias entre lo que es necesario (expresado a través de la adquisición de requisitos) y lo que es factible. El objetivo principal de la creación de un prototipo de interfaz de usuario es poder probar el diseño de interfaz de usuario, incluyendo la capacidad de utilización antes de que empiece el desarrollo real. De este modo, puede garantizar que está construyendo el sistema correcto, antes de dedicar demasiado tiempo y recursos al desarrollo.

Los prototipos de interfaz de usuario pueden ser prototipos formales o informales, ejecutables o no ejecutables, de baja fidelidad o de alta fidelidad.

Decida si un prototipo es adecuado para su proyecto. Decida de qué parte de la interfaz de usuario desea hacer un prototipo y el nivel de profundidad y de realismo de cualquier interactividad. Decid a si el prototipo es puramente una muestra o si se tiene intención de que algunos aspectos evolucionen para formar parte del producto final.

Tenga en cuenta que, para alcanzar el objetivo de realizar pruebas tempranas de la interfaz de usuario, el desarrollo del prototipo debe ser significativamente más económico que el del sistema real, al mismo tiempo que tiene las suficientes funciones para poder realizar una prueba de uso significativa.

**3.6.5. Evaluación de la interfaz**

Los puntos de calidad / medidas cuantitativas de una interfaz son muchas, pero fundamentalmente se citan los siguientes:

* Tiempo de aprendizaje (learnability)
* Velocidad de utilización (efficiency)
* Retención a lo largo del tiempo (memorability)
* Errores de los usuarios (errores)
* Satisfacción subjetiva (subjective satisfaction)

Estos criterios de evaluación responden usualmente a cualidades de más alto nivel (modelo adecuado, metáfora adecuada, consistencia, realimentación del usuario, tipografía e imagen audiovisual adecuadas.

**4. DESARROLLO**

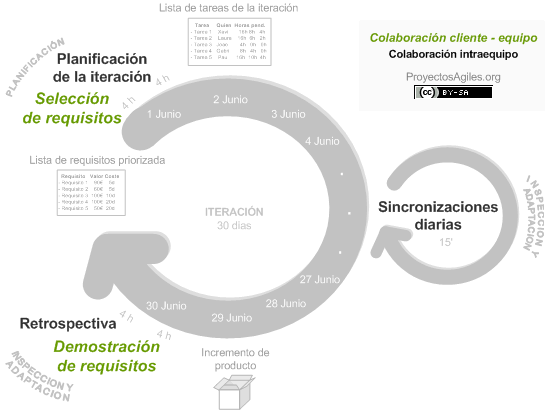
**4.1. Desarrollo.**

Desarrollar un software significa construirlo simplemente mediante su descripción. Esta es una muy buena razón para considerar la actividad de desarrollo de software como una ingeniería. En un nivel más general, la relación existente entre un software y su entorno es clara ya que el software es introducido en el mundo de modo de provocar ciertos efectos en el mismo.

**4.1.1. Métodos ágiles**

El desarrollo ágil de software envuelve un enfoque para la toma de decisiones en los proyectos de software, que se refiere a métodos de ingeniería del software basados en el desarrollo iterativo e incremental, donde los requisitos y soluciones evolucionan con el tiempo según la necesidad del proyecto. Así el trabajo es realizado mediante la colaboración de equipos auto-organizados y multidisciplinarios, inmersos en un proceso compartido de toma de decisiones a corto plazo.

En Scrum un proyecto se ejecuta en ciclos temporales cortos y de duración fija (iteraciones que normalmente son de 2 semanas, aunque en algunos equipos son de 3 y hasta 4 semanas, límite máximo de feedback de producto real y reflexión). Cada iteración tiene que proporcionar un resultado completo, un incremento de producto final que sea susceptible de ser entregado con el mínimo esfuerzo al cliente cuando lo solicite.



El proceso parte de la [lista de objetivos/requisitos priorizada](https://proyectosagiles.org/lista-requisitos-priorizada-product-backlog) del producto, que actúa como plan del proyecto. En esta lista **el**[**cliente (Product Owner)**](https://proyectosagiles.org/cliente-product-owner)[[13]](#footnote-14)**prioriza los objetivos balanceando el valor que le aportan respecto a su coste** (que el [equipo](https://proyectosagiles.org/equipo-team/) estima considerando la [Definición de Hecho](https://proyectosagiles.org/definicion-de-hecho-definition-of-done/)) y quedan repartidos en iteraciones y entregas.

Las actividades que se llevan a cabo en Scrum son las siguientes:

[Planificación de la iteración](https://proyectosagiles.org/planificacion-iteracion-sprint-planning)

El primer día de la iteración se realiza la reunión de planificación de la iteración. Tiene dos partes:

* **Selección de requisitos**(4 horas máximo). El cliente presenta al equipo la lista de requisitos priorizada del producto o proyecto. El equipo pregunta al cliente las dudas que surgen y selecciona los requisitos más prioritarios que se compromete a completar en la iteración, de manera que puedan ser entregados si el cliente lo solicita.
* **Planificación de la iteración**(4 horas máximo). El equipo elabora la [lista de tareas de la iteración](https://proyectosagiles.org/lista-tareas-iteracion-sprint-backlog)necesarias para desarrollar los requisitos a que se ha comprometido. La estimación de esfuerzo se hace de manera conjunta y los miembros del equipo se autoasignan las tareas.

[Ejecución de la iteración](https://proyectosagiles.org/ejecucion-iteracion-sprint)

Cada día el equipo realiza una [reunión de sincronización](https://proyectosagiles.org/reunion-diaria-de-sincronizacion-scrum-daily-meeting)(15 minutos máximo), normalmente delante de un[tablero físico o pizarra (Scrum Taskboard)](https://proyectosagiles.org/2010/09/26/ejemplo-tablero-pizarra-tareas-scrum-taskboard/). Cada miembro del equipo inspecciona el trabajo que el resto está realizando (dependencias entre tareas, progreso hacia el objetivo de la iteración, obstáculos que pueden impedir este objetivo) para poder hacer las adaptaciones necesarias que permitan cumplir con el compromiso adquirido. En la reunión cada miembro del equipo responde a tres preguntas:

* ¿Qué he hecho desde la última reunión de sincronización?
* ¿Qué voy a hacer a partir de este momento?
* ¿Qué impedimentos tengo o voy a tener?

Durante la iteración el [Facilitador (Scrum Master)](https://proyectosagiles.org/facilitador-scrum-master) se encarga de que el equipo pueda cumplir con su compromiso y de que no se merme su productividad.

* Elimina los obstáculos que el equipo no puede resolver por sí mismo.
* Protege al equipo de interrupciones externas que puedan afectar su compromiso o su productividad.

Durante la iteración, el cliente junto con el equipo [refinan la lista de requisitos (para prepararlos para las siguientes iteraciones) y, si es necesario, cambian o replanifican los objetivos del proyecto](https://proyectosagiles.org/replanificacion-proyecto) para [maximizar la utilidad de lo que se desarrolla](https://proyectosagiles.org/beneficios-de-scrum#flexibilidad-adaptacion) y el [retorno de inversión](https://proyectosagiles.org/beneficios-de-scrum#gestion-roi).

Inspección y adaptación

El último día de la iteración se realiza la reunión de revisión de la iteración. Tiene dos partes:

* [Demostración](https://proyectosagiles.org/demostracion-requisitos-sprint-review)(4 horas máximo). El equipo presenta al cliente los requisitos completados en la iteración, en forma de incremento de producto preparado para ser entregado con el mínimo esfuerzo. En función de los resultados mostrados y de los cambios que haya habido en el contexto del proyecto, el cliente realiza las adaptaciones necesarias de manera objetiva, ya desde la primera iteración, replanificando el proyecto.
* [Retrospectiva](https://proyectosagiles.org/retrospectiva-sprint-retrospective)(4 horas máximo). El equipo analiza cómo ha sido su manera de trabajar y cuáles son los problemas que podrían impedirle progresar adecuadamente, mejorando de manera continua su productividad. El Facilitador se encargará de ir eliminando los obstáculos identificados.

**4.1.2. Programación extrema**

La programación extrema o eXtreme Programming (de ahora en adelante, XP) es una metodología de desarrollo de la ingeniería de software formulada por Kent Beck, autor del primer libro sobre la materia, Extreme Programming Explained: Embrace Change (1999). Es el más destacado de los procesos ágiles de desarrollo de software. Al igual que éstos, la programación extrema se diferencia de las metodologías tradicionales principalmente en que pone más énfasis en la adaptabilidad que en la previsibilidad. Los defensores de la XP consideran que los cambios de requisitos sobre la marcha son un aspecto natural, inevitable e incluso deseable del desarrollo de proyectos. Creen que ser capaz de adaptarse a los cambios de requisitos en cualquier punto de la vida del proyecto es una aproximación mejor y más realista que intentar definir todos los requisitos al comienzo del proyecto e invertir esfuerzos después en controlar los cambios en los requisitos.

Se puede considerar la programación extrema como la adopción de las mejores metodologías de desarrollo de acuerdo a lo que se pretende llevar a cabo con el proyecto, y aplicarlo de manera dinámica durante el ciclo de vida del software.

Las características fundamentales del método son:

* Desarrollo iterativo e incremental: pequeñas mejoras, unas tras otras.
* [Pruebas unitarias](https://es.wikipedia.org/wiki/Prueba_unitaria) continuas, frecuentemente repetidas y automatizadas, incluyendo [pruebas de regresión](https://es.wikipedia.org/wiki/Pruebas_de_regresi%C3%B3n). Se aconseja escribir el código de la prueba antes de la codificación. Véase, por ejemplo, las herramientas de prueba [JUnit](https://es.wikipedia.org/wiki/JUnit) orientada a Java, DUnit orientada a Delphi, [NUnit](https://es.wikipedia.org/wiki/NUnit) para la plataforma.NET o [PHPUnit](https://es.wikipedia.org/wiki/PHPUnit)para PHP. Estas tres últimas inspiradas en JUnit, la cual, a su vez, se insipiró en SUnit, el primer framework orientado a realizar tests, realizado para el lenguaje de programación Smalltalk.
* [Programación en parejas](https://es.wikipedia.org/wiki/Programaci%C3%B3n_en_pareja): se recomienda que las tareas de desarrollo se lleven a cabo por dos personas en un mismo puesto. La mayor calidad del código escrito de esta manera -el código es revisado y discutido mientras se escribe- es más importante que la posible pérdida de productividad inmediata.
* Frecuente integración del equipo de programación con el cliente o usuario. Se recomienda que un representante del cliente trabaje junto al equipo de desarrollo.
* Corrección de todos los [errores](https://es.wikipedia.org/wiki/Error_de_programaci%C3%B3n) antes de añadir nueva funcionalidad. Hacer entregas frecuentes.
* [Refactorización](https://es.wikipedia.org/wiki/Refactorizaci%C3%B3n) del código, es decir, reescribir ciertas partes del código para aumentar su legibilidad y mantenibilidad pero sin modificar su comportamiento. Las pruebas han de garantizar que en la refactorización no se ha introducido ningún fallo.
* Propiedad del código compartida: en vez de dividir la responsabilidad en el desarrollo de cada módulo en grupos de trabajo distintos, este método promueve el que todo el personal pueda corregir y extender cualquier parte del proyecto. Las frecuentes pruebas de regresión garantizan que los posibles errores serán detectados.
* Simplicidad en el código: es la mejor manera de que las cosas funcionen. Cuando todo funcione se podrá añadir funcionalidad si es necesario. La programación extrema apuesta que es más sencillo hacer algo simple y tener un poco de trabajo extra para cambiarlo si se requiere, que realizar algo complicado y quizás nunca utilizarlo.

La simplicidad y la comunicación son extraordinariamente complementarias. Con más comunicación resulta más fácil identificar qué se debe y qué no se debe hacer. Cuanto más simple es el sistema, menos tendrá que comunicar sobre éste, lo que lleva a una comunicación más completa, especialmente si se puede reducir el equipo de programadores.

Roles:

* Programador: Escribe las pruebas unitarias y produce el código del sistema. Es la esencia del equipo.
* Cliente: Escribe las historias de usuario y las pruebas funcionales para validar su implementación. Asigna la prioridad a las historias de usuario y decide cuáles se implementan en cada iteración centrándose en aportar el mayor valor de negocio.
* Tester: Interpreta el pedido del cliente y ayuda al equipo de desarrollo a escribir las pruebas funcionales. Ejecuta pruebas regularmente, difunde los resultados en el equipo y es responsable de las herramientas de soporte para pruebas.
* Tracker: Es el encargado de seguimiento. Proporciona realimentación al equipo. Debe verificar el grado de acierto entre las estimaciones realizadas y el tiempo real dedicado, comunicando los resultados para mejorar futuras estimaciones.
* Entrenador (coach): Responsable del proceso global. Guía a los miembros del equipo para seguir el proceso correctamente.
* Consultor: Es un miembro externo del equipo con un conocimiento específico en algún tema necesario para el proyecto. Ayuda al equipo a resolver un problema específico. Además este tiene que investigar según los requerimientos.

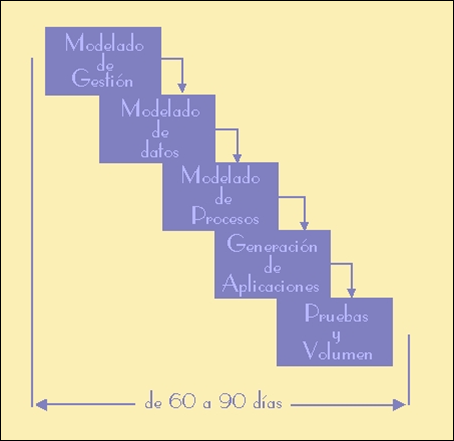
**4.1.3. Desarrollo rápido de aplicaciones**

El desarrollo rápido de aplicaciones o RAD por sus siglas en ingles Rapid Application Development.

El RAD es un proceso de desarrollo de software, desarrollado inicialmente por James Martin en 1980. El método comprende el desarrollo iterativo, la construcción de prototipos y el uso de utilidades CASE. Tradicionalmente, el desarrollo rápido de aplicaciones tiende a englobar también la usabilidad, utilidad y la rapidez de ejecución.

El Desarrollo Rápido de Aplicaciones (DRA) (Rapid Application Development RAD) es un modelo de proceso del desarrollo del software lineal secuencial que enfatiza un ciclo de desarrollo extremadamente corto. DRA es una adaptación a “Alta velocidad” en el que se logra el desarrollo rápido utilizando un enfoque de construcción basado en componentes. Si se comprenden bien los requisitos y se limita el ámbito del proyecto, el proceso DRA permite al equipo de desarrollo crear un “sistema completamente funcional” dentro de periodos cortos de tiempo. Cuando se utiliza principalmente para aplicaciones de sistemas de información, el enfoque DRA comprende las siguientes fases:

• Modelado de gestión: el flujo de información entre las funciones de gestión se modela de forma que responda a las siguientes preguntas: ¿Qué información conduce el proceso de gestión? ¿Qué información se genera? ¿Quién la genera? ¿A dónde va la información? ¿Quién la proceso?  
• Modelado de datos: el flujo de información definido como parte de la fase de modelado de gestión se refina como un conjunto de objetos de datos necesarios para apoyar la empresa. Se definen las características (llamadas atributos) de cada uno de los objetos y las relaciones entre estos objetos.  
• Modelado de proceso: los objetos de datos definidos en la fase de modelado de datos quedan transformados para lograr el flujo de información necesario para implementar una función de gestión. Las descripciones del proceso se crean para añadir, modificar, suprimir, o recuperar un objeto de datos. Es la comunicación entre los objetos.  
• Generación de aplicaciones: El DRA asume la utilización de técnicas de cuarta generación. En lugar de crear software con lenguajes de programación de tercera generación, el proceso DRA trabaja para volver a utilizar componentes de programas ya existentes (cuando es posible) o a crear componentes reutilizables (cuando sea necesario). En todos los casos se utilizan herramientas automáticas para facilitar la construcción del software.  
• Pruebas de entrega: Como el proceso DRA enfatiza la reutilización, ya se han comprobado muchos de los componentes de los programas. Esto reduce tiempo de pruebas. Sin embargo, se deben probar todos los componentes nuevos y se deben ejercitar todas las interfaces a fondo.



**Ventajas de RAD**

* Comprar puede ahorrar dinero en comparación con construir.
* Los entregables pueden ser fácilmente trasladados a otra plataforma.
* El desarrollo se realiza a un nivel de abstracción mayor.
* Visibilidad temprana.
* Mayor flexibilidad.
* Menor codificación manual.
* Mayor involucramiento de los usuarios.
* Posiblemente menos fallas.
* Posiblemente menor costo.
* Ciclos de desarrollo más pequeños.
* Interfaz gráfica estándar.

**Desventajas de RAD**

* Comprar puede ser más caro que construir.
* Costo de herramientas integradas y equipo necesario.
* Progreso más difícil de medir.
* Menos eficiente.
* Menor precisión científica.
* Riesgo de revertirse a las prácticas sin control de antaño.
* Más fallas (por síndrome de “codificar a lo bestia”).
* Prototipos pueden no escalar, un problema mayúsculo.
* Funciones reducidas (por “timeboxing”).
* Dependencia en componentes de terceros: funcionalidad de más o de menos, problemas legales.

**4.1.4. Prototipado del software**

El Modelo de prototipos, en Ingeniería de software, pertenece a los modelos de desarrollo evolutivo. El prototipo debe ser construido en poco tiempo, usando los programas adecuados y no se debe utilizar muchos recursos.

El diseño rápido se centra en una representación de aquellos aspectos del software que serán visibles para el cliente o el usuario final. Este diseño conduce a la construcción de un prototipo, el cual es evaluado por el cliente para una retroalimentación; gracias a ésta se refinan los requisitos del software que se desarrollará. La interacción ocurre cuando el prototipo se ajusta para satisfacer las necesidades del cliente. Esto permite que al mismo tiempo el desarrollador entienda mejor lo que se debe hacer y el cliente vea resultados a corto plazo.

Ventajas

* Este modelo es útil cuando el cliente conoce los objetivos generales para el software, pero no identifica los requisitos detallados de entrada, procesamiento o salida.
* También ofrece un mejor enfoque cuando el responsable del desarrollo del software está inseguro de la eficacia de un algoritmo, de la adaptabilidad de un sistema operativo o de la forma que debería tomar la interacción humano-máquina
* Se puede reutilizar el código.

Desventajas

* El usuario tiende a crearse unas expectativas cuando ve el prototipo de cara al sistema final. A causa de la intención de crear un prototipo de forma rápida, se suelen desatender aspectos importantes, tales como la calidad y el mantenimiento a largo plazo, lo que obliga en la mayor parte de los casos a reconstruirlo una vez que el prototipo ha cumplido su función. Es frecuente que el usuario se muestre reacio a ello y pida que sobre ese prototipo se construya el sistema final, lo que lo convertiría en un prototipo evolutivo, pero partiendo de un estado poco recomendado.
* En aras de desarrollar rápidamente el prototipo, el desarrollador suele tomar algunas decisiones de implementación poco convenientes (por ejemplo, elegir un lenguaje de programación incorrecto porque proporcione un desarrollo más rápido). Con el paso del tiempo, el desarrollador puede olvidarse de la razón que le llevó a tomar tales decisiones, con lo que se corre el riesgo de que dichas elecciones pasen a formar parte del sistema final.

**4.2. Reutilización del software.**

Es el proceso de creación de sistemas de software a partir de un software existente, en lugar de tener que rediseñar desde el principio.

**4.2.1. El campo de la reutilización**

Cuando los componentes ya desarrollados son similares y cumplen los requisitos para el dominio del proyecto, adquiéralos. Porque coste de la adquisición y de la integración de los componentes ya desarrollados serán casi siempre menores que el coste para desarrollar el software equivalente. Además, el riesgo es relativamente bajo.

Niveles de Reutilización

* Reutilización de Código
  1. Librerías de funciones, editores, inclusión de ficheros, mecanismos de herencia en POO, componentes, etc.
* Reutilización de Diseños
  1. No volver a inventar arquitecturas
  2. p.ej. patrones de diseño
  3. P.ej. patrones arquitectónicos (C/S, pipeline, OO, etc.)
* Reutilización de Especificaciones
  1. Reutilización de las abstracciones del dominio
  2. Debe estar asociada a la generación (semi)automática de los elementos de diseño e implementación.

**4.2.2. Patrones de diseño**

Los patrones de diseño son unas técnicas para resolver problemas comunes en el desarrollo de software y otros ámbitos referentes al diseño de interacción o interfaces.

Según la escala o nivel de abstracción hay tres categorías:

* Patrones de arquitectura: Aquellos que expresan un esquema organizativo estructural fundamental para sistemas de software.
* Patrones de diseño: Aquellos que expresan esquemas para definir estructuras de diseño (o sus relaciones) con las que construir sistemas de software.
* Dialectos: Patrones de bajo nivel específicos para un lenguaje de programación o entorno concreto.

Patrones creacionales[[14]](#footnote-15): Object Pool, Abstract Factory, Builder, Factory Method, Prototipe (Crea objetos clonados), Singleton (instancia única) y [Model View Controller (MVC)](https://es.wikipedia.org/wiki/Modelo_Vista_Controlador)  *Modelo Vista Controlador*.

Patrones estructurales[[15]](#footnote-16):

* [Adapter o Wrapper](https://es.wikipedia.org/wiki/Adapter_(patr%C3%B3n_de_dise%C3%B1o)) (Adaptador o Envoltorio): Adapta una interfaz para que pueda ser utilizada por una clase que de otro modo no podría utilizarla.
* [Bridge](https://es.wikipedia.org/wiki/Bridge_(patr%C3%B3n_de_dise%C3%B1o)) (Puente): Desacopla una abstracción de su implementación.
* [Composite](https://es.wikipedia.org/wiki/Composite_(patr%C3%B3n_de_dise%C3%B1o)) (Objeto compuesto): Permite tratar objetos compuestos como si de uno simple se tratase.
* [Decorator](https://es.wikipedia.org/wiki/Decorator_(patr%C3%B3n_de_dise%C3%B1o)) (Decorador): Añade funcionalidad a una clase dinámicamente.
* [Facade](https://es.wikipedia.org/wiki/Facade_(patr%C3%B3n_de_dise%C3%B1o)) (Fachada): Provee de una interfaz unificada simple para acceder a una interfaz o grupo de interfaces de un subsistema.
* [Flyweight](https://es.wikipedia.org/wiki/Flyweight_(patr%C3%B3n_de_dise%C3%B1o)) (Peso ligero): Reduce la redundancia cuando gran cantidad de objetos poseen idéntica información.
* [Proxy](https://es.wikipedia.org/wiki/Proxy_(patr%C3%B3n_de_dise%C3%B1o)): Proporciona un intermediario de un objeto para controlar su acceso.
* [Module](https://es.wikipedia.org/wiki/Module_(patr%C3%B3n_de_dise%C3%B1o)): Agrupa varios elementos relacionados, como clases, singletons, y métodos, utilizados globalmente, en una entidad única.

Patrones de comportamiento[[16]](#footnote-17):

* [Chain of Responsibility](https://es.wikipedia.org/wiki/Chain_of_Responsibility_(patr%C3%B3n_de_dise%C3%B1o)) (Cadena de responsabilidad): Permite establecer la línea que deben llevar los mensajes para que los objetos realicen la tarea indicada.
* [Command](https://es.wikipedia.org/wiki/Command_(patr%C3%B3n_de_dise%C3%B1o)) (Orden): Encapsula una operación en un objeto, permitiendo ejecutar dicha operación sin necesidad de conocer el contenido de la misma.
* [Interpreter](https://es.wikipedia.org/wiki/Interpreter_(patr%C3%B3n_de_dise%C3%B1o)) (Intérprete): Dado un lenguaje, define una gramática para dicho lenguaje, así como las herramientas necesarias para interpretarlo.
* [Iterator](https://es.wikipedia.org/wiki/Iterator_(patr%C3%B3n_de_dise%C3%B1o)) (Iterador): Permite realizar recorridos sobre objetos compuestos independientemente de la implementación de estos.
* [Mediator](https://es.wikipedia.org/wiki/Mediator_(patr%C3%B3n_de_dise%C3%B1o)) (Mediador): Define un objeto que coordine la comunicación entre objetos de distintas clases, pero que funcionan como un conjunto.
* [Memento](https://es.wikipedia.org/wiki/Memento_(patr%C3%B3n_de_dise%C3%B1o)) (Recuerdo): Permite volver a estados anteriores del sistema.
* [Observer](https://es.wikipedia.org/wiki/Observer_(patr%C3%B3n_de_dise%C3%B1o)) (Observador): Define una dependencia de uno-a-muchos entre objetos, de forma que cuando un objeto cambie de estado se notifique y actualicen automáticamente todos los objetos que dependen de él.
* [State](https://es.wikipedia.org/wiki/State_(patr%C3%B3n_de_dise%C3%B1o)) (Estado): Permite que un objeto modifique su comportamiento cada vez que cambie su estado interno.
* [Strategy](https://es.wikipedia.org/wiki/Strategy_(patr%C3%B3n_de_dise%C3%B1o)) (Estrategia): Permite disponer de varios métodos para resolver un problema y elegir cuál utilizar en tiempo de ejecución.
* [Template Method](https://es.wikipedia.org/wiki/Template_Method_(patr%C3%B3n_de_dise%C3%B1o)) (Método plantilla): Define en una operación el esqueleto de un algoritmo, delegando en las subclases algunos de sus pasos, esto permite que las subclases redefinan ciertos pasos de un algoritmo sin cambiar su estructura.
* [Visitor](https://es.wikipedia.org/wiki/Visitor_(patr%C3%B3n_de_dise%C3%B1o)) (Visitante): Permite definir nuevas operaciones sobre una jerarquía de clases sin modificar las clases sobre las que opera.

**4.2.3. Reutilización basada en generadores**

Los generadores de aplicación automatizan el proceso de desarrollo completo en un dominio de aplicación muy estrecho. Un generador de aplicación es una herramienta que toma especificaciones de entrada y produce programas ejecutables como salida. Las especificaciones son típicamente abstracciones de alto nivel en un dominio de aplicación específico.

**4.2.4. Marcos de trabajo de aplicaciones**

Un framework, entorno de trabajo​ o marco de trabajo​ es un conjunto estandarizado de conceptos, prácticas y criterios para enfocar un tipo de problemática particular que sirve como referencia, para enfrentar y resolver nuevos problemas de índole similar.

En el desarrollo de software, un entorno de trabajo es una estructura conceptual y tecnológica de asistencia definida, normalmente, con artefactos o módulos concretos de software, que puede servir de base para la organización y desarrollo de software. Típicamente, puede incluir soporte de programas, bibliotecas, y un lenguaje interpretado, entre otras herramientas, para así ayudar a desarrollar y unir los diferentes componentes de un proyecto.

Representa una arquitectura de software que modela las relaciones generales de las entidades del dominio, y provee una estructura y una especial metodología de trabajo, la cual extiende o utiliza las aplicaciones del dominio.

Los marcos de trabajo tienen como objetivo principal ofrecer una funcionalidad definida, auto contenida, siendo construidos usando patrones de diseño, y su característica principal es su alta cohesión y bajo acoplamiento.

Para acceder a esa funcionalidad, se construyen piezas, objetos, llamados objetos calientes, que vinculan las necesidades del sistema con la funcionalidad que este presta.

Esta funcionalidad, está constituida por objetos llamados fríos, que sufren poco o ningún cambio en la vida del framework, permitiendo la portabilidad entre distintos sistemas.

Algunos entornos de trabajo conocidos son [Spring Framework](https://es.wikipedia.org/wiki/Spring_Framework) o [Hibernate](https://es.wikipedia.org/wiki/Hibernate), donde lo esencial para ser denominados entornos de trabajo es estar constituidos por objetos casi estáticos con funcionalidad definida a nivel grupo de objetos y no como parte constitutiva de estos, por ejemplo en sus métodos, en cuyo caso se habla de una API o librería.

**4.2.5. Reutilización de sistemas de aplicaciones**

La reutilización de sistemas y aplicaciones consiste en el uso por las Administraciones públicas de las aplicaciones puestas a disposición por otras Administraciones públicas que sean titulares de los derechos de propiedad intelectual.

La reutilización de sistemas y aplicaciones comporta numerosas ventajas. En efecto, aprovechar soluciones y servicios compartidos y reutilizables basados en código abierto supone una oportunidad para la eficiencia de las Administraciones públicas, ya que permite reducir los costes de desarrollo y de tiempo en la implementación de aplicaciones e incrementar la interoperabilidad.

La reutilización de aplicaciones ahorra costes, tiempo e incrementa interoperabilidad.

**4.3. Ingeniería del software basada en componentes.**

La ingeniería de software basada en componentes (CBSE), también conocida como desarrollo basado en componentes(CBD), es una rama de la [ingeniería de software](https://es.wikipedia.org/wiki/Ingenier%C3%ADa_de_software) que enfatiza la [separación de asuntos](https://es.wikipedia.org/w/index.php?title=Separaci%C3%B3n_de_asuntos&action=edit&redlink=1), separation of concerns (SoC), por lo que se refiere a la funcionalidad de amplio rango disponible a través de un sistema de software dado. Es un acercamiento basado en la reutilización para definir, implementar, y componer [componente de software](https://es.wikipedia.org/wiki/Componente_de_software) débilmente acoplados en sistemas. Esta práctica persigue un amplio grado de beneficios tanto en el corto como el largo plazo, para el software en sí mismo y para las organizaciones que patrocinan tal software.

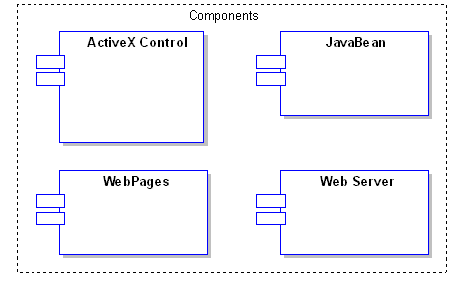
Los [ingenieros de software](https://es.wikipedia.org/wiki/Ingeniero_de_software) consideran los componentes como parte de la plataforma inicial para la [orientación a servicios](https://es.wikipedia.org/w/index.php?title=Orientaci%C3%B3n_a_servicios&action=edit&redlink=1). Los componentes juegan este rol, por ejemplo, en [servicios de web](https://es.wikipedia.org/wiki/Servicio_web) y, más recientemente, en las [arquitecturas orientadas a servicios](https://es.wikipedia.org/wiki/Arquitectura_orientada_a_servicios) (SOA), por el que un componente es convertido por el servicio web en un servicio y consiguientemente hereda otras características más allá de las de un componente ordinario.

Los componentes pueden producir o consumir eventos y pueden ser usados para las arquitecturas dirigida por eventos (EDA).

**4.3.1. Componentes y modelos de componentes**

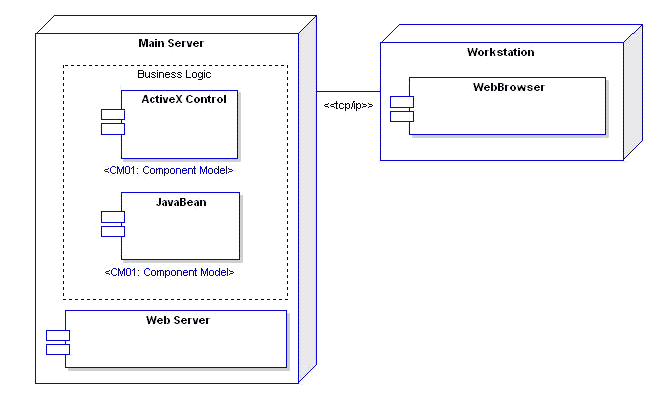
Este artículo describe cómo modelar los componentes de software y hardware en UML. El modelo de componentes ilustra los componentes de software que se usarán para construir el sistema. Se pueden construir a partir del modelo de clases y escribir desde cero para el nuevo sistema o se pueden importar de otros proyectos y de productos de terceros. Los componentes son agregaciones de alto nivel de las piezas de software más pequeñas y proveen un enfoque de construcción de bloques de “caja negra” para la elaboración de software.

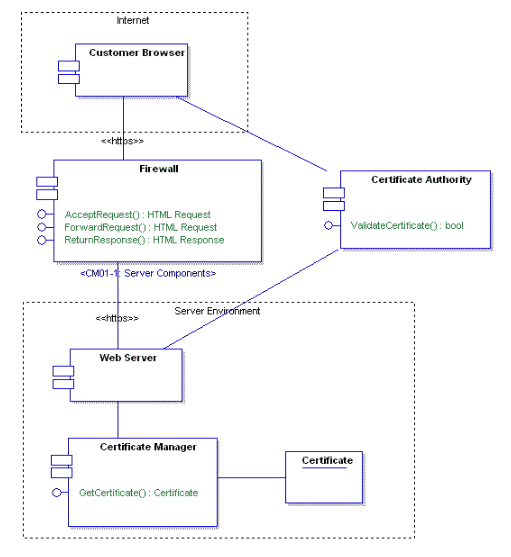
Un componente puede ser algo como un control Actives; tanto un componente de la interfaz de usuario como un servidor de reglas de negocio. Los componentes se representan gráficamente como muestra la figura siguiente:

****

El diagrama de componentes muestra la relación entre componentes de software, sus dependencias, su comunicación su ubicación y otras condiciones.

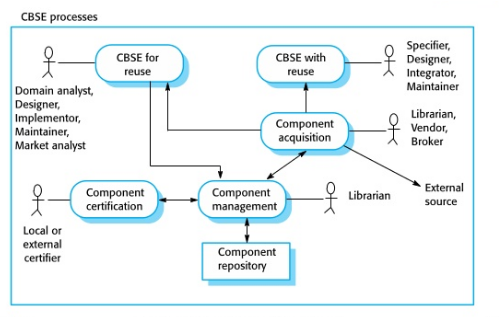
Un diagrama de despliegue muestra el despliegue físico del sistema en un ambiente de producción (o de prueba). Muestra dónde se ubican los componentes, en qué servidores, máquinas o hardware. Puede representar los enlaces de redes, el ancho de banda de la LAN, etc.

****

****

**4.3.2. El proceso CBSE**

El proceso (Component-Based Software Engineering) es la identificación de componentes. Esta implica varias subactividades.

****

**4.3.3. Composición de componentes**

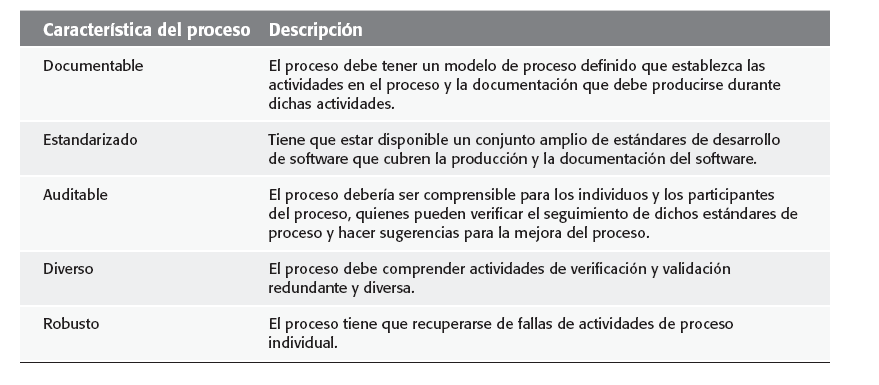
**4.4. Desarrollo de sistemas críticos.**

Los sistemas críticos se caracterizan por tener unas necesidades particulares, desde la propia concepción hasta la publicación en entornos de producción, muy especificas en las que conceptos como prevención, detección y tolerancia a defectos tienen que estar permanentemente encima de la mesa.

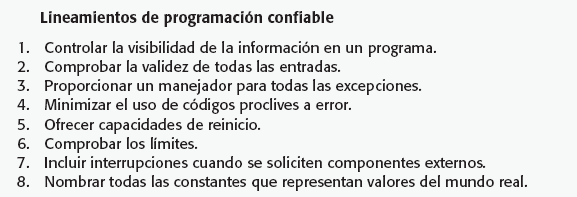
El objetivo es redundar sus sistemas y diversificarlos para garantizar su disponibilidad en el peor de los escenarios, a través de distintos Sistemas Operativos, Lenguajes de programación y Técnicas de programación.

**4.4.1. Procesos confiables**

Los procesos de software confiables están diseñados para producir software confiable. Una compañía que emplea un proceso confiable puede estar segura de que el proceso se realizó y documentó adecuadamente, y que se utilizaron técnicas de desarrollo adecuadas para el diseño de sistemas críticos. Atributos de procesos confiables:



**4.4.2. Programación confiable**

****

**4.4.3. Tolerancia a defectos**

La tolerancia a fallos o defectos es la propiedad que le permite a un sistema seguir funcionando correctamente en caso de fallo de uno o varios de sus componentes. Si disminuye su calidad de funcionamiento, la disminución es proporcional a la gravedad de la avería, en comparación con un sistema diseñado ingenuamente de forma que hasta un pequeño fallo puede causar el colapso total del sistema. Tolerancia a fallos es particularmente buscado en sistemas de alta disponibilidad.

**4.4.4. Arquitecturas tolerantes a defectos**

Los principios básicos de un sistema tolerante a fallas es bien conocido y consta de los siguientes pasos:

* Detección de la falla.
* Asegurarse que el sistema no seguirá fallando ?
* Identificar el origen de la falla. (cuál de los procesadores es el que falló)
* Reconfigurar los componentes del sistema.
* Volver el sistema a un punto en el cual pueda retomar su operación normal.

Para poder detectar la falla en algún procesador, existen varios métodos. Algunos son:

* El mismo CPU chequea internamente su lógica de operación. Si se detecta un error, envía una interrupción indicando que ha fallado y se detiene. Este mecanismo de detección es utilizado por el sistema Tandem16.
* La actividad de un CPU es replicado por otro. Así, se comparan sus salidas. En el caso de diferir una de la otra, un error ha sido detectado. Algunos de los sistemas que utilizan este método son el Stratus y Sequoia.

En el caso de producirse un error, el estado de la memoria compartida, en la mayoríade los casos, no es coherente con los datos contenidos en la cache interna del procesador que ha fallado y seguramente pueden existir otras incoherencias en los procesadores que no han fallado.

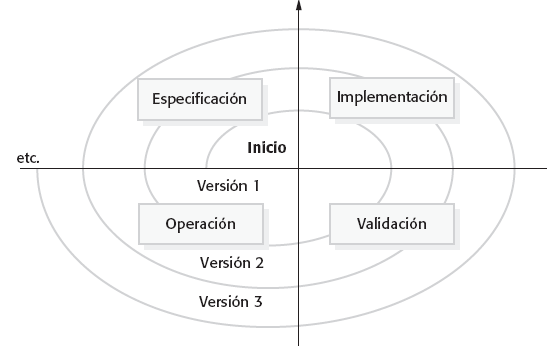
Una de los métodos de recuperación frente a falla es el "backward error recovery". Este es el mecanismo utilizado por el RSM, el cual consiste en:

* Establecer un [Punto de Recuperación](javascript:cons1();).
* Volver el sistema al estado del "Punto de Recuperación".
* Hacer "commit" al punto de recuperación.

**4.5. Evolución del software.**

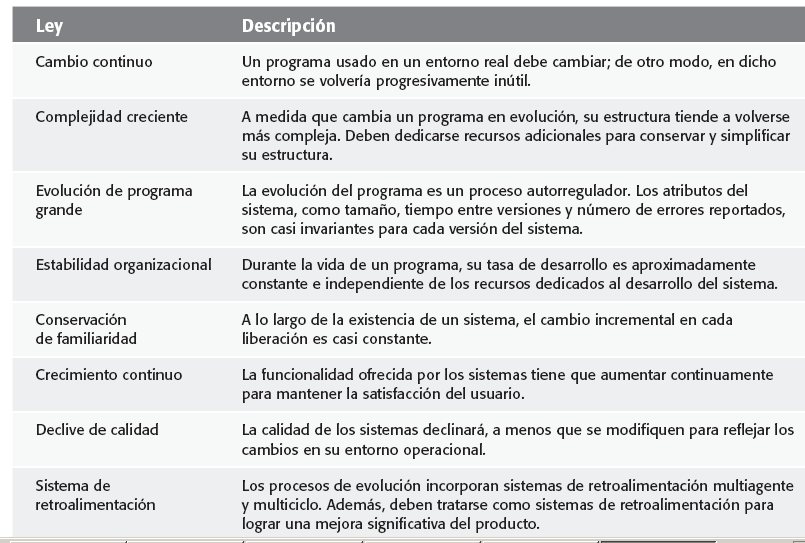
El desarrollo del software no se detiene cuando un sistema se entrega, sino que continúa a lo largo de la vida de éste. Después de distribuir un sistema, inevitablemente debe modificarse, con la finalidad de mantenerlo útil.

La evolución del software es importante porque las organizaciones invierten grandes cantidades de dinero en él y en la actualidad son completamente dependientes de dichos sistemas. Modelo en espiral de desarrollo y evolución:

****

**4.5.1. Dinámica de evolución de los programas**

Lehman y Belady investigaron la importancia de la retroalimentación en los procesos de evolución A partir de estos estudios, propusieron las “leyes de Lehman” relacionadas al cambio del sistema La primera ley afirma que el mantenimiento del sistema es un proceso inevitable. La segunda ley afirma que, conforme cambia un sistema, su estructura se degrada. La única manera de evitar que esto ocurra es invertir en mantenimiento preventivo. La tercera ley es, quizá, la más interesante y polémica de las leyes de Lehman. Sugiere que los sistemas grandes tienen una dinámica propia que se establece en una etapa temprana del proceso de desarrollo.

****

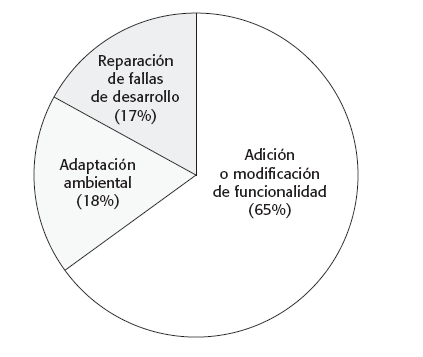
**4.5.2. Mantenimiento del software**

El mantenimiento del software es el proceso general de cambiar un sistema después de que éste se entregó.

Existen tres tipos de mantenimiento de software:

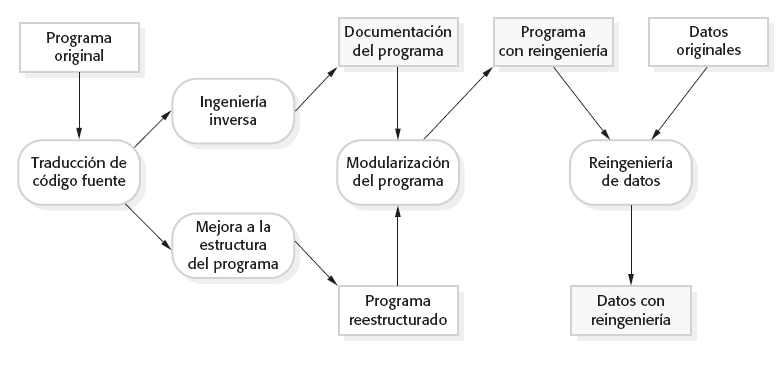
* *Reparaciones de fallas*
* *Adaptación ambiental*
* *Adición de funcionalidad*

Distribución del esfuerzo de mantenimiento:



**4.5.3. Procesos de evolución**

El proceso de evolución del sistema incluye comprender el programa que debe cambiarse y, luego, implementar dichos cambios, a continuación se visualiza un proceso de reingeniería[[17]](#footnote-18):

****

**4.5.4. Evolución de sistemas heredados**

Los sistemas heredados son sistemas antiguos que todavía son útiles y en ocasiones críticos para la operación de la empresa.

El valor empresarial de un sistema heredado y la calidad del software de aplicación y su entorno deben valorarse para determinar si el sistema tiene que sustituirse, transformarse

o mantenerse.

**5. VERIFICACIÓN Y VALIDACIÓN**

**5.1. Verificación y validación.**

Verificación[[18]](#footnote-19) y validación[[19]](#footnote-20) son dos importantes términos a menudo empleados en el contexto de los negocios, industrias y en el ámbito empresarial. Ambos tienen que ver con probar la [calidad y condición de un producto](http://diferenciaentre.info/diferencia-entre-cantidad-y-calidad/), sin embargo, no significan lo mismo.

**5.1.1. Planificación de la verificación y validación**

Esta actividad tiene como objetivo realizar la planificación de las actividades relacionadas con la Verificación del producto a ser desarrollado.

De acuerdo al documento Plan de Verificación, el responsable de Verificación deberá realizar la planificación de las actividades que se realizarán para verificar el sistema a ser desarrollado.

Esta actividad comprende:

* + 1. Planificar como se va a validar y verificar todo.
    2. Identificar y describir los métodos a ser usados para verificar que los casos de uso han sido aprobados por los analistas de requerimientos y los usuarios.
    3. Identificar y describir los métodos a ser usados para verificar que se mantienen las trazas entre los modelos.

**Entrada**

·         Especificación de Requerimientos

·         Requerimientos Suplementarios

·         Plan de Verificación y Validación

**Salida**

·         [Plan de Verificación y Validación](https://www.fing.edu.uy/inco/cursos/ingsoft/pis/memoria/dvd01/experiencia2002/ModsGX/modelo/entregab/plaverva.htm)[[20]](#footnote-21)

**Rol responsable**

·         Responsable de Verificación

**Roles involucrados**

·         No tiene

**5.1.2. Inspecciones de software**

Inspección en la Ingeniería de software, se relaciona con la Revisión por pares de cualquier producto de trabajo por personas capacitadas que buscan defectos mediante un proceso bien definido. La inspección de software es conocida también como Fagan inspection en homenaje a Michael Fagan, el creador de este popular método de inspección de software.

Las etapas en el proceso de las inspecciones son: Reunión de Planificación, Información general, preparación, reunión de inspección, repetición del trabajo y seguimiento. Etapas de la preparación, la inspección de reuniones y el trabajo repetitivo puede ser reiterado.

**5.1.3. Análisis estático automatizado**

Los analizadores estáticos de programas son herramientas de software que rastrean el texto fuente de un programa, en busca de errores no detectados por el compilador.

**5.1.4. Verificación y métodos formales**

En ingeniería de software un método formal es un camino a la construcción y análisis de modelos matemáticos que permitan una automatización del desarrollo de sistemas informáticos. Entre los métodos de verificación más utilizados, se encuentran:

* Aserciones E/S
* Precondición más débil
* Inducción estructural

**5.2. Pruebas del software.**

Las pruebas de software (en inglés software testing) son las investigaciones empíricas y técnicas cuyo objetivo es proporcionar información objetiva e independiente sobre la calidad del producto a la parte interesada o stakeholder. Es una actividad más en el proceso de control de calidad.

**5.2.1. Pruebas del sistema**

El Equipo de Proyecto comprobará cada componente que va generando para detectar posibles errores y evitar que éstos se reproduzcan en componentes o módulos posteriores. Además, cada vez que se implemente la comunicación entre dos o más módulos que estén integrados, el equipo de proyecto comprobará dicha integración. De este modo es más sencillo detectar y corregir los errores que si el sistema se encuentra totalmente desarrollado y todos sus módulos integrados. Para ello, deberá realizar las siguientes pruebas:

* **Pruebas unitarias.** Conjunto de pruebas que comprueban el correcto funcionamiento de cada componente de código por separado. Esto sirve para asegurar que cada uno de los módulos funcione correctamente por separado. Posteriormente, con las pruebas de integración, se podrá asegurar el correcto funcionamiento del sistema o subsistema en cuestión. Para la ejecución de las pruebas unitarias se deberá utilizar una herramienta que automatice el proceso, por ejemplo, JUnit. Se propone la siguiente [plantilla](http://www.juntadeandalucia.es/servicios/madeja/contenido/recurso/460) como ayuda a la definición de las pruebas unitarias.
* **Pruebas de integración.** Conjunto de pruebas que verifican la correcta integración entre todos los componentes/módulos del sistema. La necesidad de realizar las pruebas de integración viene dada por el hecho de que los módulos que forman un programa suelen fallar cuando trabajan de forma conjunta, aunque previamente se haya demostrado que funcionan correctamente de manera individual. Por ello se deberán realizar este tipo de pruebas, las cuáles asegurarán que los módulos que están relacionados se ejecutan correctamente. Con el uso de estas pruebas, se conseguirá formar el producto global a medida que se comprueba como los distintos componentes interaccionan y se comunican libres de errores. Para automatizar las pruebas de integración se pueden emplear las mismas herramientas que para las pruebas unitarias (por ejemplo, JUnit), pero los casos de pruebas por regla general serán más largos y la verificación de resultados puede requerir más de una comprobación. Se propone la siguiente [plantilla](http://www.juntadeandalucia.es/servicios/madeja/contenido/recurso/461) como ayuda a la definición de las pruebas de integración.
* **Pruebas de código estático.** Son verificaciones de código estático que todo programador debe realizar en su código para evitar errores de compilación, ejecución durante las fases posteriores. Un alto porcentaje de las pruebas se podrán automatizar en herramientas, tales como Checkstyle, PMD, Findbugs.

**5.2.2. Pruebas de componentes**

La prueba de componente también llamada prueba unitaria es por definición la prueba que se lleva a cabo luego de haber construido el componente.

**5.2.3. Diseño de casos de prueba**

Se elaborará y ejecutará el Plan de Pruebas Funcionales, cuyo objetivo es verificar los requisitos asociados a cada caso de pruebas. Para verificar que no existen defectos, se deberá incluir pruebas que impliquen acciones no permitidas por el sistema, para certificar que el sistema se comporta correctamente.

Se puede definir un **caso de prueba** como “el conjunto de entradas, condiciones de ejecución y resultados esperados" para satisfacer un objetivo concreto. La definición de los casos de pruebas se debe basar en la explotación de los distintos escenarios de ejecución de los casos de uso; en un diagrama de actividad, se puede asumir que existen distintos tipos de escenarios marcados por cada que camino que se puede trazar partiendo de su estado inicial.

**5.2.4. Automatización de las pruebas**

En las pruebas de software, la automatización de pruebas consiste en el uso de software especial (casi siempre separado del software que se prueba) para controlar la ejecución de pruebas y la comparación entre los resultados obtenidos y los resultados esperados. La automatización de pruebas permite incluir pruebas repetitivas y necesarias dentro de un proceso formal de pruebas ya existente o bien adicionar pruebas cuya ejecución manual resultaría difícil.

**5.3. Validación de sistemas críticos.**

El software es un elemento clave en los Sistemas de Seguridad Crítica, donde un fallo puede causar daños irreparables a personas, bienes o medio ambiente. Esta dependencia exige un elevado nivel de fiabilidad en este tipo de software.

Conseguir un software de calidad suficiente, se alcanza mediante desarrollos guiados en base a una serie de normas y estándares, que fuercen el uso de prácticas seguras y controles, minimizando las posibilidades de error y permitiendo un seguimiento de la evolución del mismo.

**5.3.1. Validación de la fiabilidad**

La validación es un proceso para medir la precisión, la fiabilidad y consistencia de un instrumento de prueba. Nunca lanzar un producto sin reunir primero la investigación práctica y teórica para validar que funciona como se anuncia. Hemos verificado que tenga su respectiva prueba de confiabilidad y validación especializadas para confirmar la idoneidad de los resultados.

**5.3.2. Garantía de la seguridad**

A medida que las organizaciones en todo el mundo confían cada vez más en los controles de software para proteger sus entornos informáticos y datos en la nube y en las instalaciones, el tema de ***Software Security Assurance*** adquiere cada vez más importancia. Los costos potenciales asociados con incidentes de seguridad, el surgimiento de regulaciones cada vez más complejas y los costos operativos continuos asociados con la actualización de parches de seguridad requieren que las organizaciones consideren cuidadosamente la forma en que abordan la seguridad del software y evalúan las prácticas de seguridad de sus proveedores de tecnología.

**5.3.3. Valoración de la protección**

En el derecho continental o europeo (en España mediante nuestra actual normativa de Propiedad Intelectual RDL 1/1996), el software se engloba dentro de las creaciones literarias. Su régimen jurídico se regula bajo el epígrafe de “programas de ordenador” (Art. 95 y ss. RDL 1/1996). Por tanto, el software es una obra de propiedad intelectual y no una invención susceptible de ser patentada. No por lo menos desde el punto de vista del ordenamiento jurídico europeo y dejando aparte los casos de software cuando son parte intrínseca de un hardware.

Por otro lado, como obra sujeta a las reglas de la Propiedad Intelectual, desde que comenzamos a crear un software suficientemente original, la ley nos reconoce todos los derechos sobre el mismo. Además, nos otorga toda su protección al creador sin necesidad de ningún trámite legal adicional. Este punto suele confundirse con la [Propiedad Industrial](https://delvy.es/glosario-startup/p/propiedadindustrial/) de marcas o diseños industriales. En estos casos sí estamos obligados a solicitar y obtener su registro para que la ley nos otorgue todos los derechos como titulares de esa marca o diseño industrial.

**5.3.4. Argumentos de confiabilidad y seguridad**

La IEEE define a la confiabilidad como “la habilidad que tiene un sistema o componente de realizar sus funciones requeridas bajo condiciones específicas en periodos de tiempo determinados”. La seguridad es la preservación la: confidencialidad, integridad y autenticidad según la ISO 27000.

Los errores pueden ser relacionados al diseño, a la implementación, o a la programación.

* Aunque casi todos los softwares tengan errores, la mayoría de los errores nunca serán revelados debajo de circunstancias Un atacante busca esta debilidad para atacar un sistema.
* Las organizaciones que desarrollan productos basados en software requieren de prácticas efectivas que permitan mejorar la calidad del producto. La Ingeniería de la Confiabilidad de Software es una práctica cuantitativa que puede ser implementada en organizaciones de cualquier tamaño bajo distintos modelos de desarrollo.

**6. GESTIÓN DE PERSONAL**

Las técnicas utilizadas para gestionar (contratar/despedir) empleados de cualquier organización, con el fin de aumentar su valor. Las técnicas de gestión de personal consisten en seleccionar a las personas adecuadas para el trabajo adecuado, con las habilidades adecuados, evaluar el trabajo, formar a los empleados, asegurando una comunicación efectiva dentro de los niveles y asegurarse de que los empleados están trabajando hasta sus niveles máximos de productividad**.**

**6.1. Gestión de personal.**

La gestión de personas de la empresa moderna está afectada por factores inherentes a su madurez que condicionan toda manifestación de su quehacer en los ámbitos organizativos, donde las personas convergen, para aprender, asociarse con otras, tomar decisiones, y agregar valor con lo que mejor saben hacer.

**6.1.1. Selección de personal**

La selección de personal es un proceso mediante el cual una empresa o una entidad que reemplaza al área de recursos humanos de una compañía se encarga de elegir a los empleados ideales para uno o varios puestos de trabajo. Conoce por qué el reclutamiento de personal es importante para encontrar a los candidatos perfectos.

* Un proceso que permite identificar a candidatos potenciales.
* Permite confirmar las habilidades de los futuros trabajadores.
* Permite conocer la personalidad de los postulantes.

**6.1.2. Motivación**

La motivación como práctica aislada en un determinado departamento/persona de una organización aunque puede aportar resultados, estos no los preveo como muy duraderos. La motivación debe obedecer a una política estratégica de gestión del valor humano a través practicas formalizadas a nivel interno, con el objetivo de conseguir una mayor satisfacción del conjunto de empleados como parte de un sistema (organización), redundando en un incremento de productividad y en consecuencia de los resultados. No es novedosa la afirmación que las personas motivadas trabajan más y mejor.

Pero no es posible motivar si no se conoce a quien motivar o si no se hace un análisis el método mas adecuado para ello. En éste sentido no vale el “pan para todos”. Al igual que las personas, no todas las empresas son iguales, cada sector tiene unas connotaciones especiales que harán que el negocio se organice de determinada

manera y en consecuencia de las personas que trabajan en él. Por ello creo en la importancia del conocimiento que debe tener toda persona que dirige un equipo, de los intereses y motivaciones del mismo. Ello será el primer paso para la consecución de un equipo motivado.

**6.1.3. Gestión de grupos**

La gestión de grupos es una habilidad que se puede desarrollar y que es básica para aquellos profesionales que tengan que trabajar con grupos como los educadores, docentes, monitores…

Un grupo sería un conjunto de personas que se reúnen con un fin común y que pueden estar caracterizados por un conjunto de características comunes. En el caso de la educación de calle este grupo además estará dirigido por un educador que deberá de presentar buenas habilidades sociales para el manejo del grupo.

Algunas cosas que se deben saber para la gestión de grupos

* **Se debe crear una actitud favorable en el grupo**. Evitar las etiquetar e intentar motivar a todas las personas por igual.
* **Intentar conseguir objetivo**s. Los resultados son colectivos, y lo que consigue uno afecta a los demás.
* A veces,**hay que tomar decisiones difíciles**, que al igual que los objetivos afectarán a todas las personas integrantes.
* **Hay que orientarse a la acción**. Hay que crear el plan necesario para que el grupo cumpla los objetivos propuestos.
* **Lidiar con la variedad de opiniones**. Llegar al consenso será difícil, la complejidad y los procesos de cambio serán lo normal dentro del grupo.
* **Autorregulación y autocontrol.** En ocasiones, en la gestión de grupos, cuando el grupo no funciona es probable que se tenga una sensación de desánimo. Es entonces cuando hay que desarrollar la capacidad de autocontrol, para que la gestión del grupo sea lo más eficiente posible.
* **Orientación al desarrollo**. Las personas del grupo tienen que tener la oportunidad de desarrollo y autonomía. Y hay que fomentar que cada grupo sea responsable de sus situaciones y creativo en la resolución de problemas e igualmente incentivar y recompensar cuándo esto sea así.
* **Hay que cambiar la mirada**. La persona que realiza la gestión de grupos de chic@s tiene que tener una mirada positiva, hay que mirar siempre desde la capacidad y no desde la incapacidad. Es muy importante que el educador no enfatice en ningún momento aquellas cosas que los chic@s no están preparados para hacer, sino que por el contrario, se trabajen aquellas cosas positivas que hacen bien.
* **Aprender a trabajar con las dinámicas de grupo**. Y cuando se habla de dinámicas de grupo no se habla de actividades sino más bien de fases, roles, funciones y disfunciones, que se forman en un grupo. Hay que saber tener en cuenta que cada persona toma unos roles y que estos pueden pasar por diferentes momentos y fases.
* **Paciencia. Ir deprisa no sirve de nada**. La estrategia es la mejor opción para ver como se avanza positivamente y paso a paso. Lo ideal es tener objetivos a corto plazo y concentrarse en ellos sin olvidar la meta final.

**6.1.4. El modelo de madures de la capacidad del personal**

People Capability Maturity Model es una guía de prácticas que permiten mejorar la capacidad del personal de la organización. Permite atraer, desarrollar, organizar, motivar y retener al personal que permitirá crear productos y proveer los servicios.

Es un modelo de excelencia para el negocio en general, que permite organizar las actividades de administración de las personas, con prácticas de administración del cambio, para mejorar la capacidad del personal y la efectividad de la organización. Como resultado la Organización será reconocida como un empleador deseado y su personal contará con las competencias necesarias para cubrir los objetivos del negocio.

El modelo PCMM esta estructurado en cinco niveles de madurez que evolucionan de una organización ad hoc a una enfocada a la mejora continua de las prácticas de RR.HH. En total agrupa 22 áreas de procesos agrupadas en:

* Desarrollo de la capacidad individual
* Creación de la cultura y grupos de trabajo
* Motivación y administración del desempeño
* Configuración de la plantilla

****

Modelo PCMM.

**6.2. Estimación de costes del software.**

La estimación de estos costos ha preocupado a analistas de sistema, gerentes de proyecto e ingenieros de software durante décadas.

**6.2.1. Productividad**

La productividad es la relación entre el producto obtenido, y los recursos necesarios para obtenerlo. En desarrollo, podríamos entender la productividad como la cantidad de código (se entiende bien desarrollado) que se produce con respecto a un tiempo determinado, porque desarrollar es un trabajo que se realiza en el tiempo.

Si para aumentar la productividad pretendemos que se aumenten las horas trabajadas, por la propia definición de productividad, ésta bajará, ya que hará falta más horas para llevar a cabo el proceso (aunque estas horas se hagan en menos tiempo).

Si bien un empresario típico puede pensar que la productividad aumenta (tardo menos tiempo en tener el producto y lo cobro antes), sobre todo si no se pagan esas horas extra, en realidad ese esfuerzo tiene, de forma irremediable, un efecto negativo en la productividad.

Por explicarlo con un ejemplo sencillo, si nuestras máquinas producen 20 objetos X a la hora, forzando las máquinas podremos tener 30 objetos X a la hora, so pena de que algunos de los productos sean defectuosos o las máquinas se estropeen por el sobreesfuerzo.

**6.2.2. Técnicas de estimación**

Esta actividad es realizada por el jefe de proyecto, quien es responsable de hacer dichas estimaciones lo más precisas posible. En la actualidad existen técnicas que permiten realizar esta labor aunque, lamentablemente, aún no hay técnicas maduras específicas para enfoques de desarrollo como la orientación a objetos o los sistemas expertos. A ello se suma el problema de la escasa información proporcionada por las técnicas de estimación existentes para su aplicación a ciclos de vida de desarrollo de software diferentes al de cascada, como, por ejemplo, los ciclos de vida incremental o iterativo. Debido a la problemática existente, se diseñó Tupuy que es un conjunto de técnicas que apoya en la estimación y planificación basada en Puntos de función para proyectos de desarrollo de software orientados a objetos que empleen un modelo de ciclo de vida incremental.

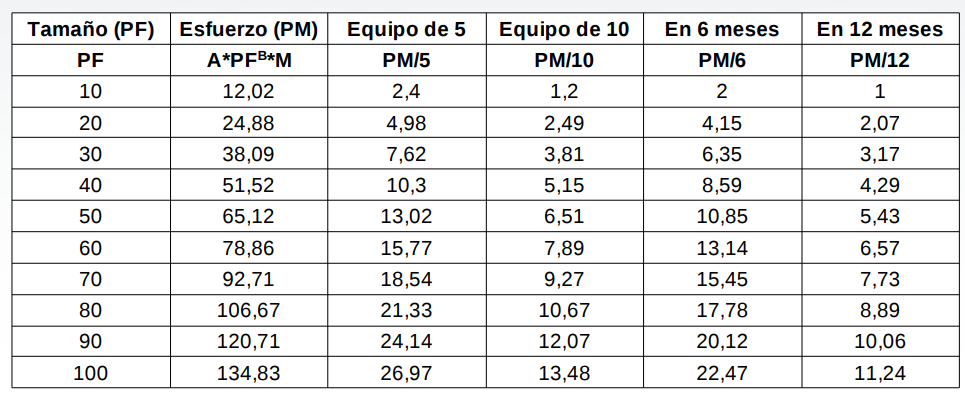
* Modelado del algoritmo de costos: se realiza un modelado con alguna métrica de software y se obtiene el costo estimado
* Opinión de expertos: se consulta a varios especialistas los cuales dan su opinión acerca del costo de proyecto, se sacan conclusiones al respecto.
* Estimación por analogía: cuando ya se han realizado proyectos similares se puede calcular la estimación de costos fácilmente.

**6.2.3. Modelado algorítmico de costes**

Si se puede tener una estimación del tamaño de la aplicación, entonces es posible calcular el esfuerzo requerido para desarrollar el sistema con una fórmula como la siguiente:

Esfuerzo=A \* TamañoB \* M

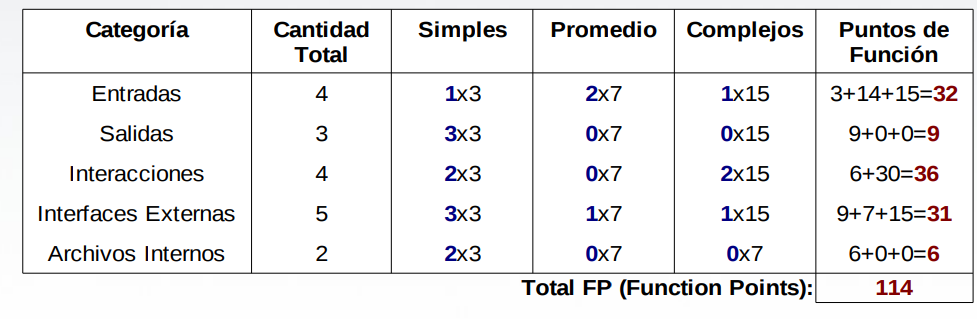
Donde el esfuerzo viene dado en P/M (Personas / Mes)

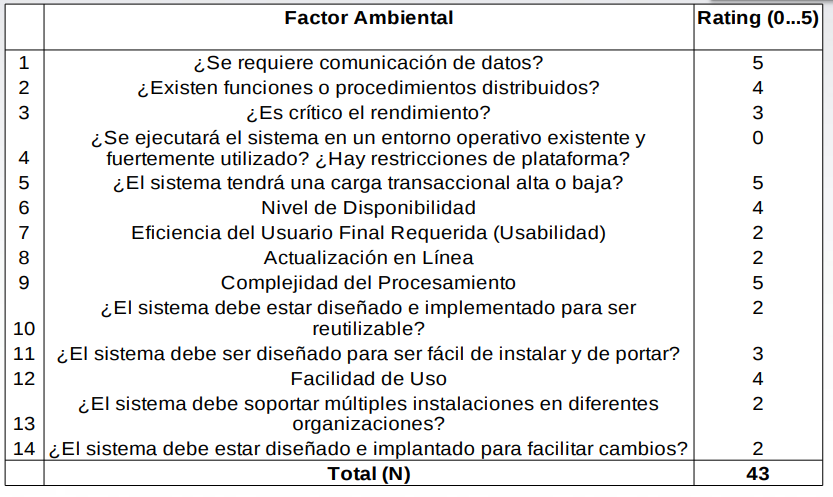


Puntos de Función

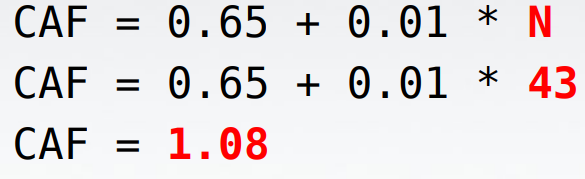
Los puntos de función miden el tamaño de un sistema en términos de la cantidad de funcionalidad del sistema. Las líneas de código en si mismas no son una métrica adecuada para medir el tamaño de un sistema. Es necesaria una métrica que sea independiente de la tecnología utilizada. Componentes:

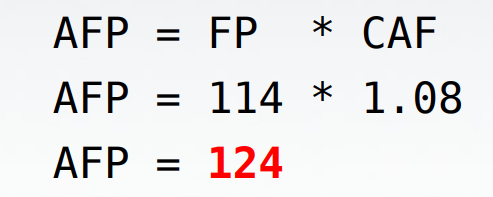
* Entradas: IU -> (Archivos / BD / Otros Sistemas)
* Salidas: (Archivos / BD / Otros Sistemas) -> IU
* Interacciones / Consultas: IU -> Archivos / BD -> UI
* Interfaces Externas: Integración con otras aplicaciones, bases de datos, etcétera externas al sistema
* Archivos (Interfaces) Internos: Integración con fuentes de datos internas.





CALCULA CAF Complexity Adjustment Factor y AFP Adjusted Funtion Points:





**6.2.3.1. El modelo de COCOMO**

El Modelo Constructivo de Costes (Constructive Cost Model) fue desarrollado por B. W. Boehm a finales de los 70 y comienzos de los 80, exponiéndolo detalladamente en su libro "Software Engineering Economics" (Prentice-Hall, 1981). COCOMO es una jerarquía de modelos de estimación de costes software que incluye submodelos básico, intermedio y detallado.



* S el número de miles de líneas de código fuente
* m(X) es un multiplicador que depende de 15 atributos
* Coeficientes ai y bi:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Modo** | **Descripción** | **Básico** | | **Intermedio** | |
| **ai** | **bi** | **ai** | **bi** |
| Orgánico | Pequeño grupo de programadores experimentados desarrollan software en un entorno familiar. | 2.4 | 1.05 | 3.2 | 1.05 |
| Semiencajado | El proyecto tiene unas fuertes restricciones, que pueden estar relacionadas con el procesador y el interface hardware. El problema a resolver es único y es difícil basarse en la experiencia. | 3.0 | 1.12 | 3.0 | 1.12 |
| Empotrado | Es un modo intermedio entre los dos anteriores. Dependiendo del problema, el grupo puede incluir una mezcla de personas experimentadas y no experimentadas. | 3.6 | 1.2 | 2.8 | 1.2 |

|  |  |  |
| --- | --- | --- |
| **Modo** | **Básico** | |
| **Coste** | **Tiempo** |
| Orgánico | Km = 2.4 Sk1.05 | td = 2.5 Km0.38 |
| Semiencajado | Km = 3.6 Sk1.20 | td = 2.5 Km0.32 |
| Empotrado | Km = 3.0 Sk1.12 | td = 2.5 Km0.35. |

Para el intermedio se emplea los índices de la tabla intermedio.

Km  se expresa en personas-mes.

Sk es el tamaño expresado en miles de líneas de código fuente.

td es el tiempo de desarrollo en meses.

Cada atributo se cuantifica para un entorno de proyecto:

|  |  |
| --- | --- |
| **DE SOFTWRE** | * **RELY**: garantía de funcionamiento requerida al software. Indica las posibles consecuencias para el usuario en el caso que existan defectos en el producto. Va desde la sola inconveniencia de corregir un fallo (*muy bajo*) hasta la posible pérdida de vidas humanas (*extremadamente alto*, software de alta criticidad). * **DATA**: tamaño de la base de datos en relación con el tamaño del programa. El valor del modificador se define por la relación: {\displaystyle D/K}D/K, donde D corresponde al tamaño de la base de datos en bytes y K es el tamaño del programa en cantidad de líneas de código. * **CPLX**: representa la complejidad del producto. |
| **DE HARDWARE** | * **TIME**: limitaciones en el porcentaje del uso de la CPU. * **STOR**: limitaciones en el porcentaje del uso de la memoria. * **VIRT**: volatilidad de la máquina virtual. * **TURN**: tiempo de respuesta requerido. |
| **DE PERSONAL** | * **ACAP**: calificación de los analistas. * **AEXP**: experiencia del personal en aplicaciones similares. * **PCAP**: calificación de los programadores. * **VEXP**: experiencia del personal en la máquina virtual. * **LEXP**: experiencia en el lenguaje de programación a usar. |
| **DE PROYECTO** | * **MODP**: uso de prácticas modernas de programación. * **TOOL**: uso de herramientas de desarrollo de software. * **SCED**: limitaciones en el cumplimiento de la planificación. |

La escala es muy bajo - bajo - nominal - alto - muy alto - extremadamente alto. Dependiendo de la calificación de cada atributo, se asigna un valor para usar de multiplicador en la fórmula (por ejemplo, si para un proyecto el atributo DATA es calificado como muy alto, el resultado de la fórmula debe ser multiplicado por 1000).

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Atributos | Valor | | | | | | |
| Muy bajo | | Bajo | Nominal | Alto | Muy alto | Extra alto |
| Atributos de software | | | | | | | |
| Fiabilidad | | 0,75 | 0,88 | 1,00 | 1,15 | 1,40 |  |
| Tamaño de Base de datos | |  | 0,94 | 1,00 | 1,08 | 1,16 |  |
| Complejidad | | 0,70 | 0,85 | 1,00 | 1,15 | 1,30 | 1,65 |
| Atributos de hardware | | | | | | | |
| Restricciones de tiempo de ejecución | |  |  | 1,00 | 1,11 | 1,30 | 1,66 |
| Restricciones de memoria virtual | |  |  | 1,00 | 1,06 | 1,21 | 1,56 |
| Volatilidad de la máquina virtual | |  | 0,87 | 1,00 | 1,15 | 1,30 |  |
| Tiempo de respuesta | |  | 0,87 | 1,00 | 1,07 | 1,15 |  |
| Atributos de personal | | | | | | | |
| Capacidad de análisis | | 1,46 | 1,19 | 1,00 | 0,86 | 0,71 |  |
| Experiencia en la aplicación | | 1,29 | 1,13 | 1,00 | 0,91 | 0,82 |  |
| Calidad de los programadores | | 1,42 | 1,17 | 1,00 | 0,86 | 0,70 |  |
| Experiencia en la máquina virtual | | 1,21 | 1,10 | 1,00 | 0,90 |  |  |
| Experiencia en el lenguaje | | 1,14 | 1,07 | 1,00 | 0,95 |  |  |
| Atributos del proyecto | | | | | | | |
| Técnicas actualizadas de programación | | 1,24 | 1,10 | 1,00 | 0,91 | 0,82 |  |
| Utilización de herramientas de software | | 1,24 | 1,10 | 1,00 | 0,91 | 0,83 |  |
| Restricciones de tiempo de desarrollo | | 1,22 | 1,08 | 1,00 | 1,04 | 1,10 |  |

**6.2.3.2. El modelo Putnam**

El modelo de Putnam o Modelo SLIM (Software LIfecycle Management), es un modelo de estimación de esfuerzo software, un método de los llamados empíricos. El método en cuestión tiene ya sus años, el artículo original que describe el método Putnam fue publicado en 1978.

****

* *T* es el tamaño en LDC.
* *C* es un factor que depende del entorno (vale 2.000 para entornos poco productivos, 8.000 para entornos buenos, 11.000 para entornos excelentes).
* *K* es el esfuerzo en personas-año, y {\displaystyle t\_{d}}td es el tiempo para completar el proyecto, medido en años.

Para utilizar la ecuación, es necesaria una estimación del tamaño en LDC, fijar C y dejar K y {\displaystyle t\_{d}}td constantes para calcular una de las dos. Despejando K se aprecia que el esfuerzo es proporcional a la cuarta potencia del tiempo necesario para la entrega. Así, si queremos entregar el trabajo en la mitad de tiempo, el esfuerzo necesario en personas-año se multiplicará por 16.

**6.2.3.3. Modelos algorítmicos de costes en la planificación**

Permite evaluar riesgos de cada opción y revela gastos financieros asociados a diferentes decisiones de gestión.

**6.2.4. Duración y personal del proyecto**

La duración del proyecto es el número de unidades de tiempo (horas, días, semanas, meses...) necesarias para llevar a cabo el proyecto. La duración del proyecto depende de la duración de cada una de las tareas y de las relaciones de dependencia que existan entre ellas.La duración del proyecto depende de la duración de cada una de las tareas y de las relaciones de dependencia que existan entre ellas. Dada una fecha de inicio, la duración del proyecto NO nos proporciona directamente la fecha de finalización del proyecto. Para conocer la fecha de finalización es necesario utilizar el calendario del proyecto y el de cada uno de los recursos que participan. Así pues, el equipo del proyecto debe contar con suficiente personal cualificado y competente para llevarlo a cabo. En lo que respecta a su especialización, experiencia y conocimientos, los miembros del equipo deben ser complementarios para que a medida que el proyecto avance el producto total sea mayor que la suma de sus partes.

**6.3. Gestión de la calidad.**

El término gestión de calidad tiene significados específicos dentro de cada sector del negocio. Esta definición, que no apunta al aseguramiento de la buena calidad por la definición más general sino a garantizar que una organización o un producto sea consistente, tiene cuatro componentes:

* Planeamiento de la calidad
* Control de la calidad
* Aseguramiento de la calidad
* Mejoras en la calidad.

La gestión de calidad se centra no solo en la calidad de un producto , servicio o la satisfacción de sus clientes, sino en los medios para obtenerla. Por lo tanto, la gestión de calidad utiliza al aseguramiento de la calidad y el control de los procesos para obtener una calidad más consistente.

**6.3.1. Calidad de proceso y producto**

La calidad vista desde el mundo de los PROCESOS nos dice que la calidad del PRODUCTO software está determinada por la calidad del PROCESO. Un buen proceso sin duda segura un buen producto.

**6.3.2. Garantía de la calidad y estándares**

**Ga**rantía de calidad del software (SQA) consiste en los medios de la supervisión tecnología de dotación lógica los procesos y los métodos aseguraban calidad. Hace esto por medio de intervenciones de sistema de gerencia de la calidad debajo de cuál se crea el sistema de software. Estas intervenciones son movidas hacia atrás por unos o más estándares, generalmente ISO 9000.

La calidad del software es el conjunto de cualidades que lo caracterizan y que determinan su utilidad y existencia. La calidad es sinónimo de eficiencia, flexibilidad, corrección, confiabilidad, mantenibilidad, portabilidad, usabilidad, seguridad e integridad. La calidad del software es medible y varía de un sistema a otro o de un programa a otro.

“La calidad del software es el grado con el que un sistema, componente o proceso cumple los requerimientos especificados y las necesidades o expectativas del cliente o usuario”.

El grupo encargado de SQA:

* Trabaja con el equipo del proyecto desde el inicio.
* Debe ser objetivo e independiente.
* Ayuda al proyecto, más que controlar sus actividades.

La actividad de SQA es el proceso de verificación de que los estándares sean aplicados correctamente. En los proyectos pequeños esto se puede realizar por el equipo de desarrollo, pero en proyectos grandes, un grupo específico se debe dedicar a este rol.

La prueba del software es tanto un arte como una ciencia. En grande, los usos complejos, tales como sistemas operativos.Diversos usos del software requieren diversos acercamientos cuando viene a la prueba, pero algunas de las tareas mas comunes del QA del software incluyen:

* **Prueba de la validación** La prueba de la validación es el acto de los datos que entran que el probador sabe para ser erróneo en un uso. Comparación de los datos Comparando la salida de un uso con parámetros específicos a un sistema previamente creado de los datos con los mismos parámetros que se saben para ser exactos.
* **Prueba de la tensión** Una prueba de tensión es cuando el software se utiliza tan pesadamente como sea posible por un período de la hora de considerar si hace frente a los altos niveles de la carga.
* **Prueba de la utilidad** A veces consiguiendo a los usuarios que son desconocedores con el software intentarlo durante algún tiempo y ofrecer la regeneración a los reveladores sobre lo que encontraron difíciles de hacer es la mejor manera de llevar a cabo mejoras a un

**6.3.2.1. ISO 9000**

Toda organización consta necesariamente de una estructura organizacional o una forma de organización de acuerdo a sus necesidades, por medio de la cual se pueden ordenar las actividades, los procesos y el funcionamiento en sí de la compañía.

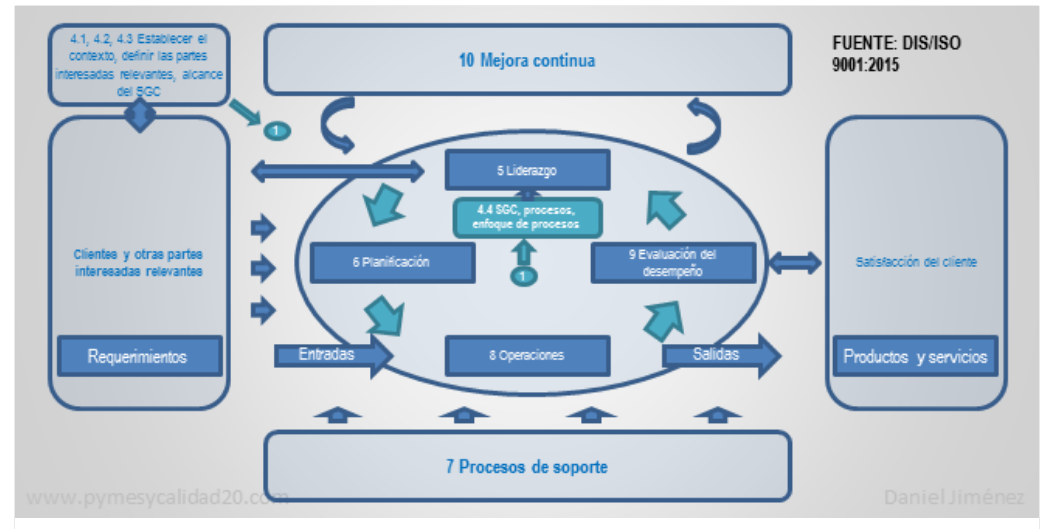
Las estructuras organizacionales son los diferentes patrones de diseño para constituir una empresa, con el fin de cumplir las metas propuestas y lograr el objetivo deseado.

ISO 9001 es una norma de sistemas de gestión de la calidad (SGC) reconocida internacionalmente. La norma ISO 9001 es un referente mundial en SGC, superando el millón de certificados en todo el mundo.

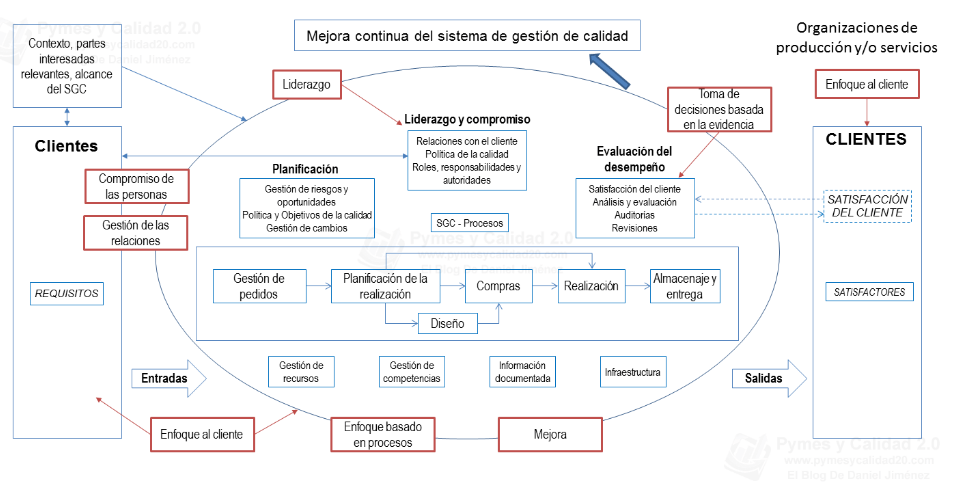
La norma ISO 9001 de sistemas de gestión de la calidad proporciona la infraestructura, procedimientos, procesos y recursos necesarios para ayudar a las organizaciones a controlar y mejorar su rendimiento y conducirles hacia la eficiencia, servicio al cliente y excelencia en el producto.



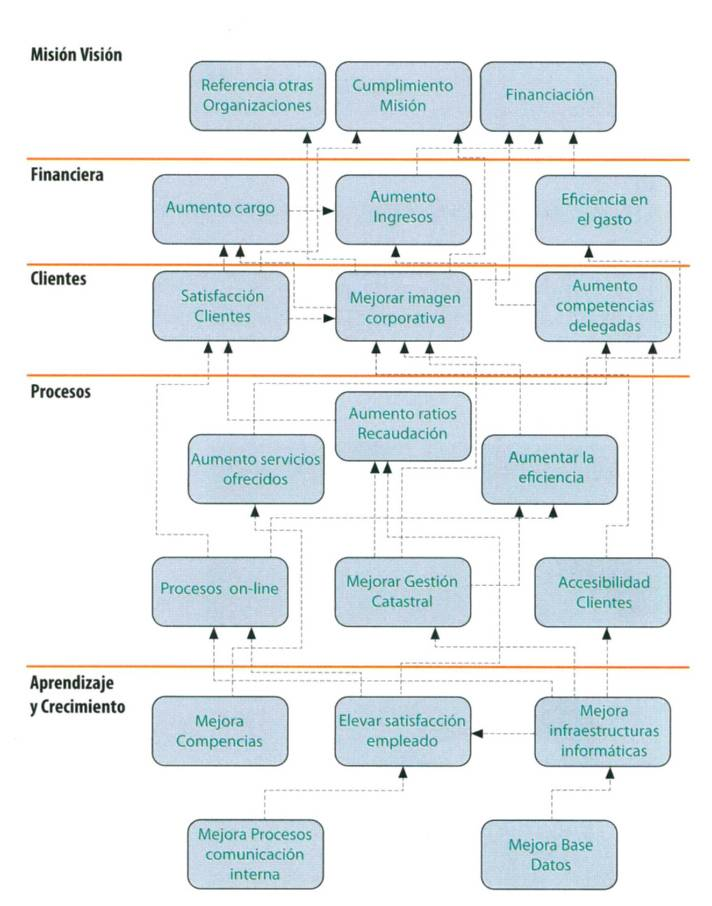
Logo ISO 9001.



Estructura de la ISO 9001.



Procesos de Norma ISO 9001.



Gestión de Procesos ISO 9001.

**6.3.2.2. Estándares[[21]](#footnote-22) de documentación**

Durante el desarrollo de un sistema, desde su concepción hasta su puesta en marcha se ha generado gran cantidad de documentos, que en muchas ocasiones se han visto modificados por documentos posteriores debido a cambios en el sistema.

Para evitar confusiones en las revisiones de la documentación se desarrollan diferentes tipos de documentos dirigidos a las personas que trabajarán con el sistema y para facilitar el mantenimiento del mismo. La documentación de un sistema debe ser marcada adecuadamente, bien organizada actualizada y completa; todos los términos utilizados deben explicarse. La documentación se hará disponible a todos los usuarios dc acuerdo a sus necesidades.

El estilo de redacción de los manuales de documentación debe ser:

* Concreto.
* Ser preciso y definir los términos utilizados.
* Utilizar párrafos cortos.
* Utilizar títulos y subtítulos.
* Utilizar formas activas en lugar de pasivas.
* No emplear frases largas que presenten hechos distintos.
* No hacer referencia a una información solamente con el número de referencia

Aún cuando las normas de documentación varían de una instalación a otra, es esencial que dentro de una organización, se utilice un solo método. El uso de procedimientos y documentación estandarizada proporciona la base de una comunicación clara y rápida, adiestramiento menos costoso del personal de sistemas, reducción de costos de almacenamiento, y otros.

Ventajas de La Estandarización[[22]](#footnote-23)

* Ayuda al ***entrenamiento*** del nuevo personal dentro y fuera de la organización de Sistemas.
* Es útil para cualquiera que tenga la ***responsabilidad*** del mantenimiento de los sistemas.
* Ayuda a los analistas y diseñadores de sistemas en el trabajo de ***integración*** de sistemas.
* Asegura que el sistema opere ***correctamente***.
* Se utilizan ***eficientemente*** los recursos que se dispongan.

Toda documentación que se relacione con un sistema, ya sea manual o por computadora, sencillo o complejo debe reunir los siguientes requisitos básicos:

* Debe ser rotulada con claridad y bien organizada, con secciones claramente indicadas, almacenarlas en carpetas e índice.
* Los diagramas deberán ser claros, no aglomerados y la escritura manuscrita deberá ser legible.
* La documentación deberá ser completa.
* Se incluirá una leyenda o explicación de los términos utilizados.
* La documentación siempre se conserva actualizada.

Asegúrese de que los estándares sean completos, actualizados, documentados y legibles.

Auditar permanentemente para que se cumplan los estándares.

Evaluar si los estándares establecidos son los requeridos y hacer los cambios necesarios para que dichos estándares sean los apropiados.

**6.3.3. Planificación de la calidad**

Debemos saber a qué nos referimos cuando hablamos de planificación de la calidad según la norma ISO 9001, por lo que durante este artículo vamos a desarrollar dicho concepto y los requisitos necesarios para llevarlo a cabo.Lo primero que debemos tener claro es que significa la planificación según la norma ISO 9001 por lo que vamos a explicarlo. La organización tiene que establecer, implantar, mantener y mejorar de forma continua su Sistema de Gestión de Calidad, en el que se deben incluir todos los procesos necesarios para ello, así como las interacciones necesarias según los requisitos que establece la norma ISO 9001.Debemos tener en cuenta los procesos que se realizan en la organización, ya que ésta debe identificar y determinar todos los procesos que sean necesarios para desarrollar las actividades que lleva a cabo definiendo todos los criterios y métodos para realizar su ejecución y controlar su eficacia, además de las medidas, análisis y mejora continua.Para poder realizar una planificación primero debemos identificar todos los procesos de la organización, ya sean internos o externos, además de las interacciones que existen entre ellos. Para ellos se suelen utilizar los mapas de procesos.

**6.3.4. Control de la calidad**

Las herramientas de control de calidad mejoran la aplicación de un Sistema de Gestión de la Calidad ISO 9001 en cualquier organización. Es la estructura que organiza evaluaciones, inspecciones, auditorias y revisiones que aseguren que se cumplan las responsabilidades asignadas, se utilicen eficientemente los recursos y se logre el cumplimiento de los objetivos del producto. Tiene la intención de mantener bajo control un proceso y eliminar las causas de los defectos en las diferentes fases del ciclo de vida de un producto.

**6.3.5. Medición y métricas del software**

Medida: proporciona indicación cuantitativa de la extensión, la cantidad, la dimensión, la capacidad o el tamaño de algún atributo de un producto o proceso.

Medición: acto de determinar una medida.

Métrica de software. Es cualquier medida o conjunto de medidas destinadas a conocer o estimar el tamaño u otra característica de un software o un sistema de información, generalmente para realizar comparativas o para la planificación de proyectos de desarrollo. La premisa es que "no podemos mejorar lo que no se puede medir".

**6.3.5.1. El proceso de medición**

Recoger datos que midan el desempeño de cada proceso. Asegurar que los procesos están apropiadamente apoyados.

**6.3.5.2. Métricas del producto**

|  |  |
| --- | --- |
| **TIPO** | **METRICAS** |
| **ANALISIS** | Funcionalidad entregada  Tamaño del sistema  Calidad de la especiación  Métricas para el modelo de diseño  Métricas arquitectónicas  Métricas al nivel de componente  Métricas de diseño de la interfaz  Métricas especializadas en diseño orientado a objetos  Métricas para el código fuente  Se usan para evaluar su complejidad, además la facilidad con que se mantiene y prueba.  Métricas de Halstead  Métricas de complejidad  Métricas de longitud |
| **ESTIMACIONES** | 1. Estimar el costo o el esfuerzo requerido para diseñar, codificar y probar el software  2. predecir el número de errores que se encontrarán durante la prueba.  3. pronosticar el número de componentes, de líneas de código proyectadas, o ambas, en el sistema implementado. |
| **DISEÑO** | Complejidad Estructural en el caso de arquitecturas jerárquicas  Complejidad de datos indica complejidad de la interfaz interna de un módulo  Complejidad del sistema es la suma de las complejidades estructural y de datos.  Métricas para el diseño orientado a objetos  Whitmire describe nueve características distintivas y mensurables de un Diseño OO:  Tamaño  Complejidad  Acoplamiento  Su ciencia  Grado de avance  Cohesión  Primitivismo  Similitud  Volatilidad |
| **PRUEBAS** | Métricas de cobertura de instrucciones y ramas  Métricas relacionadas con los defectos  Efectividad de la prueba  Métricas en el proceso |
| **CALIDAD**  **ISO 25000.** | Portabilidad,  Defectos  Usabilidad  Mantenibilidad  Fiabilidad  Corrección  Eficiencia  Efectividad  Integridad  Seguridad  Disponibilidad |

**6.3.5.3. Análisis de las mediciones**

El análisis del sistema de medición es un método para determinar si un sistema de medición es aceptable y realiza los estudios del sesgo, combinado de la precisión y de la veracidad va a permitirnos conocer la calidad de nuestras mediciones.

**6.4. Mejora de procesos.**

Es común en la actualidad escuchar el término "servicio completo", y este implica que la producción y prestación de bienes y servicios alcancen un grado de conformidad y satisfacción en todos los procesos que componen la cadena de valor, desde los procesos que afectan la calidad del producto o la prestación del servicio, como los procesos que inciden en el soporte, la postventa y los servicios complementarios.

Como es lógico, pensar en que todos los factores que inciden en la prestación de un servicio alcancen un 100% de conformidad es algo sumamente complejo, sin embargo se convierte en una gran oportunidad de mejora, en una filosofía de trabajo bajo la premisa de que **"todo puede hacerse mejor"**.

El modelo de las ocho fases, también conocido como método 8D, corresponde a una metodología sistematizada para la aplicación de mejoras en los procesos, sobre la base de la mejora enfocada. Los procesos susceptibles de abordarse con esta metodología pueden relacionarse con proyectos de:

* Reingeniería de procesos.
* Gestión de la Calidad Total.
* Gestión de operaciones y muchos más.

Las fases que componen la metodología 8D son:

* Formación del grupo de mejora (equipo)
* Definición del problema
* Implementación de soluciones de contención
* Medición y análisis: Identificación de las causas raíces
* Análisis de soluciones para las causas raíces
* Elección e implementación de soluciones raíces (comprobación)
* Prevención de reocurrencias del problema y causas raíces
* Reconocimiento del equipo de mejora

**6.4.1. Calidad de producto y proceso**

***“La calidad es aquello que asegura que se cumplen unos requisitos establecidos”***

En todo proyecto, lo primero que obtenemos son una serie de especificaciones que caracterizan el producto que se quiere desarrollar. Si desarrollamos con éxito, sin desviaciones y entregamos el producto tal y como nos solicitaron habremos conseguido que la calidad del producto sea total.

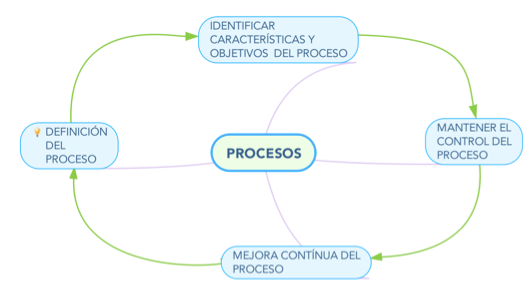
Sin embargo, para poder alcanzar el objetivo final debemos estandarizar los pasos que se deben seguir para la consecución del mismo. Si conseguimos estandarizar dichos pasos de forma que el proceso pueda ser reutilizado para diferentes productos habremos alcanzado el nivel de Calidad del Proceso. De este modo, conseguiremos que nuestro producto final sea siempre igual, con las mismas características en todo momento. Los procesos una vez definidos, deben ser documentados y controlados para poder ser repetidos.

Para que un proceso pueda ser reutilizado debe ser definido de forma que pueda ser medible y debe estar expresado en términos de beneficios para el negocio.

Un proceso constituye un conjunto de actividades diseñadas para cumplir con un objetivo específico. Un proceso toma entradas definidas y las transforma en salidas. Para que un proceso quede definido se deben incluir roles, responsabilidades, herramientas, controles de gestión (como por ejemplo: gestión de riesgos, gestión problemas, gestión de desviaciones) necesarios para alcanzar con éxito las salidas.

Los procesos describen acciones a desarrollar, dependencias, posibles desviaciones, mejoras, revisiones y, además deben ser medibles. Todo esto conforma un resultado específico en el que se debe involucrar al cliente para conocer si se han resuelto sus expectativas.

El ciclo de vida del proceso se puede visualizar en el siguiente diagrama:



Los pasos para llevar a cabo la definición de proceso son:

Identificar Características: Diseño del proceso y definición de objetivos.

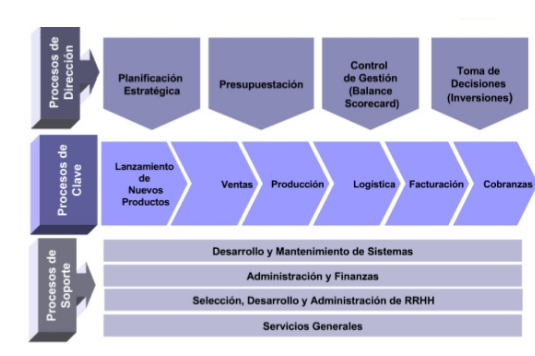
* Control del Proceso: Documentación y retroalimentación.
* Entradas: Definición de actividades, métricas, identificación de roles, desarrollo de procedimientos.
* Salidas: Entrega del producto final, incluyendo informes y revisiones del proceso.
* Mejoras: Aplicación de mejoras sobre el proceso según desviaciones y reducción de dependencias externas. Retroalimentación.

Para adaptarnos a los cambios, la calidad del proceso debe ser dinámica y estar sujeta a variaciones; dichas variaciones suponen la Mejora del Proceso. Para poder llevar a cabo una mejora del proceso éste debe revisarse para comprobar si los objetivos definidos previamente se están cumpliendo, en caso de que no sea así, debemos llevar acciones de mejora.

La calidad no solo debe incidir en el producto a entregar sino también en el proceso que ayuda a desarrollarlo. Por tanto, ambos conceptos (Calidad del Proceso y Calidad del Producto) forman una parte muy importante del proyecto, pese a que muchas veces no se le dé la importancia que merece, y es aún más relevante la acción de calidad en la unidad organizativa de cada proyecto.

**6.4.2. Clasificación de los procesos**

Los procesos de la organización se pueden dividir en distintos tipos en función de la repercusión directa que tienen sobre el producto o servicio ofrecido.

****

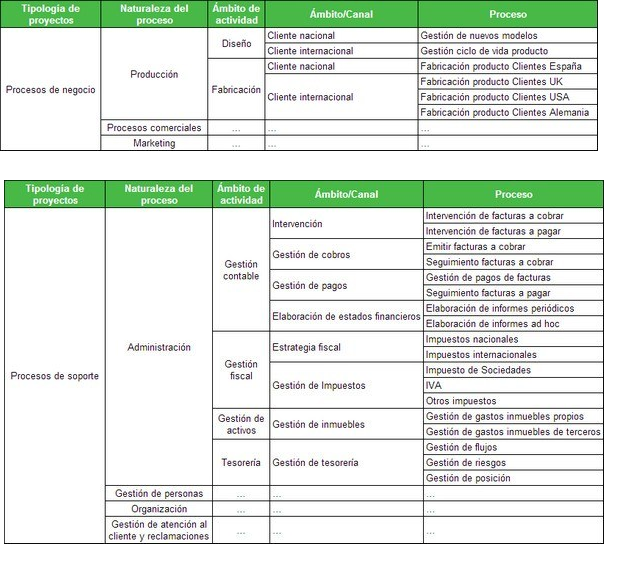
**6.4.3. Medición del proceso**

La mejora de procesos es una de las actividades que cualquier empresa debe afrontar, acompañando a las acciones del día a día (ventas, marketing, distribución, gestión de almacén, etc.). Se trata de hacer las cosas mejor para que el negocio siga siendo competitivo y pueda ofertar productos o servicios de valor para los clientes y, por tanto, generar ventas.

Una de las herramientas de la mejora de procesos para mejorar la gestión es la medición de los procesos, capturando datos del desempeño, eficiencia, eficacia, calidad de los mismos para transformarlos en información y determinar puntos de mejora. ¿Cómo puede ayudar la medición de los procesos a mejorar la gestión de una pyme?

**6.4.4. Análisis y modelado de procesos**

A modo de referencia, el número recomendado de niveles a usar en esta división de procesos en subprocesos no debería ser mayor que cuatro, estando en el quinto nivel los procesos a más bajo nivel. En el siguiente cuadro, se pueden ver dos ejemplos de división de proyectos de tipología negocio y soporte, desde la visión macro hasta la visión de proceso:

****

**6.4.5. Cambio en los procesos**

El proceso de cambio abarca todas las actividades dirigidas a ayudar a la organización para que adopte exitosamente nuevas actitudes, nuevas tecnologías y nuevas formas de hacer negocios. La administración efectiva del cambio, permite la transformación de la estrategia, los procesos, la tecnología y las personas para reorientar la organización al logro de sus objetivos, maximizar su desempeño y asegurar el mejoramiento continuo en un ambiente de negocios siempre cambiante.

Un proceso de cambio ocurre de forma muy eficiente si todos están comprometidos con él. En tanto, para que las personas se comprometan, ellas no pueden ser “atropelladas” por el proceso, como si fueran algo lejano del mismo, por que no son. En la verdad, el cambio ocurre a través de las personas. Y, para que se considere a las personas como parte del proceso de cambio es necesario conocer sus valores, sus creencias, sus comportamientos.

**6.4.6. El marco de trabajo para la mejora de procesos CMMI**

CMMI es un marco de mejora de procesos en el campo del software y sistemas que dispone actualmente de tres modelos de aplicación:

* CMMI para el desarrollo.
* CMMI para los servicios.
* CMMI para las adquisiciones.

Estos modelos, conocidos como constelaciones CMMI, constituyen conjuntos de buenas prácticas que se han ido consolidando en la industria y servicios desde que se hizo público y aplicable el primer modelo de madurez publicado en 1993. Actualmente se están aplicando en varios miles de organizaciones internacionales tanto públicas como privadas y con un amplio rango de tamaños y de tipos de aplicación de productos y servicios.

SPICE (Software Process Improvement and Capability dEtermination) es el nombre adoptado por  la serie de normas internacionales ISO/IEC 15504. Esta norma es un modelo para la evaluación de los procesos. Tiene antecedentes en otros modelos de referencia definidos en las mismas normas internacionales ISO para las TI conteniendo descripciones detalladas de los procesos, siendo la base para la recogida de evidencias para la evaluación.

* La norma ISO/IEC 15504 evoluciona con el tiempo y consta de las siguientes partes:
* ISO/IEC 15504-1: 2004  Definición de conceptos y tecnología.
* ISO/IEC 15504-2: 2003  Definición de los requisitos de evaluación y de los modelos de referencia de procesos y de evaluación.
* ISO/IEC 15504-3: 2004  Guía para la Evaluación.
* ISO/IEC 15504-4: 2004  Guía de aplicación de los resultados para la mejora de procesos.
* ISO/IEC 15504-5: 2008  Define un ejemplo de modelo de evaluación de procesos para software.
* ISO/IEC 15504-6: 2008  Define un ejemplo de modelo de evaluación de procesos para sistemas.
* ISO/IEC 15504-7: 2008  Define los niveles de madurez y su evaluación.

**6.5. Gestión de configuraciones.**

La gestión de la configuración (y de los activos) es el conjunto de procesos destinados a asegurar la calidad de todo producto obtenido durante cualquiera de las etapas del desarrollo de un sistema de información (SI), a través del estricto control de los cambios realizados sobre los mismos y de la disponibilidad constante de una versión estable de cada elemento para toda persona involucrada en el citado desarrollo.

La interfaz **gestión de la configuración** definida en [MÉTRICA](https://es.wikipedia.org/wiki/M%C3%89TRICA) v3, los elementos de configuración del [software](https://es.wikipedia.org/wiki/Software) incluyen:

* Ejecutables
* Código Fuente
* Modelos de datos
* Modelos de procesos
* Especificaciones de requisitos
* Pruebas

Y para cada uno de estos elementos se almacenará al menos:

* Nombre
* Versión
* Estado
* Localización

**6.5.1. Planificación de la gestión de configuraciones**

El plan de gestión de la configuración constituye un elemento clave para establecer y garantizar la integridad del producto durante el proceso de desarrollo. Es considerado como parte de las actividades que se cubren en el área de proceso de Configuration Management (CM), en particular para el cumplimiento de las prácticas genéricas asociadas.

La primera sección es la introducción con el resumen del plan, actividades, audiencia y cómo usarlo de manera que el usuario tenga una comprensión clara del mismo.

* Propósito, con las necesidades y la audiencia que se supone tiene el plan.
* Alcance, describe la aplicación del plan, limitaciones y suposiciones, resumen del proceso de desarrollo de software y cómo las funciones y actividades de CM se integran en el proyecto.
* Definiciones, presenta los términos clave utilizados en el documento.
* Referencias a documentos, estándares, procedimientos (internos y externos) utilizados en el plan. Identifica dónde pueden encontrarse los documentos.
* La segunda sección se relaciona la organización del equipo de CM y asignación de responsabilidades a equipos e individuos
* Estructura organizacional del equipo de CM y cómo se integra en la estructura con respecto a otros grupos como: equipo del proyecto, equipo de aseguramiento de calidad y alta gerencia, clientes, usuarios y subcontratistas involucrados en actividades de CM. Describe la composición del Comité de control de cambios y los equipos de auditoria y revisión.
* Responsabilidades y obligaciones de los individuos involucrados en actividades de CM.
* Relación de actividades de CM durante las diferentes fases del ciclo de vida.
* Descripción de las interacciones con otros grupos del proyecto como proveedores y subcontratistas.
* Responsabilidades de proveedores, subcontratistas y otras organizaciones con respecto a las funciones de CM.

La tercera sección es la que considera las actividades para la gestión de la configuración. Establece las tareas y funciones requeridas para la gestión de la configuración de acuerdo con el alcance del plan. Describe las actividades principales de CM y cómo se realizan en el proyecto.  
  
  
Identificación  
Describe como identificar, nombrar y documentar las características físicas y funcionales de los componentes de configuración. Una vez identificados los componentes son incorporados a un ambiente controlado.

* Identificación de componentes de configuración, que serán controlados por las actividades de CM. Define una lista de los componentes de configuración del proyecto, que puede mostrar en forma de estructura de árbol las interdependencias que existan.
* Nombres de componentes de configuración. Especifica el sistema para identificación, las convenciones de nombres, número de versiones y letras utilizadas para identificar los componentes.
* Adquisición de componentes de configuración. Describe cómo los componentes serán almacenados, cómo serán controlados los accesos, detalles de las bibliotecas de configuración y los procedimientos para registro de entrada y salida de componentes desde la biblioteca.

Control  
Describe los procesos para gestión de cambios como son: inicio, evaluación, implantación, revisión, aprobación y establecimiento de los cambios.

* Identificación de cambios. Describe cómo iniciar el cambio. Un cambio puede ser producto de una falla o problema o como resultado de una mejora o característica nueva. Describe los procedimientos a seguir para iniciar una solicitud de cambio o reporte de problema para comenzar la actividad de control de cambios.
* Evaluación de cambios. Describe cómo se realiza la evaluación de una solicitud de cambio. Detalla cómo manejar el análisis y clasificación del problema, así como la preparación de la persona que realiza la evaluación.
* Administración de cambios.  Describe cómo se procesa la solicitud de cambio. Establece claramente el proceso para recepción de la solicitud, enviar a evaluación, reunir al comité de control de cambios, procesar las solicitudes de cambio y demás.
* Implantación de cambios. Describe cómo, una vez que los cambios son aprobados, seleccionar al equipo de implantación, verificar y validar los cambios y enviarlos a la nueva línea base.
* Comité de control de cambios. Describe cómo funciona el comité como grupo para la decisión de aprobación o no de los cambios. Define quién tiene la autoridad en el comité y cómo se solucionan los conflictos.

Reporte de estado  
Registro del estado de los componentes de configuración y reporte a las personas interesadas.

* Identificación de necesidades de información. Describe los requerimientos de información del proyecto. Qué se requiere, por quién, frecuencia de los reportes, principalmente
* Mecanismo para obtener información. Describe cómo se obtiene la información. Es deseable que la información se encuentre en una base de datos que es alimentada por la persona que realiza la actividad y no por la que procesa la misma para generar el reporte.
* Reporte, contenido y frecuencia. Describe los diferentes reportes que se generan, su frecuencia y contenido de los reportes.
* Acceso a los datos para reporte de estado. Describe cómo se puede acceder la información para generar los reportes.
* Método de difusión de la información de los reportes de estado. Describe cómo y a quién se le distribuye la información de los reportes de estado.
* Detalle de liberaciones. Detalla información sobre qué es lo que contienen las liberaciones, a quién se libera y cuándo, en qué medio se realiza, problemas que se pueden presentar en las liberaciones del producto y forma de corregirlos, instrucciones de instalación, entre otras.

Auditorías   
Describe los tipos de auditorías que se realizan, el procedimiento que se utiliza, la frecuencia y autoridad para realizar las auditorías.

* Auditorías a realizar. Describe los diferentes tipos de auditorías que se van a realizar y cuándo serán realizadas. Normalmente se consideran: auditorías físicas, auditorías funcionales, auditorías a subcontratistas y auditorías externas.
* Componentes de configuración a auditar.
* Procedimientos de auditorías. Describe el procedimiento a utilizar en cada tipo de auditoría: quién la realiza, documentos requeridos, cómo debe llevarse a cabo y el formato para reporte de auditoría.
* Actividades de seguimiento a auditorías. Describe las actividades que se deben realizar posterior a la auditoría para la solución de las no conformidades.

Otras secciones del plan consideran:

* Control de interfaces, donde se describe la coordinación de los cambios de componentes de configuración con los cambios a los elementos de interfaz fuera del alcance del plan como los elementos de hardware, paquetes, software de apoyo y demás. Debe identificar cada uno de los elementos externos, la naturaleza de la interfaz, los grupos afectados, cómo serán controlados estos elementos y cómo serán aprobados para formar parte de la línea base.
* Control de subcontratistas, que describe las actividades necesarias para incorporar los componentes desarrollados fuera del ambiente del proyecto, en particular los componentes de proveedores. Describe las funciones y actividades de gestión de la configuración que debe realizar el subcontratista, mecanismos para asegurar que se cumplen, procedimiento para auditar los componentes entregados y los componentes que se van a suministrar. Adicionalmente describe cómo serán recibidos, probados y puestos bajo control de la configuración estos componentes y cómo se procesarán e implantarán las solicitudes de cambio.

Finalmente el plan integra las actividades en relación con el proyecto y el ciclo de vida del producto que contiene: 

* Calendario de CM. Describe la secuencia de actividades, la relación e interdependencia con el ciclo de vida del proyecto y las fechas críticas del proyecto. Identifica el ciclo de vida del proyecto, las diferentes fechas dónde se establecerán las líneas base y el calendario de las diferentes auditorías.
* Recursos para CM. Identifica las herramientas de software, técnicas, equipos, personal y capacitación requerida para la implantación de las actividades de gestión de la configuración especificadas.
* Mantenimiento del plan. Describe las actividades requeridas para mantener el plan actualizado durante el ciclo de vida del proyecto, el mecanismo para monitorizar y sincronizar con las actividades del proyecto y la persona responsable de estas actividades.

**6.5.2. Gestión del cambio**

El cambio es inherente al software computacional y genera confusión entre los ingenieros de software involucrados en un proyecto, dicha confusión surge cuando los cambios no se analizan antes de realizarlos, no se registran antes de implementarlos, no se reporta a quienes deben saberlo o no se mantiene un control sobre ellos.

Esto es la gestion de la configuacion: "El arte de coordinar el desarrollo de software para minimizar la confusión", GCS o GC .

La GC es una actividad que se desarrolla durante todo el proceso de desarrollo ya que no sabemos en que momento se originara un cambio, las actividad en este proceso se desarrollan para:

* Identificar el Cambio
* Controlar el Cambio
* Garantizar que el cambio se realizara de manera adecuada
* Reportar los cambios a todos los interesados

**6.5.3. Gestión de versiones y entregas**

En ITIL[[23]](#footnote-24) Las principales actividades de la Gestión de Entregas y Despliegues se resumen en:

* Establecer una política de planificación para la implementación de nuevas versiones.
* Desarrollar o adquirir de terceros las nuevas versiones.
* Implementar las nuevas versiones en el entorno de producción.
* Llevar a cabo los planes de back-out o retirada de la nueva versión si esto fuera necesario.
* Actualizar la DML, el DS y la CMDB.
* Comunicar y formar a los clientes y usuarios sobre las funcionalidades de la nueva versión.

**6.5.4. Construcción del sistema**

La construcción del Sistema de Información (CSI) tiene como objetivo final la construcción y prueba de los distintos componentes del sistema de información, a partir del conjunto de especificaciones lógicas y físicas del mismo, obtenido en el proceso de Diseño del Sistema de Información (DSI). Se desarrollan los procedimientos de operación y seguridad y se elaboran los manuales de usuario final y de explotación, estos últimos cuando proceda.

Para conseguir dicho objetivo, se recoge la información relativa al producto del diseño Especificaciones de construcción del sistema de información, se prepara el entorno de construcción, se genera el código de cada uno de los componentes del sistema de información y se van realizando, a medida que se vaya finalizando la construcción, las pruebas unitarias de cada uno de ellos y las de integración entre subsistemas.

Si fuera necesario realizar una migración de datos, es en este proceso donde se lleva a cabo la construcción de los componentes de migración y procedimientos de migración y carga inicial de datos.

Como resultado de dicho proceso se obtiene:

* Resultado de las pruebas unitarias.
* Evaluación del resultado de las pruebas de integración.
* Evaluación del resultado de las pruebas del sistema.
* Producto software:
  + Código fuente de los componentes.
  + Procedimientos de operación y administración del sistema.
  + Procedimientos de seguridad y control de acceso.
  + Manuales de usuario.
  + Especificación de la formación a usuarios finales.
  + Código fuente de los componentes de migración y carga inicial de datos.
  + Procedimientos de migración y carga inicial de datos.
  + Evaluación del resultado de las pruebas de migración y carga inicial de datos.

**6.5.5. Herramientas CASE para gestión de configuraciones**

La gestión de configuración de software (GCS) se encuentra en el núcleo de todos los entornos CASE. Las herramientas pueden ofrecer su asistencia en las cinco tareas principales de GCS: identificación, control de versiones control de cambios, auditoria y contabilidad de estados. La base de datos CASE proporciona un mecanismo para identificar todos los elementos de configuración y relacionarlo con otros elementos; un acceso sencillo a los elementos de configuración individuales facilita el proceso de auditoria; las herramientas de comunicación CASE pueden mejorar enormemente la contabilidad de estados (ofreciendo información acerca de los cambios a todos aquellos que necesiten conocerlos).

1. Las organizaciones son entes creados por el hombre, son un artificio adaptativo que permite no sólo sus fines sino también regular la conducta individual y social. Es decir, es un sistema social o conjuntos de personas con relación de interdependencia, deliberadamente construido para alcanzar fines específicos. Un caso particular de organizaciones son las empresas. [↑](#footnote-ref-2)
2. Riesgo es la ***posibilidad*** de que se produzca un contratiempo o una desgracia, de que alguien o algo sufra perjuicio o daño. [↑](#footnote-ref-3)
3. La Ingeniería es aquella disciplina que se ocupa del estudio y de la aplicación de los conocimientos que de este y de la experiencia resultan, para que a través de diseños, técnicas y problemas puedan ser resueltos los diferentes problemas que afectan a la humanidad. [↑](#footnote-ref-4)
4. Una parte interesada o interesados​​​ (del inglés stakeholder) hace referencia a una persona, organización o empresa que tiene interés en una empresa. [↑](#footnote-ref-5)
5. Una parte interesada o interesados​​​ (del inglés stakeholder) hace referencia a una persona, organización o empresa que tiene interés en una empresa. [↑](#footnote-ref-6)
6. Utopía es la idea, ideación o representación de una civilización ideal, fantástica, imaginaria e irrealizable. [↑](#footnote-ref-7)
7. El grado de acoplamiento mide la interrelación entre dos módulos, según el tipo de conexión y la complejidad de la interface. [↑](#footnote-ref-8)
8. Es necesario lograr que el contenido de cada módulo tenga la máxima coherencia. Para que el nº de módulos no sea demasiado elevado y complique el diseño se tratan de agrupar elementos afines y relacionados en un mismo módulo. [↑](#footnote-ref-9)
9. Para facilitar los cambios, el mantenimiento y la reutilización de módulos es necesario que cada uno sea comprensible de forma aislada. [↑](#footnote-ref-10)
10. Las capas representan una separación lógica dentro de la aplicación. En caso de que la lógica de la aplicación se distribuya físicamente en servidores o procesos independientes, estos destinos de implementación físicos independientes se conocen como niveles. Es posible, y bastante habitual, tener una aplicación de N capas que se implemente en un solo nivel. [↑](#footnote-ref-11)
11. Un sistema de tiempo real es un software cuyo correcto funcionamiento depende de los resultados producidos por el mismo y del instante de tiempo en el que se producen estos resultados. [↑](#footnote-ref-12)
12. Una interfaz de usuario, en inglés user interface, a menudo conocida por la abreviatura UI, hace referencia a la **interfaz con la que las personas interaccionan con las máquinas.** [↑](#footnote-ref-13)
13. Las responsabilidades del Cliente / Product Owner (que puede ser interno o externo a la organización que realiza el desarrollo del producto) son: Conocer el mercado y los comportamientos de los clientes / usuarios finales, con muy buena visión de Negocio. Ser el representante de todas las personas interesadas (stakeholders) para conseguir una buena definición de los objetivos del producto o proyecto y de los resultados esperados. Encargarse de que exista una priorización clara del trabajo a hacer. De este modo, dirige los resultados del proyecto para maximizar su ROI (Return Of Investment). Colaborar con el equipo para planificar, revisar y dar detalle a los objetivos de cada iteración. [↑](#footnote-ref-14)
14. Corresponden a patrones de diseño de software que solucionan problemas de creación de instancias. [↑](#footnote-ref-15)
15. Son los patrones de diseño software que solucionan problemas de composición (agregación) de clases y objetos. [↑](#footnote-ref-16)
16. Se definen como patrones de diseño software que ofrecen soluciones respecto a la interacción y responsabilidades entre clases y objetos, así como los algoritmos que encapsulan. [↑](#footnote-ref-17)
17. La reingeniería es establecer secuencias en interacciones nuevas en procesos administrativos y regulatorios. [↑](#footnote-ref-18)
18. La verificación es el proceso que consiste en confirmar que el producto en verdad encaja en lo descrito o en sus especificaciones en términos de funcionalidad. [↑](#footnote-ref-19)
19. La validación es un proceso que toma lugar en la etapa final de desarrollo de los productos, en este proceso, el producto mismo es puesto a prueba por el equipo encargado de realizarlas. Con esto se asegura que el producto alcance el grado de funcionalidad deseado. [↑](#footnote-ref-20)
20. Identificar los componentes de software y documentos que deben ser verificados. Describir las estrategias de verificación que serán usadas. Identificar los recursos necesarios y proporcionar una estimación de esfuerzo para realizar la verificación. [↑](#footnote-ref-21)
21. Estándar: Que es lo más habitual o corriente, o que reúne las características comunes a la mayoría. [↑](#footnote-ref-22)
22. Estandarización: Significa que los símbolos convencionales se usan en todos los diagramas de flujo para prescribir el sistema y que en la documentación se usen formas estandarizadas. [↑](#footnote-ref-23)
23. La Biblioteca de Infraestructura de Tecnologías de Información. [↑](#footnote-ref-24)